

# Debugging with GDB and DDT

Ramses van Zon  
SciNet HPC Consortium  
University of Toronto

June 13, 2014



# Outline

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

# Debugging basics

# Debugging basics

Help, my program doesn't work!

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

a miracle occurs

My program works brilliantly!

```
$ gcc -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 ( <code>sqrt(-0.0)</code> ), infinities
Memory access	Index out of range, uninitialized pointers.
Logic	Infinite loop, corner cases
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.
- ▶ *Debugger*: program to help detect errors in other programs.
- ▶ **You are the real debugger.**

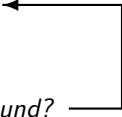
# Ways to debug

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



# What's wrong with using print statements?

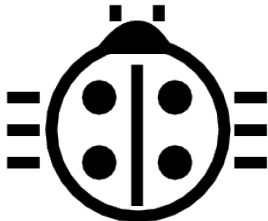
## Strategy

- ▶ 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): can be slow

In any case, graphical and text-based debuggers use the same concepts.

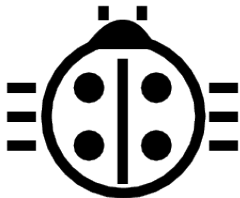
# Symbolic debuggers

## Preparing the executable

- ▶ Add required compilation flags:
  - \$ gcc/g++/gfortran -g [-gstabs]
  - \$ icc/icpc/ifort -g [-debug parallel]
  - \$ nvcc -g -G
- ▶ Optional: switch off optimization -O0

Command-line based symbolic debuggers: 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.

```
$ module load gcc/4.8.1
$ gcc -g -O0 example.c -o example
$ module load gdb/7.6
$ gdb -tui example
...
(gdb)_
```

## GDB basic building blocks



# GDB building block #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 block #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 block #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

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

# GDB building block #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

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

# GDB building block #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

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

## Demonstration GDB

```
$ ssh USER@login.scinet.utoronto.ca -X
$ ssh gpc01 -X
$ qsub -l nodes=1:ppn=8,walltime=4:00:00 -I -X
$ cd $SCRATCH
$ cp -r /scinet/course/ss2014 .
$ cd ss2014/HPC245_debug/code
$ source setup
$ cd ex1
$ make dbgtest #(or dbgtestf)
$ ulimit -c 1024
$ ./dbgtest #(or dbgtestf)
Hello
Hi
You'll find that the latter does not work. Start up
$ gdb -tui dbgtest #(or dbgtestf)
```

## Graphical symbolic debuggers



# Graphical symbolic debuggers

## Features

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

## Available on SciNet: ddd and ddt

- ▶ ddd

```
$ module load gcc ddd
```

```
$ ddd <executable compiled with -g flag>
```

- ▶ ddt

```
$ module load ddt
```

```
$ ddt <executable compiled with -g flag>
```

```
(more later)
```

# Graphical symbolic debuggers - ddd

The screenshot displays the DDD graphical debugger interface. The main window shows a C program with OpenMP parallelism. A breakpoint is set at the start of the parallel region. The program is running, and a thread list window is open, showing four threads. 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.

File Edit View Program Commands Status Source Data Help

0: th

1: f 6.50046921 2: i 51 3: th 2

```
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);
```

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) |

Display 3: th (enabled, scope main.omp\_fn.0, address 0x41401074)

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

Close Help

Run Interrupt Step Step! Next Next! Until Finish Cont Kill Up Down Undo Redo Edit Make



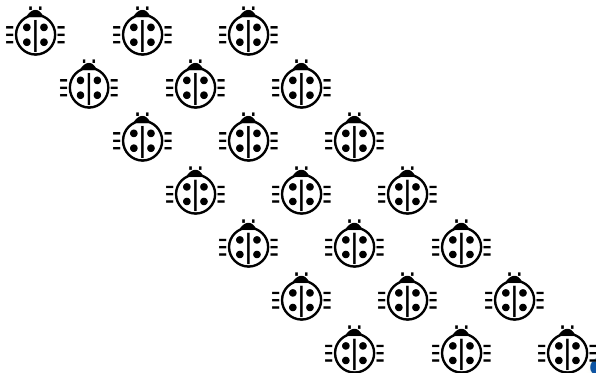
# Graphical symbolic debuggers - ddt

The screenshot shows the Alinea DDT v3.1 (on gpc-f102n084) graphical symbolic debugger interface. The main window displays the source code of `diff3d.cc` at line 105, where a `cout` statement is being executed. The code defines a 3x3 matrix `p` and calculates its determinant `ppp`.

```
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;
```

The interface includes a Project Files pane on the left, a Locals pane on the right showing variables like `argc`, `argv`, and `comm`, and a Stacks pane at the bottom left showing the current thread `main (diff3d.cc:105)`. The Evaluate pane at the bottom right shows the expression `<No symbol '*' in current context.>`.

## Parallel debugging



# Parallel debugging - 1 Shared memory

## Use gdb for

- ▶ Tracking each thread's execution and variables
- ▶ OpenMP serialization: `p omp_set_num_threads(1)`
- ▶ Stepping 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 demands specialized tools: ddt

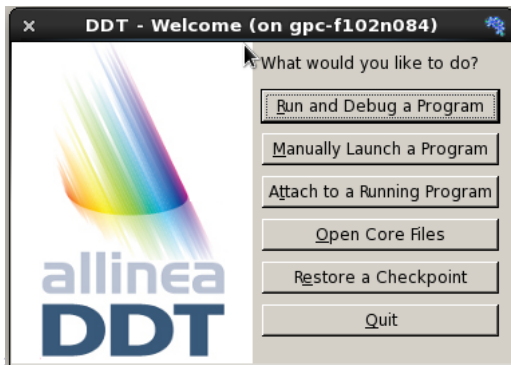
DDT



- ▶ “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:** (unchecked).
- 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)

**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: Search (Ctrl+K)

- del\_opv.cc
- del\_opvnt.cc
- delete.c
- diff3d.cc**
- distances.c
- divtf3.c

```
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	<b>main (diff3d.cc:81)</b>
4	4	mxm_event_cleanup

# User interface (2)

The screenshot displays the Alinea DDT v3.1 interface. A blue callout box at the top states: "DDT uses a tabbed-document interface." Three arrows point from this box to the "diff3d.cc" tab, the code editor window, and the "Stacks" panel.

**Session Control** Search View Help

Current Group: All

All	0	1	2	3
Root	0			
Workers	1	2	3	

Create Group

Project Files

- del\_opv.cc
- del\_opvnt.cc
- delete.c
- diff3d.cc**
- distances.c
- divtf3.c

```
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
```

Locals Current Line(s) Current Stack

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	<b>main (diff3d.cc:81)</b>
4	4	mxm_event_cleanup

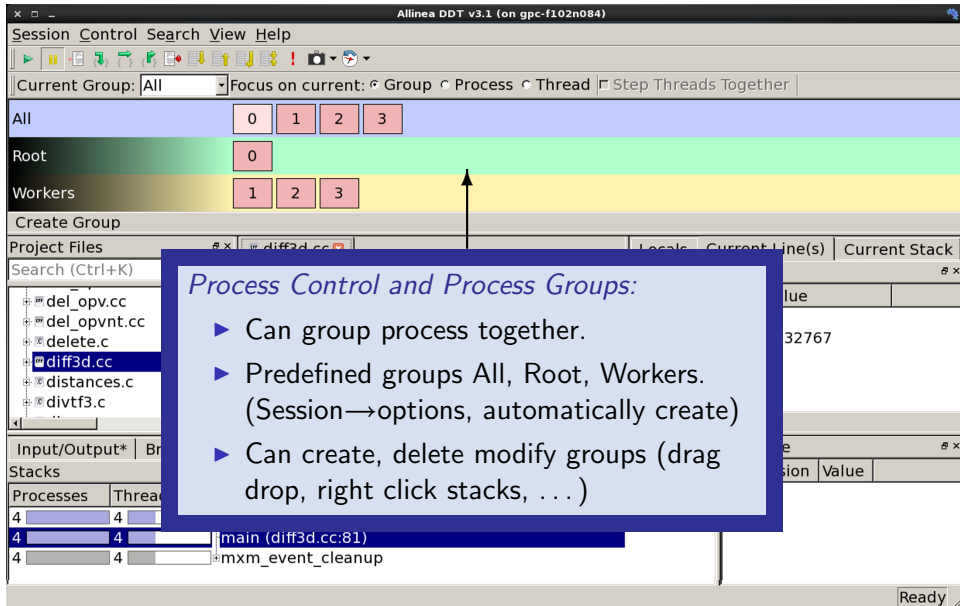
# User interface (3)

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 the source code in the 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  
89
```

The interface also shows a "Project Files" pane on the left with a tree view containing files like del\_opv.cc, del\_opvnt.cc, delete.c, diff3d.cc, distances.c, and divtf3.c. The "diff3d.cc" file is selected. The "Locals" pane on the right shows the current line(s) and the variable "rank" with a value of 32767. The "Stacks" pane at the bottom shows the current stack with three entries: gomp\_thread\_start (team.c:120), main (diff3d.cc:81), and mxm\_event\_cleanup.

# User interface (4)



The screenshot shows 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 bar is a toolbar with various icons. A status bar at the top indicates 'Current Group: All' and 'Focus on current: Group Process Thread Step Threads Together'. The main area displays a tree view of 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 'Project Files' list with 'diff3d.cc' selected, and a 'Stacks' table with columns for 'Processes' and 'Thread'. The 'Stacks' table shows three entries: 'main (diff3d.cc:81)' and 'mxm\_event\_cleanup'.

**Process Control and Process Groups:**

- ▶ Can group process together.
- ▶ Predefined groups All, Root, Workers. (Session→options, automatically create)
- ▶ Can create, delete modify groups (drag drop, right click stacks, ... )

# User interface (5)

The screenshot displays the Alinea DDT v3.1 IDE interface. At the top, the title bar reads "Alinea DDT v3.1 (on gpc-f102n084)". The menu bar includes "Session", "Control", "Search", "View", and "Help". Below the menu bar is a toolbar with various icons. The main workspace is divided into several panes:

- Current Group:** A dropdown menu showing "All", "Root", and "Workers".
- Project Files:** A tree view on the left showing files like "del\_opv.cc", "diff3d.cc", and "distances.c".
- Source Code:** The central pane shows the code for "diff3d.cc". Line 81, `int rank = MPI::COMM_WORLD.Get_rank();`, is highlighted in blue. A blue callout box with white text points to this line, stating: "Different colour coding for each group's current source line." Below the code, the "Locals" pane shows the variable `MPI::COMM_...` with a value of `rank` and `32767`.
- Stacks:** The bottom pane shows the call stack. The top entry is `gomp_thread_start (team.c:120)`, followed by `main (diff3d.cc:81)` (highlighted in blue), and `mxm_event_cleanup`.

At the bottom right, there is a "Ready" status indicator and a "compute + calcul CANADA" logo.

# 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 declaration 'MPI::COMM\_...' and its value 'rank' as '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 'Processes', 'Threads', and 'Function'. The stack contains three entries: '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 status bar indicates 'Current Group: All' and 'Focus on current: Group Process Thread Step Threads Together'. The main area shows a tree view of the project files, including 'del\_opv.cc', 'del\_opvnt.cc', 'delete.c', 'diff3d.cc', 'distances.c', and 'divtf3.c'. A blue callout box highlights the 'Breakpoints Tab' and states 'Can suspend, jump to, delete, load, save'. Below the tree view, there is a code editor showing lines 85, 86, and 87 of 'diff3d.cc'. The bottom of the interface features a 'Stacks' panel with columns for 'Processes', 'Threads', and 'Function'. The 'Function' column shows 'gomp\_thread\_start (team.c:120)', 'main (diff3d.cc:81)', and 'mxm\_event\_cleanup'. The 'main (diff3d.cc:81)' entry is highlighted in blue. The bottom right corner shows the '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>();

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 shows the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, there is a menu bar with 'Session Control Search View Help'. Below it is a toolbar with various icons. A dropdown menu is open, showing 'Current Group: All' and 'Focus on current: Group Process Thread Step Threads Together'. Below the dropdown are four buttons labeled 0, 1, 2, and 3. An arrow points from the callout box to the '2' button. The callout box contains the text: *Focus:* Choose between Group, process or thread. The main window displays a code editor with C code, a 'Current Line(s)' panel showing 'rank' with value 32767, and a 'Stacks' panel showing the current stack frame 'main (diff3d.cc:81)'. The status bar at the bottom right shows 'Ready'.

*Focus:*  
Choose between Group, process or thread

```
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 (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`, `distances.c`, and `divtf3.c`.
- Code Editor:** Displays C++ code with line numbers 82-88. The code includes `cerr` output, an `#include` for `"mpidebug.ch"`, and a call to `mpiCommit<Parameters>()`.
- Locals Panel:** Shows the current line(s) and the current stack. The variable `rank` is shown with a value of `32767`.
- Stacks Panel:** A table showing the current stack of function calls. The `main` function is highlighted.

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. At the top, there is a menu bar with 'Session', 'Control', 'Search', 'View', and 'Help'. Below the menu is a toolbar with various icons. The 'Current Group' dropdown is set to 'All'. A blue box highlights the 'Current line variables' section, which shows a tree view of MPI-related variables: 'All' (with sub-items 0, 1, 2, 3), 'Root' (with sub-item 0), and 'Workers' (with sub-items 1, 2, 3). An arrow points from this box to the 'Current Line(s)' panel on the right. The 'Current Line(s)' panel shows the current line of code (81) and its variables: 'rank' with a value of 32767. The code editor in the center shows the source code for 'diff3d.cc', with line 81 highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`. The 'Stacks' panel at the bottom shows the current stack frame: 'main (diff3d.cc:81)'. The status bar at the bottom right indicates 'Ready'.

Current line variables

Current Line(s)

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

Type: none selected

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 (11)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, the 'Session Control Search View Help' menu is visible. Below it, the 'Current Group: All' is shown. A blue box highlights the 'Local variables for process' section, which contains a grid of process group buttons: 'All' (0, 1, 2, 3), 'Root' (0), and 'Workers' (1, 2, 3). An arrow points from this box to the 'Locals' panel on the right. The 'Project Files' panel on the left shows a tree view with 'diff3d.cc' selected. The main editor window displays the source code for 'diff3d.cc', with line 81 highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`. The 'Locals' panel shows the variable 'rank' with a value of 32767. The 'Stacks' panel at the bottom shows the current stack frame: 'main (diff3d.cc:81)'. The status bar at the bottom right indicates 'Ready'.

Local variables for process

```
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) interface. At the top, there is a menu bar with 'Session', 'Control', 'Search', 'View', and 'Help'. Below the menu bar is a toolbar with various icons. A blue box labeled 'Evaluate window' is positioned over the top toolbar and the 'Current Group' dropdown menu. The 'Current Group' dropdown is set to 'All'. Below this, there are three rows of colored bars representing different group types: 'All' (blue), 'Root' (green), and 'Workers' (yellow). Each row contains several red buttons with numbers: 'All' has buttons for 0, 1, 2, 3; 'Root' has a button for 0; 'Workers' has buttons for 1, 2, 3. Below these is a 'Create Group' button. The main area is divided into several panes. On the left is a 'Project Files' pane showing a tree view of files, with 'diff3d.cc' selected. The central pane shows the source code for 'diff3d.cc', with line 81 highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`. On the right is a 'Locals' pane showing the current line(s) and the current stack. The 'Current Line(s)' pane shows 'rank' with a value of 32767. The 'Current Stack' pane shows the current stack frame: 'gomp\_thread\_start (team.c:120)'. Below the main panes is a 'Stacks' pane showing the current stack of processes and threads. The 'Stacks' pane has columns for 'Processes', 'Threads', and 'Function'. The current stack is as follows:

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

At the bottom of the interface, there is a 'Ready' button and a 'compute + calcul CANADA' logo. An arrow points from the 'Evaluate window' label to the 'Evaluate' button in the bottom right corner.

# Demonstration DDT

```
$ cd $SCRATCH/ss2014/HPC245_debug/code
```

```
$ source setup
```

```
$ cd ex2
```

```
$ make
```

```
$ ddt ex2
```

## 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 the nodes, while dashed red lines represent potential or inactive paths. 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`.



# Demonstration DDT

```
$ cd $SCRATCH/ss2014/HPC245_debug/code
```

```
$ source setup
```

```
$ cd ex3
```

```
$ make
```

```
$ ddt ex3
```

# 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

# Demonstration DDT

```
$ cd $SCRATCH/ss2014/HPC245_debug/code
```

```
$ source setup
```

```
$ cd ex4
```

```
$ make
```

```
$ ddt ex4
```

## Other features of DDT (5)

### Thread debugging

```
$ cd $SCRATCH/ss2014/HPC245_debug/code  
$ source setup  
$ cd ex5  
$ make  
$ ddt ex5
```

## Useful references

- ▶ G Wilson  
*Software Carpentry* [software-carpentry.org/3\\_0/debugging.html](http://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](http://sources.redhat.com/gdb)
- ▶ *DDT*: [www.allinea.com/products/ddt-support](http://www.allinea.com/products/ddt-support)
- ▶ *SciNet Wiki*: [wiki.scinethpc.ca](http://wiki.scinethpc.ca): Tutorials & Manuals