

Debugging with GDB and DDT

Ramses van Zon
SciNet HPC Consortium
University of Toronto

June 28, 2012

Outline

- ▶ Debugging Basics
- ▶ Debugging with the command line: GDB
- ▶ Debugging with DDT

Debugging basics



Debugging basics

Help, my program doesn't work!

```
$ icc -O3 answer.c  
$ ./a.out  
Segmentation fault
```

a miracle occurs

My program works brilliantly!

```
$ icc -O3 answer.c  
$ ./a.out  
42
```

- ▶ Unfortunately, “miracles” are not yet supported by SciNet.

Debugging:

Methodical process of finding and fixing flaws in software

Common symptoms

Errors at compile time

- ▶ Syntax errors: easy to fix
- ▶ Library issues
- ▶ Cross-compiling
- ▶ Compiler warnings

Always switch this on, and fix or understand them!

But just because it compiles does not mean it is correct!

Runtime errors

- ▶ Floating point exceptions
- ▶ Segmentation fault
- ▶ Aborted
- ▶ Incorrect output (nans)

Common issues

Arithmetic	corner cases ($\text{sqrt}(-0.0)$), infinities
Memory access	Index out of range, uninitialized pointers.
Logic	Infinite loop
Misuse	wrong input, ignored error, no initialization
Syntax	wrong operators/arguments
Resource starvation	memory leak, quota overflow
Parallel	race conditions, deadlock

What is going on?

- ▶ Almost always, a condition you are sure is satisfied, is not.
- ▶ But your programs likely relies on many such assumptions.
- ▶ First order of business is finding out what goes wrong, and what assumption is not warranted.
- ▶ A debugger is a program to help detect errors in other programs.
- ▶ **You are the real debugger.**

Ways to debug


- ▶ Preemptive:
 - ▶ Turn on compiler warnings: fix or understand them!
 - ▶ Check your assumptions (e.g. use assert).
- ▶ Inspect the exit code and read the error messages!
- ▶ Add print statements ← **No way to debug!**

Ways to debug

- ▶ Command-line based, symbolic debuggers
 - ▶ GNU debugger: *gdb*
 - ▶ Intel debugger command-line: *idbc*
- ▶ Symbolic debuggers with Graphical User Interface
 - ▶ GNU data display debugger: *ddd*
 - ▶ Intel debugger: *idb*
 - ▶ IDEs: Eclipse, NetBeans (neither on SciNet), *emacs/gdb*
 - ▶ Allinea DDT: *ddt*
 - ▶ Rogue Wave TotalView (not available at SciNet)

What's wrong with using print statements?

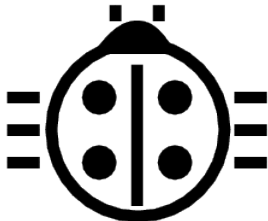
Strategy

- ▶ Constant cycle:
 1. strategically add print statements
 2. compile
 3. run
 4. analyze output
 - ▶ Removing the extra code after the bug is fixed
 - ▶ Repeat for each bug
- bug not found?* 

Problems with this approach

- ▶ Time consuming
- ▶ Error prone
- ▶ Changes memory, timing... **There's a better way!**

Symbolic debuggers



Symbolic debuggers

Features

1. Crash inspection
2. Function call stack
3. Step through code
4. Automated interruption
5. Variable checking and setting

Use a graphical debugger or not?

- ▶ Local work station: graphical is convenient
- ▶ Remotely (SciNet):
 - ▶ Some graphical debuggers slow (connection)
 - ▶ Command-line based debuggers fast (esp. gdb).
 - ▶ Ddt: gui-based, with graphics light enough to work remotely.
- ▶ Graphical and text-based debuggers use the same concepts

Symbolic debuggers

Preparing the executable

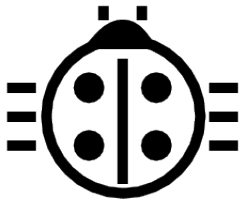
- ▶ Required: compile with `-g`.
- ▶ Optional: switch off optimization `-O0`
- ▶ Same for `gcc`, `g++`, `gfortran`, `icc`, `ifort`, `xf`, `mpif90`, `mpicc`, ...
- ▶ For `nvcc` (i.e. `cuda`), also add `-G`

Command-line based symbolic debuggers

- ▶ `gdb` ← **Focus on this one**
- ▶ `idbc` ← **Has gdb mode**

```
$ module load intel
$ icc -g -O0 example.c -o example
$ module load gdb
$ gdb example
...
(gdb)_
```

GDB



What is GDB?

- ▶ Free, GNU license, symbolic debugger.
- ▶ Available on many systems.
- ▶ Been around for a while, but still developed and up-to-date
- ▶ Text based, but has a '-tui' option.

GDB command summary

help	h	print description of command
run	r	run from the start (+args)
backtrace/where	ba	function call stack
break	b	set breakpoint
delete	d	delete breakpoint
continue	c	continue
step	s	step into function
next	n	continue until next line
print	p	print variable
quit	q	quit
finish	fin	continue until function end
set variable	set var	change variable
down	do	go to called function
tbreak	tb	set temporary breakpoint
until	unt	continue until line/function
up	up	go to caller
watch	wa	stop if variable changes
watch	wa	stop if variable changes
quit	q	quit gdb

GDB basic building blocks



GDB building blocks #1: Inspect crashes

Inspecting core files

Core = file containing state of program after a crash

- ▶ needs max core size set (`ulimit -c <number>`)
- ▶ gdb reads with `gdb <executable> <corefile>`
- ▶ it will show you where the program crashed

No core file?

- ▶ can start gdb as `gdb <executable>`
- ▶ type `run` to start program
- ▶ gdb will show you where the program crashed if it does.

GDB building blocks #2: Function call stack

Interrupting program

- ▶ Press Ctrl-C while program is running in gdb
- ▶ gdb will show you where the program was.

Stack trace

- ▶ From what functions was this line reached?
- ▶ What were the arguments of those function calls?

`gdb` commands

<code>backtrace</code>	function call stack
<code>continue</code>	continue
<code>down</code>	go to called function
<code>up</code>	go to caller

GDB building blocks #3: Step through code

Stepping through code

- ▶ Line-by-line
- ▶ Choose to step into or over functions
- ▶ Can show surrounding lines or use `-tui`

`gdb` commands

<code>list</code>	list part of code
<code>next</code>	continue until next line
<code>step</code>	step into function
<code>finish</code>	continue until function end
<code>until</code>	continue until line/function

GDB building blocks #4: Automatic interruption

Breakpoints

- ▶ `break [file:]<line>|<function>`
- ▶ each breakpoint gets a number
- ▶ when run, automatically stops there
- ▶ can add conditions, temporarily remote breaks, etc.

Related gdb commands

<code>delete</code>	<code>unset breakpoint</code>
<code>condition</code>	<code>break if condition met</code>
<code>disable</code>	<code>disable breakpoint</code>
<code>enable</code>	<code>enable breakpoint</code>
<code>info breakpoints</code>	<code>list breakpoints</code>
<code>tbreak</code>	<code>temporary breakpoint</code>

GDB building blocks #5: Variables

Checking a variable

- ▶ Can print the value of a variable
- ▶ Can keep track of variable (print at prompt)
- ▶ Can stop the program when variable changes
- ▶ Can change a variable (“what if ...”)

`gdb` commands

<code>print</code>	<code>print variable</code>
<code>display</code>	<code>print at every prompt</code>
<code>set variable</code>	<code>change variable</code>
<code>watch</code>	<code>stop if variable changes</code>

Graphical symbolic debuggers



Graphical symbolic debuggers

Features

- ▶ Nice, more intuitive graphical user interface
- ▶ Front to command-line based tools: Same concepts
- ▶ Need graphics support (qsub -X -I ...)

Available on SciNet

- ▶ ddd
\$ module load gcc ddd
\$ ddd <executable compiled with -g flag>
- ▶ idb
\$ module load intel java *Java slow remotely*
\$ idb <executable compiled with -g flag>
- ▶ ddt
\$ module load ddt
(more later)

Graphical symbolic debuggers - ddd

The screenshot displays the DDD graphical debugger interface. The main window shows a C program with OpenMP parallelism:

```
float f=0.0;
int i, th;
#pragma omp parallel for default(none) private(i,th) shared(f)
for (i = 0; i<100; i++) {
    double g;
    th = omp_get_thread_num();
    printf("%d\n",th);
    g = sqrt(0.25*i+th);
    f += g;
}

printf("result = %f\n", f);
```

A red stop sign icon is visible on the left side of the code editor. The top status bar shows the current thread: "3: th" with a value of 2. The bottom status bar shows: "Display 3: th (enabled, scope main.omp_fn.0, address 0x41401074)".

A "Threads" window is open, listing the threads:

```
Threads
4 Thread 0x41e02940 () at add.c:17
3 Thread 0x41401940 () at add.c:17
2 Thread 0x40a00940 () at add.c:17
1 Thread 0x2aaaab8d3d20 () at add.c:17
```

The control panel on the right includes buttons for Run, Interrupt, Step, Next, Until, Cont, Up, Undo, Edit, Step!, Next!, Finish, Kill, Down, Redo, and Make.

The bottom console shows the following GDB output:

```
Breakpoint 1, main.omp_fn.0 (.omp_data_i=0x7fffffff9f0)
(gdb) c
Continuing.
[Switching to Thread 0x40a00940 (LWP 25170)]

Breakpoint 1, main.omp_fn.0 (.omp_data_i=0x7fffffff9f0) at add.c:17
(gdb) graph display i
(gdb) graph display th
(gdb) c
Continuing.
2
0
1
[Switching to Thread 0x41401940 (LWP 25171)]

Breakpoint 1, main.omp_fn.0 (.omp_data_i=0x7fffffff9f0) at add.c:17
(gdb) |
```

Graphical symbolic debuggers - idb

The screenshot shows the Intel(R) Debugger interface. The main window displays the source code of a C program named `add.c`. The code is as follows:

```
9 float f=0.0;
10 int i, th;
11 #pragma omp parallel for default(none) private(i,th) shared(f)
12 for (i = 0; i<100; i++) {
13     double g;
14     th = omp_get_thread_num();
15     printf("%d\n",th);
16     g = sqrt(0.25*i+th);
17     f += g;
18 }
19
20 printf("result = %f\n", f);
```

The line `f += g;` is highlighted in blue. Below the code editor is a **Threads** window showing a list of threads:

ID	Type	OS ID	Thread Library ID	Execution Attribute	Location
▼ \$allthreads					
1		nati 2606	469387906204	thawed	void main.omp_fn.0(void) "/home/rzon/C
2	↔	nati 2608	1084623168	thawed	void main.omp_fn.0(void) "/home/rzon/C
3		nati 2608	1113561408	thawed	void main.omp_fn.0(void) "/home/rzon/C
4		nati 2608	1124051264	thawed	<opaque> __write_nocancel(void)
\$currentopenmpteam					

The status bar at the bottom indicates the current thread is `0x000000000400887` in `main.omp_fn.0` at `"/home/rzon/Courses/snugdebug/ex2/add.c":17`.

Graphical symbolic debuggers - ddt

Alinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

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

All 0 1 2 3

Create Group

Project Files

Search (Ctrl+F)

```
95 p.runtime = ini.get_double("runtime", 1.0e5);
96 p.dt = ini.get_double("dt", 0.2);
97 p.dc = ini.get_double("dc", 2.0);
98 p.l[0] = ini.get_double("lx", 10);
99 p.l[1] = ini.get_double("ly", 10);
100 p.l[2] = ini.get_double("lz", 10);
101 p.n[0] = ini.get_long("nx", 10);
102 p.n[1] = ini.get_long("ny", 10);
103 p.n[2] = ini.get_long("nz", 10);
104
105 cout << "l = "
106 << p.l[0] << " ";
107 << p.l[1] << " ";
108 << p.l[2] << " \n";
109 << "n = "
110 << p.n[0] << " ";
111 << p.n[1] << " ";
112 << p.n[2] << " \n";
113
114 // points per processor
115 double ppp = (p.n[0]*p.n[1]*p.n[2])/size;
116 n.dim[0] = n.dim[1] = n.dim[2] = 1;
```

Locals

Variable Name	Value
DdtOverDx2	
-argc	2
-argv	0x7fffff6c
-comm	
-coords	
-dfield	0x17
-dims	
-field	0x7ffff6e2
-fullNn	
-ini	
-lastt	14073729
-negProc	
-negSlabin	
-negSlabOut	
-npoints	14073735
-nthrds	2
-oldprogress	-1342464
-origin	
-p	
-periods	

Input/Output* Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output

Stacks

Processes	Threads	Function
1	1	+ _kmp_launch_monitor
1	1	+ _kmp_launch_worker
1	1	+bb_openib_async_thread
1	1	main (diff3d.cc:105)
1	1	+service_thread_start

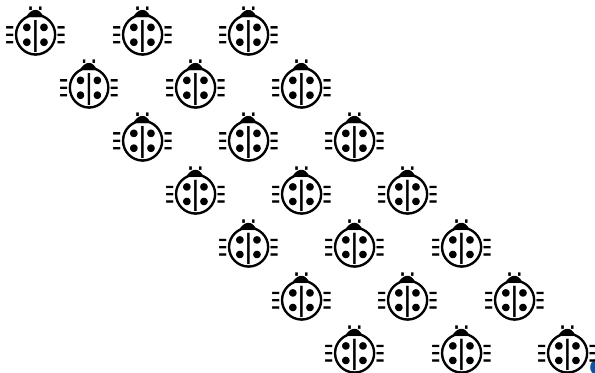
Evaluate

Expression	Value
	<No symbol '*' in current context.>

3 processes playing

compute + calcul CANADA

Parallel debugging



Parallel debugging

- ▶ Challenge: Simultaneous execution
- ▶ Shared memory:
 - OpenMP* (Open Multi-Processing)
 - pthread*s (POSIX threads)
 - ▶ Private/shared variables
 - Intel compiler extra flag: `-debug parallel`
 - ▶ Race conditions
- ▶ Distributed memory:
 - MPI* (Message Passing Interface)
 - ▶ Communication
 - ▶ Deadlock
- ▶ Hard to solve: some commercial debuggers do a good job.
But let's see how the command-line ones handle it.

Parallel debugging - 1 Shared memory

Use gdb for

- ▶ Track each thread's execution and variables
- ▶ OpenMP serialization: `p omp_set_num_threads(1)`
- ▶ Step into OpenMP block: `break` at first line!
- ▶ Thread-specific breakpoint: `b <line> thread <n>`

Use helgrind for

- ▶ Finding race conditions:

```
$ module load valgrind
$ valgrind --tool=helgrind <exe> &> out
$ grep <source> out
```

where <source> is the name of the source file where you suspect race conditions (valgrind reports a lot more)

Parallel debugging - 2 Distributed memory

Multiple MPI processes

- ▶ Your code is running on different cores!
- ▶ Where to run debugger?
- ▶ Where to send debugger output?
- ▶ Much going on at same time.
- ▶ No universal free solution.

Good approach

1. Write your code so it can run in serial: perfect that first.
2. Deal with communication, synchronization and deadlock on *smaller* number of MPI processes/threads.
3. Only then try full size.

Parallel debugging - 2 Distributed memory

Advanced gdb (not recommended!)

- ▶ You want `#proc` terminals with gdb for each process?
- ▶ Possible, but brace yourself!
- ▶ Small number of procs:
 1. Start terminals: by default X forwarding from compute nodes
 2. Submit your job on scinet
 3. Make sure its runs: `checkjob -v`
 4. From each terminal, ssh into the appropriate nodes
 5. Do `top` or `ps -C <exe>` to find process id (pid)
 6. Attach debugger with `gdb -pid <pid>`.
 7. This will interrupt the process.

Parallel debugging - 2 Distributed memory

Advanced tricks

Wait, so the program started already?

- ▶ Yes, and that's probably not what you want.
- ▶ Instead, put infinite loop into your code:

```
int j=1;  
while(j) sleep(5);
```
- ▶ Once attached, go “up” until at while loop.
- ▶ do “set var j=0”
- ▶ now you can step, continue, etc.

Now let's take a look at DDT...

DDT

The screenshot shows the Alinea DDT v3.1 debugger interface. The title bar reads "Alinea DDT v3.1 (on gpc-f102n)". The menu bar includes "Session", "Control", "Search", "View", and "Help". Below the menu is a toolbar with various execution and navigation icons. The status bar shows "Current Group: All" and "Focus on current: Group", with checkboxes for "Process" and "Thread", and a "Step Threads Together" option. A group bar displays "All" and four colored buttons (0, 1, 2, 3). The "Project Files" pane on the left shows a tree view with "Source Files" expanded to "diff3d.cc". The main editor window displays the following C++ code:

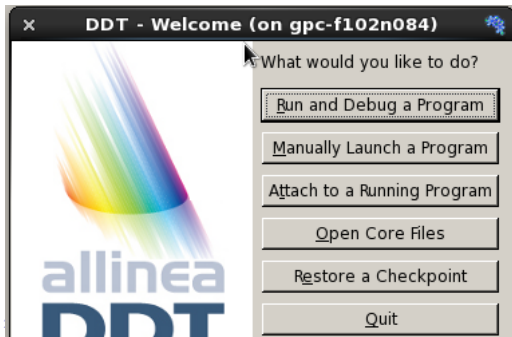
```
95     p.runtime = ini.get_double("runtime", 1.0e5);
96     p.dt      = ini.get_double("dt", 0.2);
97     p.dc      = ini.get_double("dc", 2.0);
98     p.l[0]    = ini.get_double("lx", 10);
99     p.l[1]    = ini.get_double("ly", 10);
100    p.l[2]    = ini.get_double("lz", 10);
101    p.n[0]    = ini.get_long ("nx", 10);
102    p.n[1]    = ini.get_long ("ny", 10);
103    p.n[2]    = ini.get_long ("nz", 10);
104
105    cout << "l = "
106         << p.l[0] << " "
107         << p.l[1] << " "
108         << p.l[2] << " \n"
109         << "n = "
```



- ▶ “Distributed Debugging Tool”
- ▶ Powerful GUI-based commercial debugger by *Allinea*.
- ▶ Supports C, C++ and Fortran
- ▶ Supports MPI, OpenMP, threads, CUDA and more
- ▶ Available on all SciNet clusters (GPC, TCS, ARC, P7)
- ▶ Available on SHARCNET’s kraken, requin, orca and monk.

Launching ddt

- ▶ Load your compiler and MPI modules.
- ▶ Load the ddt module: `$ module load ddt`
- ▶ Start ddt with one of these:
 - `$ ddt`
 - `$ ddt <executable compiled with -g flag>`
 - `$ ddt <executable compiled with -g flag>`
 - `<arguments>`
- ▶ First time: create config file: OpenMPI (skip other steps)
- ▶ Then gui for setting up debug session.



Run and Debug a Program (session setup)

The image shows the DDT (Data Display Tool) interface for running a program. The main window is titled "DDT - Run (on gpc-f102n084)". It contains several sections for configuring the execution environment:

- Application:** /home/s/scinet/rzon/Code/diff3d/diff3d
- Arguments:** (empty)
- Input File:** (empty)
- Working Directory:** (empty)
- MPI:** 2 processes, OpenMPI. Number of processes: 2. Implementation: OpenMPI, no queue. mpirun arguments: (empty).
- OpenMP:** 4 threads. Number of OpenMP threads: 4.
- CUDA:** (disabled).
- Memory Debugging:** Minimal, No guard pages, Backtraces, Preload.
- Environment Variables:** none.
- Plugins:** none.

The "Memory Debugging Options" sub-dialog is open, showing the following settings:

- Preload the memory debugging library:** Language: C++, threads
- Note:** Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually.
- Heap Debugging:**
 - Minimal (fewest tests, picks up invalid pointers passed to memory functions)
 - Runtime (fast, basic tests including fence-post checking, null handling)
 - Low (adds minimal heap checking, overwriting of allocated/freed space)
 - Medium (adds full heap checking, always relocates block on realloc)
 - High (adds checking for arguments to common functions)
 - Custom: (empty)
- Heap Overflow/Underflow Detection:**
 - Add guard pages to detect out of bounds heap access
 - Guard pages: 1 Add guard pages: After
- Advanced:**
 - Specify heap-check interval: 100
 - Store stack backtraces for memory allocations
 - Only enable for these processes: 0-1 100% Select All x2 x0.5 1%

Buttons for "Run", "Cancel", "OK", and "Cancel" are visible at the bottom of the dialog boxes.

User interface (1)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) user interface. The top menu bar includes Session, Control, Search, View, and Help. Below the menu is a toolbar with various icons for session management. The main area is divided into several panels:

- Session Control:** Shows 'Current Group: All' and 'Focus on current: Group Process Thread'. A 'Step Threads Together' checkbox is present. Below this, a tree view shows 'All' (0, 1, 2, 3), 'Root' (0), and 'Workers' (1, 2, 3).
- Project Files:** A search bar (Ctrl+K) and a file list including del_opv.cc, del_opvnt.cc, delete.c, diff3d.cc (selected), distances.c, and divtf3.c.
- Source Code:** The diff3d.cc file is open, showing C code. Line 81 is highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`.
- Locals:** A table showing the current line's local variables:

Variable Name	Value
MPI::COMM_...	
rank	32767
- Stacks:** A table showing the current stack frames:

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup
- Input/Output*, Breakpoints, Watchpoints, Stacks, Tracepoints, Tracepoint Output:** A row of tabs for different debugging features.
- Evaluate:** A table for evaluating expressions:

Expression	Value
------------	-------

The status bar at the bottom right shows 'Ready'.

User interface (2)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) user interface. A blue callout box at the top center contains the text "DDT uses a tabbed-document interface." with arrows pointing to the "diff3d.cc" tab in the Project Files window and the code editor window.

Session Control Search View Help

Current Group: A

All: 0 1 2 3

Root: 0

Workers: 1 2 3

Create Group

Project Files: diff3d.cc

Search (Ctrl+K)

Code Editor (diff3d.cc):

```
74 }
75 // MPI::COMM_WORLD.Abort(1);
76 }
77
78 const int nthreads = get_num_threads();
79 const int root = 0;
80 const int size = MPI::COMM_WORLD.Get_size();
81 int rank = MPI::COMM_WORLD.Get_rank();
82
83 cerr << "nthreads=" << nthreads << endl;
84
85 // #include "mpidebug.ch"
86
87 mpiCommit<Parameters>();
88
```

Locals: Current Line(s) Current Stack

Current Line(s)

Variable Name	Value
MPI::COMM_...	
rank	32767

Type: none selected

Input/Output* Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Evaluate

Stacks

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

User interface (3)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. A callout box with a blue border and white background contains the text: "When the session begins, DDT automatically finds source code from information compiled in the executable." An arrow points from this box to line 81 of the source code in the diff3d.cc file. The code editor shows the following code:

```
74     }
75     // MPI::COMM_WORLD.Abort(1);
76 }
77
78 const int nthreads = get_num_threads();
79 const int root = 0;
80 const int size = MPI::COMM_WORLD.Get_size();
81 int rank = MPI::COMM_WORLD.Get_rank();
82
83 cerr << "nthreads=" << nthreads << endl;
84
85 //include "mpidebug.ch"
86
87 mpiCommit<Parameters>();
88
```

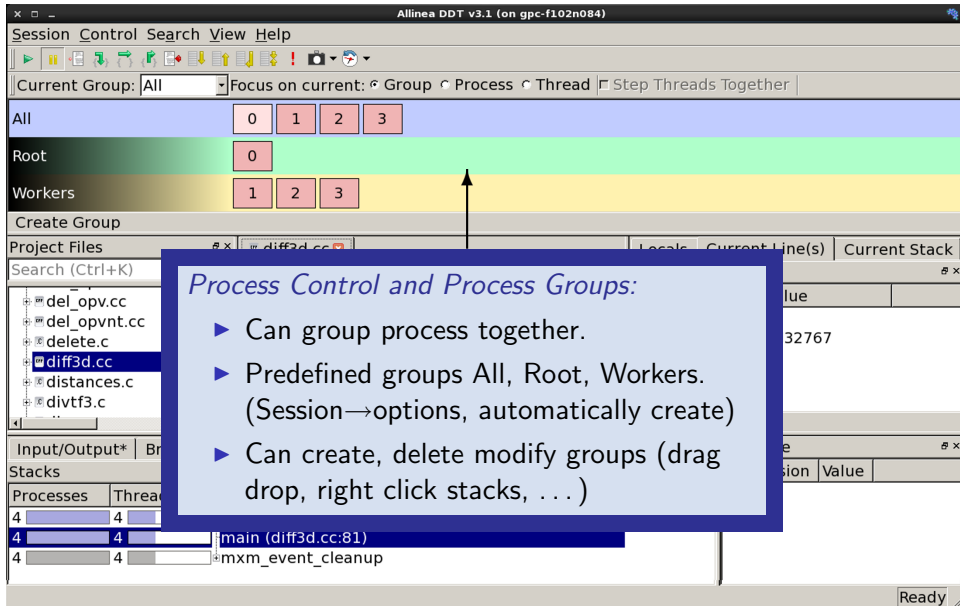
The stack window at the bottom shows the following information:

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

The Locals window shows the current line(s) and the value of the rank variable:

Variable Name	Value
MPI::COMM_...	
rank	32767

User interface (4)



The screenshot shows the Allinea DDT v3.1 (on gpc-f102n084) interface. The top menu bar includes Session, Control, Search, View, and Help. Below the menu is a toolbar with various icons. The main area displays process groups: 'All' (blue bar with sub-groups 0, 1, 2, 3), 'Root' (green bar with sub-group 0), and 'Workers' (yellow bar with sub-groups 1, 2, 3). An arrow points from the 'Workers' group to a callout box. The callout box, titled 'Process Control and Process Groups:', contains three bullet points: 'Can group process together.', 'Predefined groups All, Root, Workers. (Session→options, automatically create)', and 'Can create, delete modify groups (drag drop, right click stacks, ...)'. The bottom of the interface shows a file browser with 'diff3d.cc' selected, and a stack view with 'main (diff3d.cc:81)' and 'mxm_event_cleanup' visible.

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

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

All 0 1 2 3

Root 0

Workers 1 2 3

Create Group

Project Files diff3d.cc

Search (Ctrl+K)

del_opv.cc
del_opvnt.cc
delete.c
diff3d.cc
distances.c
divtf3.c

Input/Output* Br

Stacks

Processes Thread

4 4
4 4 main (diff3d.cc:81)
4 4 mxm_event_cleanup

Ready

compute • calcul
CANADA

User interface (5)

The screenshot shows the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, there is a menu bar with 'Session Control Search View Help' and a toolbar with various icons. Below the toolbar, the 'Current Group' is set to 'All'. A callout box with a blue border and white background contains the text: 'Different colour coding for each group's current source line.' An arrow points from this box to line 81 in the code editor. The code editor shows the following code:

```
74     }
75     // MPI::COMM_WORLD.Abort(1);
76 }
77
78 const int nthrds = get_num_threads();
79 const int root = 0;
80 const int size = MPI::COMM_WORLD.Get_size();
81 int rank = MPI::COMM_WORLD.Get_rank();
82
83 cerr << "nthrds=" << nthrds << endl;
84
85 //include "mpidebug.ch"
86
87 mpiCommit<Parameters>();
88
```

The code editor also has a 'Locals' panel on the right showing the current line(s) and the current stack. The 'Current Line(s)' panel shows the variable 'rank' with a value of 32767. The 'Current Stack' panel shows the function 'main (diff3d.cc:81)'. Below the code editor, there are several panels: 'Input/Output*', 'Breakpoints', 'Watchpoints', 'Stacks', 'Tracepoints', 'Tracepoint Output', and 'Evaluate'. The 'Stacks' panel is expanded, showing the following stack frames:

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

User interface (6)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, there is a menu bar with 'Session', 'Control', 'Search', 'View', and 'Help'. Below the menu is a toolbar with various icons. A dropdown menu for 'Current Group' is set to 'All', and 'Focus on current' is set to 'Group'. There are four buttons labeled '0', '1', '2', and '3'. A blue callout box with the text 'Session Control Dialog: Control program execution, e.g., play/continue, pause, step into, step over, step out' is overlaid on the interface. The main area shows a project tree on the left with files like 'del_opv.cc', 'diff3d.cc', and 'distances.c'. The central pane shows C++ code with line numbers 77-88. The right pane shows a variable 'rank' with a value of 32767. At the bottom, there are tabs for 'Input/Output*', 'Breakpoints', 'Watchpoints', 'Stacks', 'Tracepoints', and 'Tracepoint Output'. The 'Stacks' tab is active, showing a table with columns for 'Processes', 'Threads', and 'Function'. The stack entries are: 'gomp_thread_start (team.c:120)', 'main (diff3d.cc:81)', and 'mxm_event_cleanup'.

Session Control Dialog:
Control program execution, e.g., play/continue, pause, step into, step over, step out

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

User interface (7)

The screenshot displays the Allinea DDT v3.1 (on gpc-f102n084) interface. At the top, there is a menu bar with 'Session', 'Control', 'Search', 'View', and 'Help'. Below the menu is a toolbar with various icons. A 'Current Group' dropdown is set to 'All', and 'Focus on current' options for 'Group', 'Process', and 'Thread' are visible. A 'Step Threads Together' checkbox is also present. The main area shows a tree view of groups: 'All' (with sub-items 0, 1, 2, 3), 'Root' (with sub-item 0), and 'Workers' (with sub-items 1, 2, 3). Below this is a 'Project Files' section showing 'diff3d.cc'. A 'Search (Ctrl+K)' field is also visible. A blue callout box with the text 'Breakpoints Tab' and 'Can suspend, jump to, delete, load, save' points to the 'Breakpoints' tab in the bottom panel. The bottom panel also shows 'Input/Output*', 'Watchpoints', 'Stacks', 'Tracepoints', and 'Tracepoint Output' tabs. The 'Stacks' tab is active, showing a table with columns for 'Processes', 'Threads', and 'Function'. The table contains three entries: 'gomp_thread_start (team.c:120)', 'main (diff3d.cc:81)', and 'mxm_event_cleanup'. The 'main' entry is highlighted in blue. The bottom right corner shows a 'Ready' status and the 'compute + calcul CANADA' logo.

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

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

All 0 1 2 3

Root 0

Workers 1 2 3

Create Group

Project Files diff3d.cc Locals Current Line(s) Current Stack

Search (Ctrl+K)

Breakpoints Tab
Can suspend, jump to, delete, load, save

del_opv.cc
del_opvnt.cc
delete.c
diff3d.cc
distances.c
divtf3.c

85 // #include "mpidebug.ch"
86
87 mpiCommit<Parameters>();
88

Type: none selected

Input/Output* Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Evaluate

Stacks Expression Value

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

Ready

compute + calcul CANADA

User interface (8)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, the 'Focus on current' control is highlighted with a red box, showing radio buttons for 'Group', 'Process', and 'Thread', with 'Group' selected. Below this, a row of four colored buttons (0, 1, 2, 3) is visible, with an arrow pointing to button '2'. A blue callout box with a white border contains the text: *Focus:* Choose between Group, process or thread. The interface also shows a file explorer on the left with 'diff3d.c' selected, a code editor in the center showing MPI-related code, and a variable viewer on the right showing 'rank' with a value of 32767. At the bottom, the 'Stacks' panel shows the current stack with 'main (diff3d.cc:81)' highlighted.

Session Control Search View Help

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

All 0 1 2 3

Root 0

Workers

Create Group

Project Files

Search (Ctrl+K)

del_opv.cc
del_opvnt.cc
delete.c
diff3d.c
distances.c
divtf3.c

```
75 // MPI::COMM_WORLD.Abort(1);  
76 }  
77  
78 const int nthreads = get_num_threads();  
79 const int root = 0;  
80 const int size = MPI::COMM_WORLD.Get_size();  
81 int rank = MPI::COMM_WORLD.Get_rank();  
82  
83 cerr << "nthreads=" << nthreads << endl;  
84  
85 // #include "mpidebug.ch"  
86  
87 mpiCommit<Parameters>();  
88
```

Current Line(s)

Variable Name	Value
MPI::COMM_...	
rank	32767

Type: none selected

Input/Output* Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Evaluate

Stacks

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

Ready

User interface (9)

The screenshot shows the Alinea DDT v3.1 interface. A callout box with a blue border contains the following text:

Stacks: Current and Parallel

- ▶ Tree of functions (merged)
- ▶ Click on branch to see source
- ▶ Hover to see process ranks
- ▶ Use to gather processes in new groups

The interface includes a menu bar (Session Control, Search, View, Help), a toolbar, and a main workspace. The workspace is divided into several panels:

- File Explorer:** Shows a tree of files including `delete.c`, `diff3d.cc` (selected), `distances.c`, and `divtf3.c`.
- Code Editor:** Displays C++ code with line numbers 82-88. The code includes `cerr`, `nthrds`, `mpidebug.ch`, and `mpiCommit<Parameters>()`.
- Locals Panel:** Shows `MPI::COMM_...` and `rank` with a value of `32767`.
- Stacks Panel:** A table showing the current stack of function calls.
- Input/Output, Breakpoints, Watchpoints, Tracepoints, Tracepoint Output, Evaluate:** A row of tabs for debugging features.

Processes	Threads	Function
4	4	<code>gomp_thread_start (team.c:120)</code>
4	4	<code>main (diff3d.cc:81)</code>
4	4	<code>mxm_event_cleanup</code>

User interface (10)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. The 'Current Group' is set to 'All'. The 'Current line variables' window is open, showing the following data:

Group	0	1	2	3
All				
Root	0			
Workers	1	2	3	

The 'Current Line(s)' window is also open, showing the current line of code and its variables:

Variable Name	Value
MPI::COMM_...	
rank	32767

The 'Current Line(s)' window also shows the current line of code:

```
81 int rank = MPI::COMM_WORLD.Get_rank();
```

The 'Stacks' window is open, showing the current stack of processes and threads:

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

User interface (11)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, there is a menu bar with 'Session Control Search View Help' and a toolbar with execution controls. Below this is a 'Current Group: All' section. A blue box highlights the 'Local variables for process' area, which is a table with columns for process groups and their ranks. The groups are 'All', 'Root', and 'Workers'. The 'All' group has ranks 0, 1, 2, 3. The 'Root' group has rank 0. The 'Workers' group has ranks 1, 2, 3. Below this is a 'Create Group' section. The main area is divided into 'Project Files' and 'diff3d.cc'. The 'Project Files' list includes del_opv.cc, del_opvnt.cc, delete.c, diff3d.cc (selected), distances.c, and divtf3.c. The 'diff3d.cc' file is open, showing C code. Line 81 is highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`. To the right of the code editor is a 'Locals' panel showing the current line(s) and a table of local variables. The table has columns for 'Variable Name' and 'Value'. The variable 'rank' is shown with a value of 32767. Below the code editor is a 'Stacks' panel with tabs for 'Input/Output*', 'Breakpoints', 'Watchpoints', 'Stacks', 'Tracepoints', and 'Tracepoint Output'. The 'Stacks' panel shows a table with columns for 'Processes', 'Threads', and 'Function'. The stack includes 'gomp_thread_start (team.c:120)', 'main (diff3d.cc:81)' (selected), and 'mxm_event_cleanup'. At the bottom right, there is a 'Ready' status bar and a 'compute + calcul CANADA' logo.

Local variables for process				
All	0	1	2	3
Root	0			
Workers	1	2	3	

```
74     }
75     // MPI::COMM_WORLD.Abort(1);
76   }
77
78   const int nthreads = get_num_threads();
79   const int root = 0;
80   const int size = MPI::COMM_WORLD.Get_size();
81   int rank = MPI::COMM_WORLD.Get_rank();
82
83   cerr << "nthreads=" << nthreads << endl;
84
85   //include "mpidebug.ch"
86
87   mpiCommit<Parameters>();
88
```

Variable Name	Value
MPI::COMM_...	
rank	32767

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

User interface (12)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) user interface. The top menu bar includes Session, Control, Search, View, and Help. Below the menu is a toolbar with various icons. The main area is divided into several panels:

- Session Control:** Shows 'Current Group: All' and a list of groups: All (0, 1, 2, 3), Root (0), and Workers (1, 2, 3).
- Project Files:** A tree view on the left shows files like del_opv.cc, del_opvnt.cc, delete.c, diff3d.cc (selected), distances.c, and divtf3.c. The main editor window shows the code for diff3d.cc, with line 81 highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`.
- Locals:** A table showing the current line(s) and variable values. The variable 'rank' is shown with a value of 32767.
- Stacks:** A table showing the current stack frames. The top frame is 'main (diff3d.cc:81)', which is highlighted in blue. Other frames include 'gomp_thread_start (team.c:120)' and 'mxm_event_cleanup'.
- Evaluate Window:** A blue box labeled 'Evaluate window' is overlaid on the top right, with an arrow pointing to the 'Evaluate' button in the bottom right corner.

Variable Name	Value
MPI::COMM_...	
rank	32767

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup

Other features of DDT (1)

- ▶ Some of the user-modified parameters and windows are saved by right-clicking and selecting a save option in the corresponding window (Groups; Evaluations)
- ▶ DDT can load and save sessions.
- ▶ *Find* and *Find in Files* in the Search menu.
- ▶ *Goto line* in Search menu (or Ctrl-G)
- ▶ Synchronize processes in group: Right-click, “Run to here”.
- ▶ View multiple source codes simultaneously: Right-click, “Split”
- ▶ Right-click power!

Other features of DDT (2)

- ▶ Signal handling: SEGV, FPE, PIPE,ILL
- ▶ Support for Fortran modules
- ▶ Change data values in evaluate window
- ▶ Examine pointers (vector, reference, dereference)
- ▶ Multi-dimensional arrays
- ▶ Viewer

Other features of DDT (3)

Message Queue

- ▶ View → show message queue
- ▶ produces both a graphical view and table for active communications
- ▶ Helps to find e.g. deadlocks

The screenshot shows the 'DDT - Message Queues' window. On the left, a communication graph displays three nodes (0, 1, 2) arranged in a triangle. Solid red arrows indicate active communication paths between nodes, while dashed red lines represent other potential connections. On the right, there are three control panels:

- Select queues to show:** Includes checkboxes for Send, Receive, and Unexpected.
- Ranks:** Includes radio buttons for Show local ranks and Show global ranks.
- Select communicator:** A list box containing `MPI_COMM_WORLD` (highlighted), `MPI_COMM_SELF`, and `MPI_COMM_NULL`.

Other features of DDT (4)

Memory debugging

- ▶ Select “memory debug” in Run window
- ▶ Stops on error (before crash or corruption)
- ▶ Check pointer (right click in evaluate)
- ▶ View, overall memory stats

Useful references

- ▶ G Wilson
Software Carpentry software-carpentry.org/3_0/debugging.html
- ▶ N Matloff and PJ Salzman
The Art of Debugging with GDB, DDD and Eclipse
- ▶ *GDB*: sources.redhat.com/gdb
- ▶ *DDT*: www.allinea.com/products/ddt-support
- ▶ *SciNet Wiki*: wiki.scinethpc.ca: Tutorials & Manuals