

DDT Debugging Techniques

Carlos Rosales
carlos@tacc.utexas.edu

Scaling to Petascale 2010
July 7, 2010



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Debugging Parallel Programs

- Usual problems
 - Memory access issues
 - Special cases not accounted for in the code
 - Wrong arguments to functions
 -



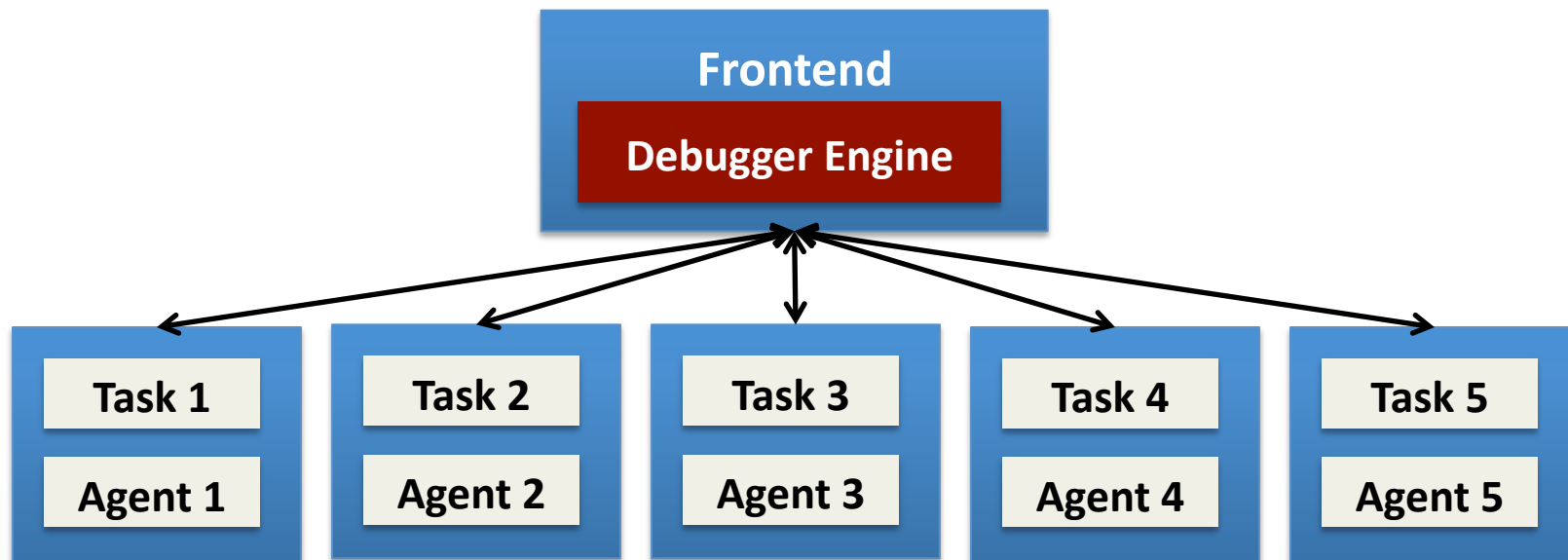
- New set of problems
 - Work distribution and partitioning
 - Arguments to parallel function calls
 - Racing conditions (shared memory / OMP)
 - Deadlocks (MPI)
 - ...

The failure of printf/write

- Usually buffered by system
 - May not appear on screen/file even after executed
 - If called from all tasks they create too much data
- When un-buffered (or flushed)
 - Not ordered (because of delays between tasks)
 - Code must be “instrumented” manually
- May change code behavior (introduces delays)
- Time Consuming
- Inaccurate

Parallel Debuggers

- Parallel applications themselves
- Overhead is of concern for large-scale runs
- Frontend + distributed *agents* attached to each task
- In DDT agents are instances of a modified GDB serial debugger



About DDT

Allinea Distributed Debugger Tool

www.allinea.com

- Multiplatform
- Supports all MPI distributions
- Capable of debugging large scale OMP/MPI
- Comprehensive
 - Memory checking
 - MPI message tracking
- Useful Graphical User Interface

setting up a debug session

CONFIGURATION



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Using DDT

- Compile your code using the standard debug flags:

```
% mpicc -g -O0 ./srcFile.c
```

```
% mpif90 -g -O0 ./srcFile.f90
```

- Load the DDT module:

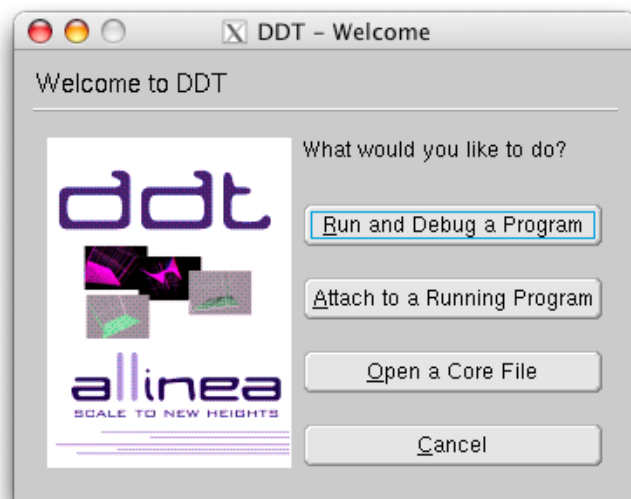
```
% module load ddt
```

```
% module list
```

- Start up DDT:

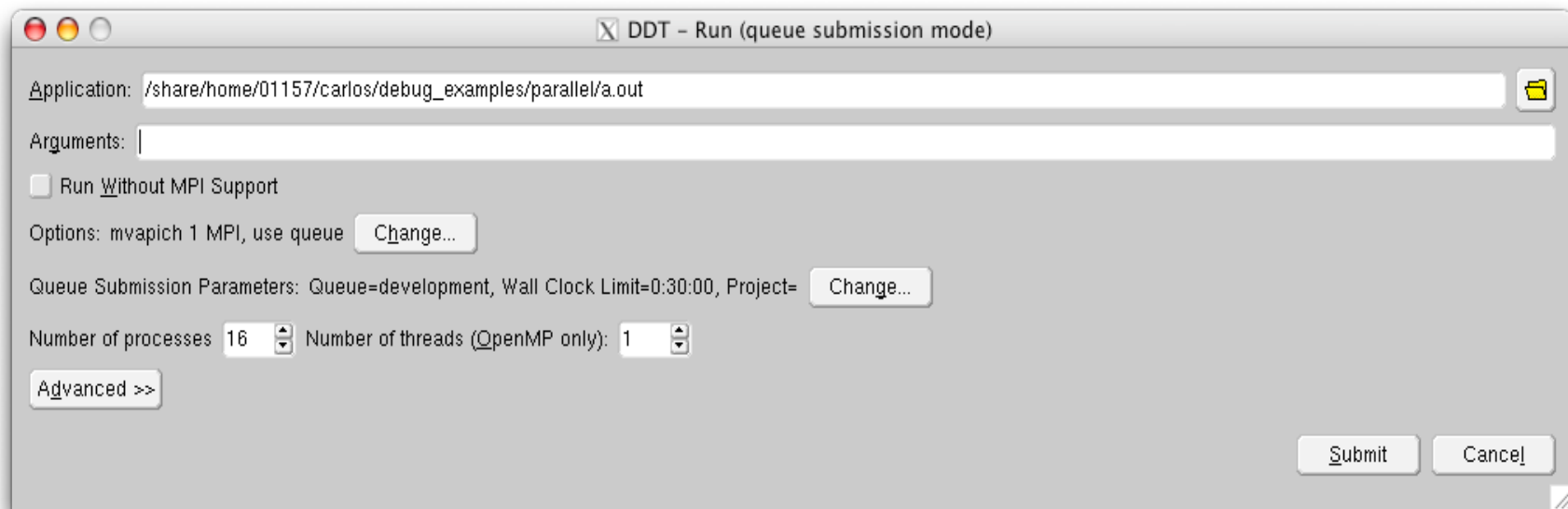
```
% ddt ./a.out
```

Configuration: Welcome Screen



- Three ways of starting a debug session
 - Run and debug
 - Attach to a running program
 - Open core dump file

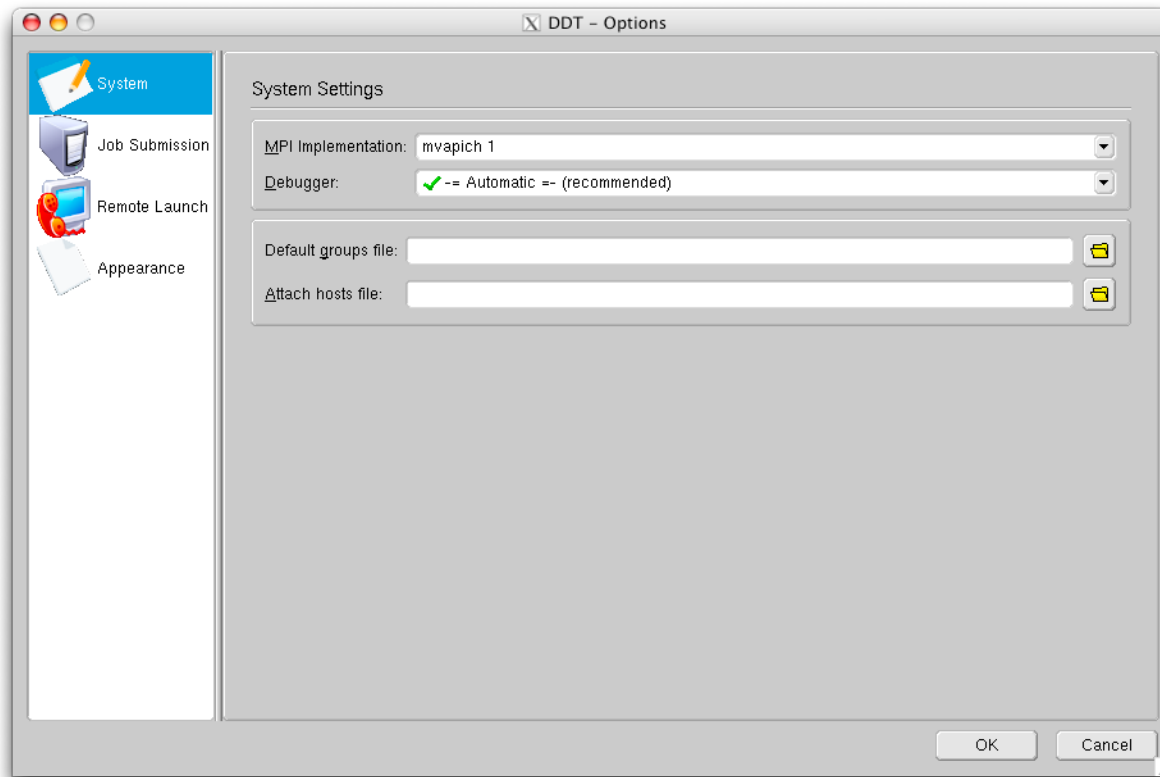
Configuration: Job Submission



The image shows a macOS-style dialog box titled "DDT - Run (queue submission mode)". It contains the following fields and controls:

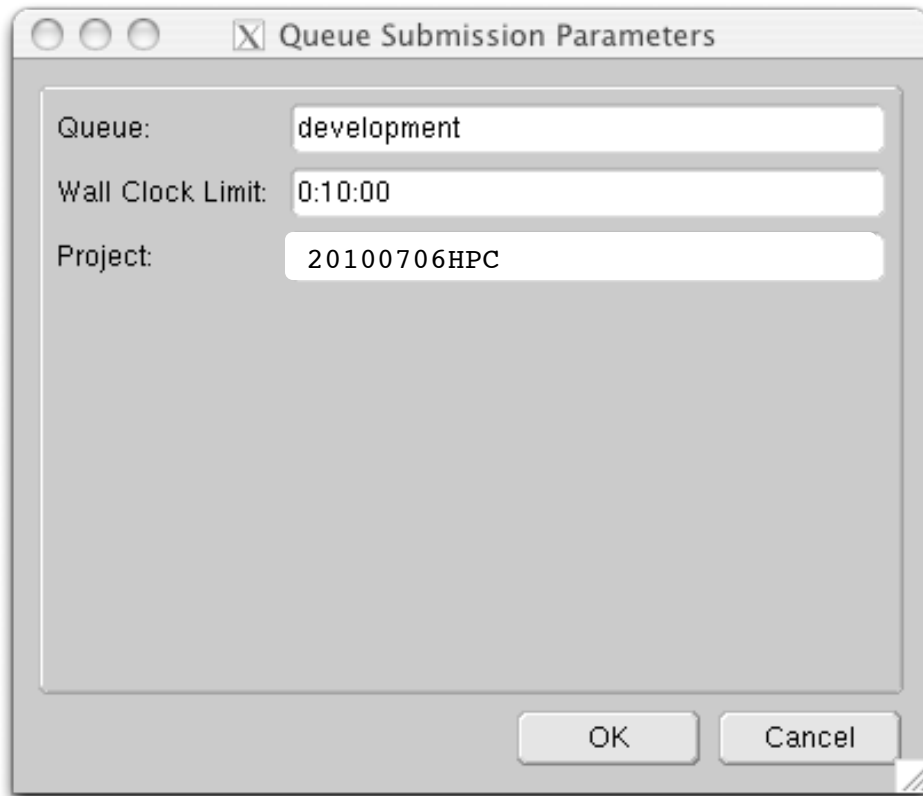
- Application:** A text field containing the path `/share/home/01157/carlos/debug_examples/parallel/a.out` with a folder icon on the right.
- Arguments:** An empty text field.
- Run Without MPI Support:** A checkbox that is currently unchecked.
- Options:** A label followed by the text "mvapich 1 MPI, use queue" and a "Change..." button.
- Queue Submission Parameters:** A label followed by the text "Queue=development, Wall Clock Limit=0:30:00, Project=" and a "Change..." button.
- Number of processes:** A spinner box set to the value 16.
- Number of threads (OpenMP only):** A spinner box set to the value 1.
- Advanced >>** A button to expand the configuration options.
- Submit** and **Cancel** buttons at the bottom right.

Configuration: Options



- Choose the correct version of MPI
 - mvapich 1
 - mvapich 2
 - openMPI
- Leave Debugger on the Automatic setting

Configuration: Queue Parameters



Queue Submission Parameters

Queue: development

Wall Clock Limit: 0:10:00

Project: 20100706HPC

OK Cancel

- Choose the queue
- Set the Wall Clock Limit (H:MM:SS)
- Set your project code - for this training class use 20100706HPC

Configuration: Memory Checks

DDT - Run (queue submission mode)

Application: /share/home/01157/carlos/debug_examples/parallel/a.out

Arguments:

☐ Run Without MPI Support

Options: mvapich 1 MPI, use queue [Change...](#)

Queue Submission Parameters: Queue=development, Wall Clock Limit=0:30:00, Project= [Change...](#)

Number of processes: 16 Number of threads (OpenMP only): 1

<< Advanced

Plugins: No plugins found. To install a plugin, copy its ".xml" file into the ddt/plugins directory. The documentation contains more information about writing and installing plugins.

MPIRun Arguments:

Extra environment variables:
(for MPI script and application)

☐ Pause when the program reaches exit or _exit

☒ Pause when the program reaches abort or has a fatal MPI error

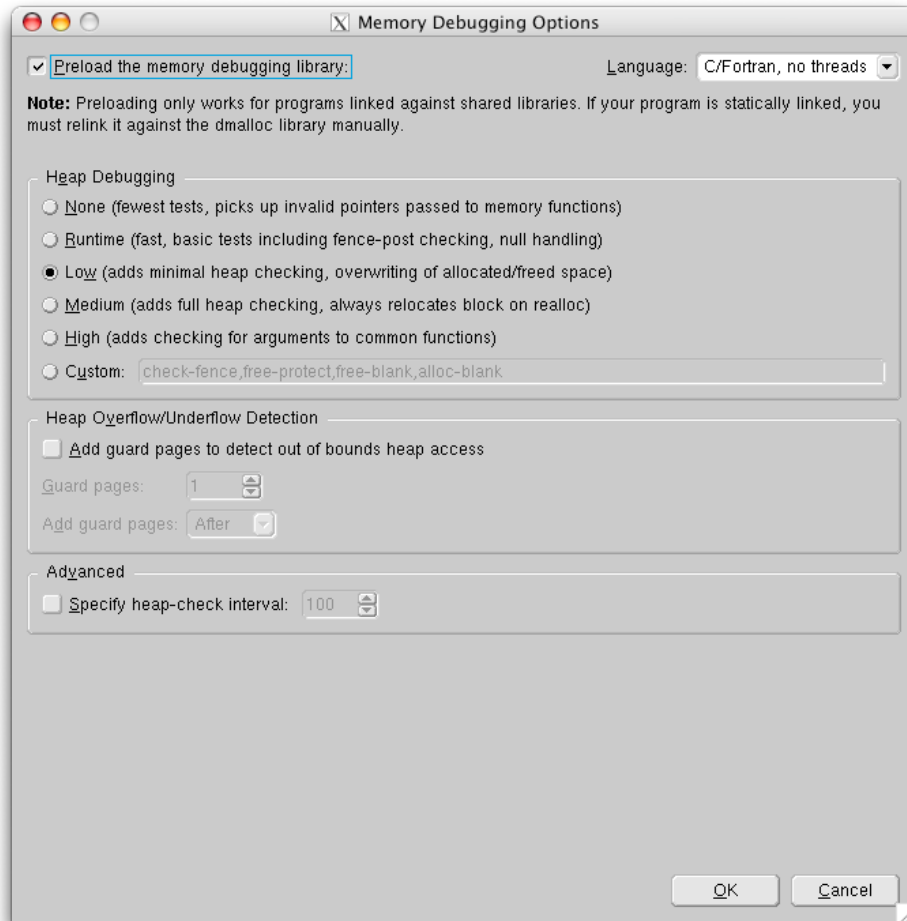
Input file:

☒ Enable Memory Debugging [Settings...](#)

[Submit](#) [Cancel](#)

- Open the Advanced tab.
- Enable Memory Debugging (bottom left check box)
- Open the Memory Debug Settings

Configuration: Memory Options



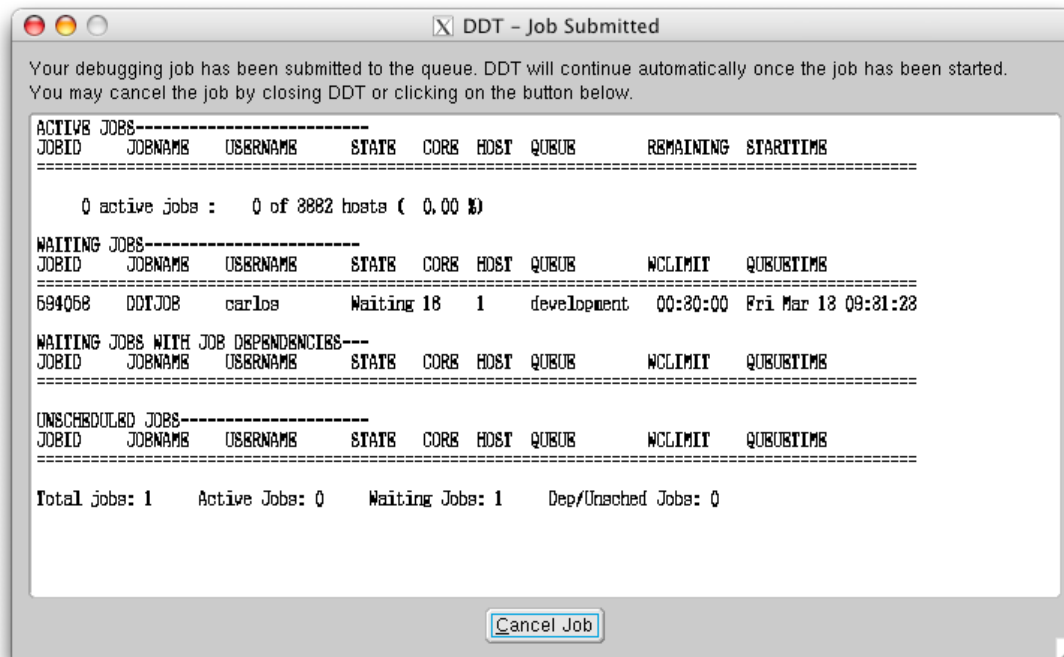
- Change the Heap Debugging option from the default **Runtime** to **Low**
- Even the option None provides some memory checking
- Leave Heap and Advanced unchecked unless you really know that you need them...

Memory Checks and Running Time

- The higher the level of memory checking the longer the execution time
- Runtime overhead is NOT LINEAR with the level of memory checking
- Often a low level of memory checking is sufficient to debug non-pathological issues

Job Queuing

Add any necessary arguments to the program. Click the Submit button.
A new window will open:



The job is submitted to the specified queue.

An automatically refreshing job status window appears.

The debug session will begin when the job starts.

DDT: The debug session

The screenshot shows the Allinea Distributed Debugging Tool (DDT) v2.3.1 interface. The top bar contains menu items (Session, Control, Search, View, Help) and a toolbar with icons for running, stepping, and other debugging actions. Below the toolbar is a 'Process controls' bar with a 'Current Group' dropdown set to 'All' and radio buttons for 'Group', 'Process', and 'Thread'. A 'Process groups window' displays a tree view with 'All' (16 processes), 'Root' (0), and 'Workers' (15). The 'Project Navigator' on the left shows a tree of 'Project Files', 'Source Tree', 'Header Files', and 'Source Files'. The 'Code window' in the center displays the source code for 'deadlock_mem.c'. The 'Variable window' on the right shows 'Local Variables' with 'nprocs' set to 4210165. The 'Stack view and output window' at the bottom left shows a stack trace for 'main (deadlock_mem.c:30)'. The 'Evaluation window' at the bottom right has tabs for 'Expression' and 'Value'. Arrows point from labels to these specific components.

Process controls

Process groups window

Project navigation window

Code window

Variable window

Stack view and output window

Evaluation window

Right click on a line to set a breakpoint.

Most operations available as buttons in the process controls bar

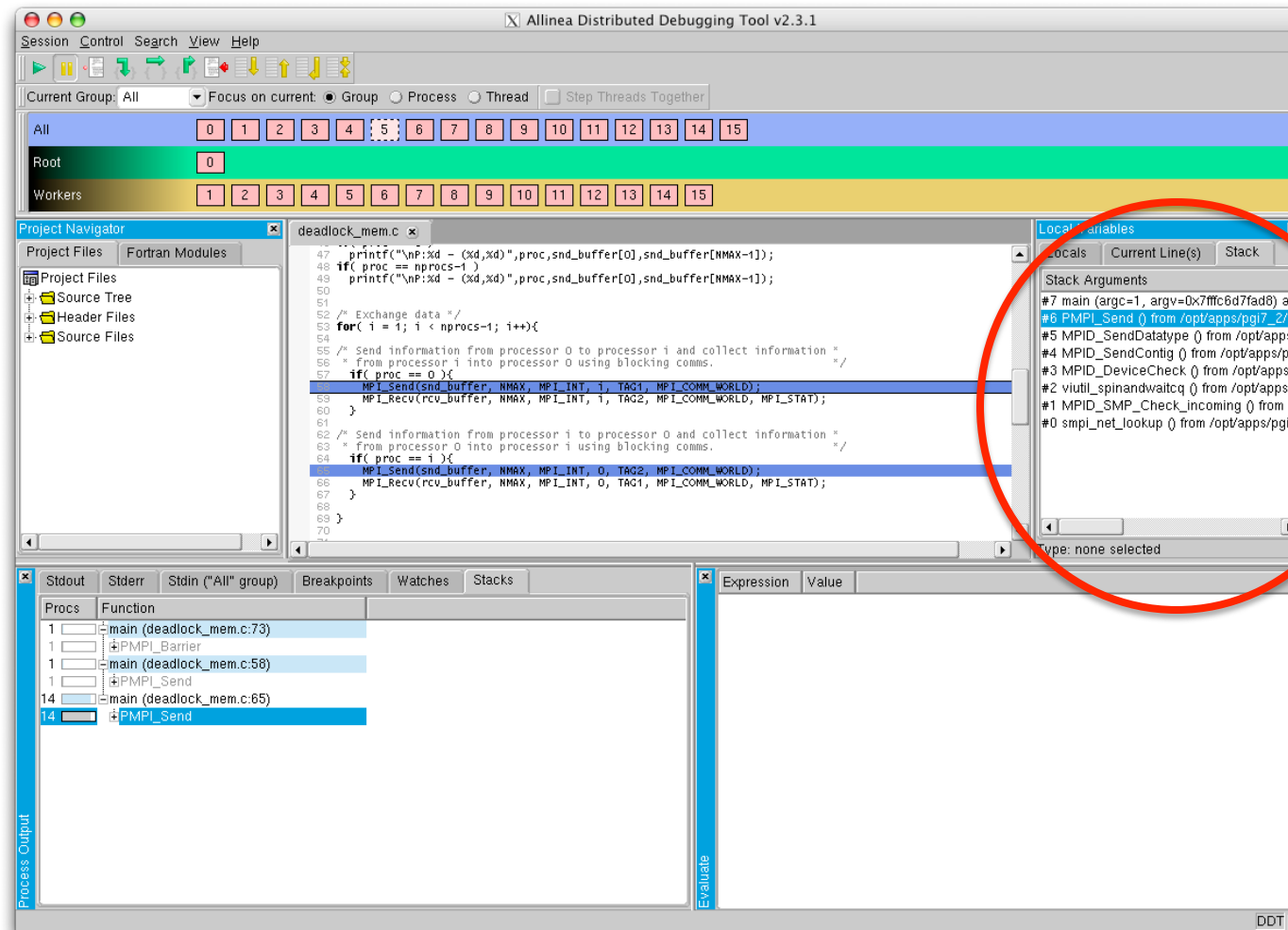
focus, breakpoints and watches

PROCESS CONTROL



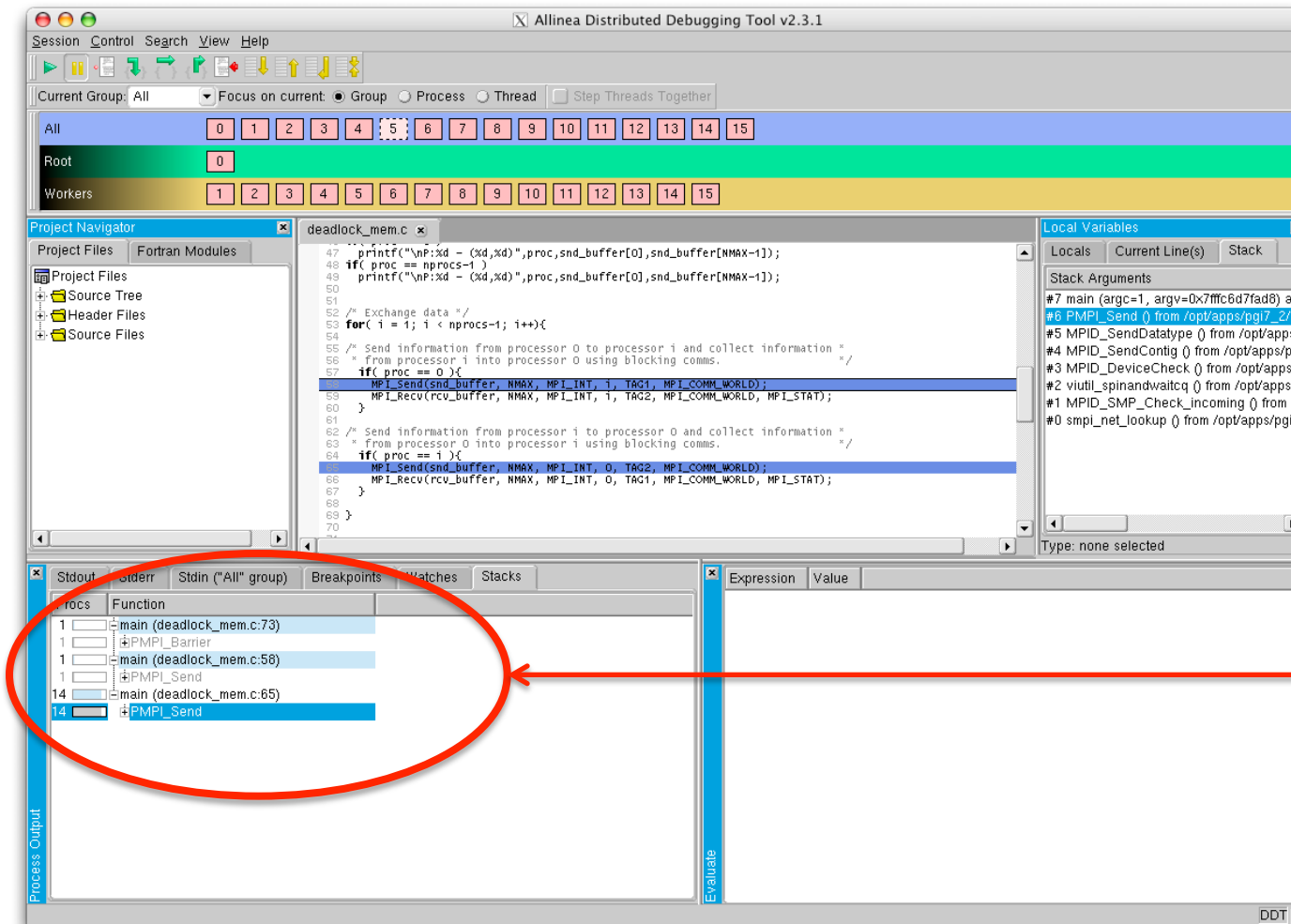
THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Stack Window



Local variables for the currently selected line and stack information is visible here

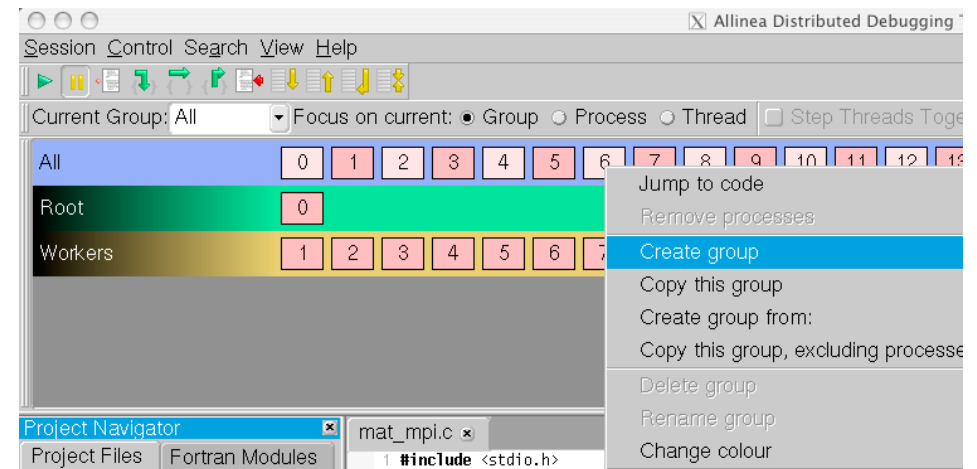
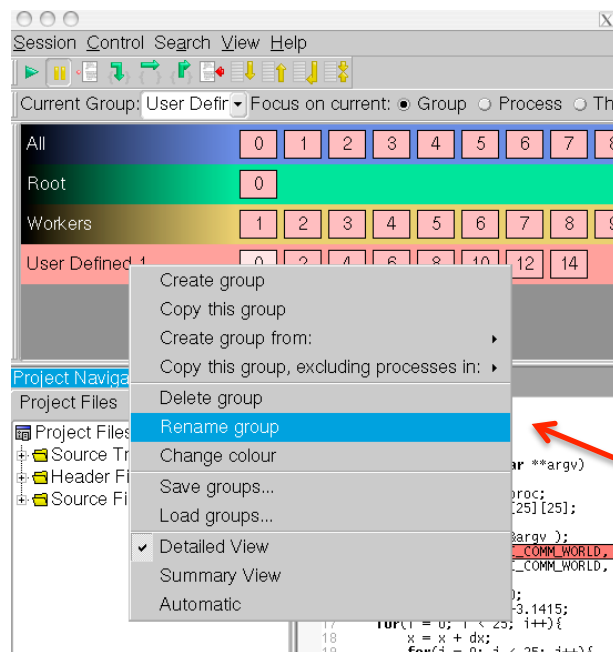
Parallel Stack View



- Shows position of each thread / process in the code.
- Hover over any function to see a list of processes at that position.

Creating Process Groups

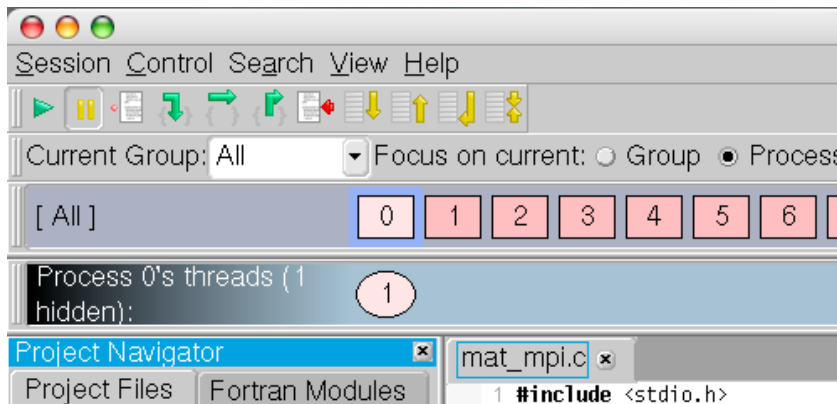
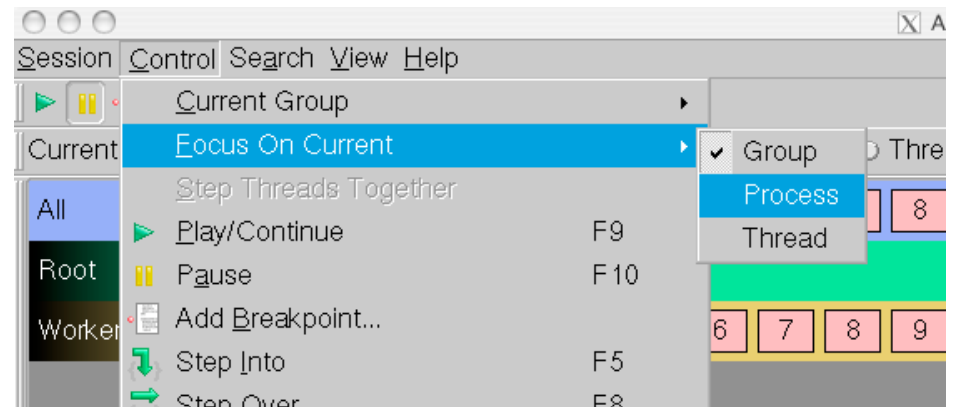
- Ctr-Click to select
- Control->Create Group



- Group is created from selection with default name
- Right-Click the group to edit properties

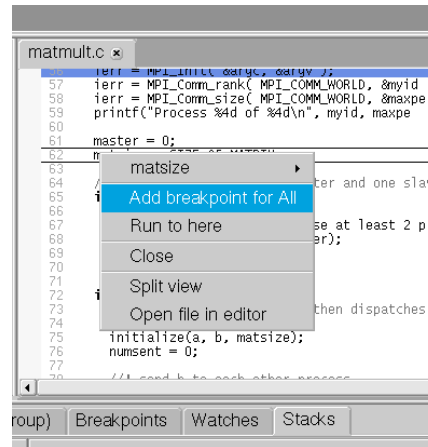
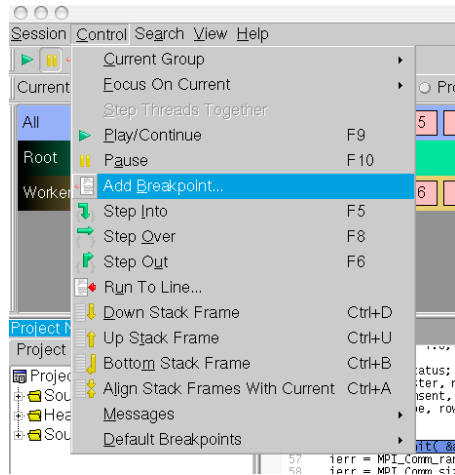
Changing Focus

- Control->Focus
- Select
 - Group
 - Process
 - Thread
- Or direct selection button

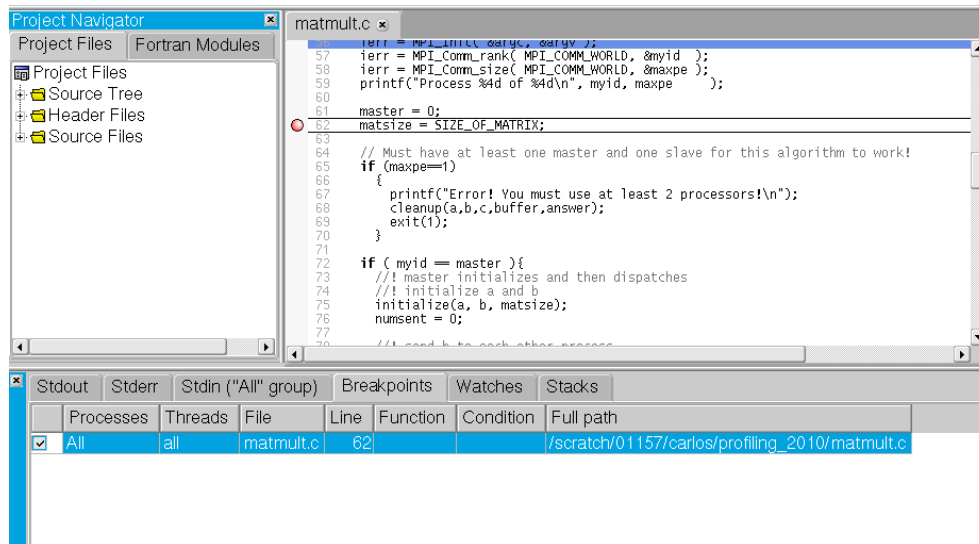


- Process control window changes to reflect selection

Inserting Breakpoints

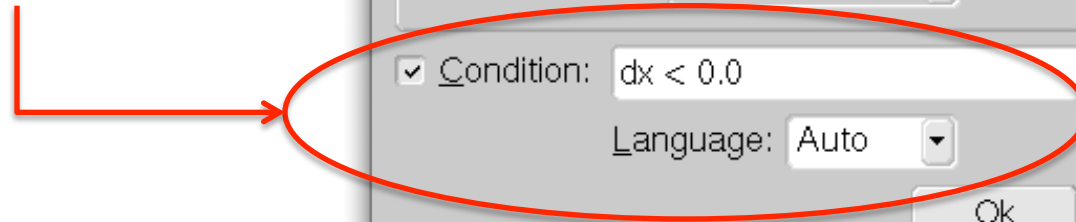


- Double-click on any line
- Right-click on any line
- Control -> Add Breakpoint
- Breakpoint icon



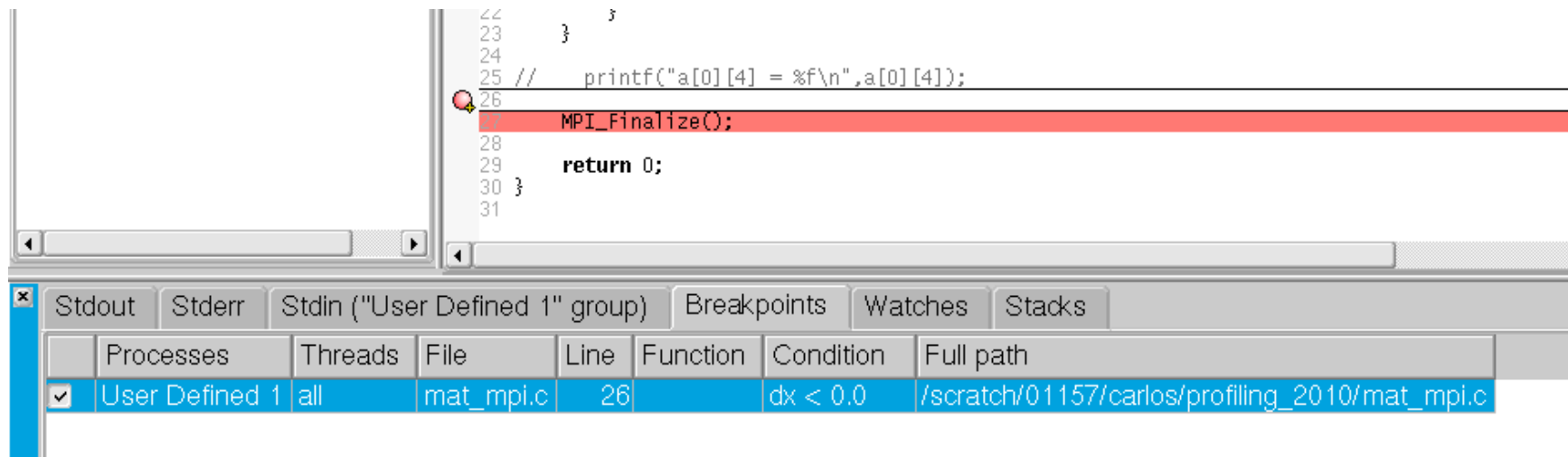
Conditional Breakpoints

- Right-Click on any breakpoint in the breakpoint window
- Edit Breakpoint
- Write condition in appropriate language



Conditional Breakpoints

- Change will be reflected in breakpoint window



Inserting Watches

- Right-Click in Watch Window
- Right-Click on Local/Evaluation Windows

The screenshot illustrates the steps to insert a watch in a debugger. The top window displays the source code of a C program. The bottom window shows the 'Watches' tab with a table containing 'dx'. A right-click context menu is open over the 'dx' entry, with 'Add To Watches' highlighted. A red arrow points from the 'Add To Watches' option to the 'Watches' tab.

```
6 int main(int argc, char **argv)
7 {
8     int i, j, proc, nproc;
9     float dx, x, y, a[25][25];
10
11     MPI_Init(&argc, &argv);
12     MPI_Comm_rank(MPI_COMM_WORLD, &proc);
13     MPI_Comm_size(MPI_COMM_WORLD, &nproc);
14
15     dx = 2*3.1415/25.0;
16     x = -3.1415; y = -3.1415;
17     for(i = 0; i < 25; i++){
18         x = x + dx;
19         for(j = 0; j < 25; j++){
20             y = y + dx;
21             a[i][j] = sin(x)*cos(y);
22         }
23     }
24
25     // printf("a[0][4] = %f\n", a[0][4]);
26
27     MPI_Finalize();
28 }
```

Process	Expression	Stack frame
0	dx	#0 main

Watches

- Add Watch...
- Delete
- Clear All

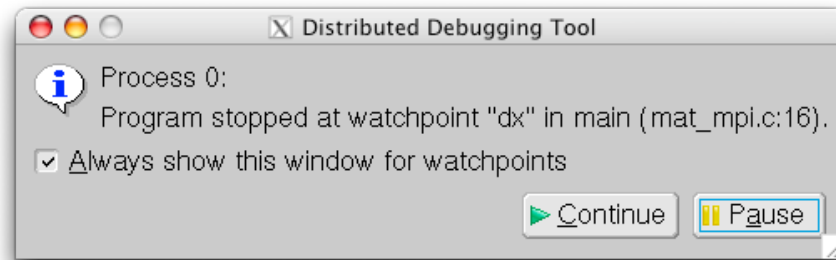
Local Variables

Variable Name	Value
x	
y	

Watches

- Add To Evaluations
- Add To Watches
- Edit Type...
- Copy Value
- View Array (MDA)
- View Across Processes (CPC)
- View Across Threads (CTC)
- Check Pointer Is Valid
- View Pointer Details
- Find Variable In Files

Watch Point Activation



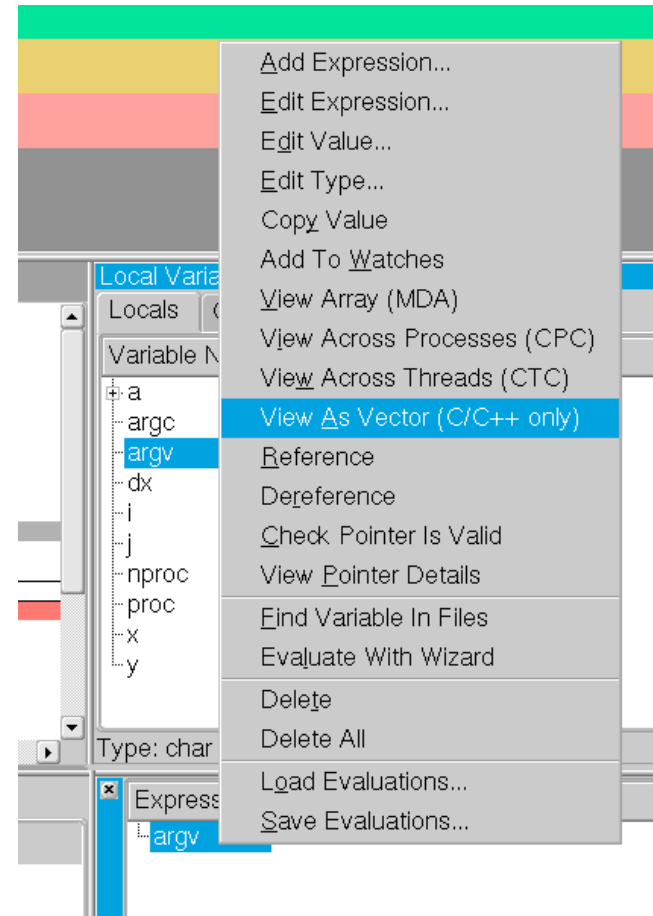
- When a watch point is triggered DDT will pause execution and issue a warning
- This gives you the opportunity to analyze the current execution state and the values of your variables

multidimensional array viewer, memory stats, parallel message queue

VIEWING AND EDITING DATA

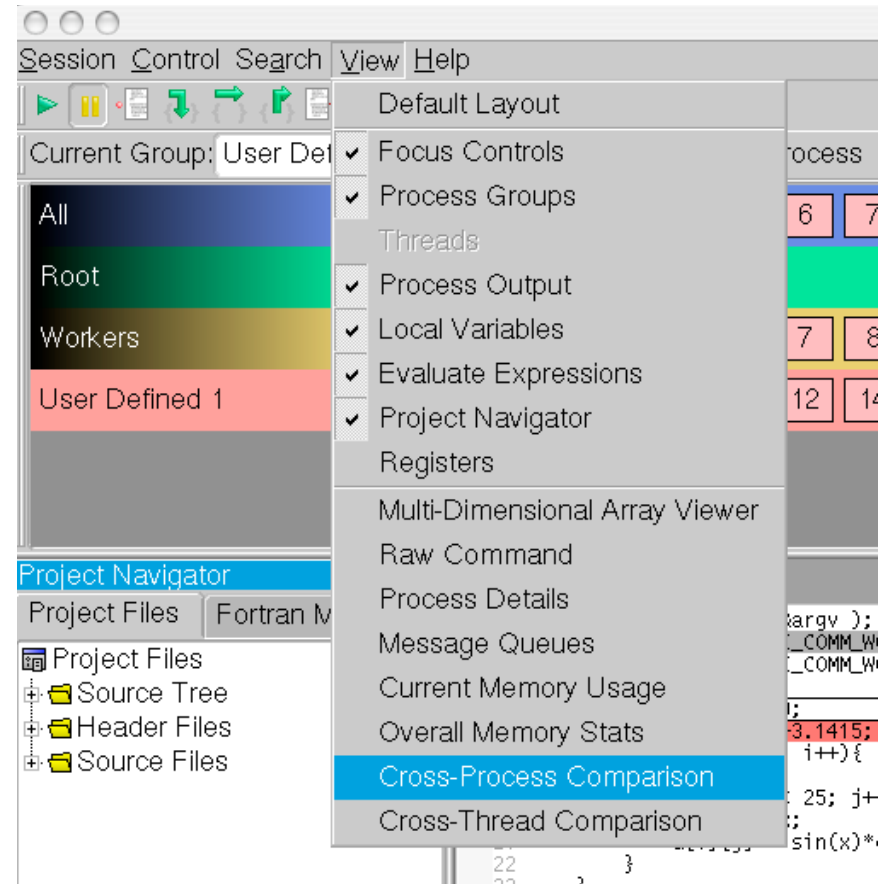
Viewing and Editing Variables

- Local variables appear in the variable window
- Drag & drop variables from Variable window to Evaluation window
- Right-click the variable name in the Evaluation window and select “Edit Value” or “Edit Type”
- View as vector to see dynamically allocated values



Data Values Across Processes

- View->Cross-Process Comparison
- View variable values across all processes in a group
- View->Cross-Thread Comparison
- View variable values across all threads in a process



Data Values Across Processes (2)

Value comparison

Compare across:

- ☒ Processes in current group (User Defined 1, 8 procs)
- ☐ Threads in current process (Process 0, with 1 thread)

Expression:

☐ Array mode - evaluate for i, from to inclusive

☐ Limit comparison to s.f. ☒ Align stack frames

Process(es)	Value
0, 2, 4, 6, 8, 10, 12, 14	0.251320004

Statistical Info

Compare across:

- ☒ Processes in current group (User Defined 1, 8 procs)
- ☐ Threads in current process (Process 0, with 1 thread)

Expression:

☐ Array mode - evaluate for i, from to inclusive

☐ Limit comparison to s.f. ☒ Align stack frames

Minimum value:

Maximum value:

Range:

Variance:

Median:

Mean:

Visual comparison

Compare across:

- ☒ Processes in current group (User Defined 1, 8 procs)
- ☐ Threads in current process (Process 0, with 1 thread)

Expression:

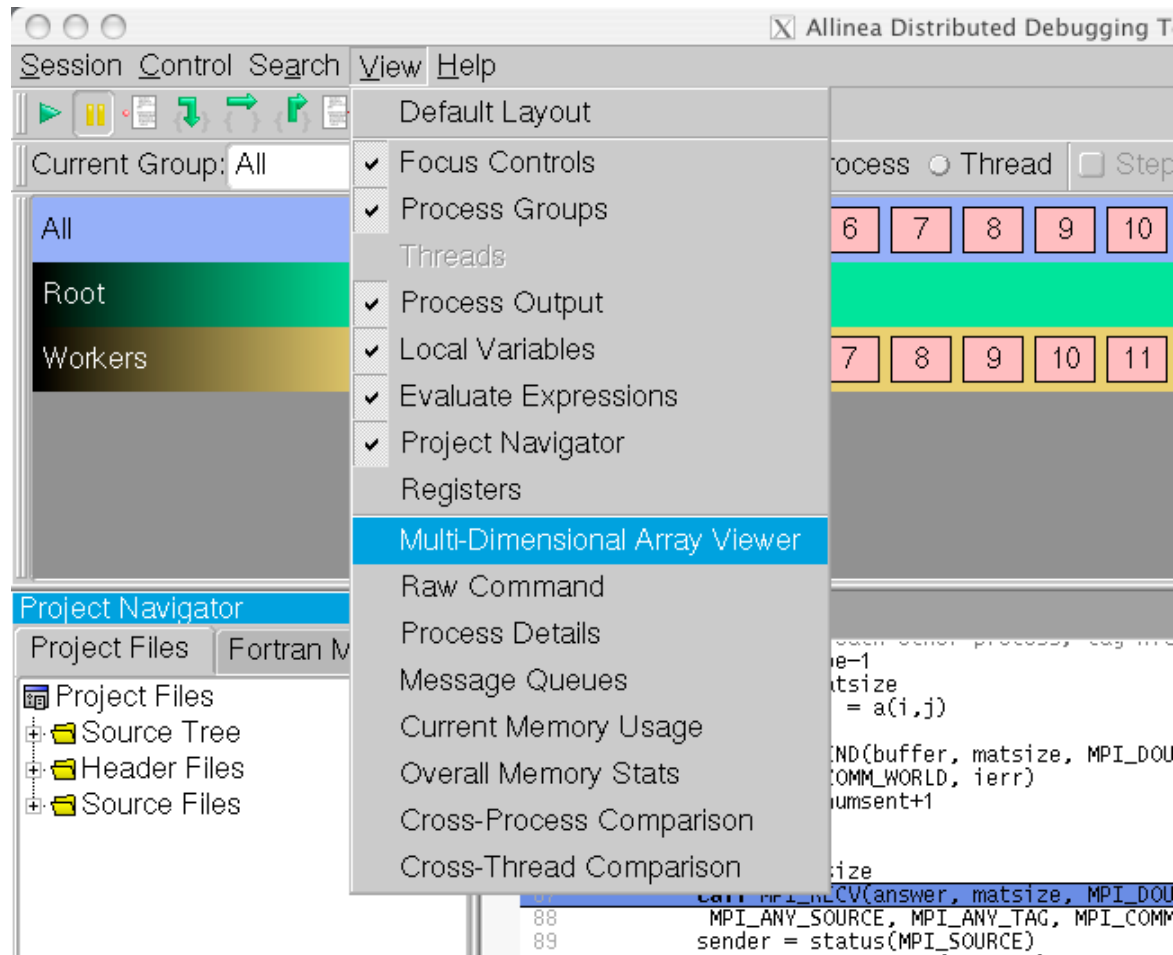
☐ Array mode - evaluate for i, from to inclusive

☐ Limit comparison to s.f. ☒ Align stack frames

Comparison of a single value



Array data access



Multi-Dimensional Array Viewer

DDT - Multi-Dimensional Array Viewer

Array Expression:

Range of \$i

From:

To:

Display:

Range of \$j

From:

To:

Display:

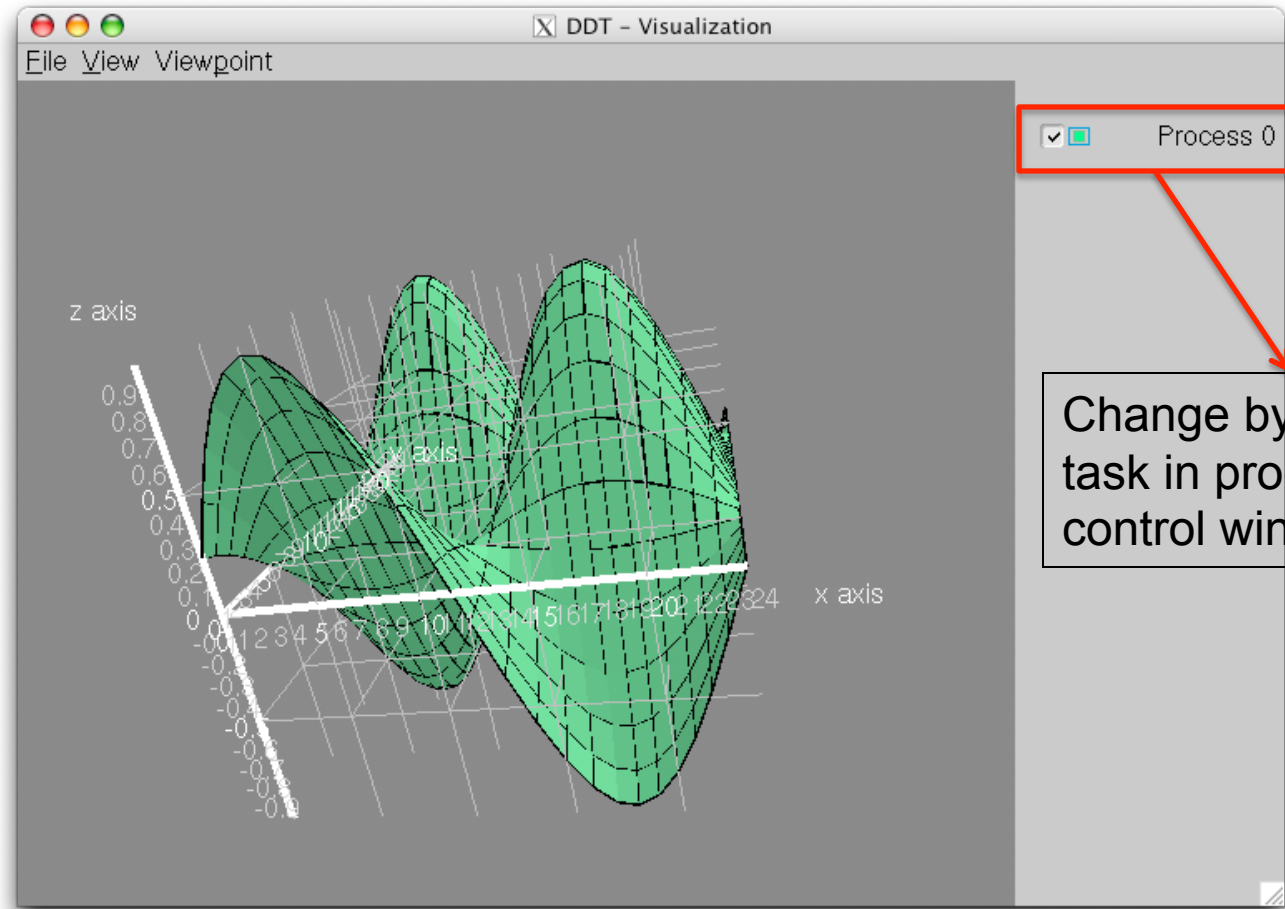
☐ Auto-update

Aggregate Function:

☐ Filter:

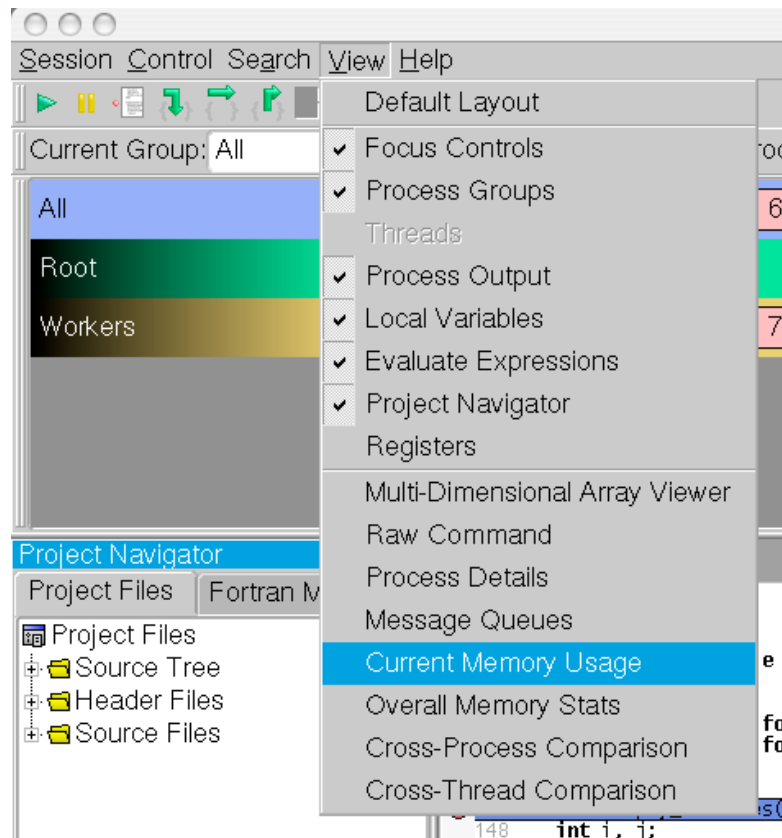
One of these for each dimension

Data Visualization

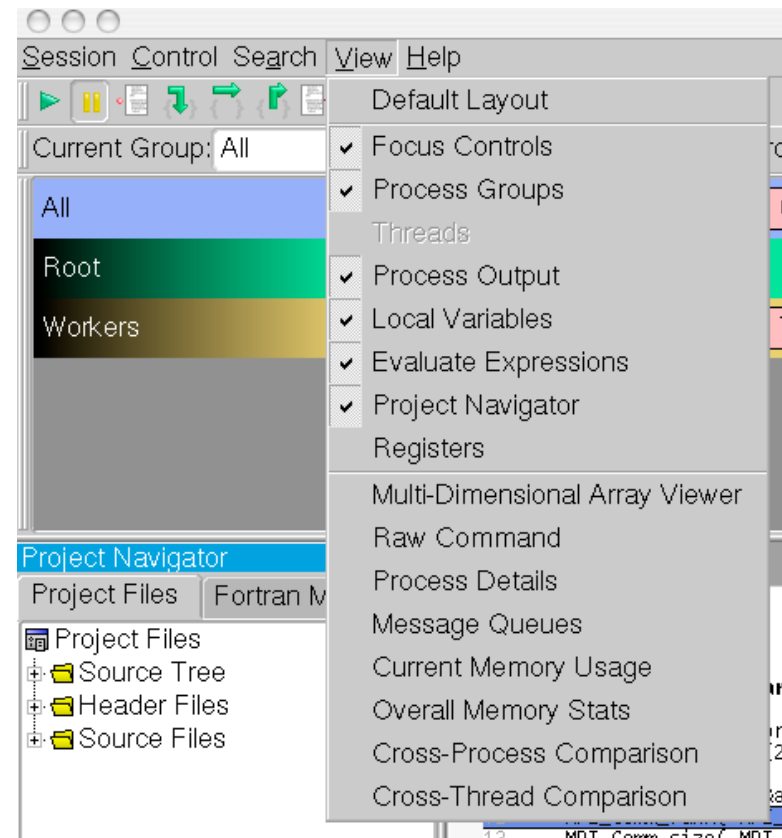


Analyzing Memory Usage

Current Memory Usage

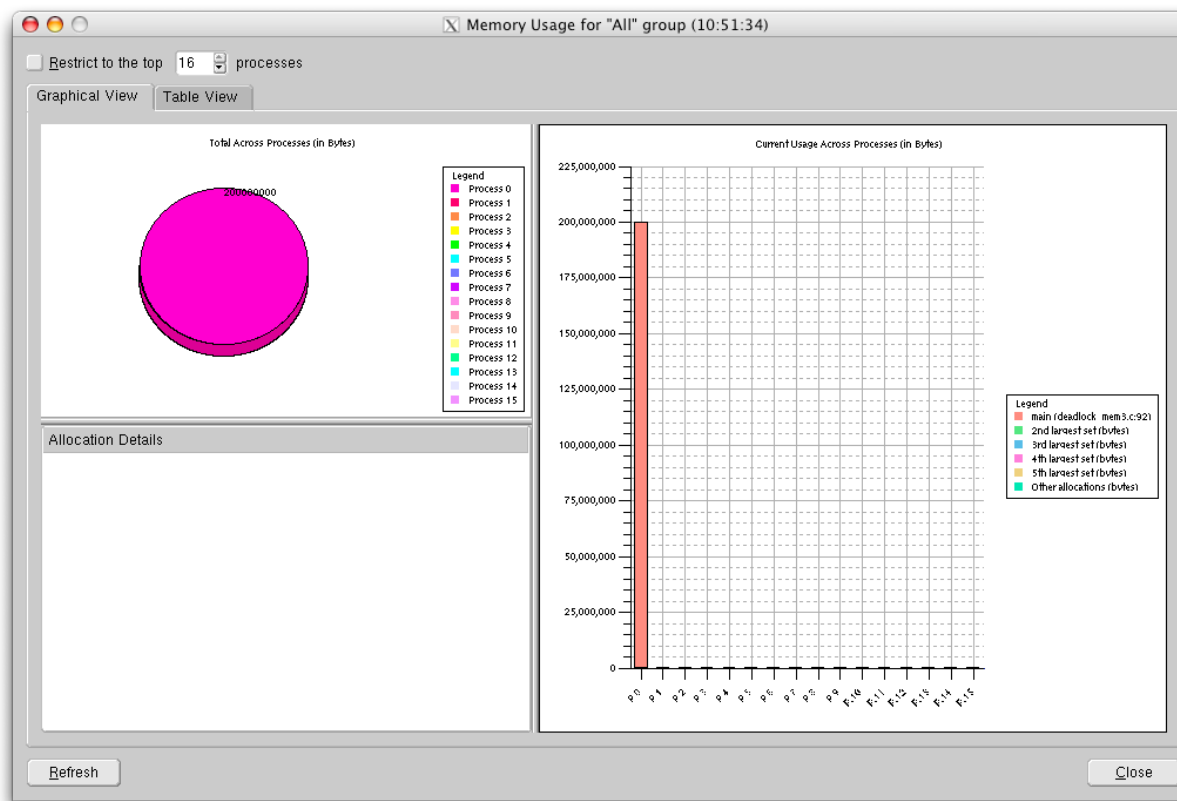


Overall Memory Stats



Current memory Usage

Go to View -> Current Memory Usage

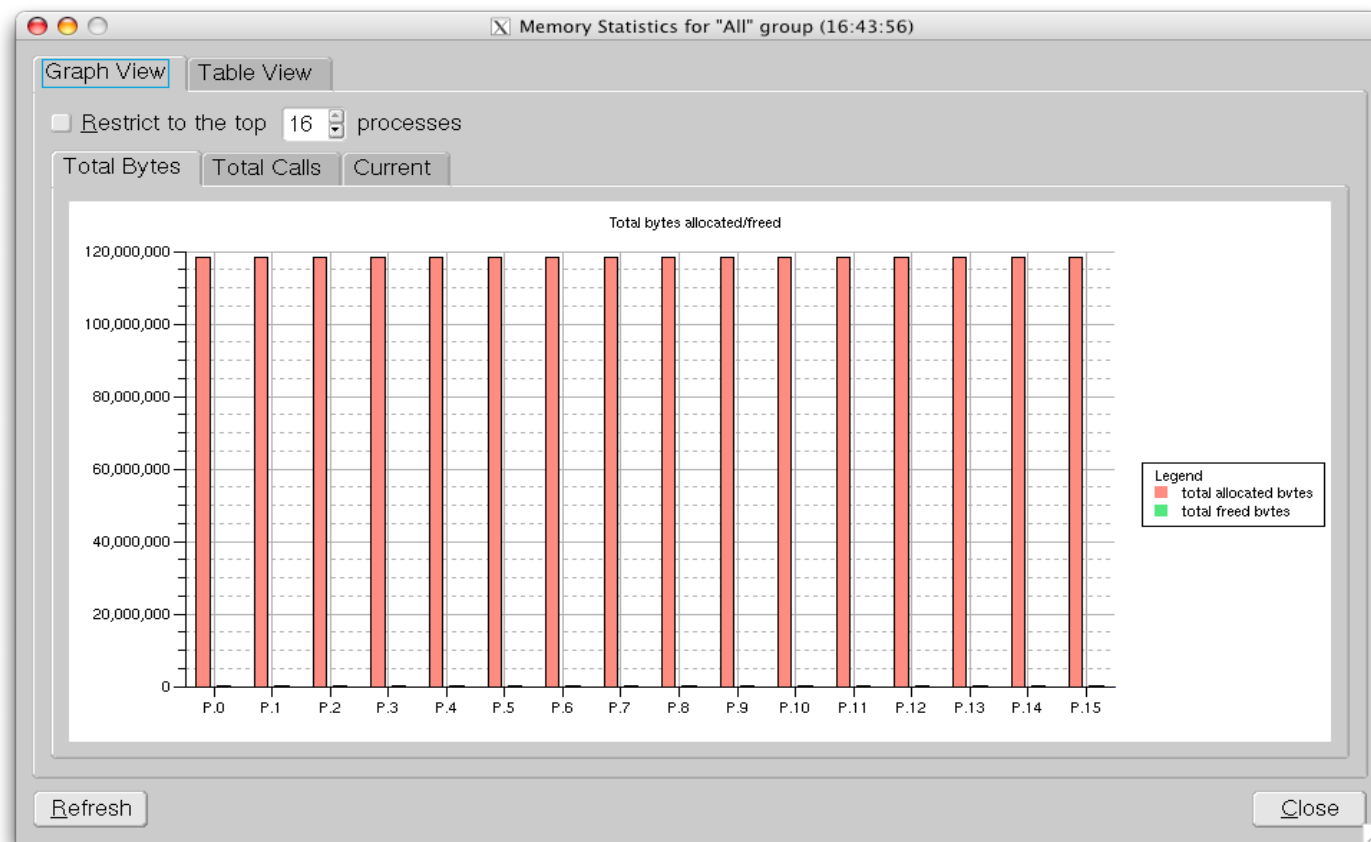


Process 0 is using much more memory than the others.

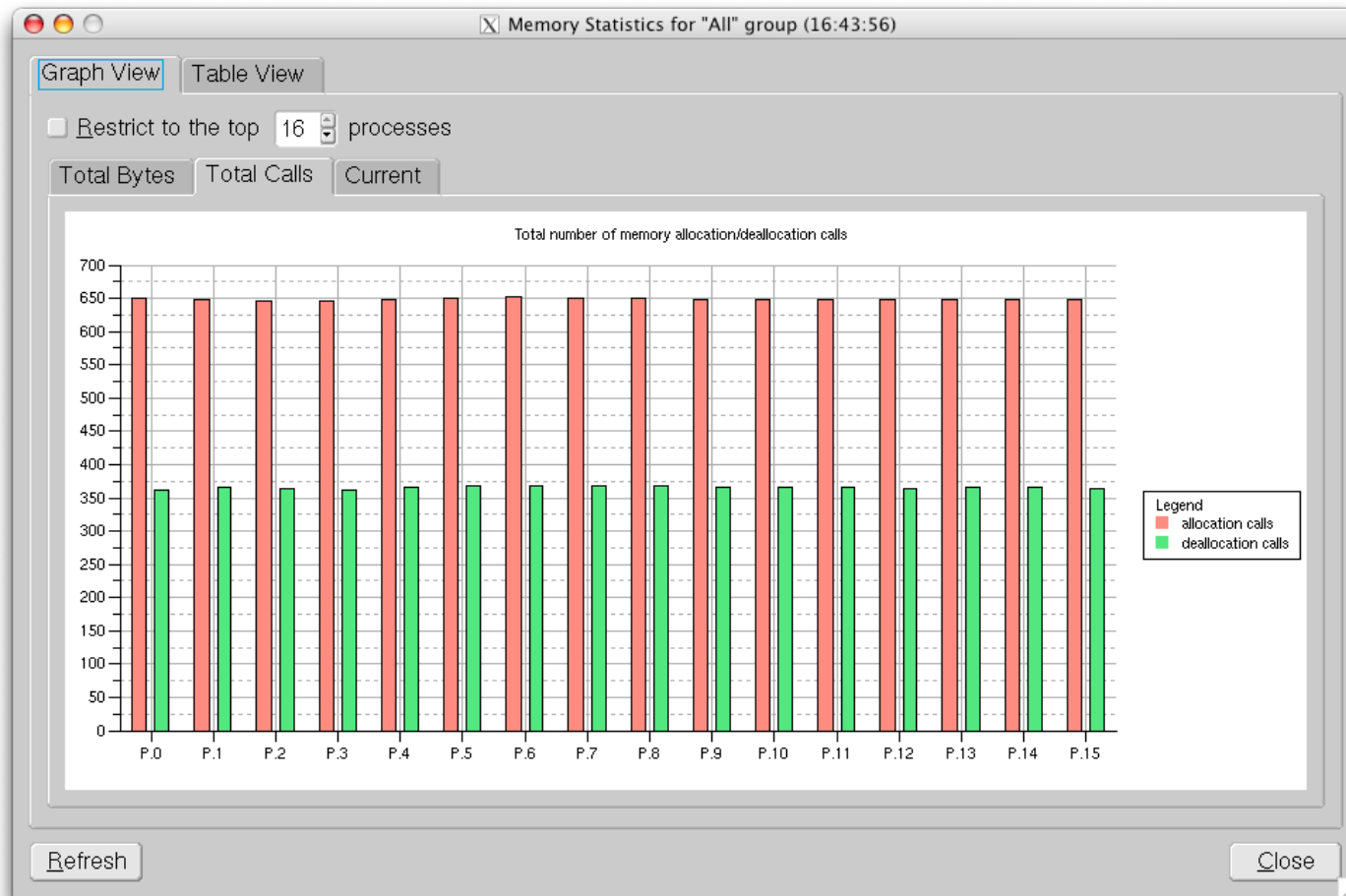
This looks like a memory leak.

Overall Memory Stats (Size)

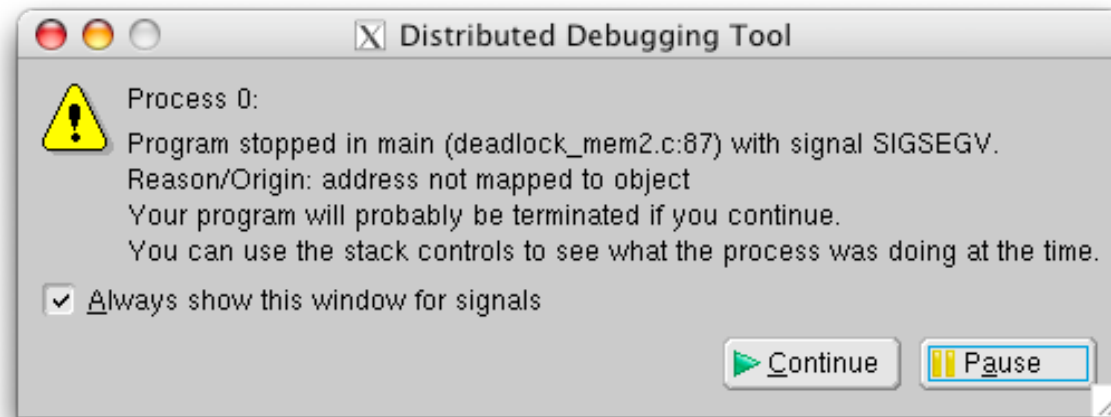
Go to View -> Overall Memory Stats



Overall Memory Stats (Calls)



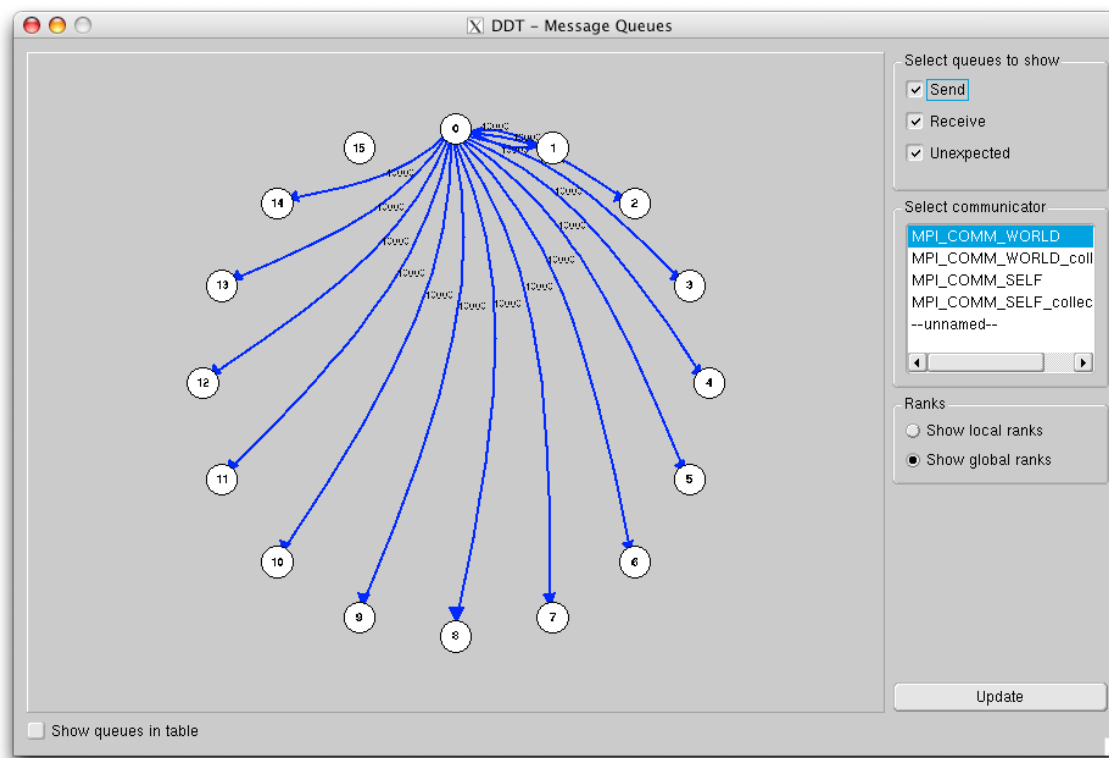
Segmentation Faults



- DDT tells you the exact line in the code where the problem occurs.
- DDT tells you the precise reason that caused the problem.

Message Queues

Go to View -> Message Queues



Click on bottom left for additional information (message sizes, etc...)

Pending messages in the "Unexpected" queue are indicative of MPI problems

HANDS-ON

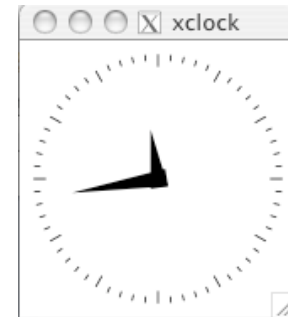


THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Setup

- Login to Ranger or Lonestar:
 - `ssh -X username@ranger.tacc.utexas.edu`
 - `ssh -X username@lonestar.tacc.utexas.edu`
- Make sure you can export graphics to your laptop screen:
 - `xclock`

If you do not see a clock, contact an instructor



- If you are in Ranger swap the PGI and Intel compilers:
 - `module swap pgi intel`
- Untar the lab files:
 - `cd`
 - `tar xvf ~train00/ddt_petascale.tar`
- Change directories and ls to see the files:
 - `cd ddt_petascale`
 - `ls`

Hands-On Code

- Using the same examples you already studied with TotalView
- You should find C and F90 versions of a single code, with indicative names:
 - **cell_seq.c, cell_seq.f90**
 - Baseline code
 - **cell_mpi_bug.c, cell_mpi_bug.f90**
 - MPI code to be debugged
 - **cell_omp_bug.c, cell_omp_bug.f90**
 - OpenMP code to be debugged
- Both the MPI and OMP codes contain bugs that you must find with DDT and fix.

Instructions

- Compile the three given codes with the -g -O0 flags:
 - `icc -g -O0 cell_seq.c -o seq`
 - `mpicc -g -O0 cell_mpi_bug.c -o mpi`
 - `icc -g -O0 -openmp cell_omp_bug.c -o omp`
- Load the ddt module
 - `module load ddt`
- Run the sequential code using DDT and write down the answer.
- Run the MPI and OMP codes using DDT and try to find and correct all errors, until you obtain an output identical to that of the sequential run.