

Tutorial

Debugging parallel code on distributed systems

Sergey Mashchenko
SHARCNET
McMaster University

Outline

- Part I: overview of DDT
 - Introduction
 - Preparing your program
 - Launching DDT
 - User interface
 - Controlling Program Execution
 - Variables and data
 - Memory Debugging
- Part II: using DDT
 - Running DDT on SHARCNET
 - Hands on

Part I: overview of DDT

Intro

- The Distributed Debugging Tool (DDT) is a powerful commercial debugger with a graphical user interface (GUI).
- It's main intended use is for debugging parallel MPI codes, though it can also be used with serial and parallel threaded (OpenMP / pthreads) programs.
- The product was developed by Alinea (U.K.). It is installed on many SHARCNET clusters. The current version is 1.10.

Intro (2)

- DDT supports C, C++, and Fortran 77 / 90 / 95 / 2003.
- Detailed documentation (the Beginner's guides for C and Fortran, and the Advanced User Guide) is available as PDF files on clusters where DDT is installed (currently these are **requin**, **dolphin**, **megaladon**, and **bruce**), in

`/opt/sharcnet/ddt/current/doc`

Preparing your program

- The code has to be compiled with the switch **-g**, which tells the compiler to generate symbolic information required by any debugger. Normally, all optimizations have to be turned off. For example,

`>g77 -g -o program program.f`

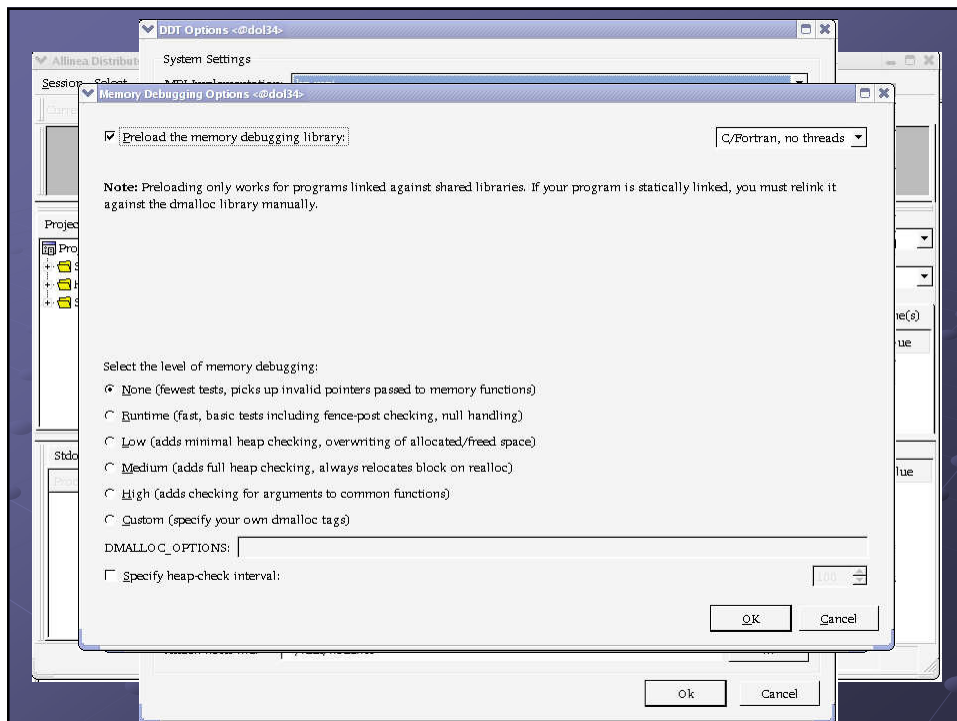
`>mpicc -g -o code code.c`

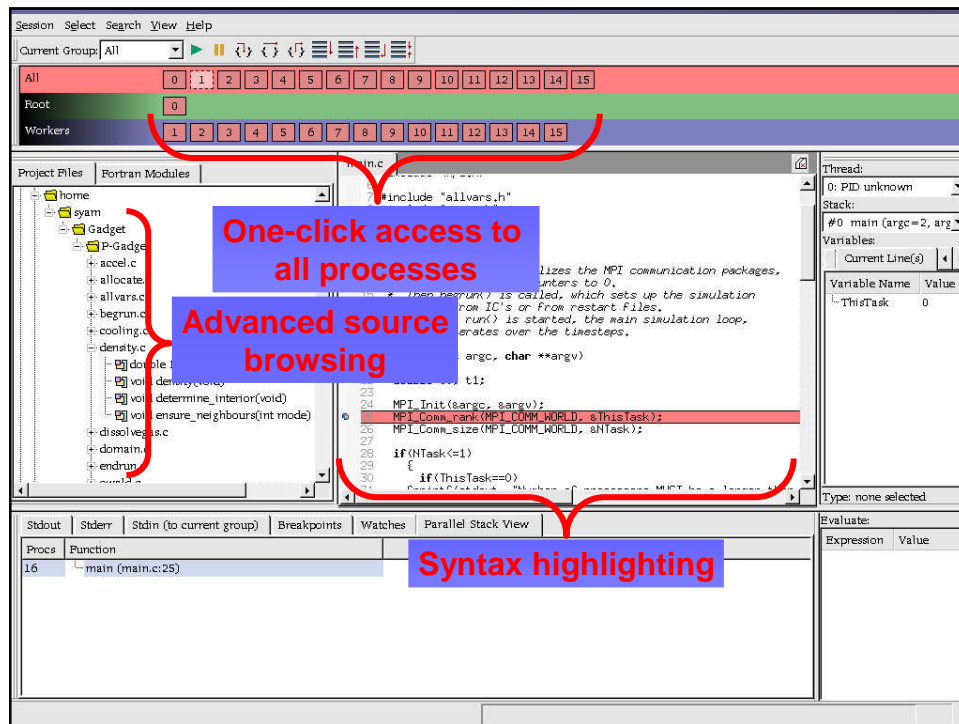
Launching DDT

• Simply type

```
>ddt program [optional arguments]
```

• After a couple of seconds, you will be presented with the following window:





User interface

- DDT uses a tabbed-document interface.
- Each component of DDT is a dockable window, which may be dragged around by a handle. Components can also be double-clicked, or dragged outside of DDT, to form a new window.
- Most of the user-modified parameters and windows are saved by right-clicking and selecting a save option in the corresponding window.

User interface (2)

- DDT also has the ability to load and save all these options concurrently to minimize the inconvenience in restarting sessions. Saving the session stores such things as process groups, the contents of the Evaluate window and more.
- When DDT begins a session, source code is automatically found from the information compiled in the executable.

User interface (3)

- The **`Find`** and **`Find In Files`** dialogs are found from the **`Search`** menu. The **`Find`** dialog will find occurrences of an expression in the currently visible source file. The **`Find In Files`** dialog searches all source and header files associated with your program and lists the matches in a result box. Click on a match to display the file in the main Source Code window and highlight the matching line.

User interface (4)

- DDT has a **jump to line** function which enables the user to go directly to a line of code (in the `Search` menu, or **Ctrl-G**).

Controlling Program Execution

- Process Control And Process Groups
 - Ability to group many process together, with a one-click access to the whole group
 - Three predefined groups: All, Root, Workers.
 - Groups can be created, deleted, or modified by the user at any time



Controlling Program Execution (2)

- Starting, Stopping And Restarting A Program
 - Session Control dialog
- Stepping Through A Program
 - Play/Continue, Pause, Step Into, Step Over, Step Out
- Setting Breakpoints
 - All breakpoints are listed under the breakpoints tab.
 - You can suspend, delete, load or save breakpoints.

Controlling Program Execution (3)

- Breakpoints can easily be made conditional:

Stdout	Stderr	Stdin (to current group)	Breakpoints	Watches	Parallel Stack View
Group	Filename	Line	Function	Condition	Full path
<input checked="" type="checkbox"/>	All	accel.c	54		/home/syam/Gadget/P-Gadget/accel.
<input checked="" type="checkbox"/>	All	global.c	63	X > 0	/home/syam/Gadget/P-Gadget/global

Stdout	Stderr	Stdin	Breakpoints	Watches	Parallel Stack View	
Filename		Line	Function	Condition	Full path	
<input checked="" type="checkbox"/>	re_center3.37.F	162		X .gt. 0	/home/syam/turbo2/re_center3.37.F	
<input checked="" type="checkbox"/>	re_center3.37.F	223			/home/syam/turbo2/re_center3.37.F	

Controlling Program Execution (4)

● Synchronizing Processes

- “Run to here” command (right mouse click)

● Setting A Watch

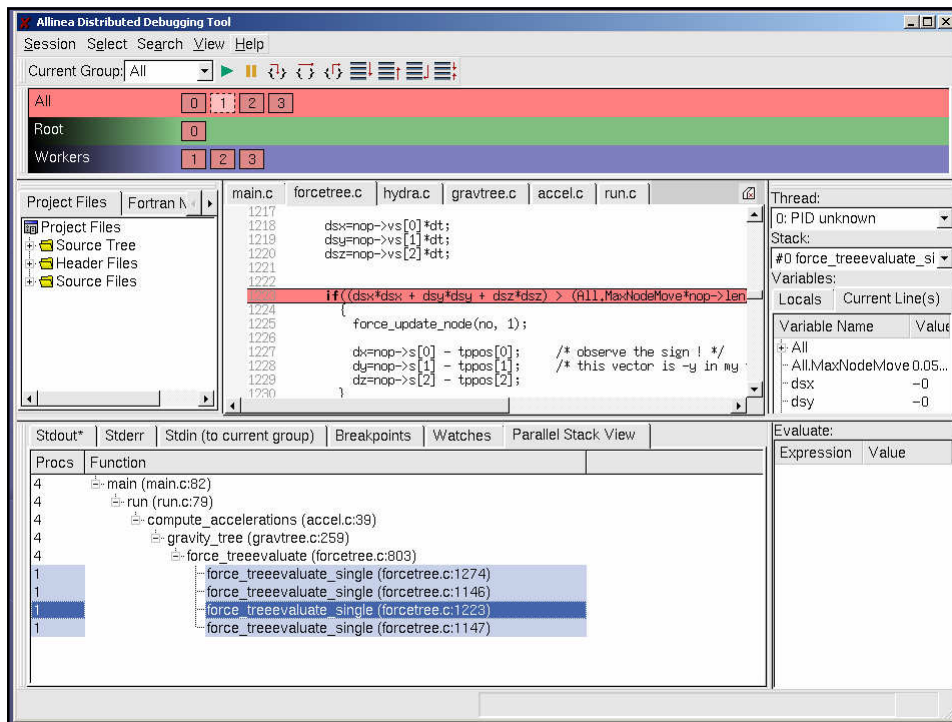
● Working with stacks

- Examining
- Aligning

Controlling Program Execution (5)

● Viewing Stacks in Parallel

- Parallel Stack View tab.
- Shows a tree of functions, merged from every process in the group
- Click on any branch to see its location in the Source Code viewer
- Hover the mouse to see the list of the process ranks at this location in the code
- Can automatically gather the processes at a function together in a new group



Controlling Program Execution (6)

Browsing Source Code

- Highlights the current source line
- Different color coding for synchronous and asynchronous state in the group

Simultaneously Viewing Multiple Files

- Right click to split the source window into two or more, with each one having its own set of tabs.

Controlling Program Execution (7)

● Signal Handling: will stop on the following signals:

- SIGSEGV (segmentation fault)
- SIGFPE (Floating Point Exception)
- SIGPIPE (Broken Pipe)
- SIGILL (Illegal Instruction)

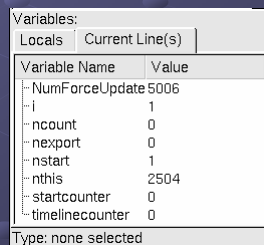
Variables and data

● Current Line tab

- Viewing all the variable for the current line(s)
(click and drag for multiple lines)

● Locals tab

- Shows all local variables for the process



Variables:	
Locals Current Line(s)	
Variable Name	Value
NumForceUpdate	5006
i	1
ncount	0
nexport	0
nstart	1
nthis	2504
startcounter	0
timelinecounter	0

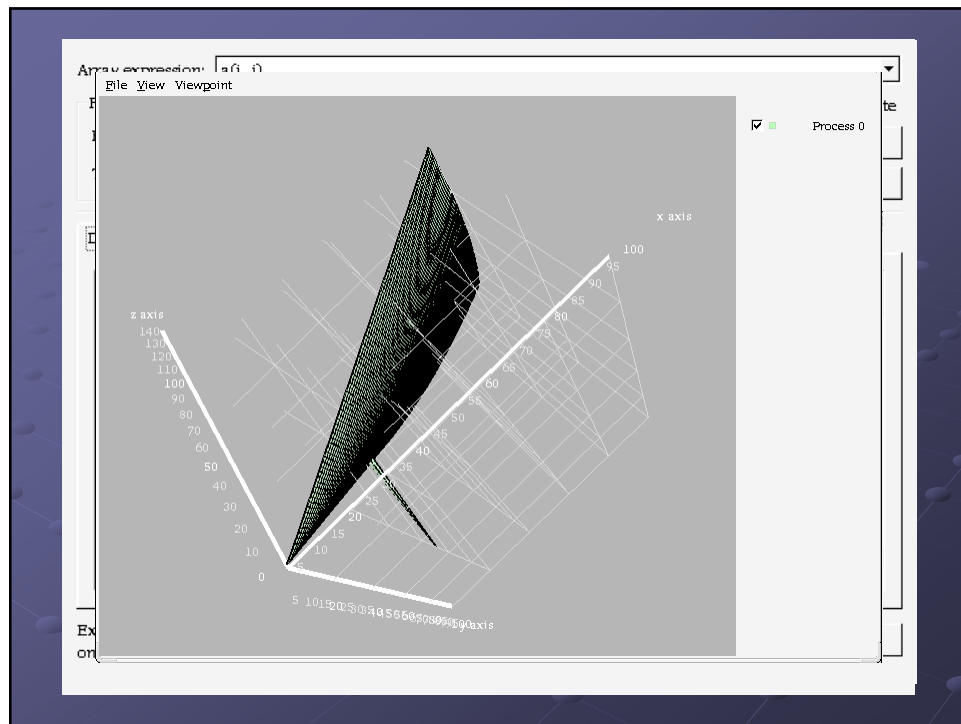
Type: none selected

Variables and data (2)

- Evaluate window
 - Can be used to view values for arbitrary expressions and global variables
- Support for Fortran modules
 - Fortran Modules tab in the Project Navigator window
- Changing Data Values
 - Right-click and select “edit value”

Variables and data (3)

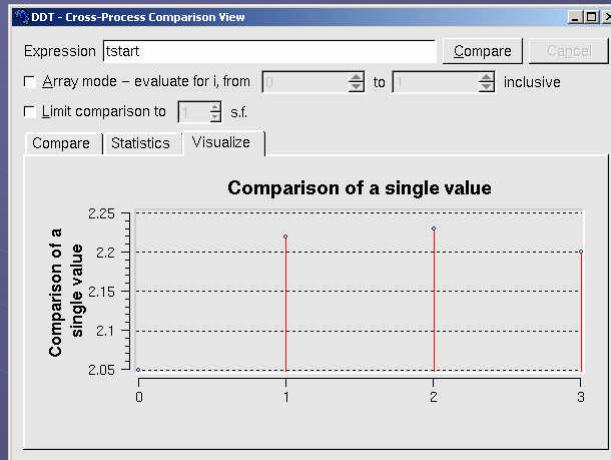
- Examining Pointers
 - Drag a pointer into the Evaluate window
 - Can be viewed as Vector, Reference, or Dereference (right-click to choose).
- Multi-Dimensional Arrays
 - Can be viewed in the Variable View
 - Multi-Dimensional Array viewer – visualization of a 2-D slice of an array using OpenGL



Variables and data (4)

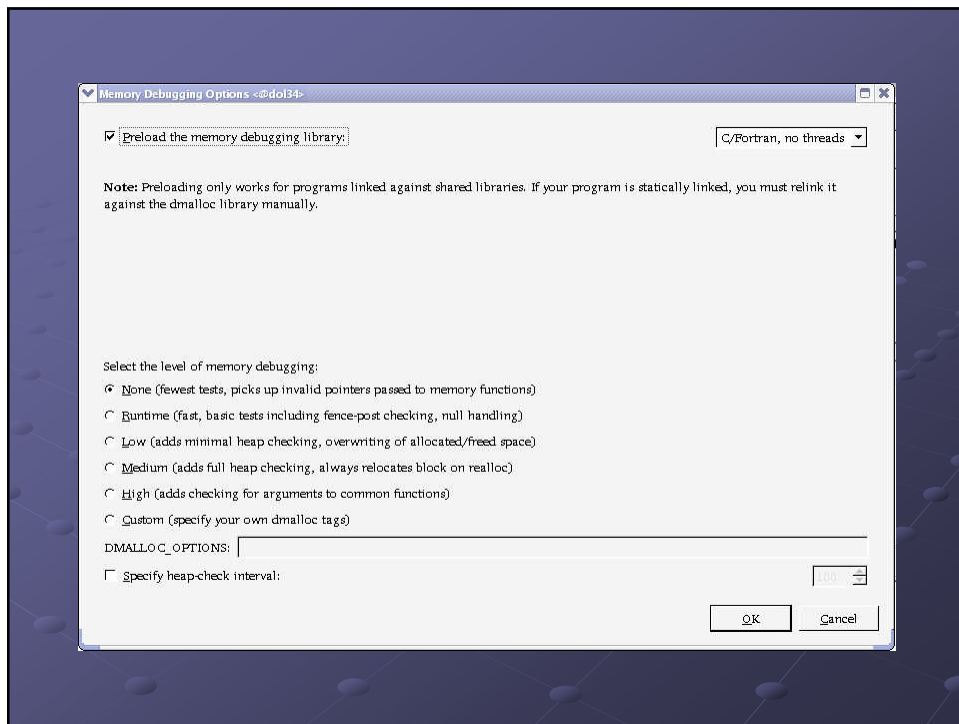
● Cross-Process Comparison

- Right-click on a variable name, or
- From View menu (type in any valid expression)
- Three modes – Compare, Statistics, and Visualize



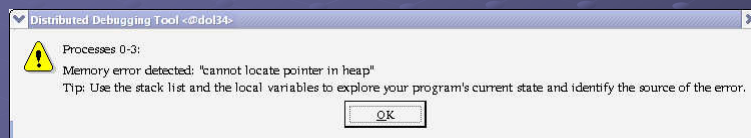
Memory Debugging

- Can intercept memory allocation and de-allocation calls and perform lots of complex heap- and bounds- checking.
- Off by default, can be turned on before starting a debugging session.
- Different levels of memory debugging – from minimal to high.



Memory Debugging (2)

- On error, stops with a message like



- Check Validity

- In Evaluate window, right-click to "Check pointer is valid"
- Useful for checking function arguments

Memory Debugging (3)

Detecting memory leaks

- 'View->Current Memory Usage' window
- Shows current memory usage across processes in the group
- Click on a color bar to get allocation details
- For more details, choose 'Table View'

Memory Usage for "All" group (18:07:51)

Graphical View Table View

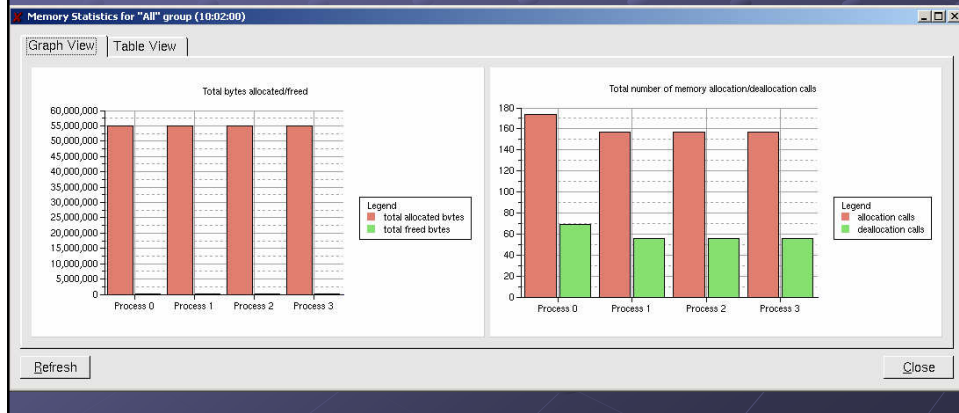
Process	Location	# Pointers	Size (bytes)
3	allocate.c:30(allocate_commbuffers)	1 ptr	52428800
2	allocate.c:30(allocate_commbuffers)	1 ptr	52428800
1	allocate.c:30(allocate_commbuffers)	1 ptr	52428800
0	allocate.c:30(allocate_commbuffers)	1 ptr	52428800
3	forcetree.c:1802(force_treeallocate)	1 ptr	1020136
2	forcetree.c:1802(force_treeallocate)	1 ptr	1020136
1	forcetree.c:1802(force_treeallocate)	1 ptr	1020136
0	forcetree.c:1802(force_treeallocate)	1 ptr	1020136
3	allocate.c:115(allocate_memory)	1 ptr	720000
2	allocate.c:115(allocate_memory)	1 ptr	720000
1	allocate.c:115(allocate_memory)	1 ptr	720000
0	allocate.c:115(allocate_memory)	1 ptr	720000
3	hmpmp_reqblkalloc	1 ptr	299040
2	hmpmp_reqblkalloc	1 ptr	299040
1	hmpmp_reqblkalloc	1 ptr	299040
0	hmpmp_reqblkalloc	1 ptr	299040
3	allocate.c:142(allocate_memory)	1 ptr	210000
2	allocate.c:142(allocate_memory)	1 ptr	210000
1	allocate.c:142(allocate_memory)	1 ptr	210000
0	allocate.c:142(allocate_memory)	1 ptr	210000
3	forcetree.c:2425(nqb_treeallocate)	1 ptr	80000

Refresh Close

Memory Debugging (4)

Memory Statistics

- Menu option 'View->Memory Statistics'



Part II: using DDT

Running DDT on SHARCNET

- The environment has to be set properly to run X-windows (graphical) applications on our clusters.
 - Microsoft Windows: X-win32 (commercial program), or cygwin (not easy to set up)
 - Linux / Unix: add “ForwardX11 yes” – in `/etc/ssh/ssh_config`, or `~/.ssh/config`
 - To test, ssh to a cluster and type `xterm`

Running DDT on SHARCNET (2)

- On SHARCNET, DDT is configured to use the test queue
 - Almost immediate allocation
 - One hour limit per debugging session
 - Other jobs may be suspended

Hands on

