DDT Debugging Techniques

Carlos Rosales
carlos@tacc.utexas.edu

Scaling to Petascale 2010
July 7, 2010

Debugging Parallel Programs

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

➕

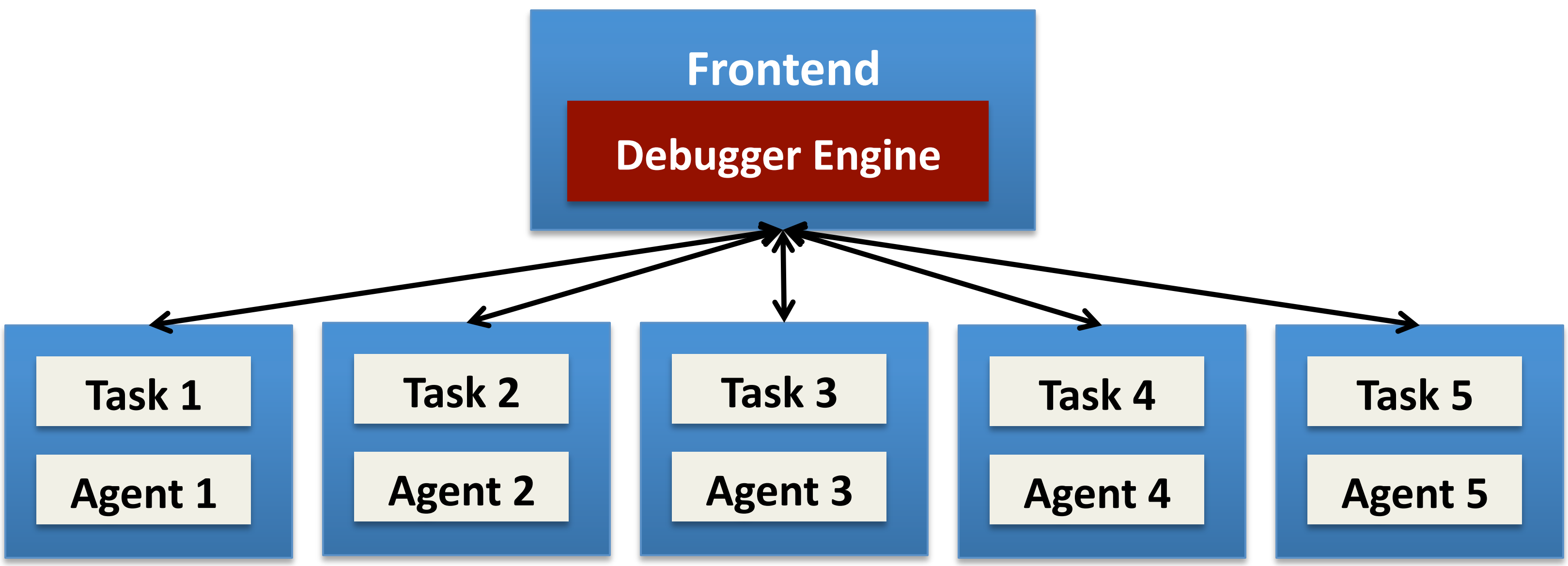
- New set of problems
  - Work distribution and partitioning
  - Arguments to parallel function calls
  - Racing conditions (shared memory / OMP)
  - Deadlocks (MPI)
  - …

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 **D**istributed **D**ebugger **T**ool 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

Using DDT

- Compile your code using the standard debug flags:

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

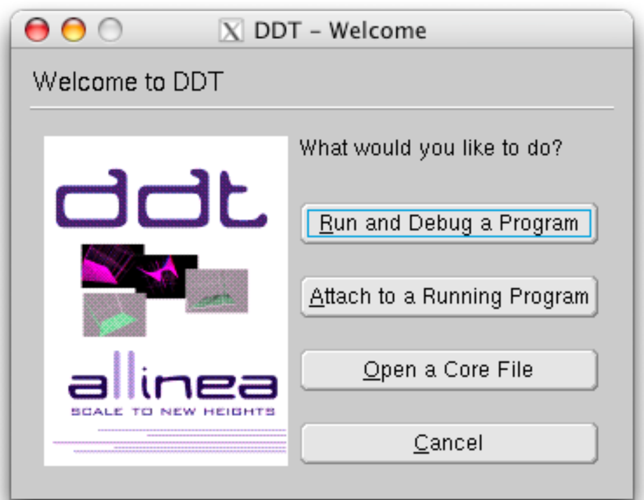
- Load the DDT module:

```
% module load ddt
% module list
```

- Start up DDT:

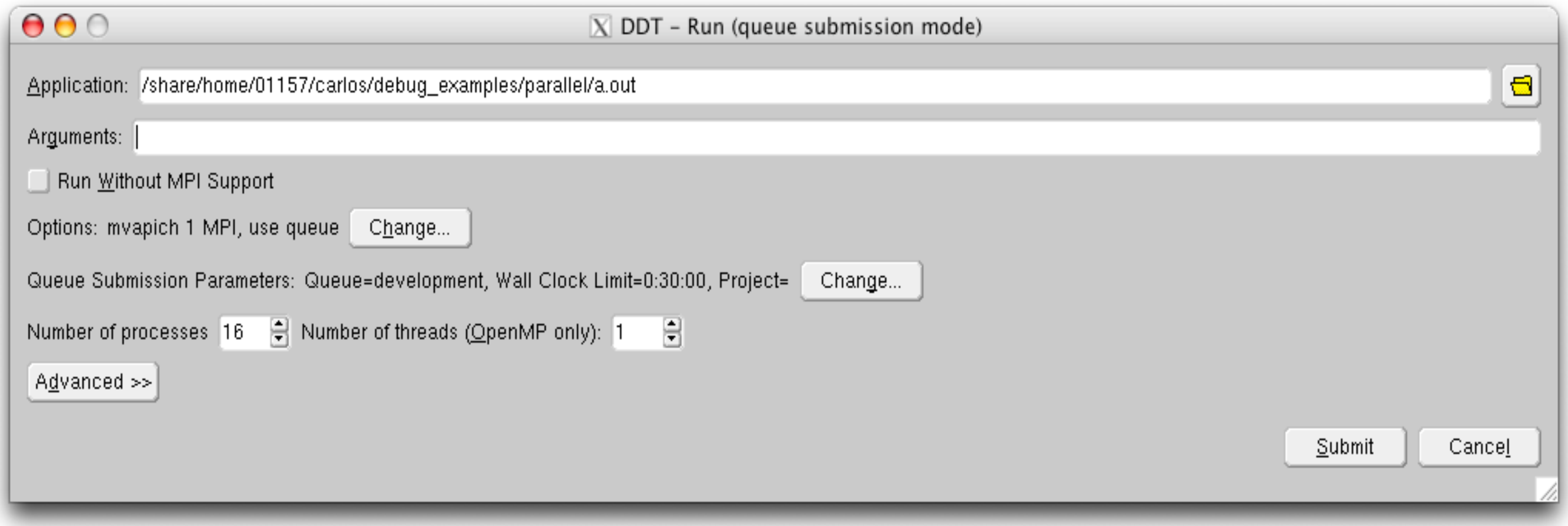
```
% ddt ./a.out
```

Configuration: Welcome Screen

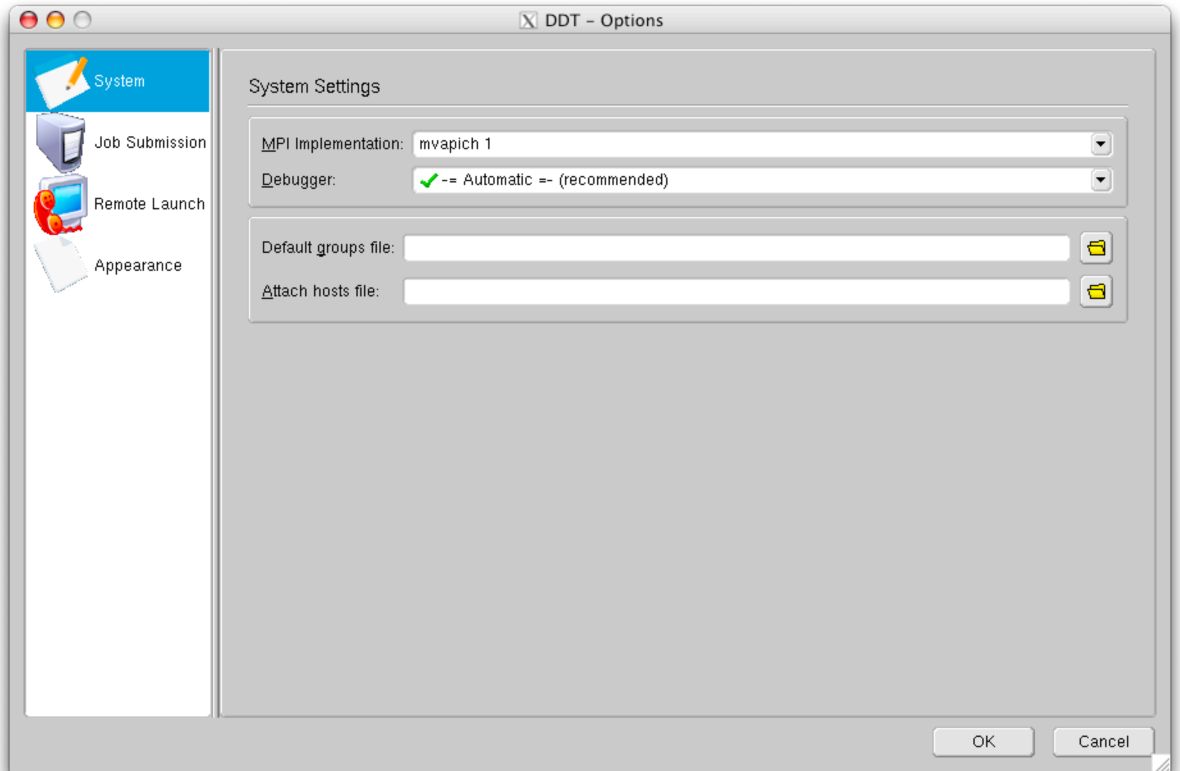


- Three ways of starting a debug session
  - Run and debug
  - Attach to a running program
  - Open core dump file

Configuration: Job Submission

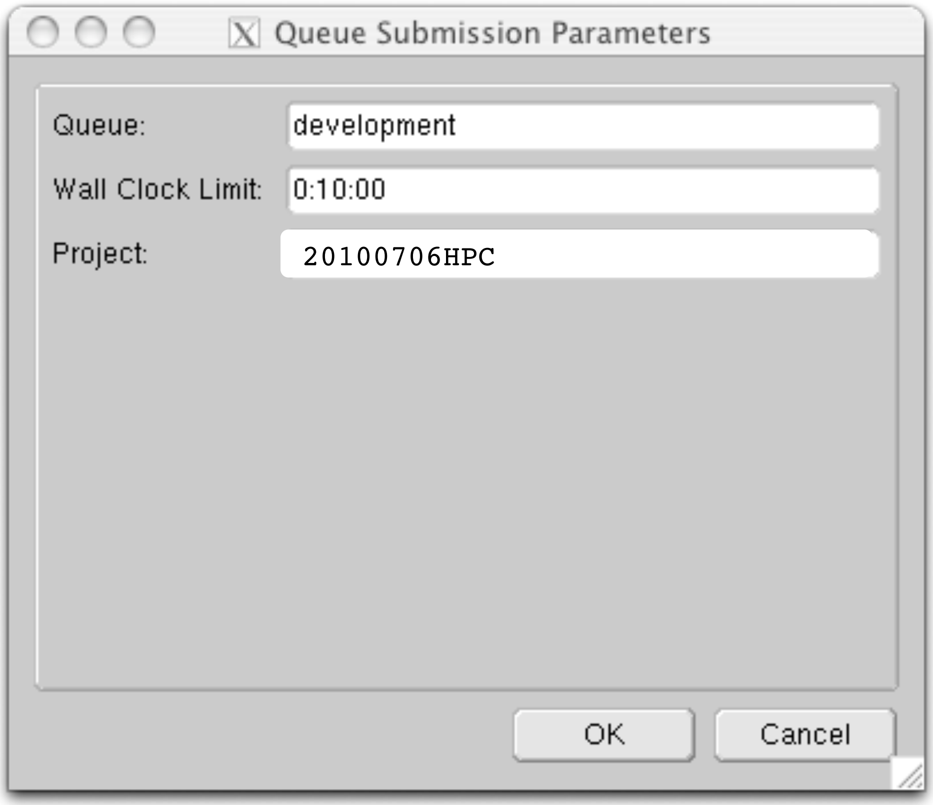


Configuration: Options



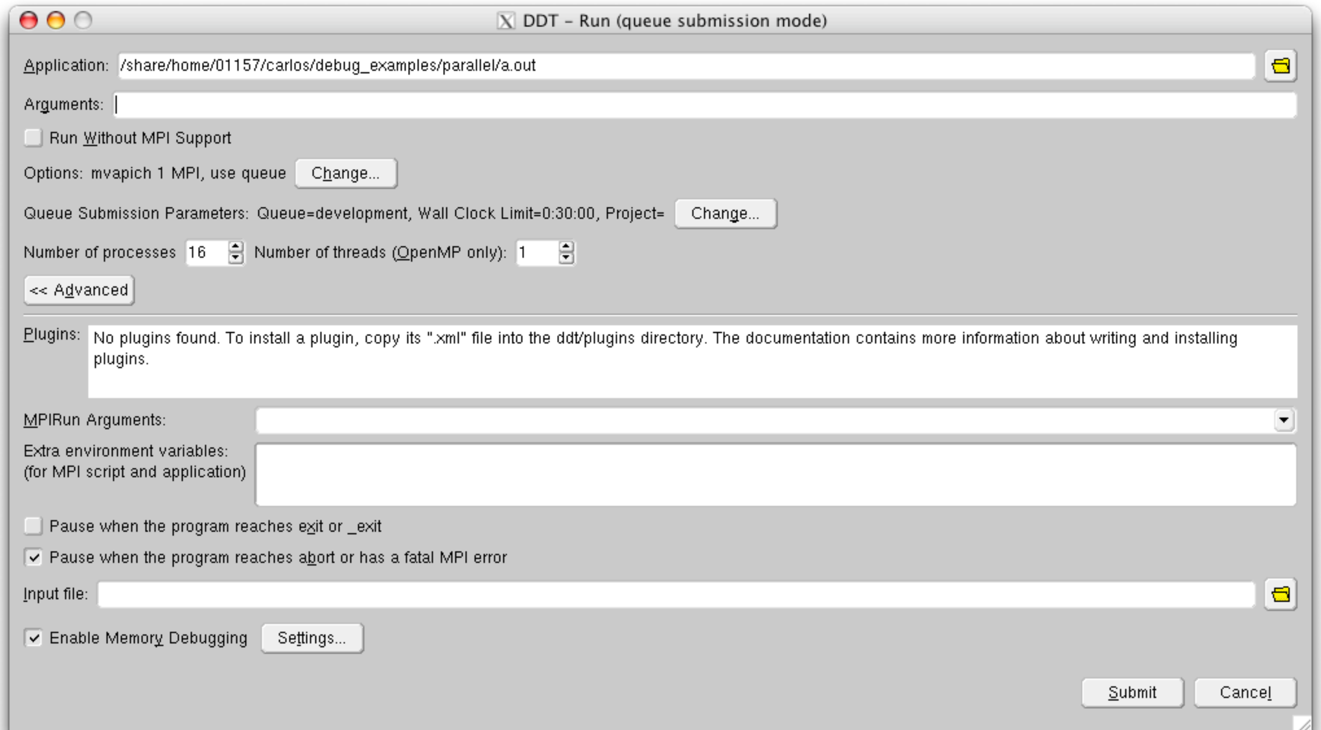
- Choose the correct version of MPI
  - mvapich 1
  - mvapich 2
  - openMPI
- Leave Debugger on the Automatic setting

Configuration: Queue Parameters



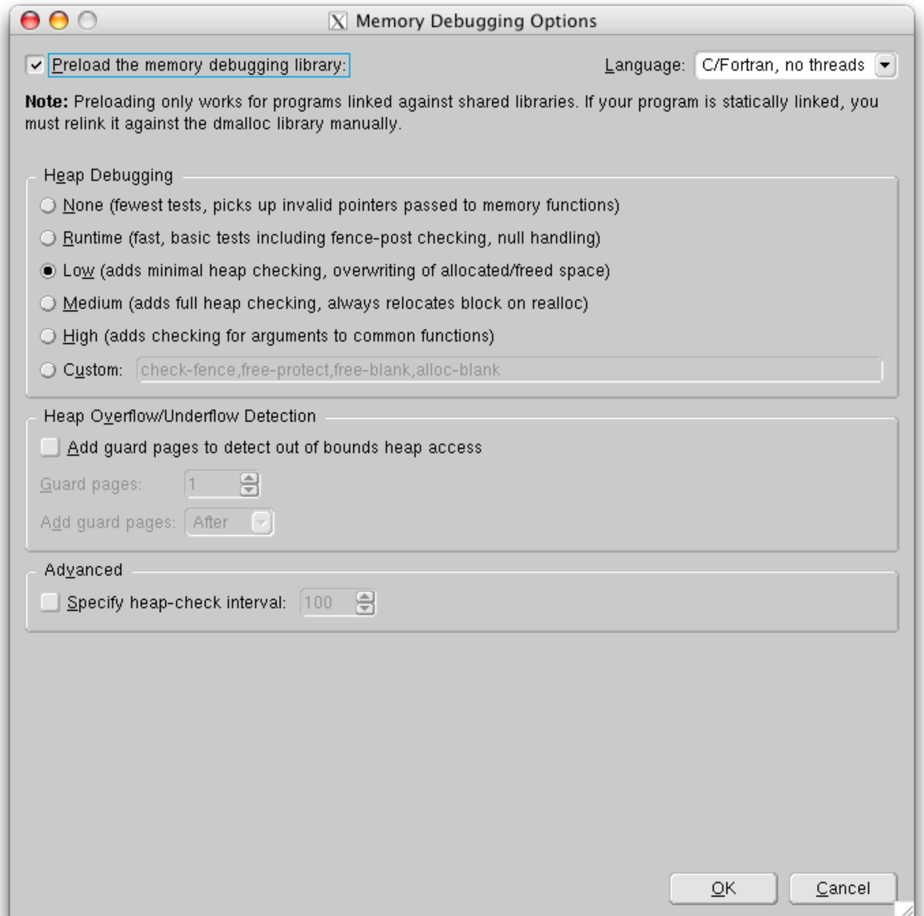
- Choose the queue
- Set the Wall Clock Limit (H:MM:SS)
- Set your project code - for this training class use 20100706HPC

Configuration: Memory Checks



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

Configuration: Memory Options



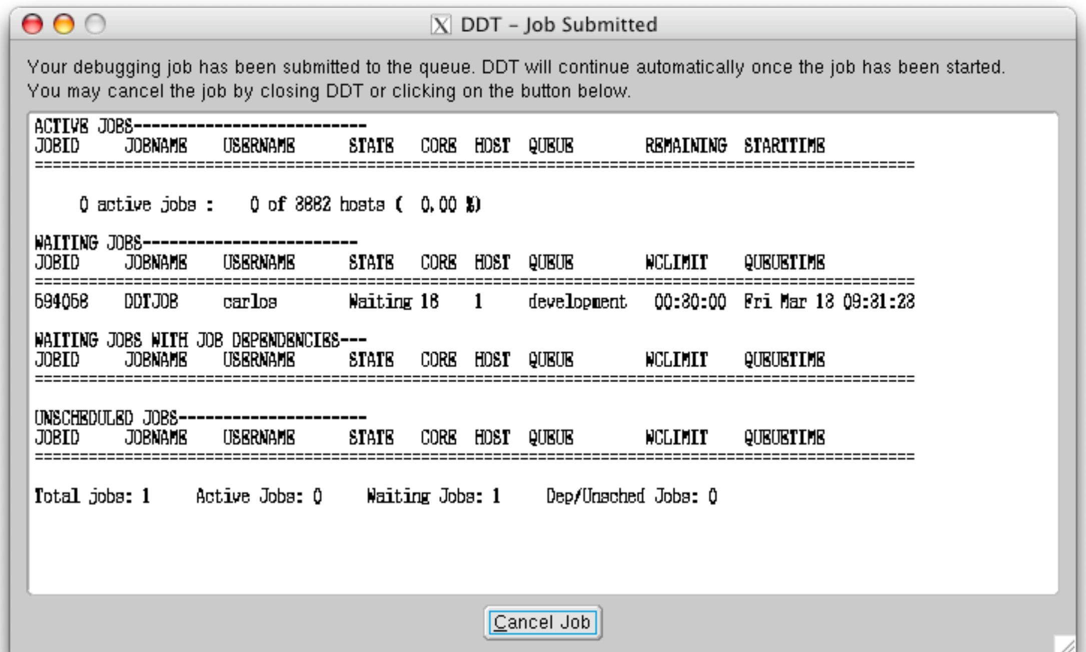
- Change the Heap Debugging option from the default **Runtime** to **Low**
- Even the option None provides some memory checking
- Leave Heap and Advanced unchecked unless you really know that you need them…

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:

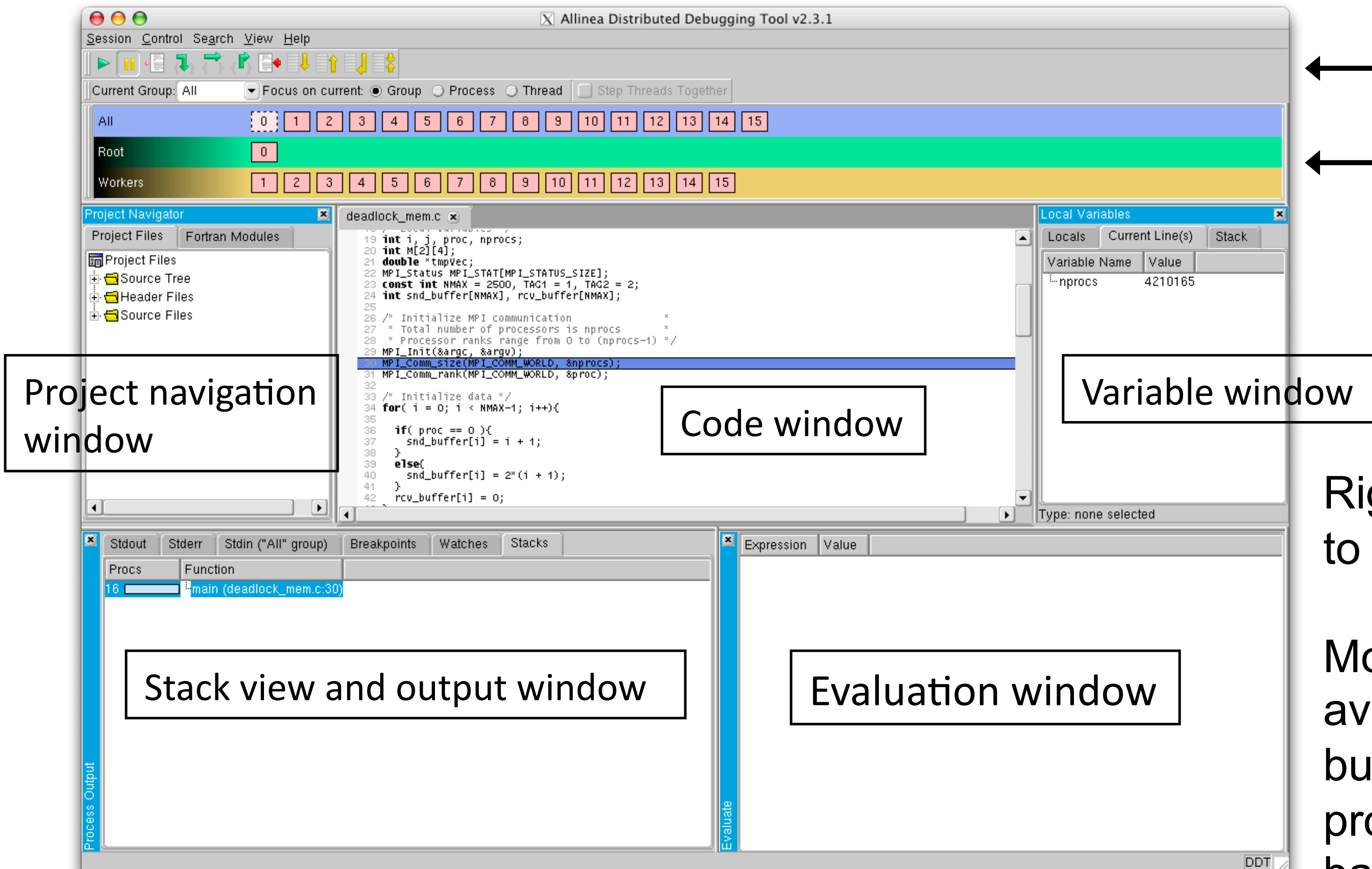


The job is submitted to the specified queue.

An automatically refreshing job status window appears.

The debug session will begin when the job starts.

DDT: The debug session



Process controls

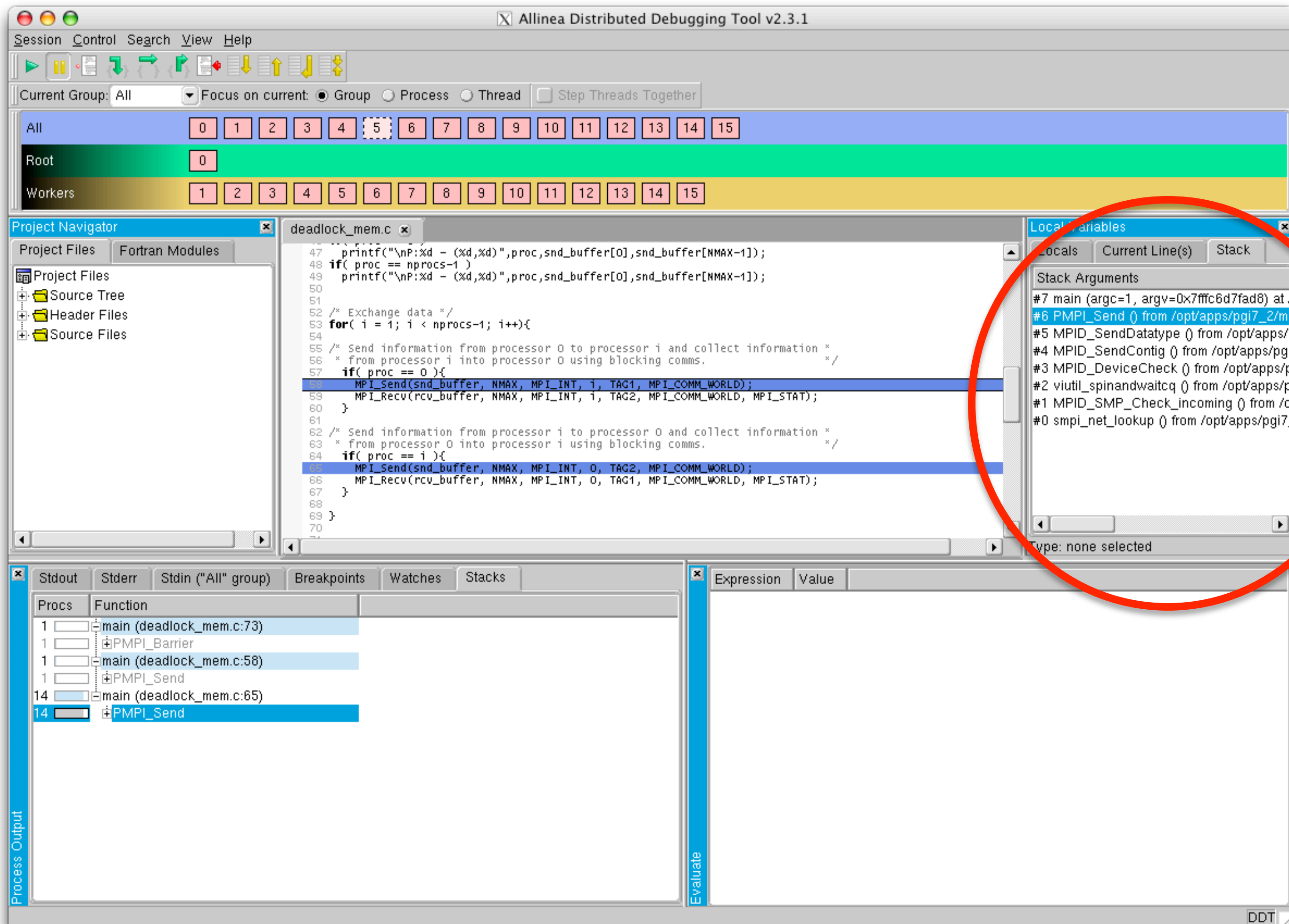
Process groups window

Right click on a line to set a breakpoint.

Most operations available as buttons in the process controls bar focus, breakpoints and watches

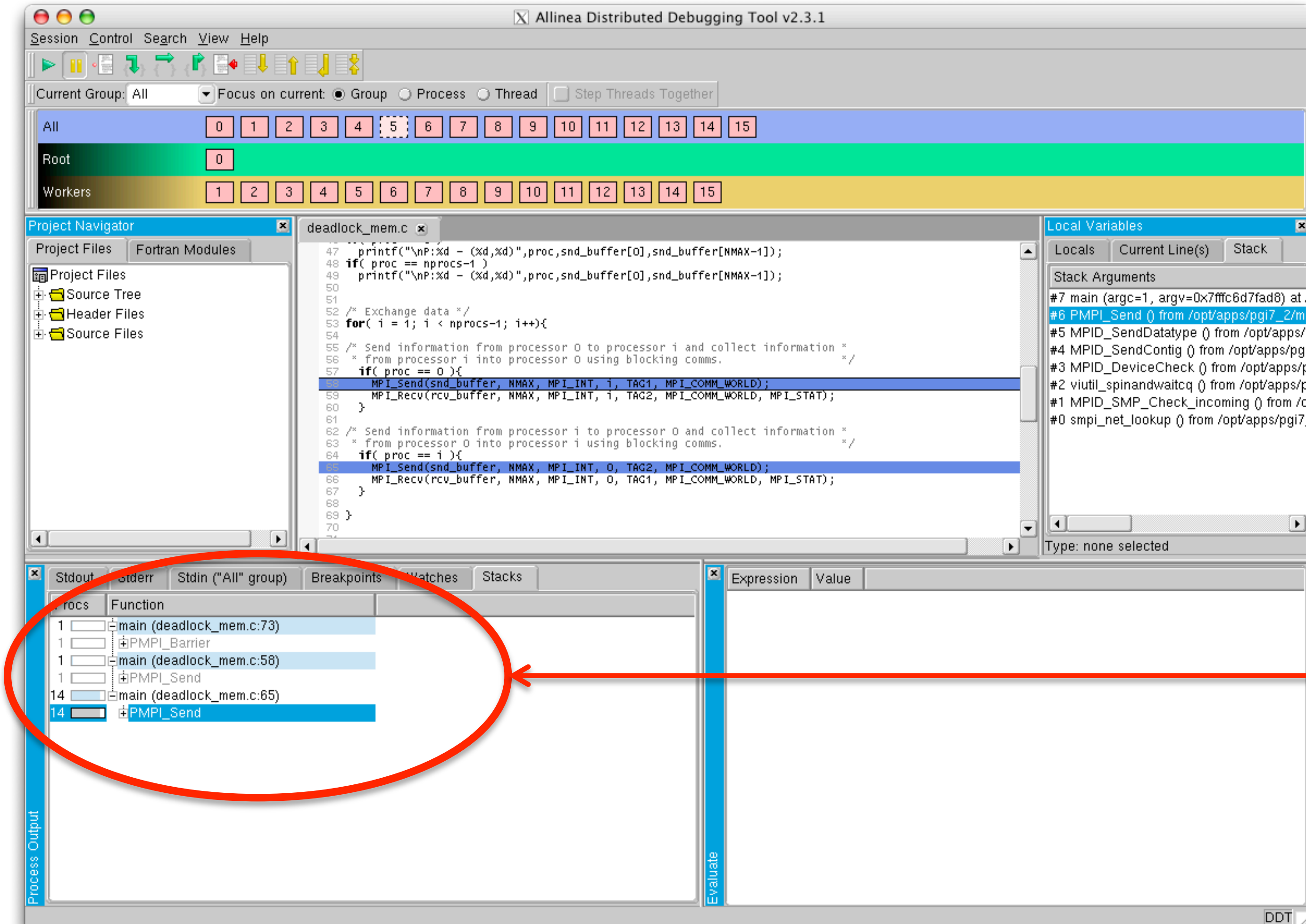
PROCESS CONTROL

Stack Window



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

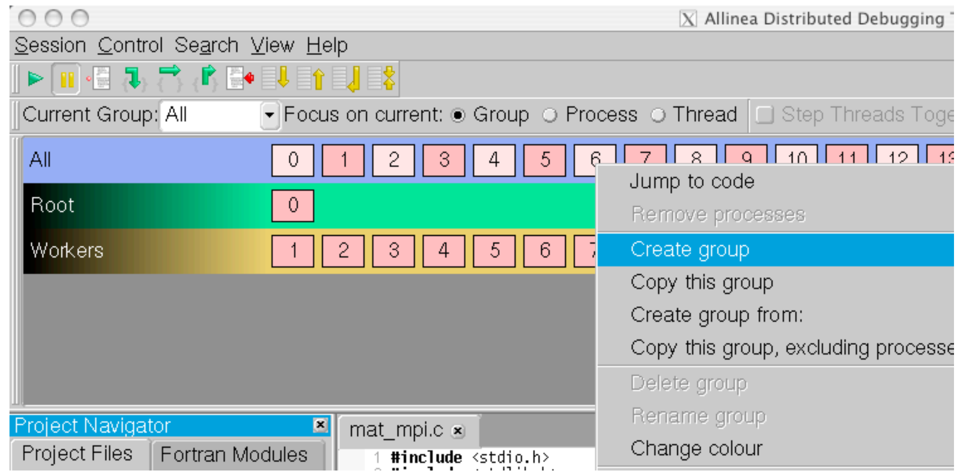
Parallel Stack View

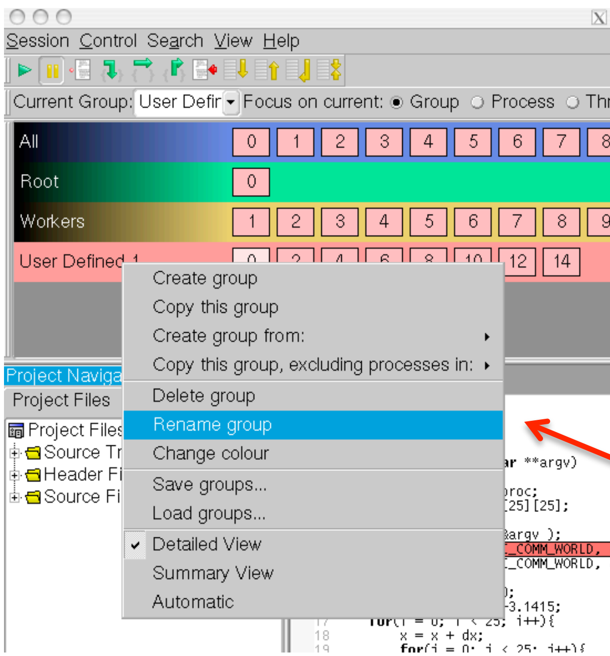


- Shows position of each thread / process in the code.
- Hover over any function to see a list of processes at that position.

Creating Process Groups

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

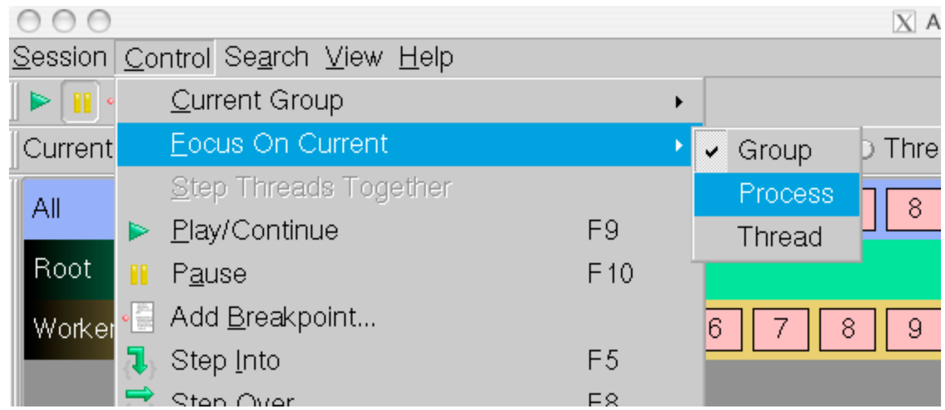


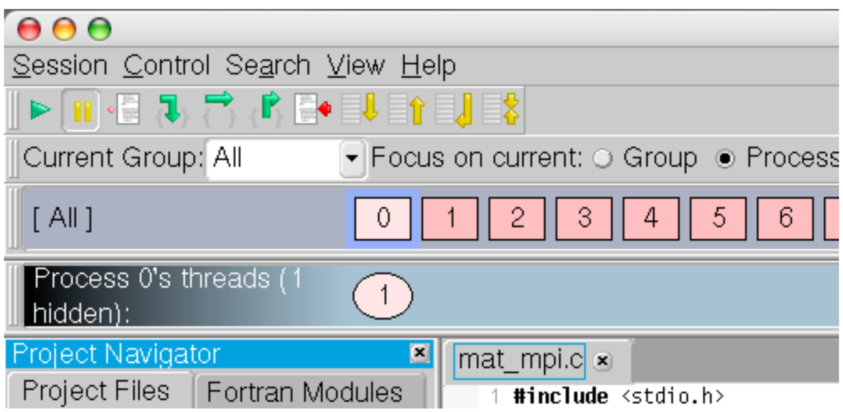


- Group is created from selection with default name
- Right-Click the group to edit properties

Changing Focus

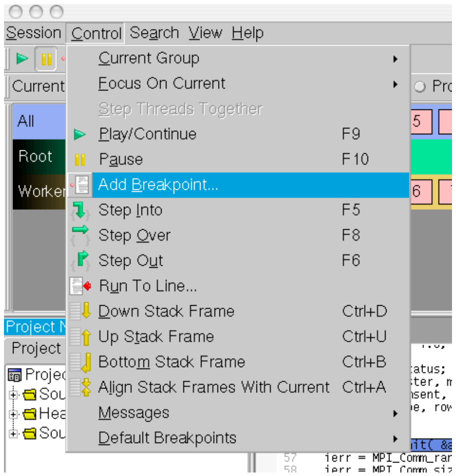
- Control->Focus
- Select
  - Group
  - Process
  - Thread
- Or direct selection button

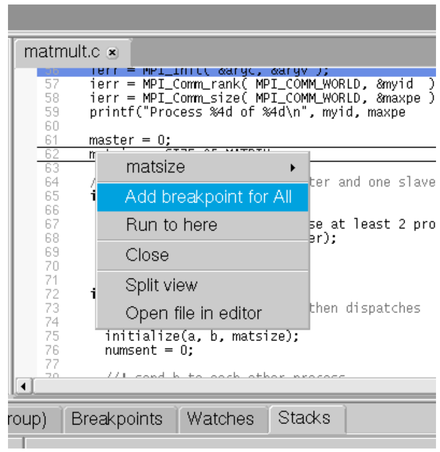




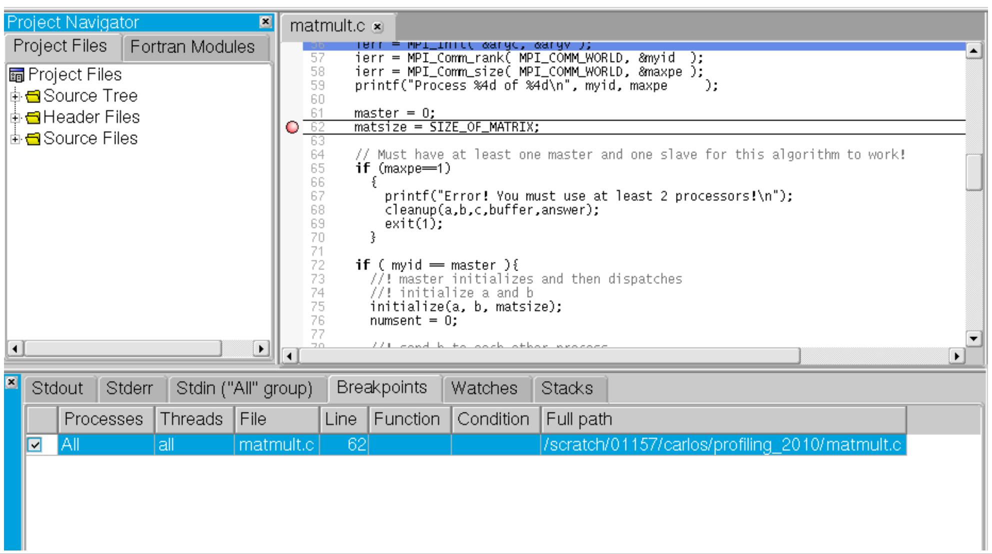
- Process control window changes to reflect selection

Inserting Breakpoints



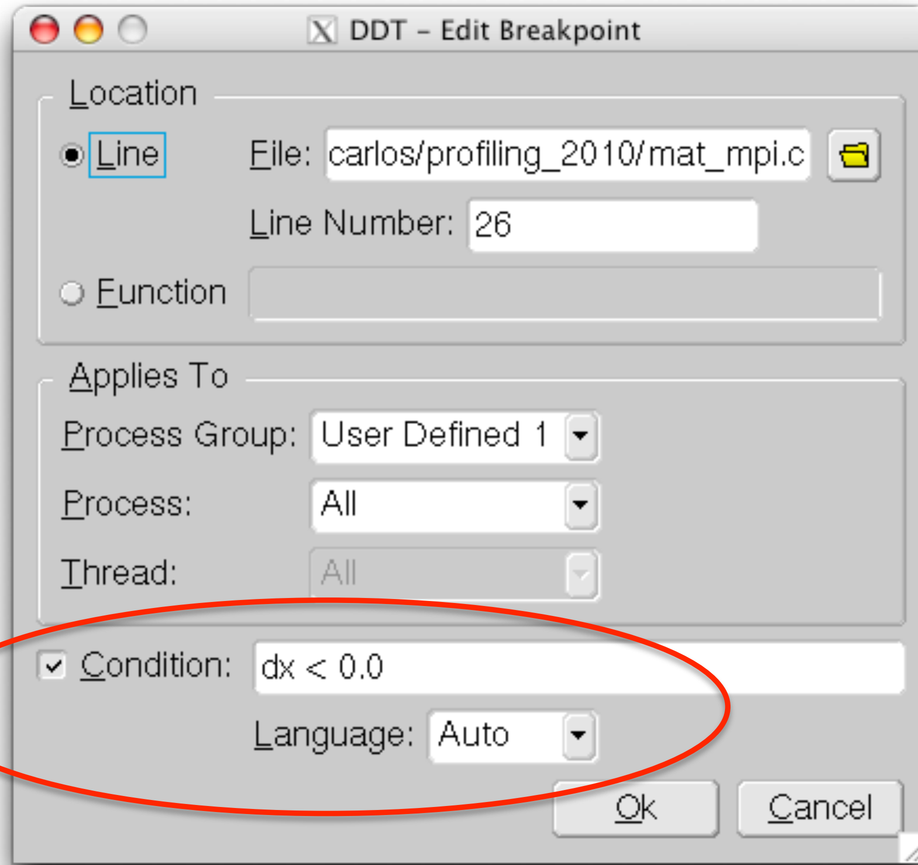


- Double-click on any line
- Right-click on any line
- Control -> Add Breakpoint
- Breakpoint icon



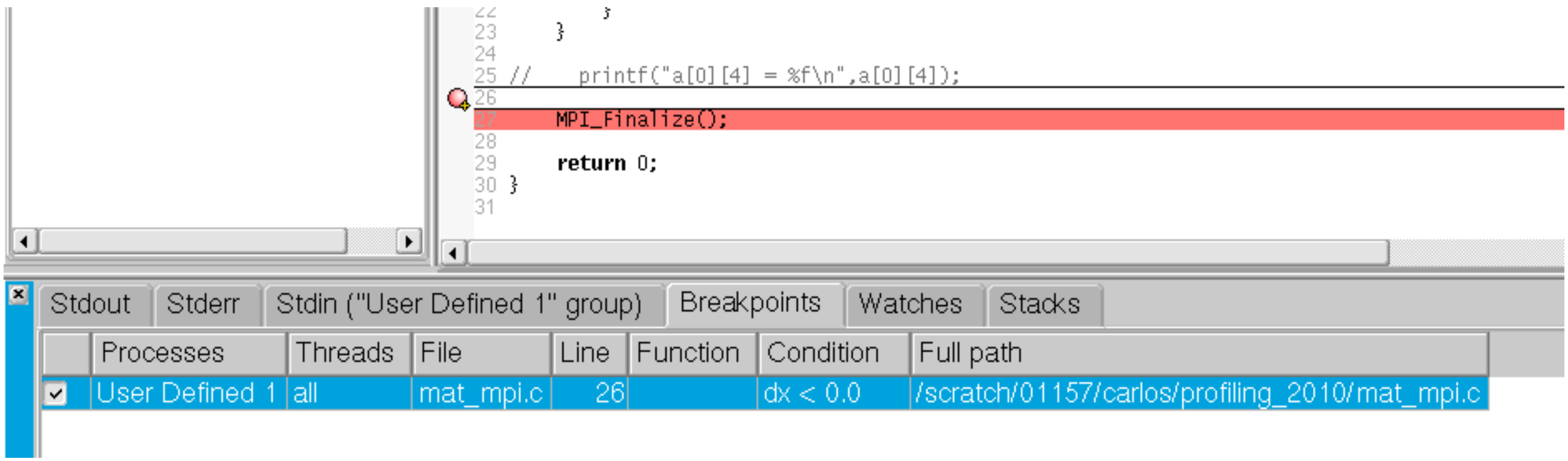
Conditional Breakpoints

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



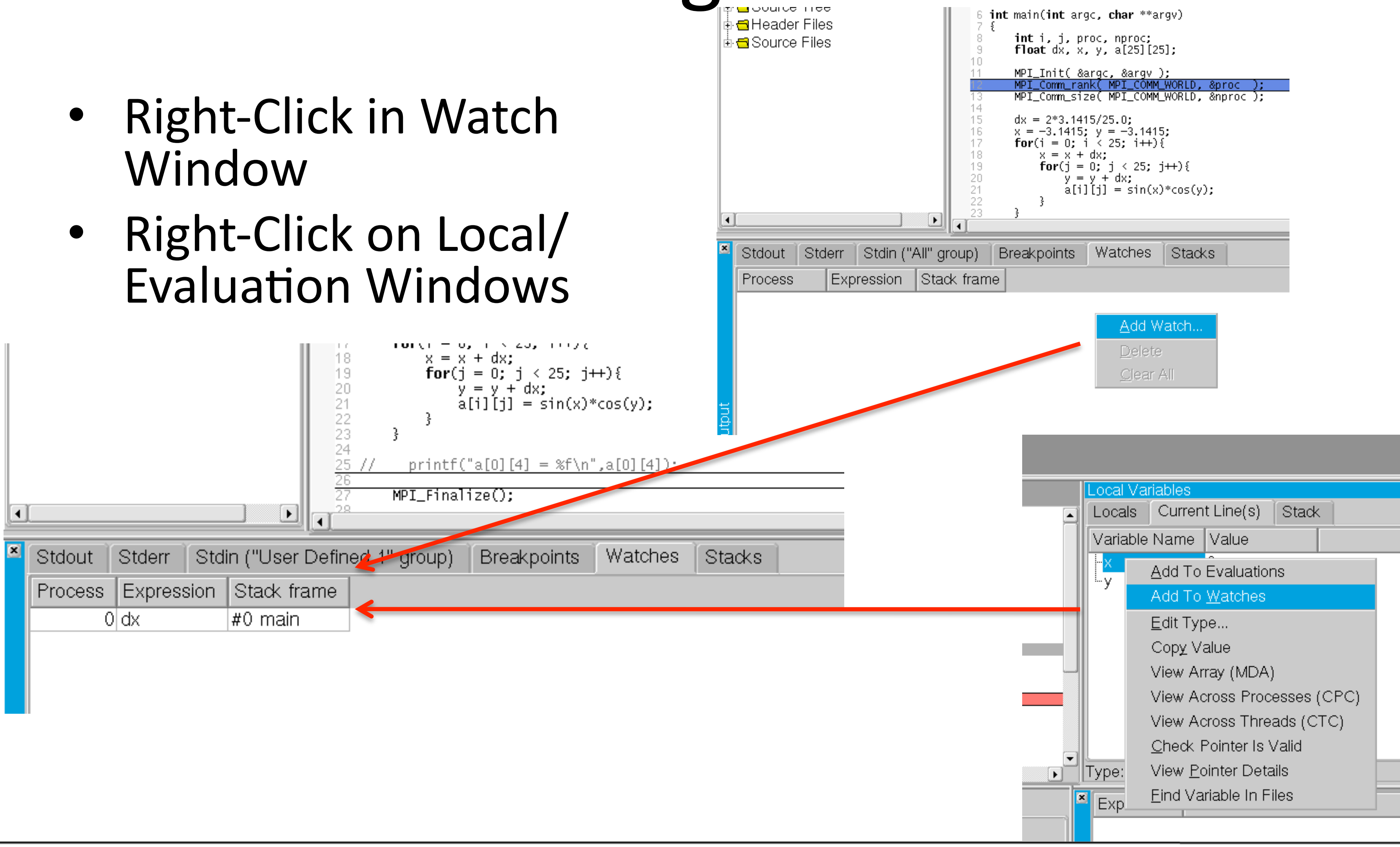
Conditional Breakpoints

- Change will be reflected in breakpoint window

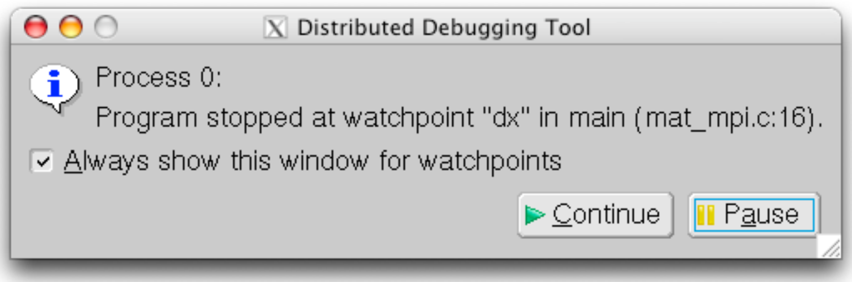


Inserting Watches

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



Watch Point Activation



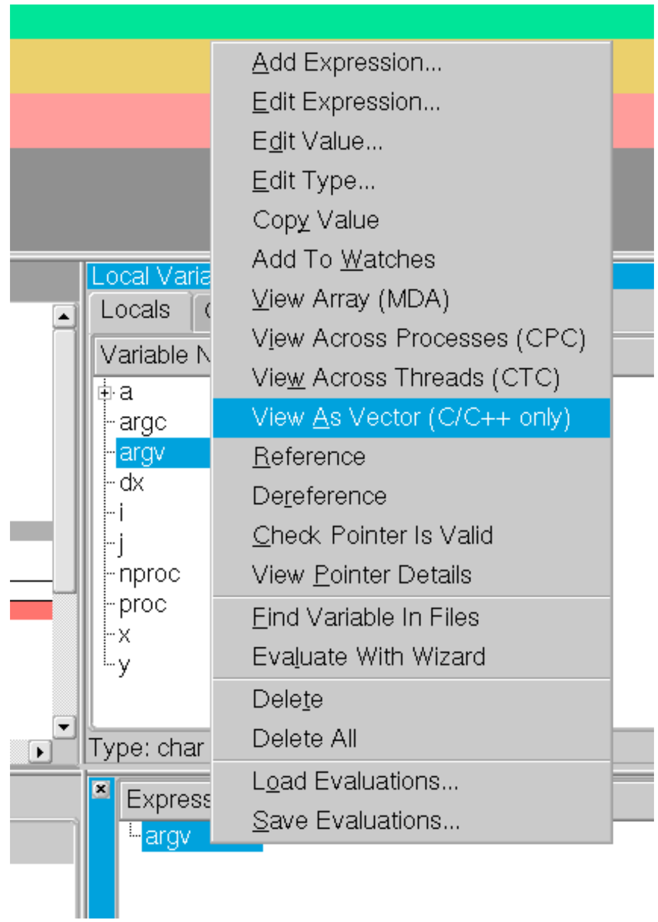
- When a watch point is triggered DDT will pause execution and issue a warning
- This gives you the opportunity to analyze the current execution state and the values of your variables

multidimensional array viewer, memory stats, parallel message queue

VIEWING AND EDITING DATA

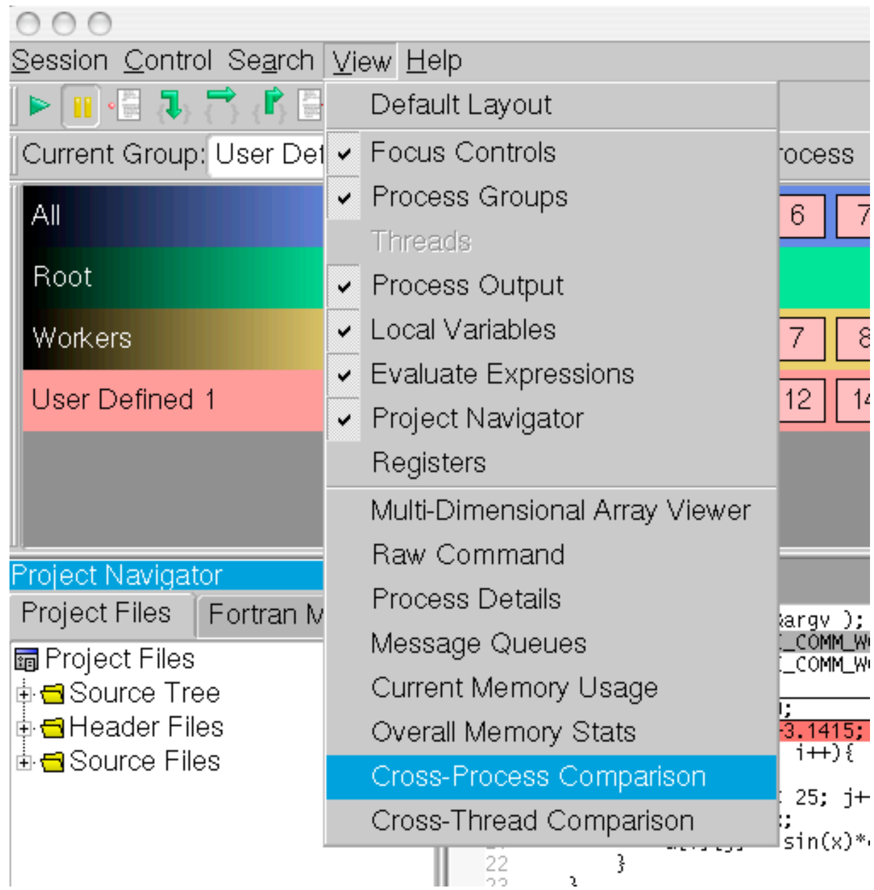
Viewing and Editing Variables

- 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

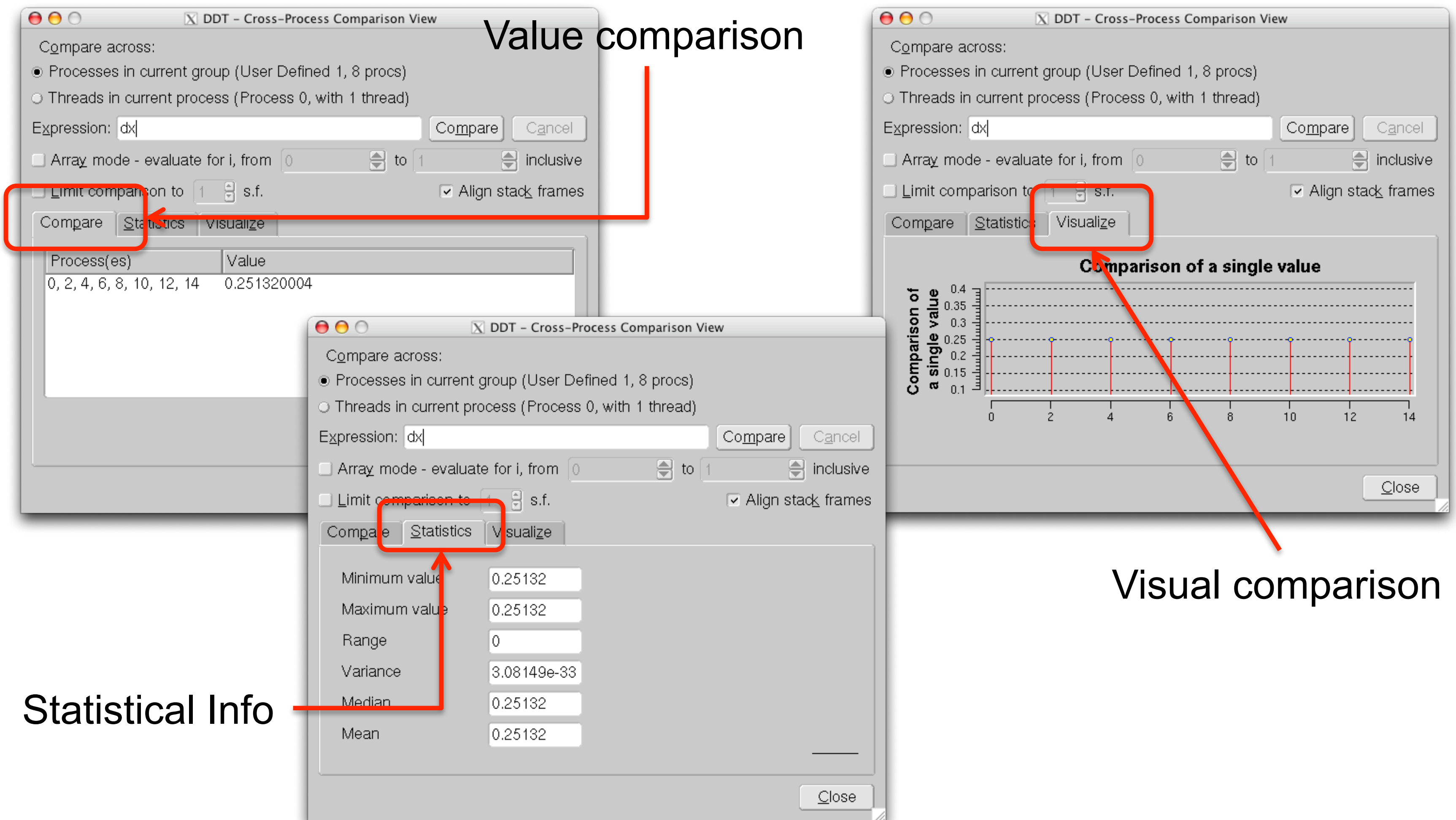


Data Values Across Processes

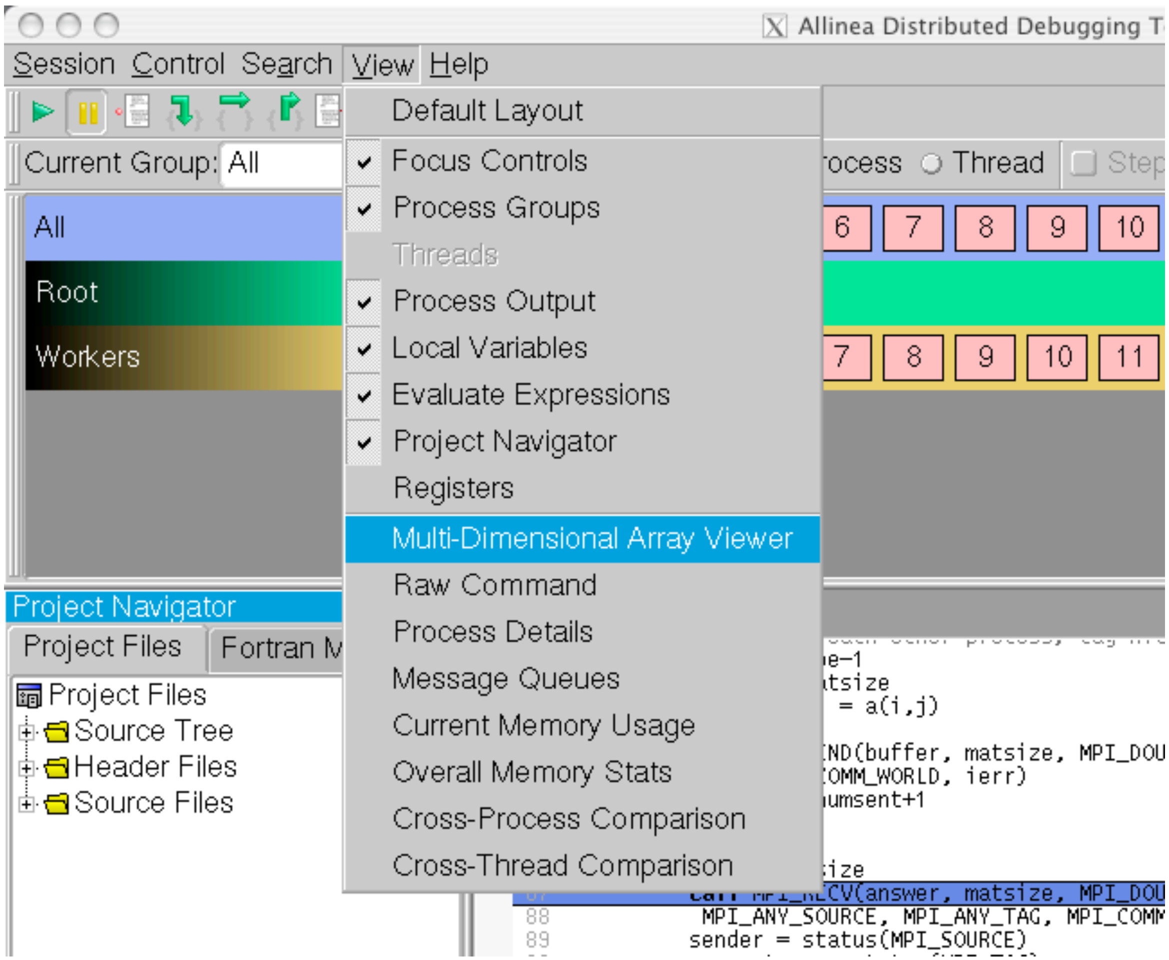
- View->Cross-Process Comparison
- View variable values across all processes in a group
- View->Cross-Thread Comparison
- View variable values across all threads in a process



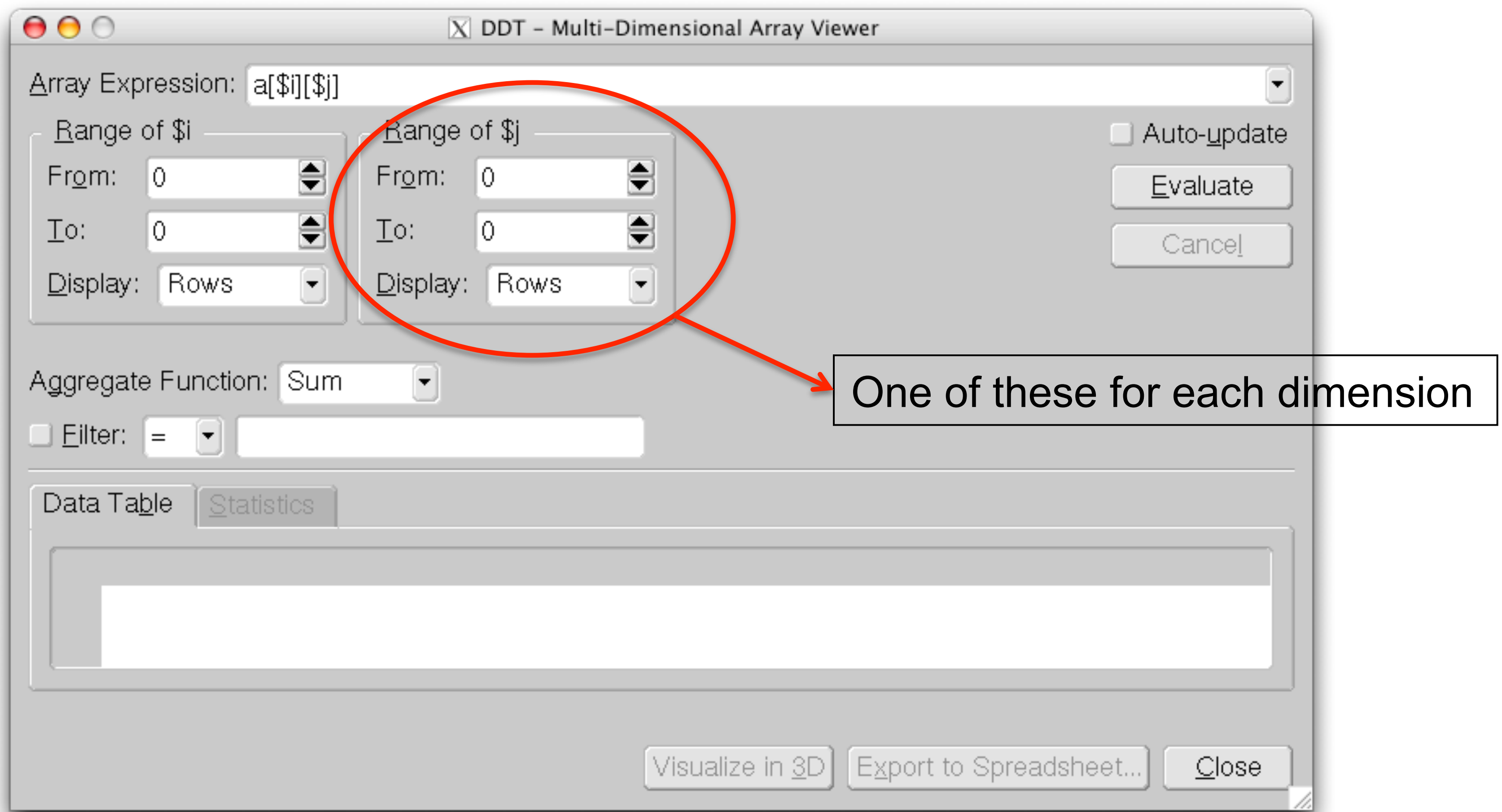
Data Values Across Processes (2)



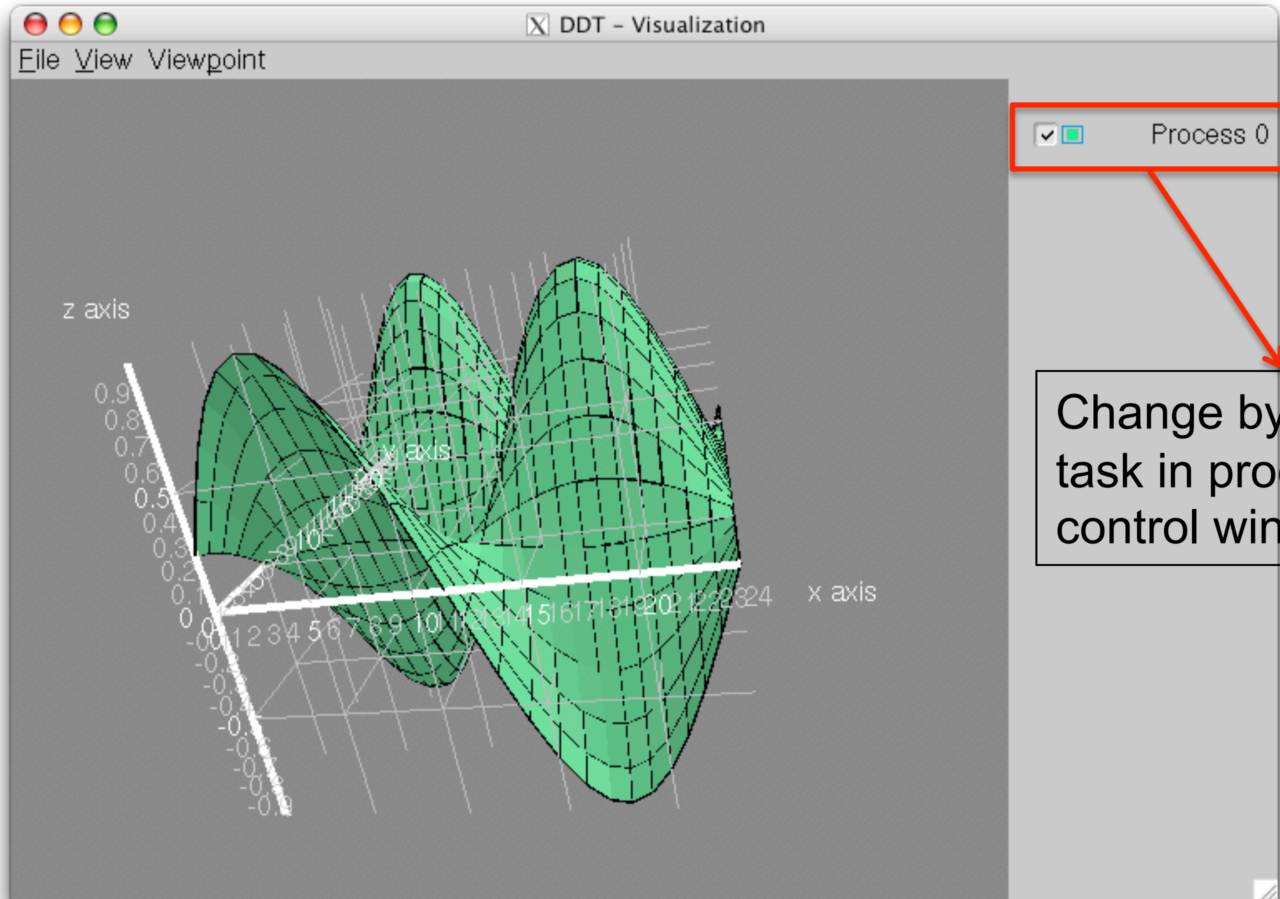
Array data access



Multi-Dimensional Array Viewer



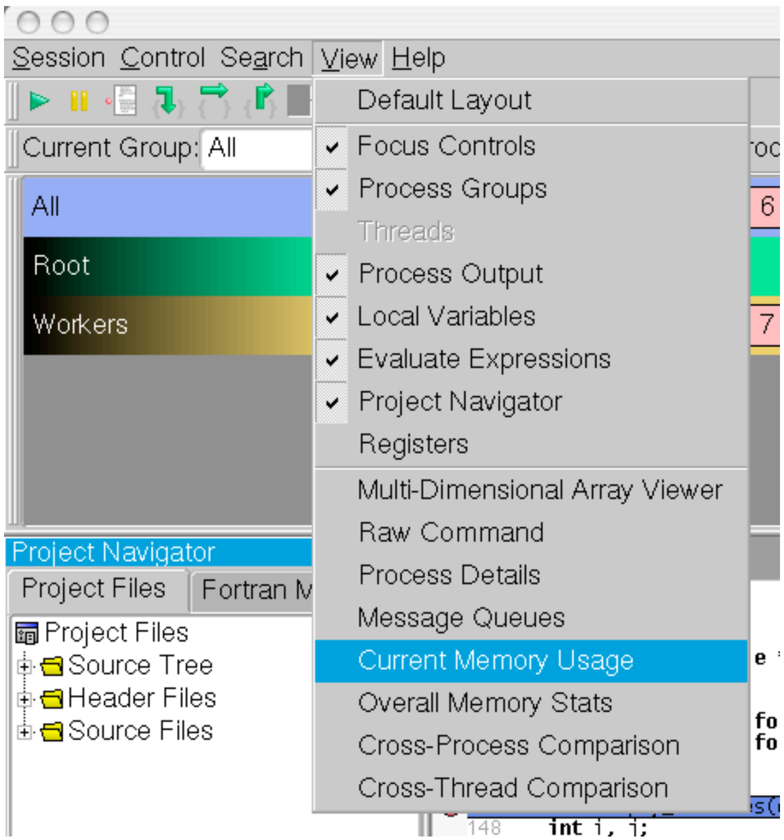
Data Visualization



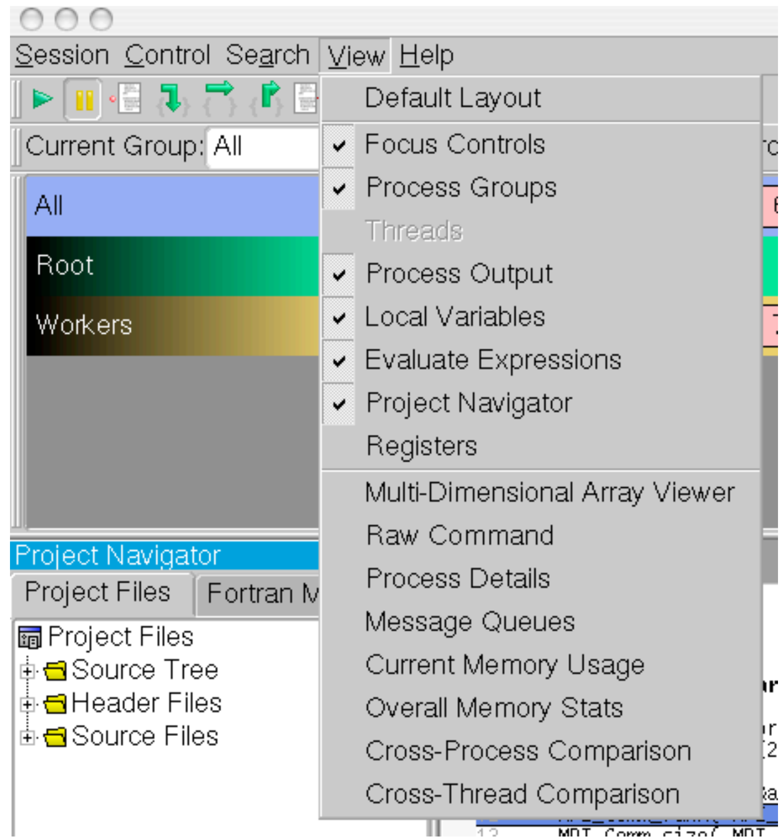
Change by selecting task in process control window

Analyzing Memory Usage

Current Memory Usage

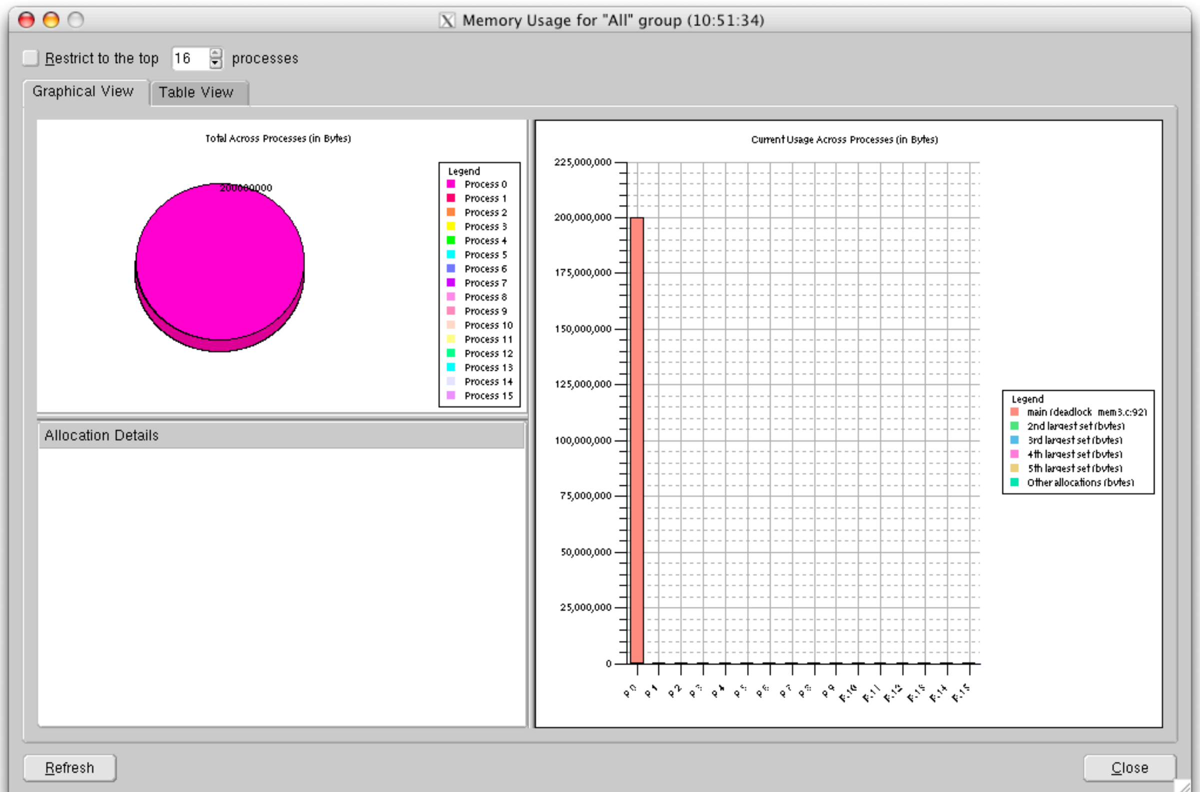


Overall Memory Stats



Current memory Usage

Go to View -> Current Memory Usage

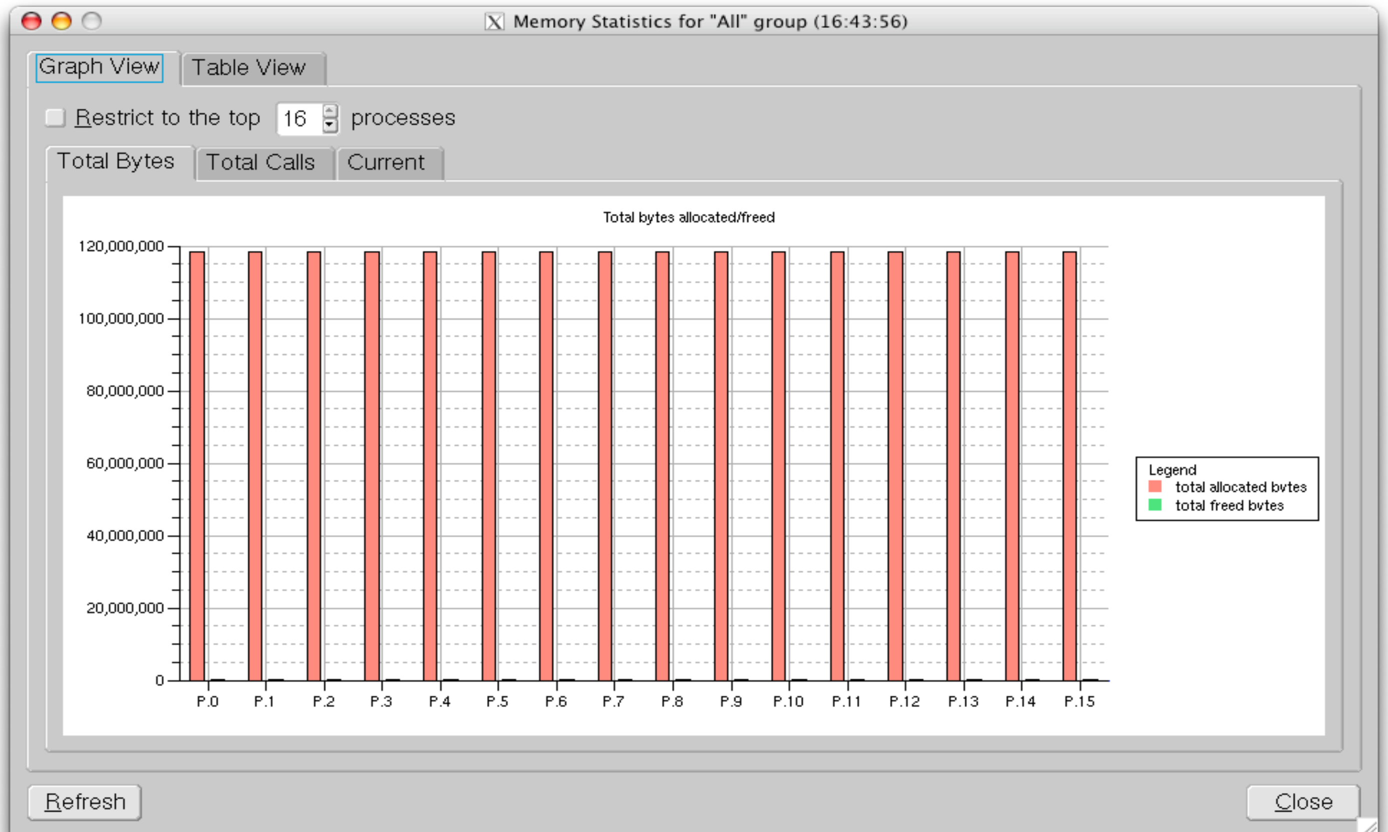


Process 0 is using much more memory than the others.

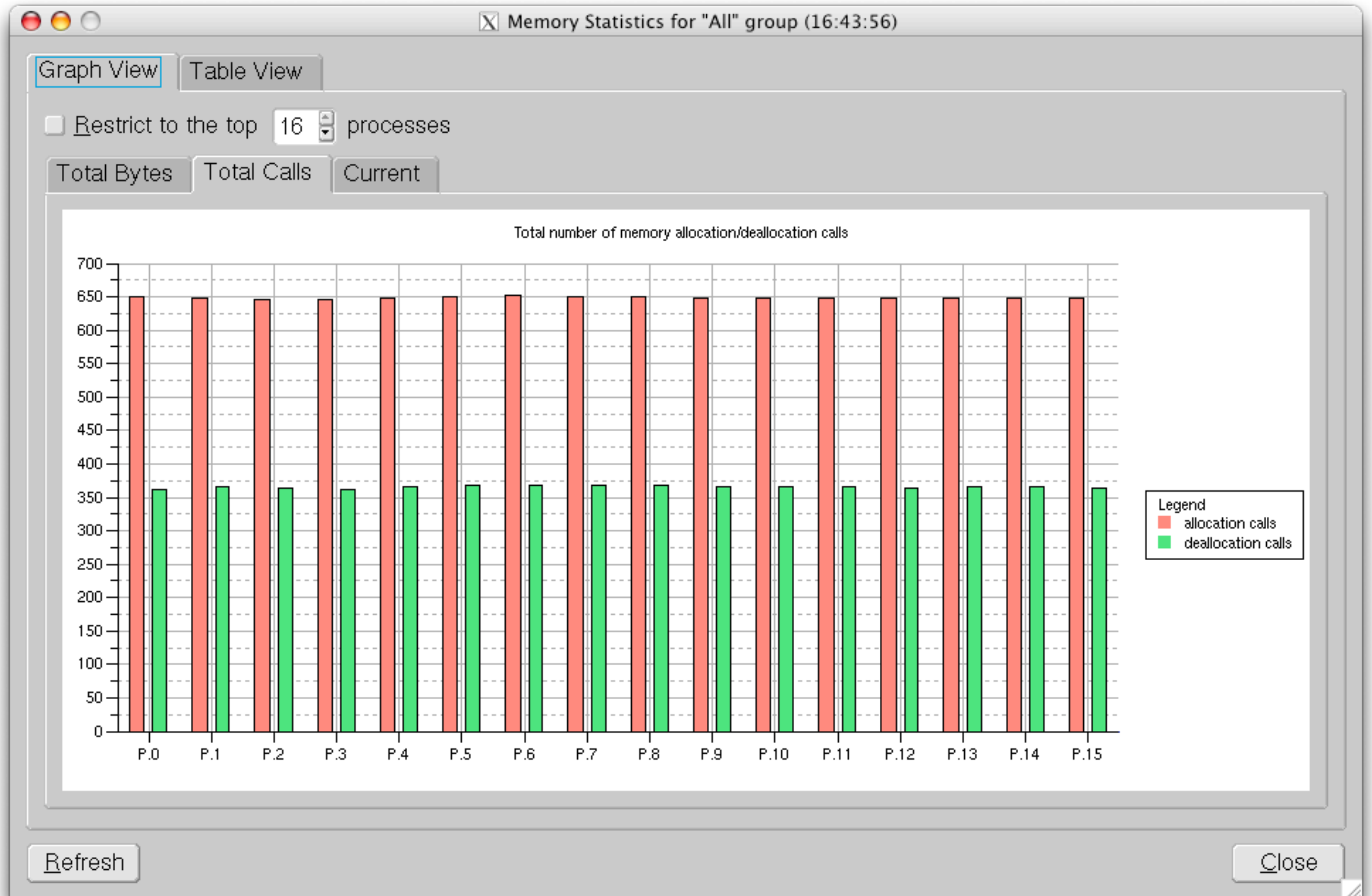
This looks like a memory leak.

Overall Memory Stats (Size)

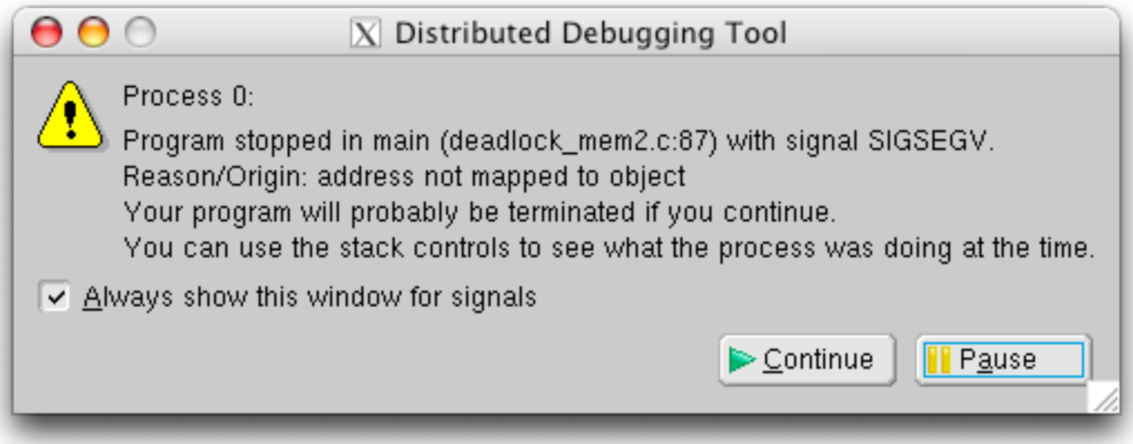
Go to View -> Overall Memory Stats



Overall Memory Stats (Calls)



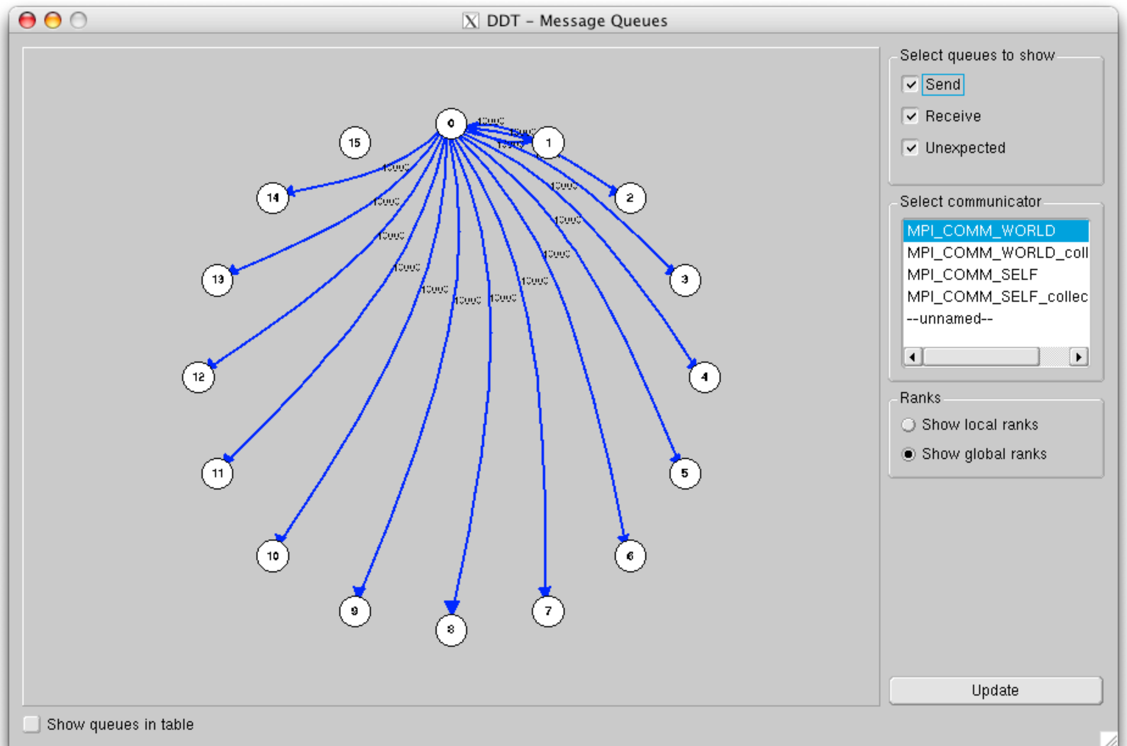
Segmentation Faults



- 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

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.