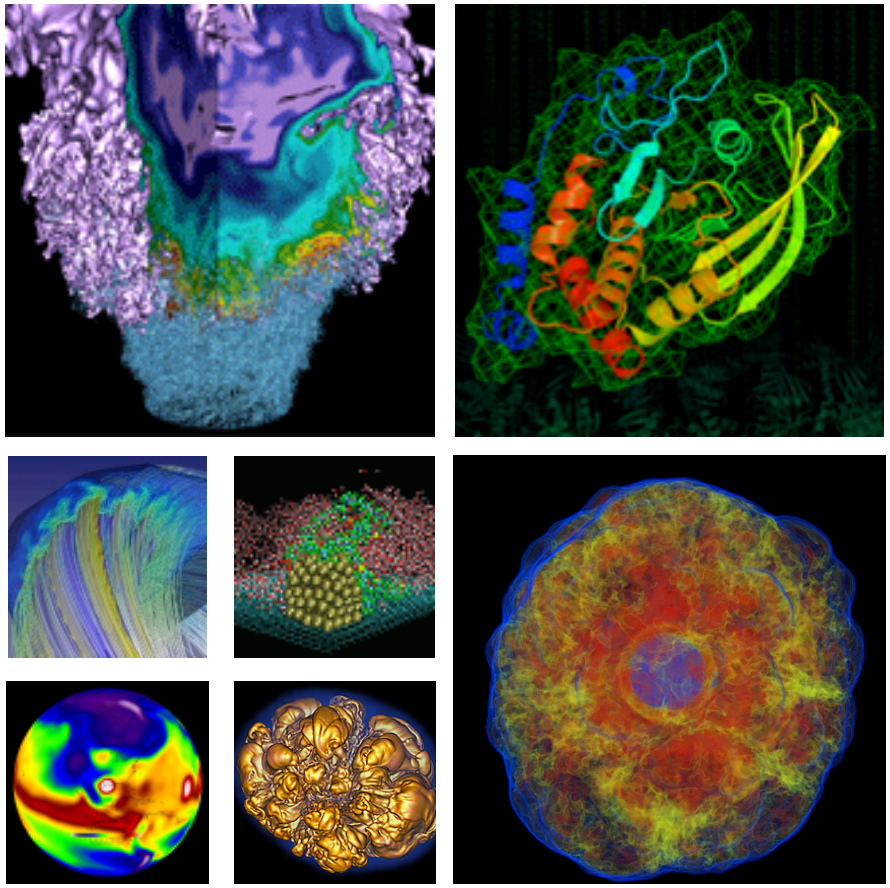
Debugging with DDT



Woo-Sun Yang

NERSC User Services Group

February 15, 2013

Why a debugger?

- **Your code fails and you want to know why**
- **You control the pace of running the code and examine execution flow or variables to see if it is running as expected (better than a print statement!)**
- **Typical scenario**
  - Set a place in your program where you want your program to stop execution
  - Let your program run until the place is reached
  - Check variables
- **Gdb is good but we need to control multiple processors for a parallel program**

DDT

- **Distributed Debugging Tool by Allinea**
- **Graphical debugger capable of debugging**
  - Serial
  - MPI
  - OpenMP
  - CAF
  - UPC
  - CUDA (NERSC doesn't have a license on Dirac)
- **Intuitive and simple user interfaces**
- **Available on Hopper, Carver and Edison**
- **Can debug for up to 8192 tasks at NERSC**
  - Shared among users and machine

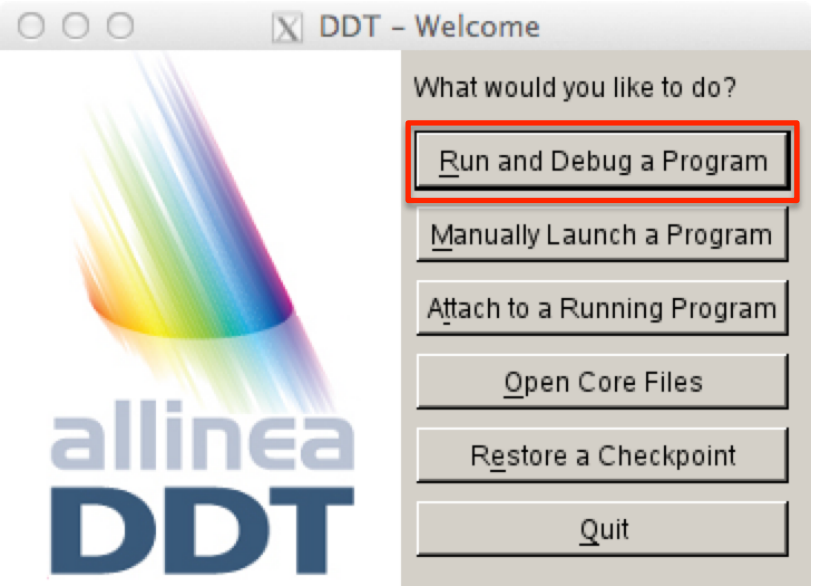
Starting DDT

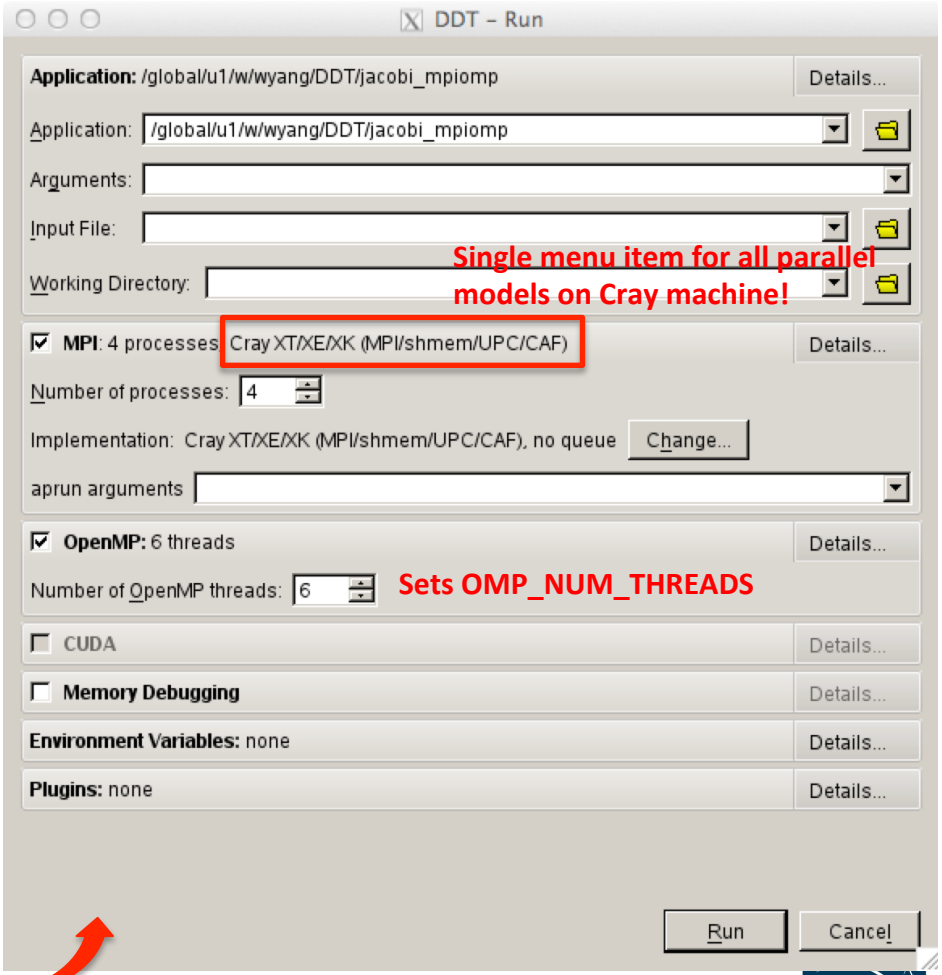
- Compile the code with the -g flag
- Start DDT in an interactive batch session

```
% ftn -g prog.f                          # Hopper/Edison
% mpif90 -g prog.f                       # Carver

% qsub -I -lmppwidth=24 -q debug -V      # Hopper/Edison
% qsub -I -lnodes=1:ppn=8 -q debug -V    # Carver
...
% cd $PBS_O_WORKDIR
% module load ddt

% ddt ./a.out
```

Starting DDT (cont'd)

- **Set program name, parallel programming model, number of MPI tasks and/or threads**





DDT window

Action buttons

Process groups

Allinea DDT v3.2.1-27702

Session Control Search View Help

Current Group: All Focus on current: Group Process Thread Step Threads

Process/thread control

All 0 1 2 3

Create Group

Project Files

Search (Ctrl+K)

Project Files
- Source Tree
- Header Files
- Source Files

jacobi_mpiomp.f90

```
201   integer n, js, je
202   real    u(0:n,js-1:je+1)
203   integer i, j, joff, np, myid
204   real    h
205   integer ierr
206
207   call mpi_comm_size(mpi_comm_world,np,ierr)
208   call mpi_comm_rank(mpi_comm_world,myid,ierr)
209
210   joff = myid * ((n + 1) / np)      ! j-index offset
211
212   h = 1.0 / n
213
214   if (myid == 0) then
215 !$omp parallel do
216     do i=0,n
217       u(i,js) = (i * h)**2
218     enddo
219 !$omp end parallel do
220   endif
```

Locals | Current Line(s) | Current Stack

Current Line(s)

Variable Name	Value
joff	92160000
myid	0
n	9999
np	4

Sparklines

Local Variables or Variables in the current line(s)

Type: none selected

Input/Output | Breakpoints | Watchpoints | Stacks | Tracepoints | Tracepoint Output

Stacks

Processes	Function
4	jacobi_mpiomp (jacobi_mpiomp.f90:45)
4	init_fields (jacobi_mpiomp.f90:191)
4	set_bc (jacobi_mpiomp.f90:210)

Parallel Stack/IO/action points/…

Evaluate

Expression	Value
ierr > 0	.FALSE.

Evaluation

Action Points

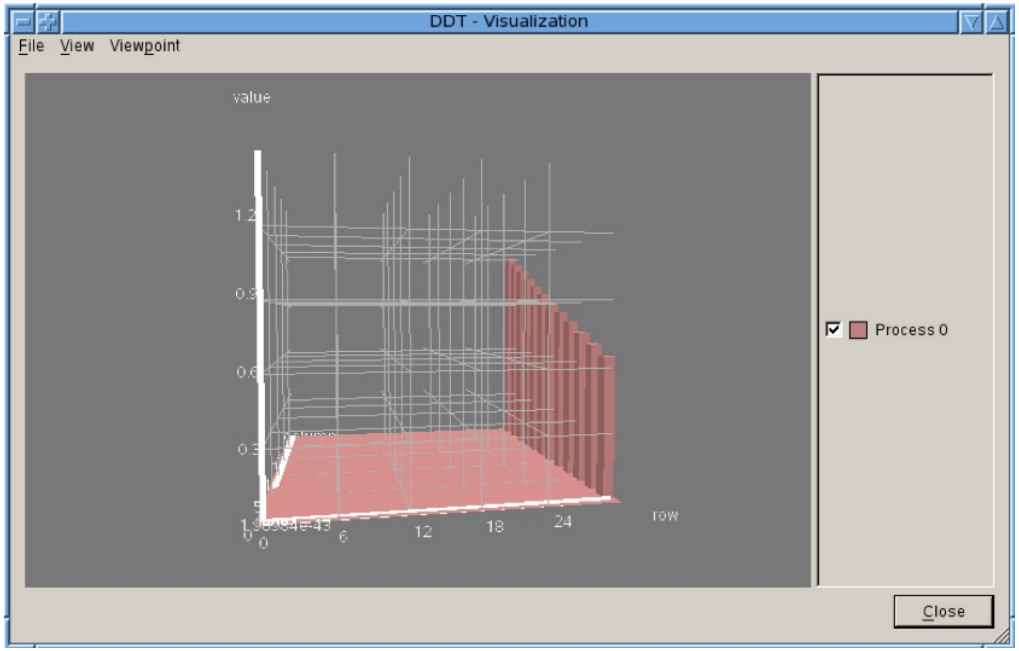
- **Make a code do something when a certain condition is met**
- **Breakpoint**
  - Stops execution when a selected line (breakpoint) is reached
  - Double click on a line to create one; there are other ways, too
- **Watchpoint for variables or expressions**
  - Stops every time a variable or expression changes its value
- **Tracepoint**
  - Prints the file and line number and variable's value when a selected line is reached
  - Just like saying "Hi, I'm here"
- **Can add a condition for an action point**
  - Useful inside a loop
- **Can be enabled or disabled**
- **Action points are listed in the lower left panel of DDT window**

Many ways to check data

- **Right-click on a variable for a quick summary**
- **Variable pane**
- **Evaluate pane**
- **Sparkline for variation over tasks or threads**
- **Display values of a variable over tasks or threads**
  - '**Compare Across Processes**' or '**Compare Across Threads**'
  - Right click on a variable and then select
- **MDA (Multi-dimensional Array)**
- **…**

Multi-dimensional Array Viewer (MDA)

- **MDA can be selected in *View* menu, or by right-clicking on an array and then selecting *View Array***
- **Visualize an array**
- **Can filter values by setting a condition**
- **Can be useful for quickly identifying a problem**
  - The following shows odd values in an array section. It is because the array is not properly initialized when entering a subroutine – it is using garbage values.



  - Incorrect MPI communication can display something similar in ghost node regions

Memory debugging

- **Intercept calls to system memory allocation library, to get memory usage and monitor correct usage**
- **Useful for**
  - Detecting memory leaks
  - Detecting heap overflow/underflow (out-of-bound array references)
  - Finding out memory usage stats

Building code for memory debugging

- **Carver**
  - Build as usual

- **Building dynamic binary on Hopper and Edison**
  - After loading the ddt module

Compiler	Commands

PGI, Cray	`% ftn -g -c prog.f`
	`% ftn -dynamic -o prog prog.o ${DDT_LINK_DMALLOC} \`
	`  --Wl,--allow-multiple-definition`

GNU, Intel	`% ftn -g -c prog.f` `% ftn -dynamic -o prog prog.o ${DDT_LINK_DMALLOC} -zmuldefs`

Building code for memory debugging (cont'd)

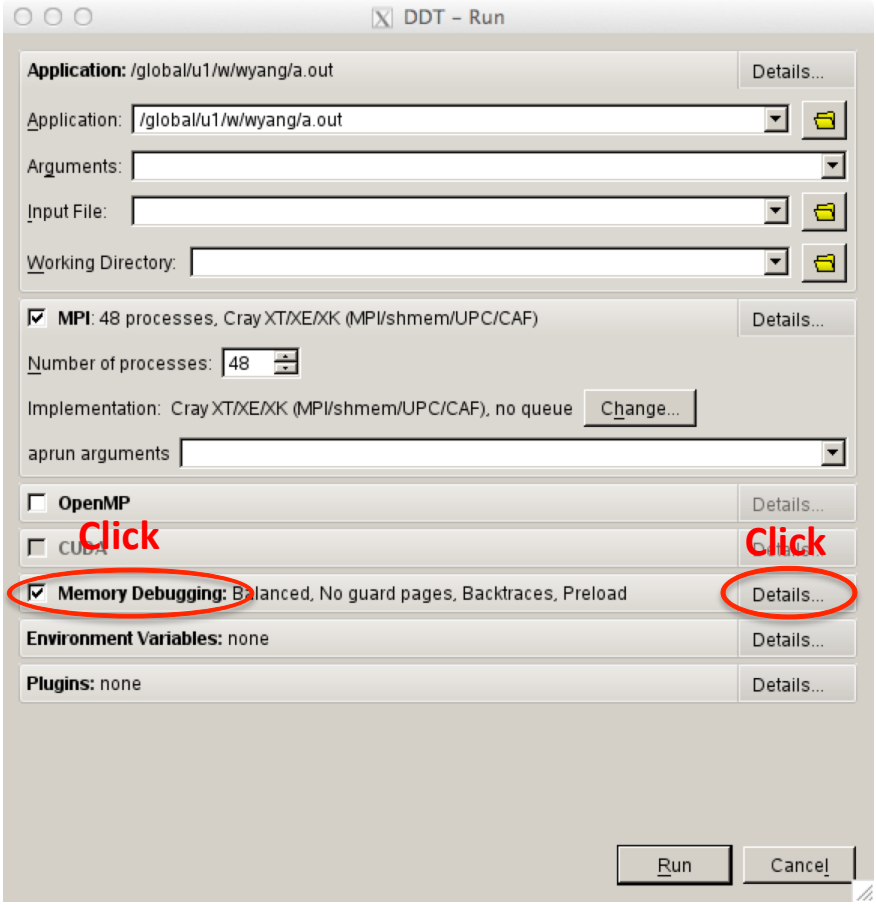
- **Static linking on Hopper and Edison**
  - A lot more complicated than the dynamic linking case
  - Modify the last liker line and rerun it
  - For detailed info, see the '*Memory Debugging*' section in http://www.nersc.gov/users/software/debugging-and-profiling/ddt/

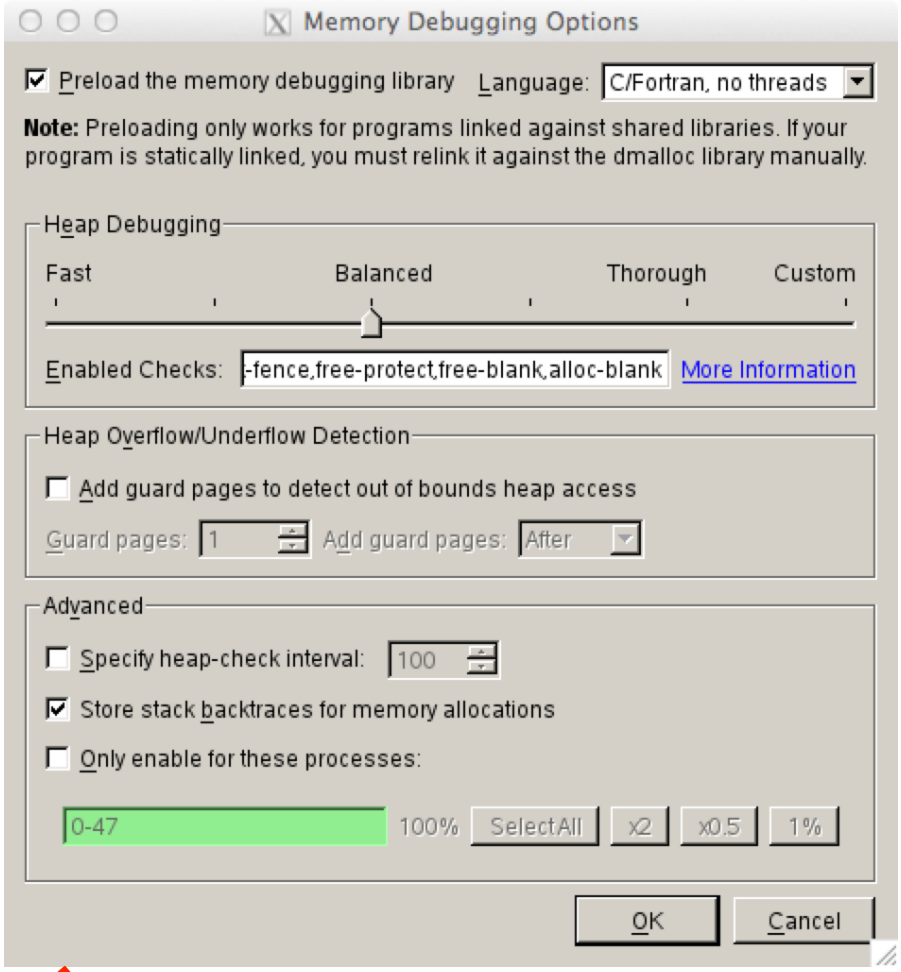
- **NERSC provides utility scripts to help with this task:**

```
% module load ddt
% ftn -g -c prog.f
% static_linking_ddt_md ftn -o prog prog.o
          # instead of 'ftn -o prog prog.o'
```

- **Use *static_linking_ddt_md_th* for a threaded code**

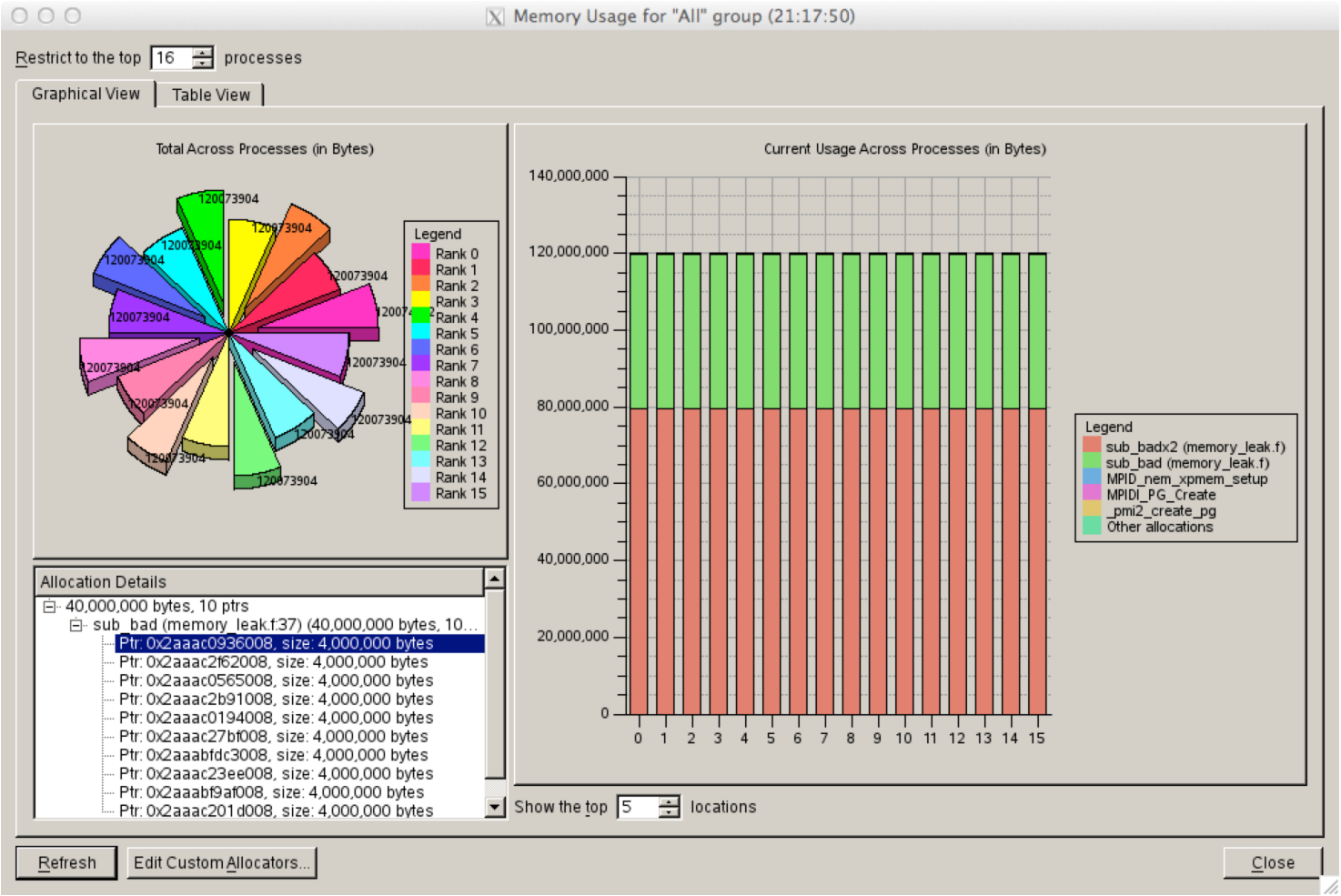
Starting memory debugging





Current memory usage (memory debugging)

- ***Current Memory Usage*** **in** ***View*** **menu**



Memory statistics (memory debugging)

- *Memory Statistics* in *View* menu
- Can be useful for detecting memory leaks

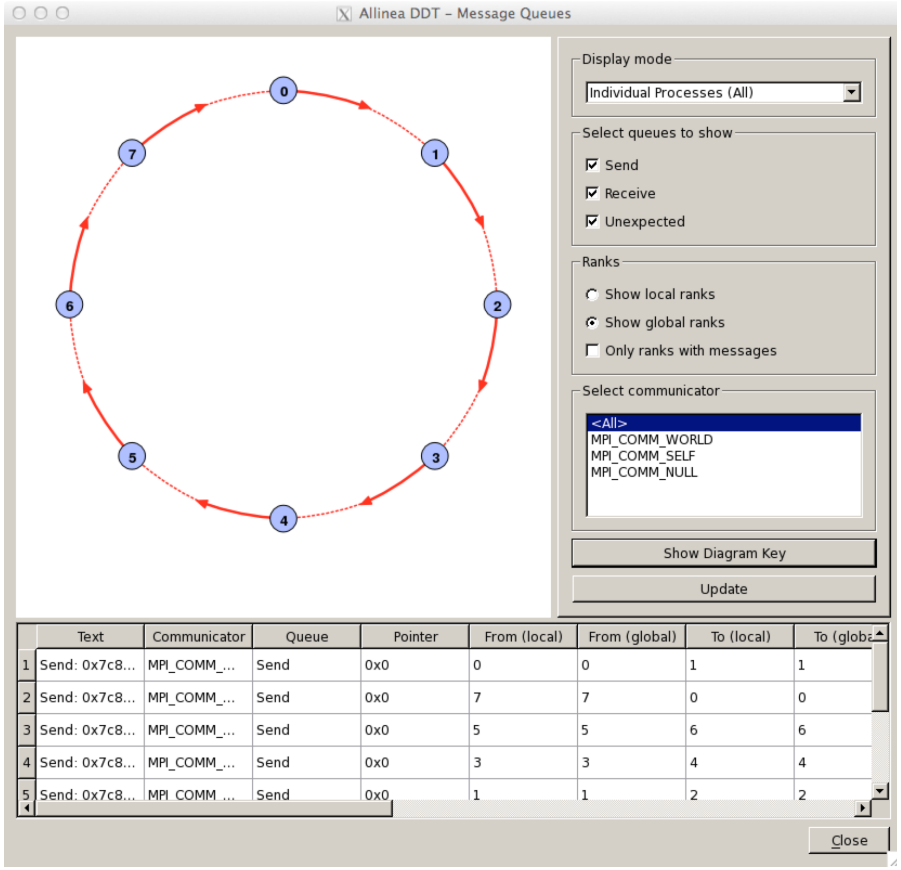
Message queues

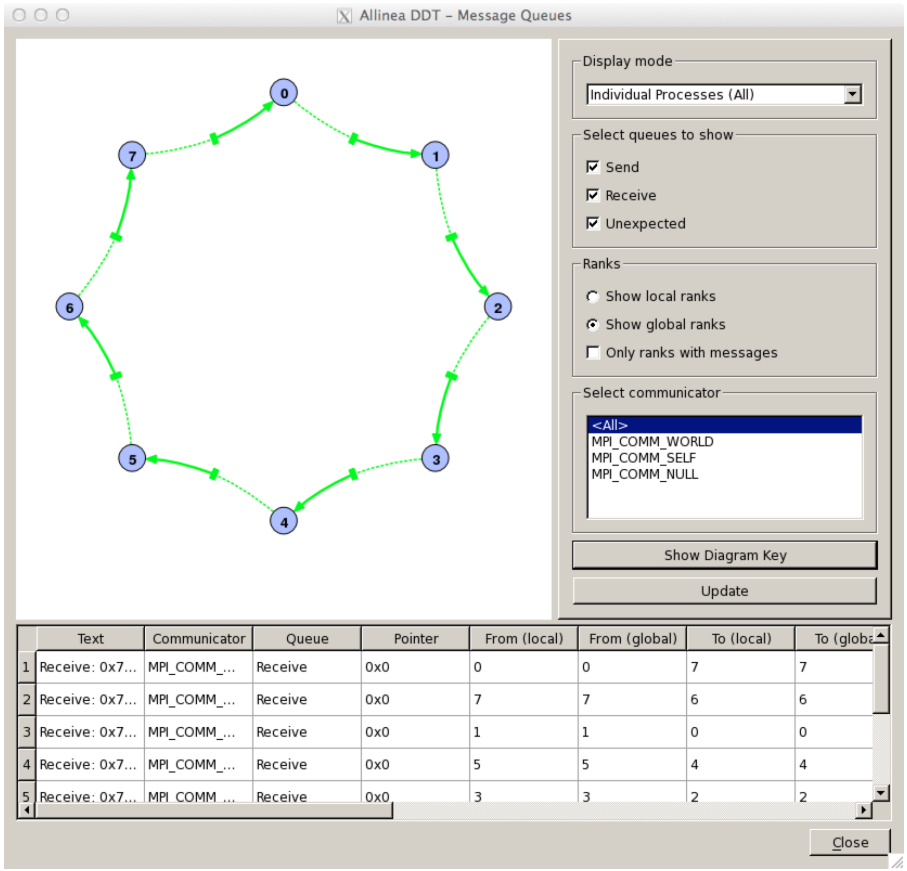
- **Examine status of internal MPI message buffers**
- **Can detect a communication deadlock**
- **Message queue debugging only available on Carver**
- **Three queues are examined**
  - Send Queue
  - Receive Queue
  - Unexpected Message Queue (are you able to see this on Carver?)
- **How to examine these queues: select *View* > *Message Queues***

Message queues examples

```
deadlock with a large message
call mpi_send(sbuf,n,mpi_real,nbr_r,tag,mpi_comm_world,ierr)
call do_something(tbuf,n_t)
call mpi_recv(rbuf,n,mpi_real,nbr_l,tag,mpi_comm_world,stat,ierr)
```

```
always deadlock
call mpi_recv(rbuf,n,mpi_real,nbr_l,tag,mpi_comm_world,stat,ierr)
call do_something(tbuf,n_t)
call mpi_send(sbuf,n,mpi_real,nbr_r,tag,mpi_comm_world,ierr)
```





More Help?

- **User guide on each machine**
  - $DDT_DOCDIR/userguide.pdf
  - From DDT: Help > User Guide
- **http://www.nersc.gov/users/software/debugging-and-profiling/ddt/**
- **NUG2012 DDT tutorial**
  - Some outdated (and incorrect) information there, but detailed info with working code examples can be found there
- **http://www.allinea.com/**

National Energy Research Scientific Computing Center