

Debugging with GDB and DDT

Ramses van Zon
SciNet HPC Consortium
University of Toronto

May 10, 2013



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?

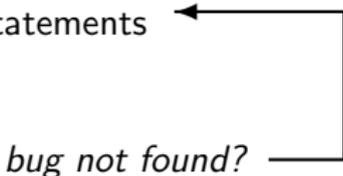
- ▶ Almost always, a condition you are sure is satisfied, is not.
- ▶ But your programs likely relies on many such assumptions.
- ▶ First order of business is finding out what goes wrong, and what assumption is not warranted.
- ▶ *Debugger*: program to help detect errors in other programs.
- ▶ **You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

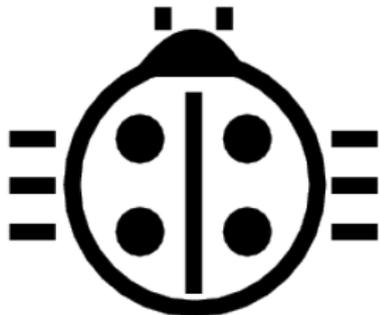
Strategy

- ▶ 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.

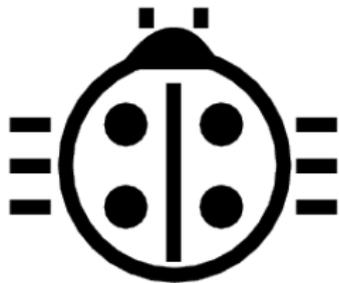
Symbolic debuggers

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
$ gcc -g -O0 example.c -o example
$ module load gdb
$ 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

list	list part of code
next	continue until next line
step	step into function
finish	continue until function end
until	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 remove breaks, etc.

Related gdb commands

delete	unset breakpoint
condition	break if condition met
disable	disable breakpoint
enable	enable breakpoint
info breakpoints	list breakpoints
tbreak	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

<code>print</code>	print variable
<code>display</code>	print at every prompt
<code>set variable</code>	change variable
<code>watch</code>	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/ss2013 .
$ cd ss2013/HPC106_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 red stop sign icon is visible on the left side of the code editor. A small window titled "Threads" is open, showing a list of threads. A control panel on the right side of the interface contains buttons for "Run", "Interrupt", "Step", "Next", "Until", "Cont", "Up", "Down", "Undo", "Edit", "Finish", "Kill", "Redo", and "Make".

```
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);
```

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

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)

Graphical symbolic debuggers - ddt

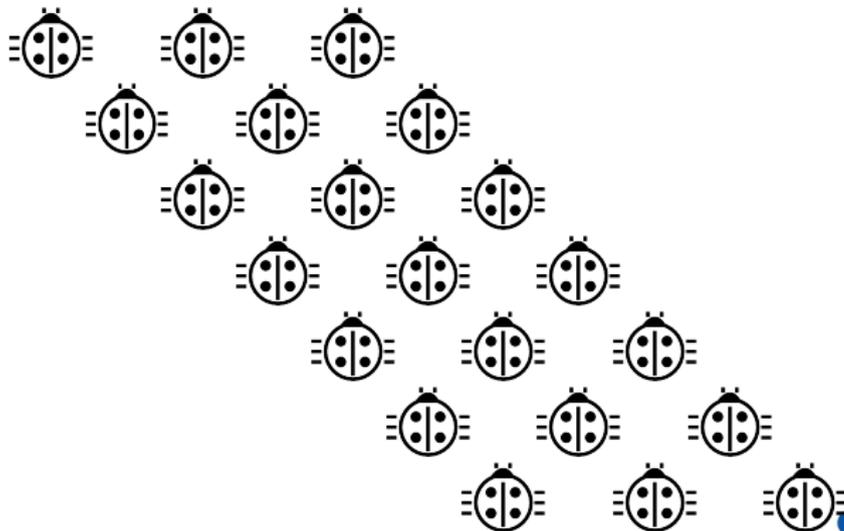
The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) graphical symbolic debugger. The interface is divided into several panels:

- Top Panel:** Contains session control buttons (run, stop, step, etc.) and a status bar showing "Current Group: All" and "Focus on current: Group".
- Project Files:** A tree view on the left showing the project structure, including source files like types.h, uio.h, and diff3d.cc.
- Source Code:** The main window shows the source code of diff3d.cc. Line 105 is highlighted, containing the statement `cout << "i" << endl;`. The code defines a double array `p` and prints its elements.
- Locals:** A table on the right showing the current stack frame's local variables. The table has columns for "Variable Name" and "Value".
- Stacks:** A table at the bottom left showing the stack of processes and threads. The current thread is highlighted.
- Evaluate:** A panel at the bottom right for evaluating expressions. The current expression is `<No symbol 'i' in current context.>`.

Variable Name	Value
argc	2
argv	0x7fffff9c
comm	
coords	
dfield	0x17
dims	
field	0x7ffff6e2
fullnn	
ini	
lastt	14073729
negProc	
negSlabin	
negSlabOut	
npoints	14073735
nthreads	2
oldprogress	-1342464
origin	
p	
periods	

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

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

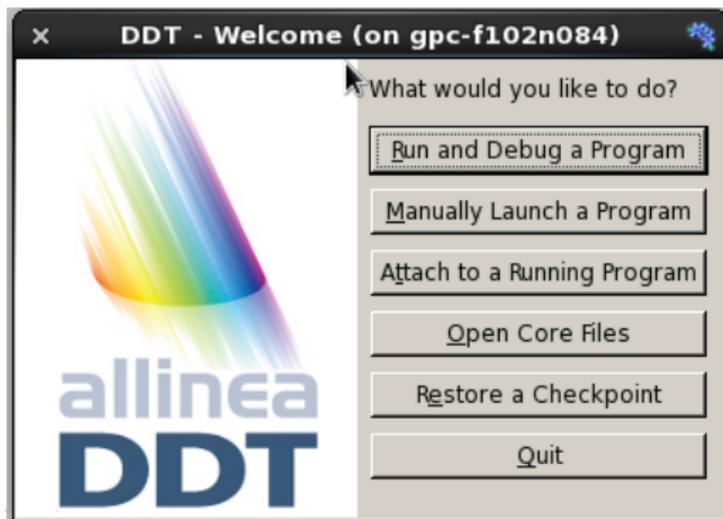
DDT



- ▶ “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 two overlapping dialog boxes from the DDT (Data Display Tool) interface. The background dialog is titled "DDT - Run (on gpc-f102n084)" and contains configuration options for running a program. The foreground dialog is titled "Memory Debugging Options (on gpc-f102n084)" and provides settings for memory debugging.

DDT - Run (on gpc-f102n084)

- Application: /home/s/scinet/rzon/Code/diff3d/diff3d
- 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**
- Memory Debugging:** Minimal, No guard pages, Backtraces, Preload
- Environment Variables: none
- Plugins: none

Memory Debugging Options (on gpc-f102n084)

- 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%]

User interface (1)

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
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	main (diff3d.cc:81)
4	4	mxm_event_cleanup

Ready

compute • calcul
CANADA

User interface (2)

Alinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

Current Group: All

All: 0 1 2 3

Root: 0

Workers: 1 2 3

Create Group

Project Files: 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

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

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

DDT uses a tabbed-document interface.

User interface (3)

The screenshot displays the Alinea DDT v3.1 (on gpc-f102n084) interface. A blue callout box with white text states: "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 editor.

The source code editor shows the following code in `diff3d.cc`:

```
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
```

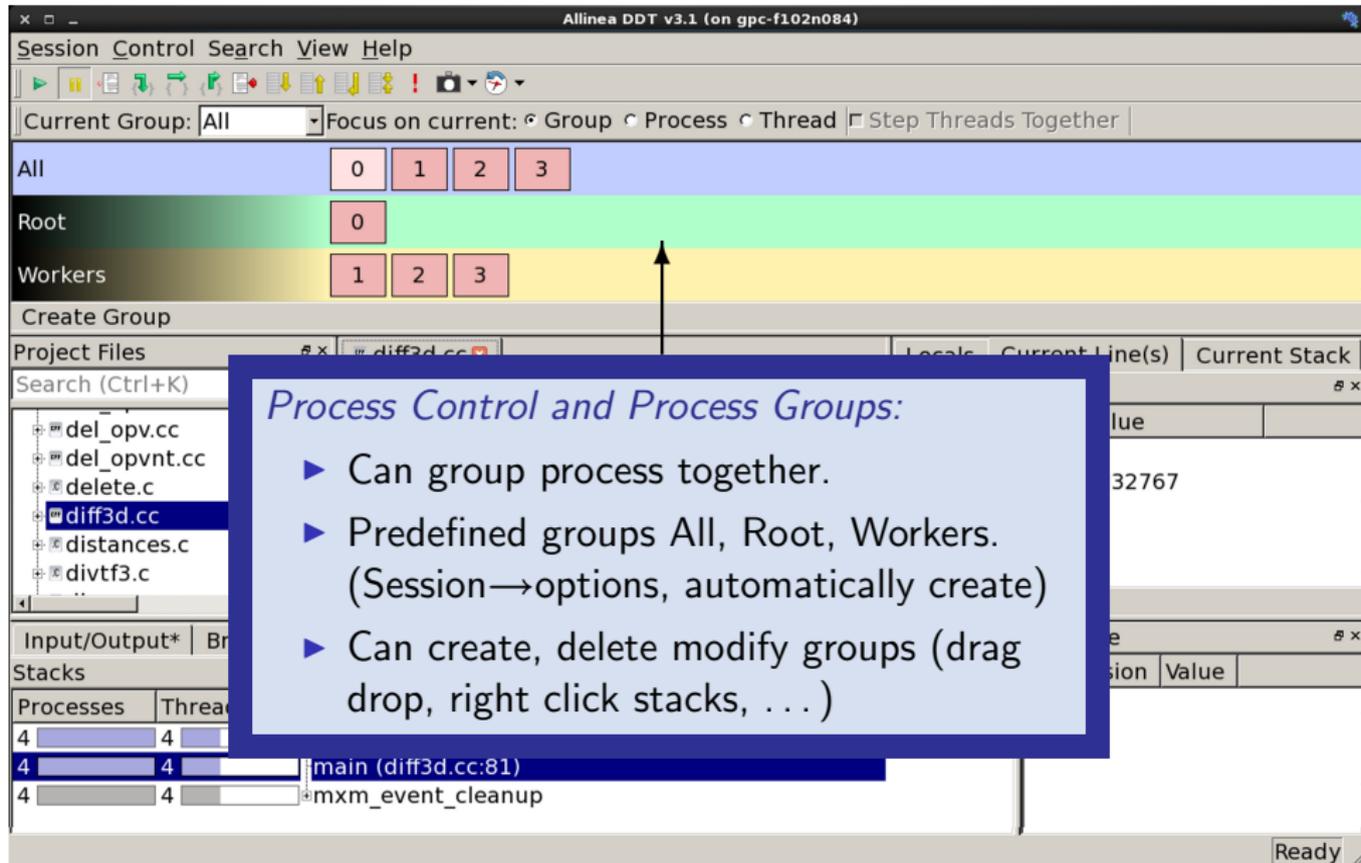
The variable watch window shows the value of the `rank` variable:

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

The Stacks window shows the current call stack:

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

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. Below the callout box, the 'Project Files' pane shows a tree view with files like del_opv.cc, del_opvnt.cc, delete.c, diff3d.cc (selected), distances.c, and divtf3.c. The 'Stacks' pane shows a list of processes and threads, with the top entry being 'main (diff3d.cc:81)'. The bottom status bar shows 'Ready'.

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 (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 text points to the code editor, stating: 'Different colour coding for each group's current source line.' The code editor shows the file 'diff3d.cc' with 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 lines are color-coded: line 81 is blue, line 82 is yellow, line 83 is green, and line 84 is red. The 'Project Files' pane on the left shows a tree view with 'diff3d.cc' 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 'main (diff3d.cc:81)' 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 (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 tree view on the left shows a hierarchy: 'All' (with sub-items 0, 1, 2, 3), 'Root' (with sub-item 0), and 'Workers' (with sub-items 1, 2, 3). The 'Project Files' pane shows 'diff3d.cc' selected. The 'Search (Ctrl+K)' pane is empty. The 'Breakpoints Tab' is highlighted with a blue box and contains the text: 'Breakpoints Tab' and 'Can suspend, jump to, delete, load, save'. Below this, a code editor shows lines 85-87 of 'diff3d.cc' with a breakpoint set at line 81. The 'Input/Output*' pane is empty. The 'Stacks' pane shows a table with columns 'Processes', 'Threads', and 'Function'. The 'Evaluate' pane is empty.

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

User interface (8)

The screenshot shows the Alinea DDT v3.1 (on gpc-f102n084) interface. At the top, the 'Focus on current' dropdown menu is open, showing options for 'Group', 'Process', and 'Thread'. A callout box with a blue border and the text 'Focus: Choose between Group, process or thread' points to this menu. Below the menu, a row of buttons labeled '0', '1', '2', and '3' is visible, with an arrow pointing to button '2'. The interface also shows a file explorer on the left, a code editor in the center displaying C code, and a variable viewer on the right showing 'rank' with a value of 32767. At the bottom, there are tabs for 'Input/Output*', 'Breakpoints', 'Watchpoints', 'Stacks', 'Tracepoints', and 'Tracepoint Output', with the 'Stacks' tab selected. The 'Stacks' panel shows a list of processes and threads, with the top entry 'gomp_thread_start (team.c:120)' 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.cc
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 window with several panels:

- Current Line(s):** Shows the current line of code in the source file.
- Current Stack:** A tree view of the current stack of functions.
- Locals:** A table showing local variables and their values.
- Stacks:** A table showing the stack of processes and threads.

The **Stacks** panel is currently selected and displays the following data:

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

The **Locals** panel shows the following data:

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

The **Current Line(s)** panel shows the following code:

```
82  
83 cerr << "nthreads=" << nthreads << endl;  
84  
85 // #include "mpidebug.ch"  
86  
87 mpiCommit<Parameters>();  
88
```

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. A blue box highlights the 'Current Group' dropdown, which is set to 'All'. A callout box labeled 'Current line variables' points to the 'rank' variable in the 'Current Line(s)' panel. The 'Current Line(s)' panel shows the variable 'rank' with a value of 32767. The code editor in the center shows the source code for 'diff3d.cc', with line 81 highlighted. The 'Stacks' panel at the bottom shows the current stack of processes and threads, with 'main (diff3d.cc:81)' selected. The 'Stacks' panel has columns for 'Processes', 'Threads', and 'Function'. The 'Stacks' panel also has a 'Stacks' tab selected, and the 'Evaluate' panel is visible on the right.

Current Group: All

Current line variables

All: 0, 1, 2, 3

Root: 0

Workers: 1, 2, 3

Create Group

Project Files: Search (Ctrl+K)

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

Ready

compute • calcul CANADA

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 icons. Below this is a 'Current Group: All' field. The main area is divided into several sections:

- Process Groups:** A grid showing process groups for 'All', 'Root', and 'Workers'. 'All' has 4 processes (0-3), 'Root' has 1 process (0), and 'Workers' has 3 processes (1-3).
- Project Files:** A list of files including 'del_opv.cc', 'del_opvnt.cc', 'delete.c', 'diff3d.cc' (selected), 'distances.c', and 'divtf3.c'.
- Code Editor:** Shows the source code for 'diff3d.cc'. Line 81 is highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`.
- Locals:** A table showing local variables for the current line. The variable 'rank' has a value of 32767.
- Stacks:** A table showing the current stack frames. The top frame is 'main (diff3d.cc:81)', which is highlighted.

A blue box labeled 'Local variables for process' points to the 'Locals' panel. The 'Stacks' panel shows the current stack frame 'main (diff3d.cc:81)' highlighted in blue.

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 window is divided into several panes:

- Session Control:** Shows the current group as 'All' and a list of groups: All (0, 1, 2, 3), Root (0), and Workers (1, 2, 3).
- Project Files:** A file explorer showing a directory structure with files like del_opv.cc, del_opvnt.cc, delete.c, diff3d.cc (selected), distances.c, and divtf3.c.
- Code Editor:** Displays the source code for diff3d.cc. Line 81 is highlighted: `int rank = MPI::COMM_WORLD.Get_rank();`.
- Locals:** A table showing the current state of local variables:

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

Processes	Threads	Function
4	4	gomp_thread_start (team.c:120)
4	4	main (diff3d.cc:81)
4	4	mxm_event_cleanup
- Evaluate Window:** A panel at the bottom right with tabs for Input/Output*, Breakpoints, Watchpoints, Stacks, Tracepoints, Tracepoint Output, and Evaluate. The Evaluate tab is active, showing a table with columns for Expression and Value.

A blue box labeled "Evaluate window" is positioned over the top part of the interface, with an arrow pointing to the Evaluate tab in the bottom right pane.

Demonstration DDT

```
$ cd $SCRATCH/ss2013/HPC106_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 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/ss2013/HPC106_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/ss2013/HPC106_debug/code
```

```
$ source setup
```

```
$ cd ex4
```

```
$ make
```

```
$ ddt ex4
```

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