

Debugging and Profiling Lab

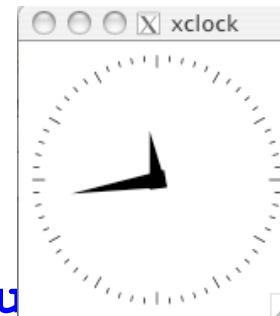
Carlos Rosales, Kent Milfeld, Yaakoub Y. El Kharma,
Victor Eijkhout
carlos@tacc.utexas.edu



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Setup

- Login to stampede or lonestar:
 - `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

- Get the lab files:
 - <https://bitbucket.org/VictorEijkhout/parallel-computing-book>
 - [booksources/projects-public/debug_lab](https://bitbucket.org/VictorEijkhout/booksources/projects-public/debug_lab)

DEBUGGING LAB



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Finding a deadlock with DDT

- In this example we will use **DDT** to debug a code that deadlocks.
- Compile the deadlock example:

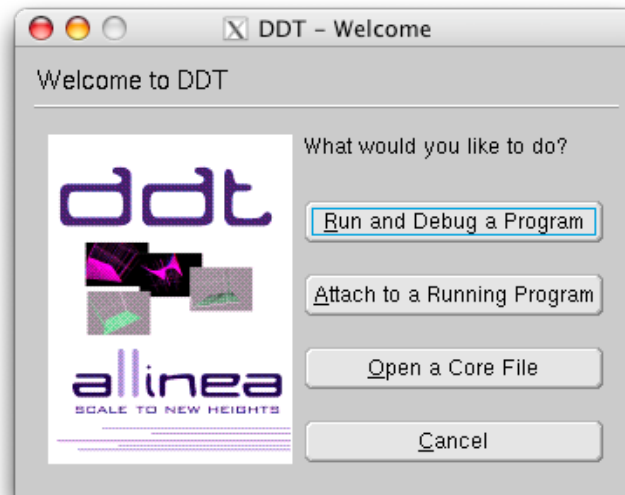
```
% cd debug_lab  
% mpicc -g -O0 ./deadlock.c
```
- Load the DDT module:

```
% module load ddt
```
- Start up DDT:

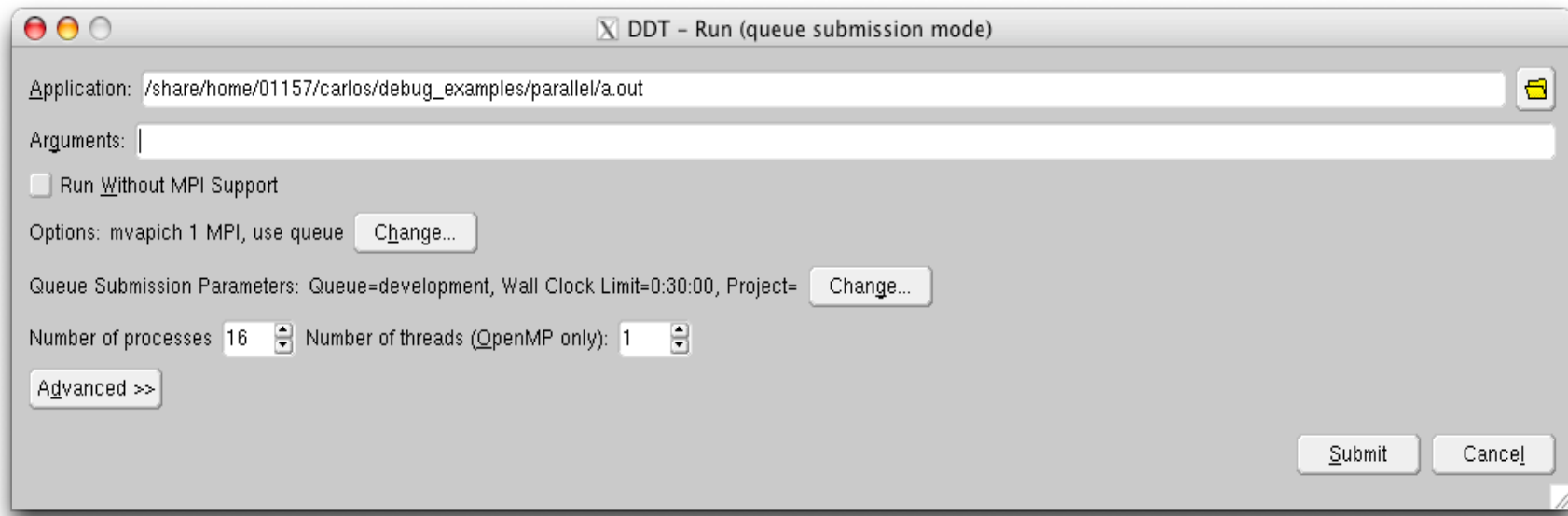
```
% ddt ./a.out
```

Configure DDT: Welcome

When you see the welcome screen below click the button that says “Run and Debug a Program”.



Configure DDT: Job Submission

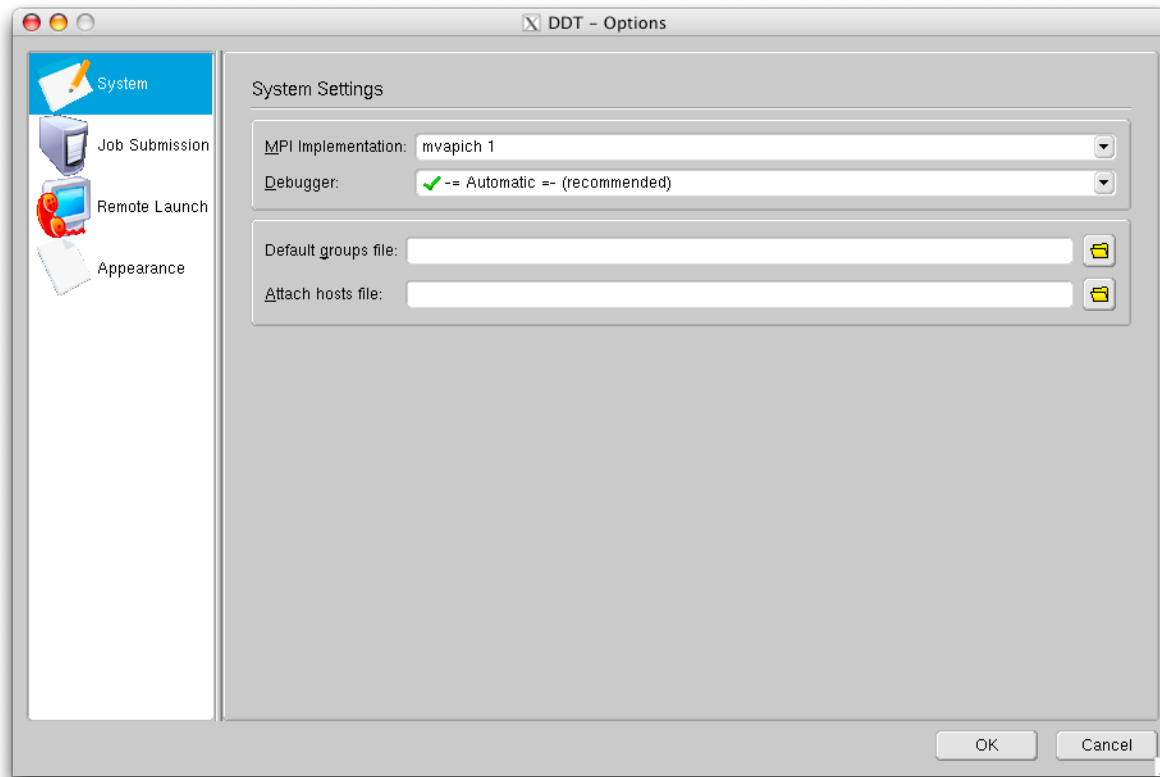


Don't click submit yet! We need to configure:

- General Options
- Queue Submission Parameters
- Processor and thread number
- Advanced Options

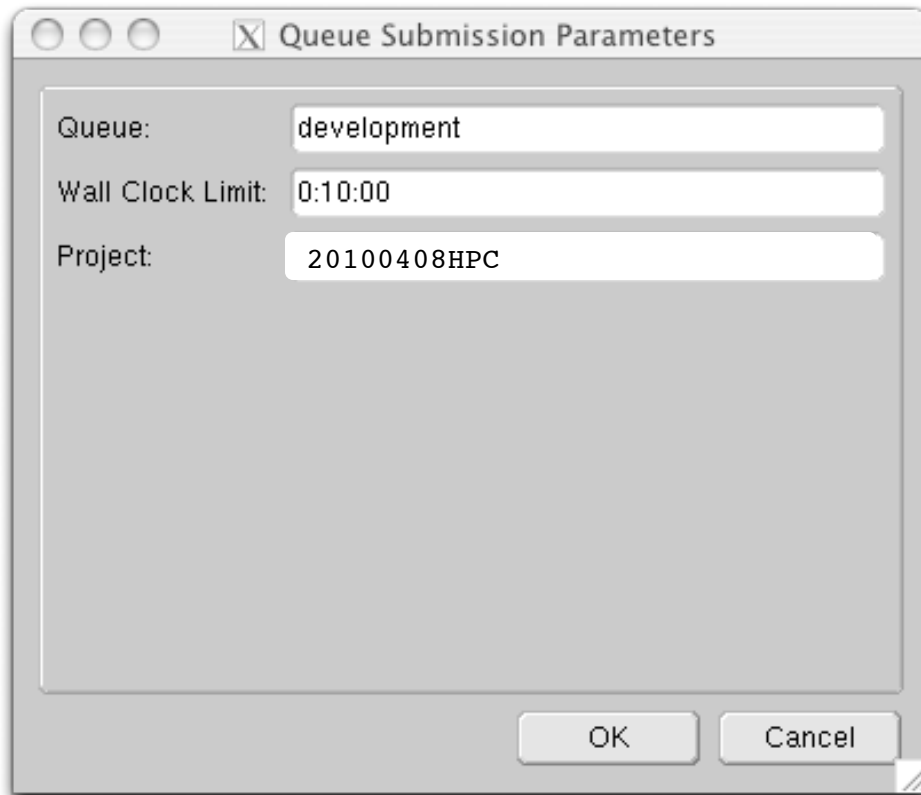
Click on Options -> Change

Configure DDT: Options



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

Configure DDT: Queue Parameters



Queue Submission Parameters

Queue: development

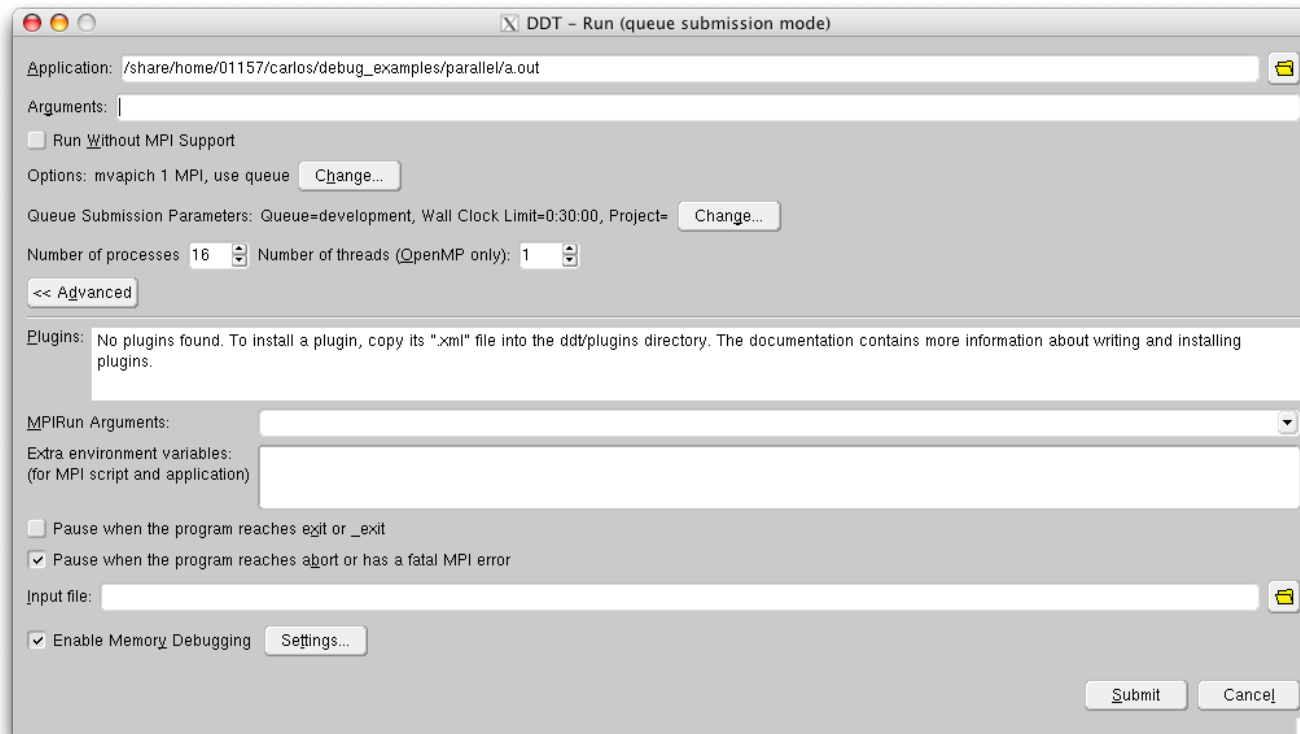
Wall Clock Limit: 0:10:00

Project: 20100408HPC

OK Cancel

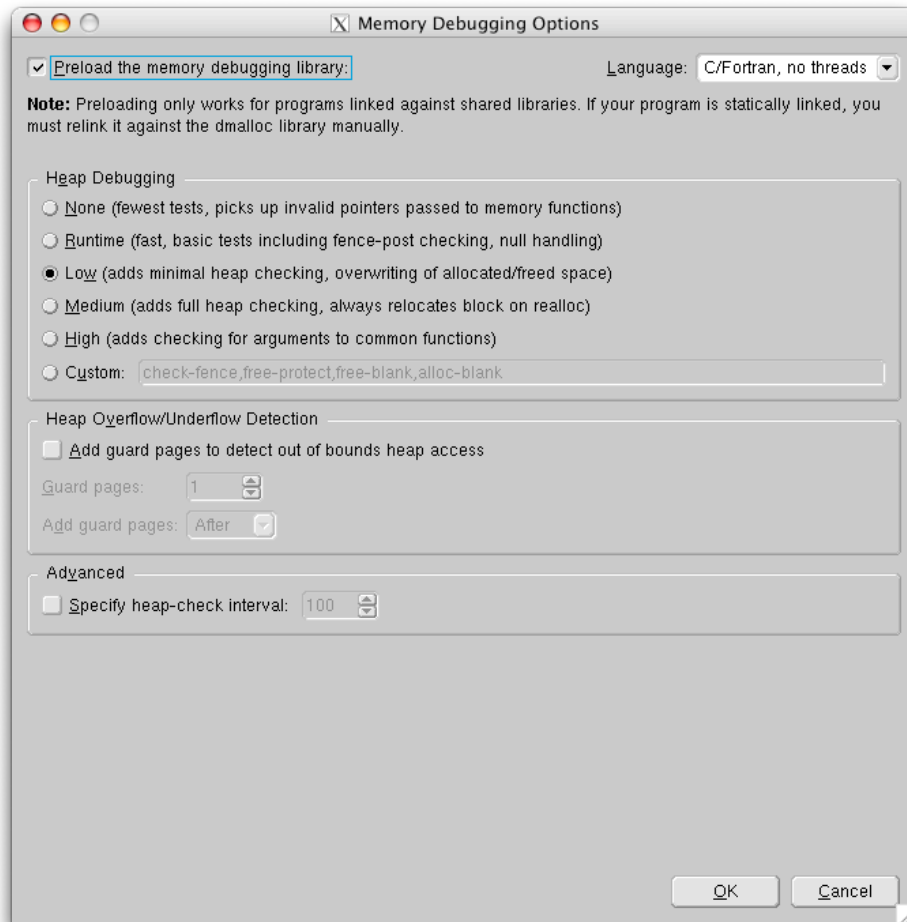
- Choose the “development” queue
- Set the Wall Clock Limit to 10 minutes (H:MM:SS)
- Set your project code - for this class use TACC-PCSE
- Click OK and double check that you have selected 12/16 CPUs / 1 thread in the main Job Submission window.

Configure DDT: Memory Checks



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

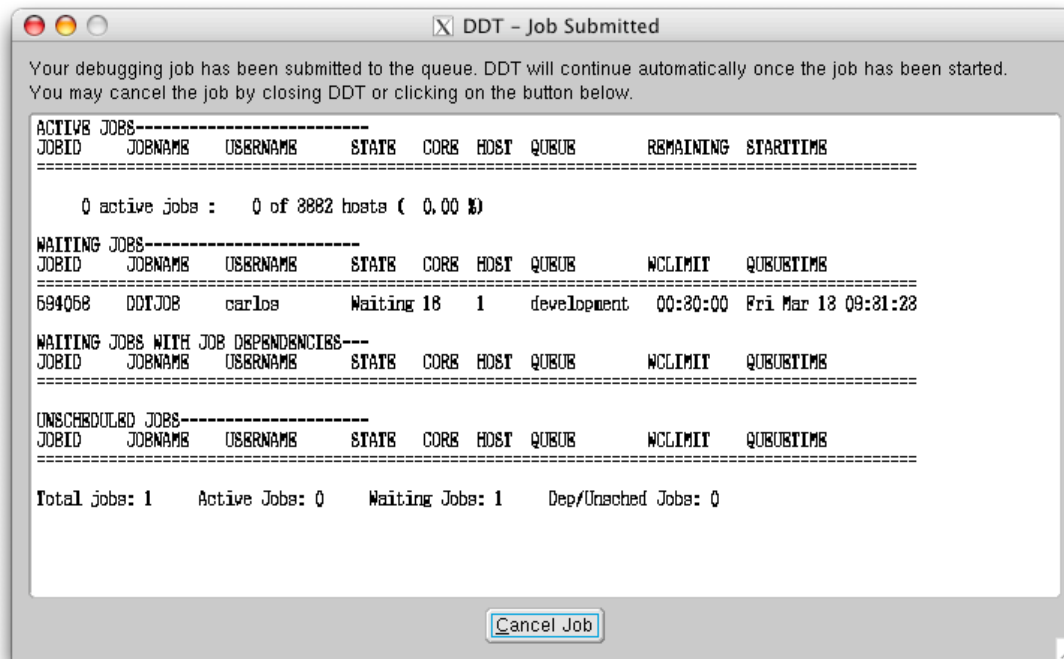
Configure DDT: Memory Options



- Change the Heap Debugging option from the default **Runtime** to **Low**
- Even the option None provides some memory checking
- Leave Heap and Advanced unchecked

DDT: Job Queuing

Add any necessary arguments to the program (none for the example)
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 displays the Allinea Distributed Debugging Tool (DDT) v2.3.1 interface. The top toolbar includes buttons for Session, Control, Search, View, and Help. Below the toolbar, a 'Current Group' dropdown is set to 'All', and radio buttons allow focusing on the current Group, Process, or Thread. A 'Step Threads Together' checkbox is also present. The main area shows a hierarchical view of process groups: 'All' (blue bar with 16 sub-items), 'Root' (green bar with 1 sub-item), and 'Workers' (yellow bar with 16 sub-items). The 'Project Navigator' on the left shows a tree of Project Files, Fortran Modules, Source Tree, Header Files, and Source Files. The central 'Code window' displays the source code for 'deadlock_mem.c', with line 31 highlighted. The 'Local Variables' window on the right shows a table with 'Variable Name' and 'Value' columns, containing 'nprocs' with the value 4210165. The bottom section contains a 'Stack view and output window' with tabs for Stdout, Stderr, Stdin ('All' group), Breakpoints, Watches, and Stacks; the 'Procs' tab is active, showing 'main (deadlock_mem.c:30)'. To its right is an 'Evaluation window' with 'Expression' and 'Value' tabs. Labels with arrows point to the 'Process controls' (top toolbar), 'Process groups window' (hierarchical view), 'Project navigation window' (Project Navigator), 'Code window' (source code), 'Variable window' (Local Variables), 'Stack view and output window' (bottom left), and 'Evaluation window' (bottom right).

Process controls

Process groups window

Project navigation window

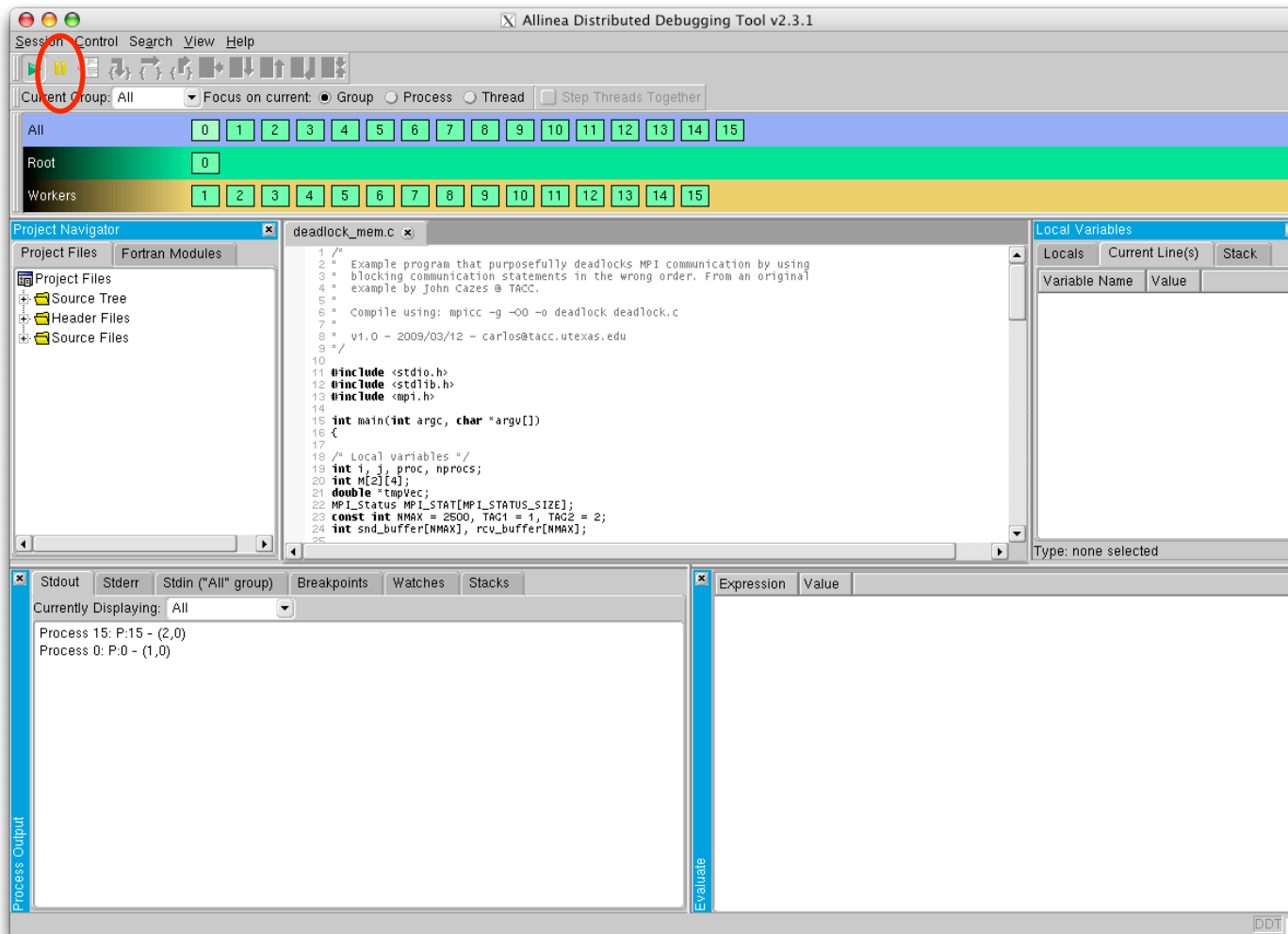
Code window

Variable window

Stack view and output window

Evaluation window

DDT: Program Hangs

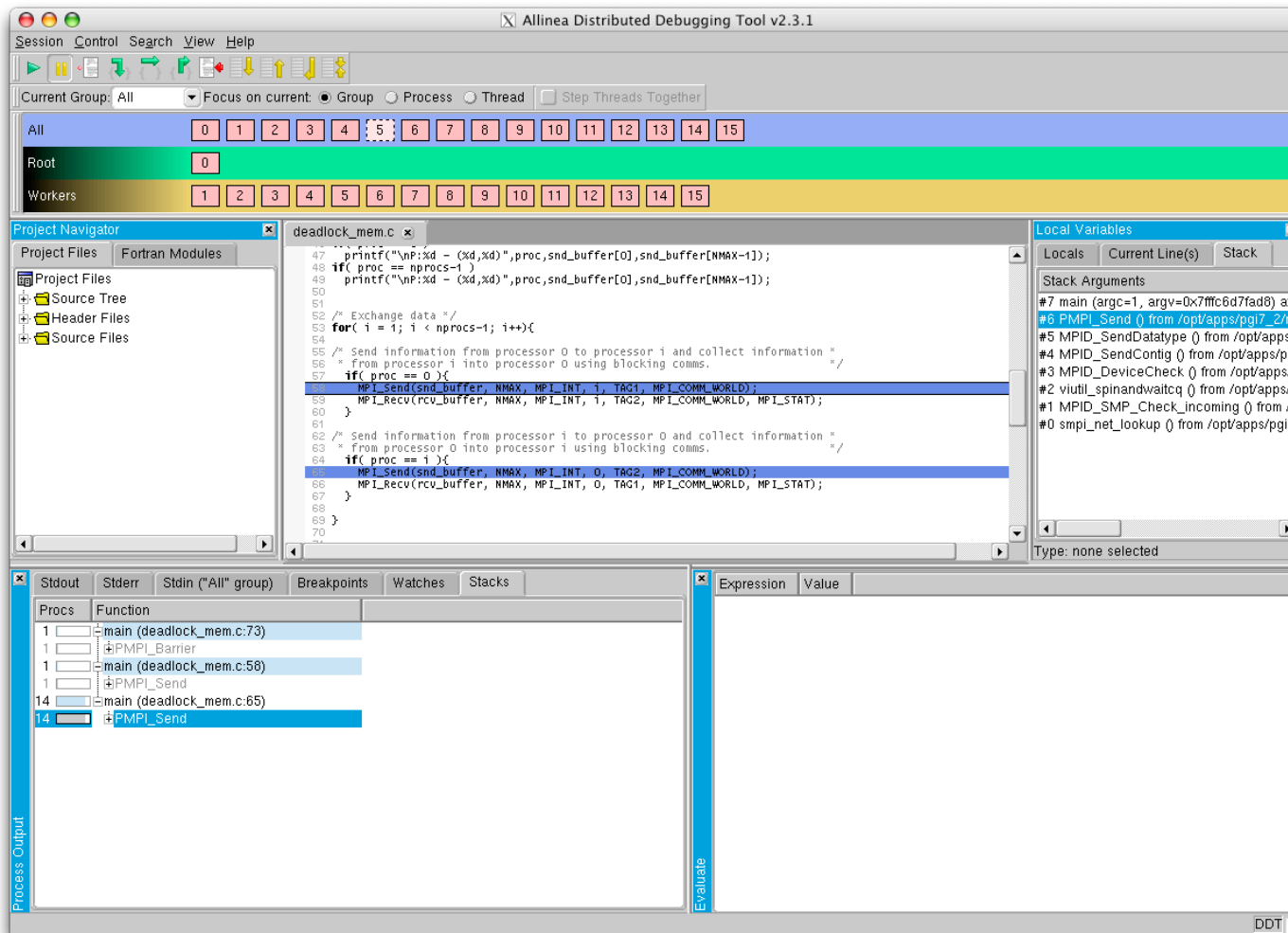


The output we expect does not appear in the Stdout window.

No active communication between procs.

Stop execution to analyze the program status (top left).

DDT: Stacks

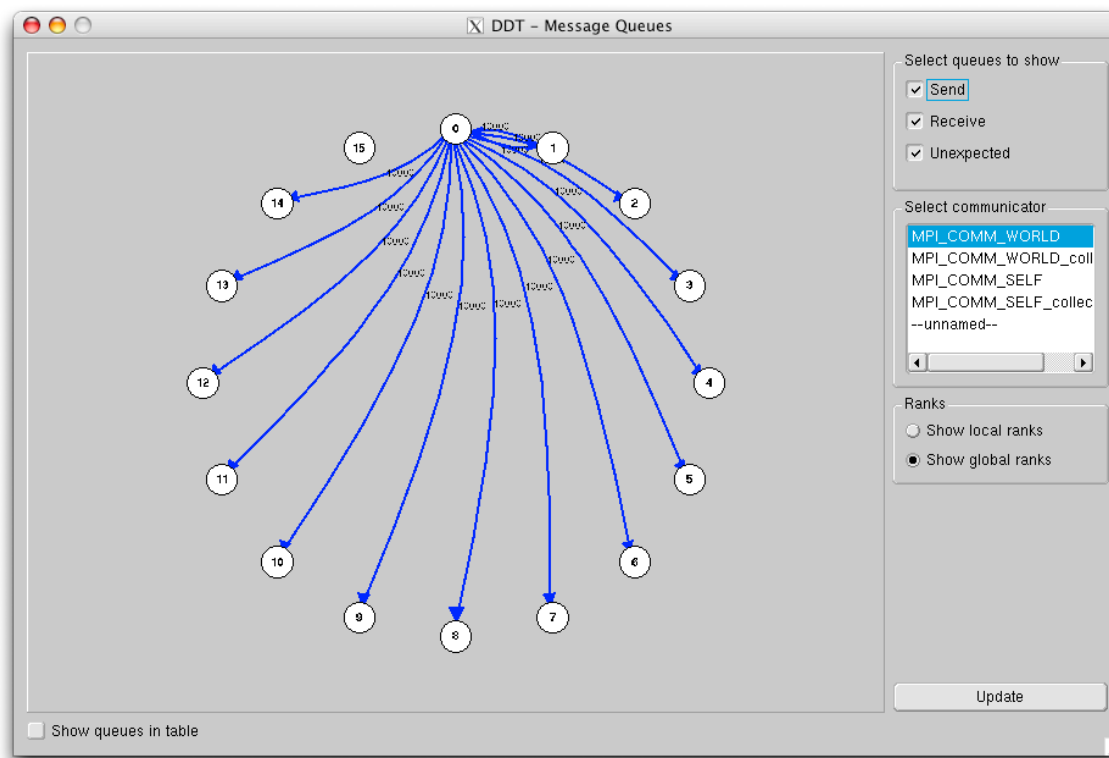


On the bottom left window select the Stacks view.

All processors seem to be stuck on a MPI_Send().

DDT: Message Queues

Go to View -> Message Queues



There are uncompleted Send messages everywhere!

You can double-check that all communications are in the “Unexpected queue” (select on top right)

This is characteristic of a deadlock.

Find the source of the deadlock in the code.

PARALLEL SCALABILITY LAB



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Parallel Scalability: IPM

- In this example you will use **IPM** to evaluate the scalability of a matrix multiplication code.
- Load the IPM module:
 - `module load ipm`
 - `module list`
- Compile the matmult.c or matmult.f90 source with the **-g** flag:
 - `mpicc -g./matmult.c`
 - `mpif90 -g./matmult.f90`
- Open the Sun Grid Engine script **ipm_job.sge** and make sure the following lines appear before the ibrun command is invoked:
 - `export LD_PRELOAD=$TACC_IPM_LIB/libipm.so`
 - `export IPM_REPORT=full`

Parallel Scalability: IPM

- Submit the job through the SGE queue system:
 - `qsub ./ipm_job.sge`
- When the job is done IPM will generate an xml file with a name like:
 - `username.1298314568.32191.0`
- Have a look at the basic text report by typing:
 - `ipm_parse username.1298314568.32191.0`
- You can also read the full text report:
 - `ipm_parse -full username.1298314568.32191.0`

Parallel Scalability: IPM

- Try transforming the output file to HTML:
 - `ipm_parse -html username.1298314568.32191.0`
- A new directory containing an **index.html** file will be created. You can copy this directory to your laptop and view the contents with any web browser.
- In your laptop, open the index.html file and explore the different performance data provided by IPM.

Parallel Scalability: mpiP

- In this example you will use **mpiP** to evaluate the scalability of a matrix multiplication code.
- Load the mpiP module:
 - `module load mpiP`
 - `module list`
- Compile the matmult.c or matmult.f90 source with the flags required to link in the mpiP library:
 - `mpicc -g -L$TACC_MPIP_LIB -lmpiP -lbfd -liberty ./matmult.c`
 - `mpif90 -g -L$TACC_MPIP_LIB -lmpiP -lbfd -liberty ./matmult.f90`
- Set the environmental variables that control mpiP data collection behavior:
 - `setenv MPIP '-t 10 -k 2'`

Parallel Scalability: mpiP

- Submit the job through the SGE queue system:
 - `qsub ./parallel_job.sge`
- The initial submission using 2 processing cores only (-pe 2way 16). Check execution and MPI times in the .mpiP file created.
- Change the submission script to use 4 cores (-pe 4way 16), 8 and 16, and build a table with the execution times.
- Does the execution time decrease linearly with the number of cores? Why?

SIZE	2 cores	4 cores	8 cores	16 cores
1000 x 1000				
2000 x 2000				

PROFILING LAB



THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Profiling with Tau: Compilation

- Load the papi and tau modules:
 - `module load papi`
 - `module load tau`
- Set the TAU_MAKEFILE environmental variable
 - `setenv TAU_MAKEFILE $TACC_TAU_LIB/Makefile.tau-multiplecounters-mpi-papi-pdt-pgi`
- If you have changed to the Intel compiler use instead:
 - `setenv TAU_MAKEFILE $TACC_TAU_LIB/Makefile.tau-icpc-multiplecounters-mpi-papi-pdt`
- Compile the matrix multiplication example using the Tau compiler wrappers:
 - `tau_cc.sh matmult.c`
 - `tau_f90.sh matmult.f90`

Profiling with Tau: Job Script

- Open **tau_job.sge** and make sure the following lines - which define the hardware counters to measure- appear before the ibrun invocation:
 - `export COUNTER1=GET_TIME_OF_DAY`
 - `export COUNTER2=PAPI_FP_OPS`
 - `export COUNTER3=PAPI_L1_DCM`
- Submit the job through the batch queue system:
 - `qsub tau_job.sge`
- When the job completes execution you should have three new directories:
 - `MULTI__GET_TIME_OF_DAY`
 - `MULTI__PAPI_FP_OPS`
 - `MULTI__PAPI_L1_DCM`

Profiling with Tau: Analysis

- Analyze the results:
 - `paraprof`
- Get used to the interface
 - Unstack the bars to get a clearer view
 - Open a window with the function names corresponding to each color
- Generate a derived metric that gives you the floating point operation to L1 data cache miss ratio
- Remember that you can copy these directories and analyze them in your own laptop as well