Debugging

Ramses van Zon
SciNet HPC Consortium
University of Toronto

July 17, 2015

SciNet compute • calcul CANADA

Outline

- Debugging Basics
- Debugging with the command line: GDB
- Memory debugging with the command line: valgrind
- (Parallel) Debugging with DDT

SciNet compute • calcul CANADA

Debugging basics

SciNet compute • calcul CANADA

Debugging basics

Debugging basics

Help, my program doesn't work!

SciNet compute • calcul CANADA

Debugging basics

Help, my program doesn't work!

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

SciNet compute • calcul CANADA

Debugging basics

Help, my program doesn't work!

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

↓

a miracle occurs

↓

SciNet compute • calcul CANADA

Debugging basics

Help, my program doesn't work!

a miracle occurs

My program works brilliantly!

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

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
```


SciNet compute • calcul CANADA

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.

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

Common symptoms

Errors at compile time

SciNet compute • calcul CANADA

Common symptoms

Errors at compile time

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

SciNet
compute • calcul
CANADA

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!

SciNet compute • calcul CANADA

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!

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

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)

SciNet compute • calcul CANADA

Common issues

Arithmetic	corner cases (`sqrt(-0.0)`), 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

SciNet compute • calcul CANADA

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.**

SciNet compute • calcul CANADA

How to avoid debugging:

- Write better code.
    - Simpler, clear, straightforward code.
    - Modularity (no global variables or 10,000-line functions)
    - Avoid 'cute' tricks (no obfuscated C code winners)
- Don't write code, use existing libraries
- Write (simple) tests for each part of your code
- Use version control so you can 'roll back'.

SciNet compute • calcul CANADA

Debugging Workflow

First things first:

- As soon as you are convinced there is a real problem, create the simplest situation in which it reproducibly occurs.
- This is science: model, hypothesis, experiment, conclusion.
- Try a smaller problem size, turning off physical effects with options, etc. until you have a simple, fast repeatable example of the bug.
- Try to narrow it down to a particular module/function/class. For fortran, switch on bounds checking (**`-fbounds-check`**.)
- Now you're ready to start debugging.

SciNet compute • calcul CANADA

Ways to debug

Ways to debug

- Preemptive:
  - Turn on compiler warnings: fix or understand them!

    ```
    $ gcc/gfortran -Wall
    ```

  - Check your assumptions (e.g. use **`assert`**).

SciNet compute • calcul CANADA

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!

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

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!**

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

- Constant cycle:

What's wrong with using print statements?

Strategy

- Constant cycle:
  1. strategically add print statements

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

- Constant cycle:
  1. strategically add print statements
  2. compile

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

- Constant cycle:
  1. strategically add print statements
  2. compile
  3. run

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

- Constant cycle:
  1. strategically add print statements
  2. compile
  3. run
  4. analyze output

SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

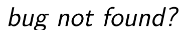
- Constant cycle:
  1. strategically add print statements
  2. compile
  3. run
  4. analyze output *bug not found?*

SciNet compute • calcul CANADA

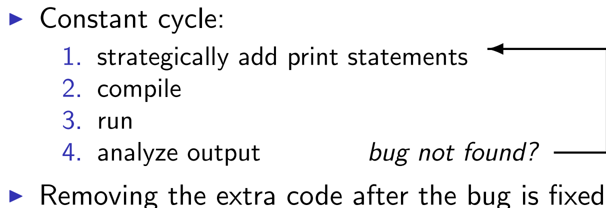
What's wrong with using print statements?

Strategy

- Constant cycle:
  1. strategically add print statements
  2. compile
  3. run
  4. analyze output



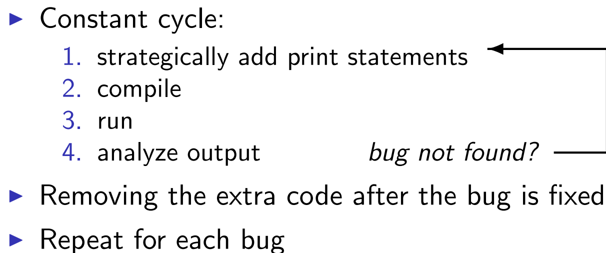
What's wrong with using print statements?



SciNet compute • calcul CANADA

What's wrong with using print statements?

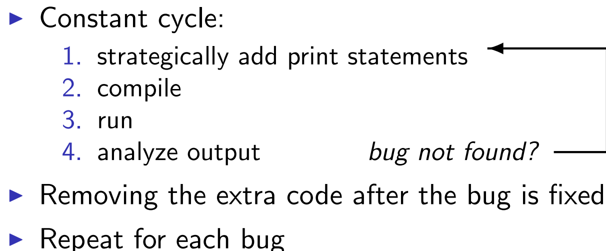
Strategy



SciNet compute • calcul CANADA

What's wrong with using print statements?

Strategy

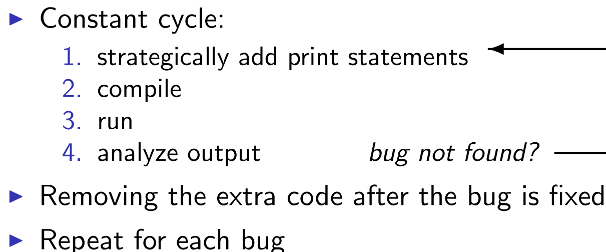


Problems with this approach

What's wrong with using print statements?

Strategy

- Constant cycle:



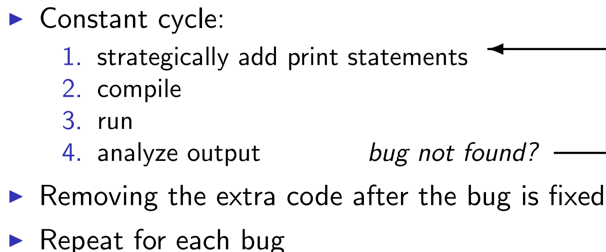
- Removing the extra code after the bug is fixed
- Repeat for each bug

Problems with this approach

- Time consuming
- Error prone
- Changes memory, timing...

What's wrong with using print statements?

Strategy

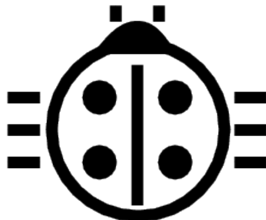


Problems with this approach

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

SciNet compute • calcul CANADA

Symbolic debuggers



Symbolic debuggers

Features

SciNet compute • calcul CANADA

Symbolic debuggers

Features

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

SciNet compute • calcul CANADA

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?

SciNet compute • calcul CANADA

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.

SciNet compute • calcul CANADA

Symbolic debuggers

Preparing the executable

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

Symbolic debuggers

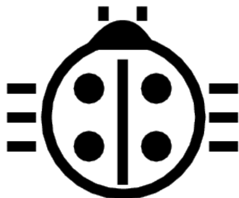
Preparing the executable

- Add required compilination 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

SciNet compute • calcul CANADA

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/4.7.2
$ gcc -Wall -g -O0 example.c -o example
$ module load gdb/7.6
$ gdb -tui example
...
(gdb)_
```

SciNet compute • calcul CANADA

GDB basic building blocks

Demonstration of GDB features

- We will look at the features of gdb using a running example.
- Example reads integers from command line and sums them.
- There's a C and a Fortran version.

```
$ ssh USER@login.scinet.utoronto.ca -X
$ ssh gpc01 -X
$ qsub -l nodes=1:ppn=8,walltime=8:00:00 -I -X -qteach
$ cp -r /scinet/course/ss2015/debug $SCRATCH
$ source $SCRATCH/debug/code/setup
$ cd $SCRATCH/debug/code/bugexample
$ make bugexample #(or make bugexample_f)
```

SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

Inspecting core files

SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

Inspecting core files

Core = file containing state of program after a crash

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

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?

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.

SciNet compute • calcul CANADA

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.

Related gdb commands

Command	Description
`run`	run the executable from the start
`list`	list code lines (where current execution is, or range)

SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

```
$ ulimit -c 1024
$ ./bugexample #(or ./bugexample_f)
Give some integers as command-line arguments
$ ./bugexample 1 3 5
Segmentation fault (core dumped)
```


SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

```
$ ulimit -c 1024
$ ./bugexample #(or ./bugexample_f)
Give some integers as command-line arguments
$ ./bugexample 1 3 5
Segmentation fault (core dumped)

$ gdb ./bugexample core.2387 # core number varies
```


SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

```
$ ulimit -c 1024
$ ./bugexample #(or ./bugexample_f)
Give some integers as command-line arguments
$ ./bugexample 1 3 5
Segmentation fault (core dumped)

$ gdb ./bugexample core.2387 # core number varies

GNU gdb (GDB) 7.6
Copyright (C) 2013 Free Software Foundation, Inc.
...
Reading symbols from debug/code/bugexample/bugexample...done.
[New LWP 3817]
warning: Can't read pathname for load map: Input/output error.
Core was generated by `./bugexample 1 3 5'.
Program terminated with signal 11, Segmentation fault.
#0  0x4007d5 in sum_integers (n=3, a=0x4) at intlisttools.c:30
30          s += a[i];
(gdb)
```

GDB building block #1: Inspect crashes

```
...
Program terminated with signal 11, Segmentation fault.
#0  0x4007d5 in sum_integers (n=3, a=0x4) at intlisttools.c:30
30          s += a[i];
```

This points at the line where the error is detected.

SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

```
...
Program terminated with signal 11, Segmentation fault.
#0  0x4007d5 in sum_integers (n=3, a=0x4) at intlisttools.c:30
30          s += a[i];
```

This points at the line where the error is detected. More context:

SciNet compute • calcul CANADA

GDB building block #1: Inspect crashes

```
...
Program terminated with signal 11, Segmentation fault.
#0  0x4007d5 in sum_integers (n=3, a=0x4) at intlisttools.c:30
30          s += a[i];
```

This points at the line where the error is detected. More context:

```
(gdb) list
```

GDB building block #1: Inspect crashes

```
...
Program terminated with signal 11, Segmentation fault.
#0  0x4007d5 in sum_integers (n=3, a=0x4) at intlisttools.c:30
30          s += a[i];
```

This points at the line where the error is detected. More context:

```
(gdb) list

25 /* Compute the sum of the array of integers */
26 int sum_integers(int n, int* a)
27 {
28     int i, s;
29     for (i=0; i<n; i++)
30         s += a[i];
31     return s;
32 }
(gdb)
```


SciNet compute • calcul CANADA

GDB building block #2: Function call stack

Interrupting program

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

SciNet compute • calcul CANADA

GDB building block #2: Function call stack

Interrupting program

- Press Crtl-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?

SciNet compute • calcul CANADA

GDB building block #2: Function call stack

Interrupting program

- Press Crtl-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

backtrace	function call stack
continue	continue
down	go to called function
up	go to caller

SciNet compute • calcul CANADA

GDB building block #2: Function call stack

```
...
(gdb) list
25 /* Compute the sum of the array of integers */
26 int sum_integers(int n, int* a)
27 {
28     int i, s;
29     for (i=0; i<n; i++)
30         s += a[i];
31     return s;
32 }
(gdb)  backtrace
#0 0x4007d5 in sum_integers (n=3,a=0x4) at intlisttools.c:30
#1 0x40082a in process (argc=4,argv=0x7fff0b89ce58) at process.c
#2 0x4006d3 in main (argc=4,argv=0x7fff0b89ce58) at bugexample.c
(gdb)
```


SciNet compute • calcul CANADA

GDB building block #3: 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 . . . ")

SciNet compute • calcul CANADA

GDB building block #3: 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

print	print variable
display	print at every prompt
set variable	change variable
watch	stop if variable changes

GDB building block #3: Variables

Remember: We were looking at a seg fault in **`s += a[i]`**.

```
(gdb) print i
0
(gdb) print a[0]
Cannot access memory at address 0x4
(gdb) print a
0x4
(gdb) up
#1  0x000000000040082a in process (argc=4, argv=0x7fff0b89ce58)
11      int s = sum_integers(n, arg);
(gdb) print arg
$1 = (int *) 0x4
(gdb) list
7 void process(int argc, char** argv)
8 {
9      int* arg = read_integer_arguments(argc, argv);
10      int n = argc-1;
11      int s = sum_integers(n, arg);
12      print_integers(n, arg);
13      printf("Sum of integers is: %d\n", s);
```

SciNet compute • calcul CANADA

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.

SciNet compute • calcul CANADA

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

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

SciNet compute • calcul CANADA

GDB building block #4: Automatic interruption

```
...
(gdb) list
7 void process(int argc, char** argv)
8 {
9     int* arg = read_integer_arguments(argc, argv);
10     int n = argc-1;
11     int s = sum_integers(n, arg);
12     print_integers(n, arg);
13     printf("Sum of integers is: %d\n", s);
14     free(arg);
15 }
(gdb) break read_integer_arguments
Breakpoint 1 at 0x4006ec: file intlisttools.c, line 8.
```


SciNet compute • calcul CANADA

GDB building block #4: Automatic interruption

```
...
(gdb) list
7 void process(int argc, char** argv)
8 {
9     int* arg = read_integer_arguments(argc, argv);
10     int n = argc-1;
11     int s = sum_integers(n, arg);
12     print_integers(n, arg);
13     printf("Sum of integers is: %d\n", s);
14     free(arg);
15 }
(gdb) break read_integer_arguments
Breakpoint 1 at 0x4006ec: file intlisttools.c, line 8.

(gdb) run 1 3 5
```

SciNet compute • calcul CANADA

GDB building block #4: Automatic interruption

```
...
(gdb) list
7 void process(int argc, char** argv)
8 {
9     int* arg = read_integer_arguments(argc, argv);
10     int n = argc-1;
11     int s = sum_integers(n, arg);
12     print_integers(n, arg);
13     printf("Sum of integers is: %d\n", s);
14     free(arg);
15 }
(gdb) break read_integer_arguments
Breakpoint 1 at 0x4006ec: file intlisttools.c, line 8.

(gdb) run 1 3 5

Starting program: debug/code/bugexample/bugexample 1 3 5
Breakpoint 1, read_integer_arguments (n=4, a=0x7fffffffc9b8)
    at intlisttools.c:8
8     int* result = malloc(sizeof(int)*(n-1));
(gdb)
```


SciNet compute • calcul CANADA

GDB building block #5: Step through code

Stepping through code

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

SciNet compute • calcul CANADA

GDB building block #5: 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

Command	Description
`list`	list part of code
`next`	continue until next line
`step`	step into function
`finish`	continue until function end
`until`	continue until line/function

SciNet compute • calcul CANADA

GDB building block #5: Step through code

```
...
(gdb) list 6,14
6 int* read_integer_arguments(int n, char** a)
7 {
8     int* result = malloc(sizeof(int)*(n-1));
9     int i;
10     /* convert every argument, but skip '0', because it is ju
11         executable name */
12     for (i=1;i<n;i++)
13         result[i] = atoi(a[i]);
14 }
(gdb) display result
1: result = (int *) 0x0
(gdb) next
12     for (i=1;i<n;i++)
1: result = (int *) 0x601010
(gdb) until 14
```

SciNet compute • calcul CANADA

GDB building block #5: Step through code

```
(gdb) until 14
read_integer_arguments (n=4,a=0x7fffffffc9b8) at intlisttools.c:
14 }
1: result = (int *) 0x601010
(gdb) finish
Run till exit from #0  read_integer_arguments (n=4,
    a=0x7fffffffc9b8) at intlisttools.c:14
0x000000000040080c in process (argc=4, argv=0x7fffffffc9b8)
    at process.c:9
9     int* arg = read_integer_arguments(argc, argv);
Value returned is $3 = (int *) 0x4
(gdb)
```


SciNet compute • calcul CANADA

GDB building block #5: Step through code

```
(gdb) until 14
read_integer_arguments (n=4,a=0x7fffffffc9b8) at intlisttools.c:
14 }
1: result = (int *) 0x601010
(gdb) finish
Run till exit from #0  read_integer_arguments (n=4,
    a=0x7fffffffc9b8) at intlisttools.c:14
0x000000000040080c in process (argc=4, argv=0x7fffffffc9b8)
    at process.c:9
9     int* arg = read_integer_arguments(argc, argv);
Value returned is $3 = (int *) 0x4
(gdb)
```

He, why is the result variable equal to **`0x601010`** while the value returned is **`0x4`**?

SciNet compute • calcul CANADA

GDB building block #5: Step through code

```
(gdb) until 14
read_integer_arguments (n=4,a=0x7fffffffc9b8) at intlisttools.c:
14 }
1: result = (int *) 0x601010
(gdb) finish
Run till exit from #0  read_integer_arguments (n=4,
    a=0x7fffffffc9b8) at intlisttools.c:14
0x000000000040080c in process (argc=4, argv=0x7fffffffc9b8)
    at process.c:9
9     int* arg = read_integer_arguments(argc, argv);
Value returned is $3 = (int *) 0x4
(gdb)
```

He, why is the result variable equal to **`0x601010`** while the value returned is **`0x4`**?

Contradicts your assumption of what the program does.
The program is always right, you are wrong.

SciNet compute • calcul CANADA

GDB building block #5: Step through code

Why is the result variable equal to **`0x601010`** while the value returned is **`0x4`**?

SciNet compute • calcul CANADA

GDB building block #5: Step through code

Why is the result variable equal to **0x601010** while the value returned is **0x4**?

```
(gdb) list read_integer_arguments,+7
7 {
8     int* result = malloc(sizeof(int)*(n-1));
9     int i;
10     /* convert every argument, but skip '0', because it is ju
11         executable name */
12     for (i=1;i<n;i++)
13         result[i] = atoi(a[i]);
14 }
```


SciNet compute • calcul CANADA

GDB building block #5: Step through code

Why is the result variable equal to **0x601010** while the value returned is **0x4**?

```
(gdb) list read_integer_arguments,+7
7 {
8     int* result = malloc(sizeof(int)*(n-1));
9     int i;
10     /* convert every argument, but skip '0', because it is ju
11         executable name */
12     for (i=1;i<n;i++)
13         result[i] = atoi(a[i]);
14 }
```

Aargh! Forgot the return statement!

SciNet compute • calcul CANADA

GDB building block #5: Step through code

Why is the result variable equal to **0x601010** while the value returned is **0x4**?

```
(gdb) list read_integer_arguments,+7
7 {
8     int* result = malloc(sizeof(int)*(n-1));
9     int i;
10     /* convert every argument, but skip '0', because it is ju
11        executable name */
12     for (i=1;i<n;i++)
13         result[i] = atoi(a[i]);
14 }
```

Aargh! Forgot the return statement!

Feeling like an idiot is a common side-effect of debugging.

SciNet compute • calcul CANADA

GDB command summary

help	**h**	print description of
run	**r**	run from the start (+args)
backtrace/where	**ba**	function call stack
list	**l**	list code lines
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
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
quit	**q**	quit gdb

SciNet compute • calcul CANADA

Memory Debugging

SciNet compute • calcul CANADA

Memory Checking: Valgrind

- Memory errors do not always give segfaults
- Commonly have to go *way* out of bounds to get a segfault.
- Write into other variable - hard to find problem.
- Valgrind - intercepts each memory call and checks them.
- Finds illegal accesses, uninitialized values, memory leaks.
- Warning: Quite verbose, typically, and, if you use external libraries, sometimes false positives. debugging too.

SciNet compute • calcul CANADA

Valgrind example

```
$ valgrind ./bugexample 1 3 5
==909== Memcheck, a memory error detector
==909== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward
==909== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyr
==909== Command: ./bugexample 1 3 5
==909==
==909== Invalid write of size 4
==909==    at 0x400741: read_integer_arguments (intlisttools.c:1
==909==    by 0x40080B: process (process.c:9)
==909==    by 0x4006D2: main (bugexample.c:12)
==909==  Address 0x51c304c is 0 bytes after a block of size 12 a
==909==    at 0x4C2636D: malloc (vg_replace_malloc.c:291)
==909==    by 0x4006FF: read_integer_arguments (intlisttools.c:8
==909==    by 0x40080B: process (process.c:9)
==909==    by 0x4006D2: main (bugexample.c:12)
==909==
==909== Invalid read of size 4
...
```

SciNet compute • calcul CANADA

Valgrind example (continued)

```
==909== HEAP SUMMARY:
==909==     in use at exit: 12 bytes in 1 blocks
==909==   total heap usage: 1 allocs, 0 frees, 12 bytes allocate
==909==
==909== LEAK SUMMARY:
==909==    definitely lost: 12 bytes in 1 blocks
==909==    indirectly lost: 0 bytes in 0 blocks
==909==      possibly lost: 0 bytes in 0 blocks
==909==    still reachable: 0 bytes in 0 blocks
==909==         suppressed: 0 bytes in 0 blocks
==909== Rerun with --leak-check=full to see details of leaked me
==909==
==909== For counts of detected and suppressed errors, rerun with
==909== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 6 f
Segmentation fault
$ _
```

SciNet compute • calcul CANADA

Valgrind recommendations

- Using valgrind on mature codes often shows lots of errors. Now, some may not be an issue (e.g. dead code or false positives from libraries), but hard to know.
- So: start using valgrind early in development.
- Program modularly, and create small unit tests, on which you can comfortably use valgrind.
- Apart from this basic valgrind usage, there are other tools availble with valgrind to deal cache performance, to get more detailed memory leak information, to detect race conditions, etc. (some of which we'll discuss later).

Graphical symbolic debuggers

SciNet compute • calcul CANADA

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)

SciNet compute • calcul CANADA

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)
```


SciNet compute • calcul CANADA

Graphical symbolic debuggers - ddd

DDD: /home/rzon/Courses/snugdebug/ex2/add.c (on gpc-f102n084)

File Edit View Program Commands Status Source Data Help

(): th

Lookup Find» Break Watch Print Display Plot Hide Rotate Set Undisp

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);

}
```

DDD: Threads (on gpc-f102n084)

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

DDD: Run, Interrupt, Step, Stepi, Next, Nexti, Until, Finish, Cont, Kill, Up, Down, Undo, Redo, Edit, Make

```
Breakpoint 1, main.omp_fn.0 (.omp_data_i=0x7fffffffd9f0
(gdb) c
Continuing.
[Switching to Thread 0x40a00940 (LWP 25170)]

Breakpoint 1, main.omp_fn.0 (.omp_data_i=0x7fffffffd9f0) 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=0x7fffffffd9f0) at add.c:17
(gdb)
```

Display 3: th (enabled, scope main.omp_fn.0, address 0x41401074)

Graphical symbolic debuggers - ddt

DDT

SciNet compute • calcul CANADA

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.
- Part of the "Allinea Forge" suite, which also includes a 'profiler' called *MAP*.

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>
```

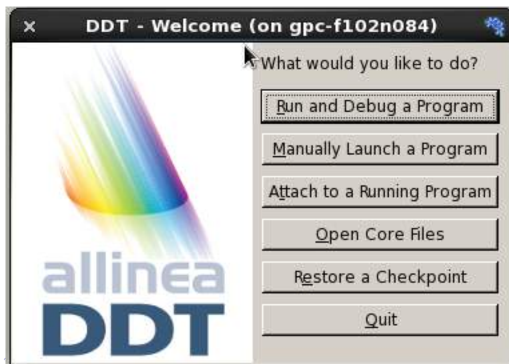
- First time: create config file: OpenMPI (skip other steps)
- Then gui for setting up debug session.

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)

DDT - Run (on gpc-f102n084)

Application: /home/s/scinet/rzon/Code/diff3d/diff3d — Details

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

☑ **MPI**: 2 processes, OpenMPI — Details

Number of processes: 2

Implementation: OpenMPI, no queue — Change...

mpirun arguments

☑ **OpenMP:** 4 threads — Details

Number of OpenMP threads: 4

☐ **CUDA** — Details

☑ **Memory Debugging:** Minimal, No guard pages, Backtraces, Preload — Details...

Environment Variables: none — Details

Plugins: none — Details

Run — Cancel

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:

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%

OK — Cancel

SciNet compute • calcul CANADA

Run and Debug a Program (session setup)

DDT - Run (on gpc-f102n084)

Application: /home/s/scinet/rzon/Code/diff3d/diff3d — Details

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

☑ **MPI**: 2 processes, OpenMPI — Details

Number of processes: 2

Implementation: OpenMPI, no queue — Change...

mpirun arguments

☑ **OpenMP:** 4 threads — Details

Number of OpenMP threads: 4

☐ **CUDA** — Details

☑ **Memory Debugging:** Minimal, No guard pages, Backtraces, Preload — Details...

Environment Variables: none — Details

Plugins: none — Details

Run — Cancel

Memory Debugging Options (o

☑ Preload the memory debugging library: Lang

Note: Preloading only works for programs linked a
program is statically linked, you must relink it again

Heap Debugging

- (•) Minimal (fewest tests, picks up invalid pointers
- ( ) Runtime (fast, basic tests including fence-post
- ( ) Low (adds minimal heap checking, overwriting
- ( ) Medium (adds full heap checking, always reloc
- ( ) High (adds checking for arguments to commo
- ( ) Custom:

Heap Overflow/Underflow Detection

☐ Add guard pages to detect out of bounds hea

Guard pages: 1 Add guard pages: Afte

Advanced

☐ Specify heap-check interval: 100

☑ Store stack backtraces for memory allocations

☐ Only enable for these processes:

0-1 100% Selec

DDT - Run (on gpc-f102n084)

Application: /home/s/scinet/rzon/Code/diff3d/diff3d — Details

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

[x] **MPI**: 2 processes, OpenMPI — Details

Number of processes: 2

Implementation: OpenMPI, no queue — Change...

mpirun arguments

[x] **OpenMP:** 4 threads — Details

Number of OpenMP threads: 4

[ ] **CUDA** — Details

[x] **Memory Debugging:** Minimal, No guard pages, Backtraces, Preload — Details...

Environment Variables: none — Details

Plugins: none — Details

Memory De

[x] Preload the memory de

Note: Preloading only wor
program is statically linked

Heap Debugging

(o) Minimal (fewest tests,

( ) Runtime (fast, basic te

( ) Low (adds minimal he

( ) Medium (adds full hea

( ) High (adds checking f

( ) Custom:

Heap Overflow/Underflow

[ ] Add guard pages to c

Guard pages: 1

Advanced

[ ] Specify heap-check in

[x] Store stack backtrace

[ ] Only enable for these

0-1

DDT - Run (on gpc-f102n084)

Application: /home/s/scinet/rzon/Code/diff3d/diff3d — Details

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

☑ **MPI**: 2 processes, OpenMPI — Details

Number of processes: 2

Implementation: OpenMPI, no queue — Change...

mpirun arguments

☑ **OpenMP:** 4 threads — Details

Number of OpenMP threads: 4

☐ **CUDA** — Details

☑ **Memory Debugging:** Minimal, No guard pages, Backtraces, Preload — Details...

Environment Variables: none — Details

Plugins: none — Details

☑ Preloa

Note: Pr
program

Heap D

Mini
Runt
Low
Med
High
Cust

Heap O

☐ Add
Guard

Advanc

☐ Spe
☑ Stor

Application: /home/s/scinet/rzon/Code/diff3d/diff3d Details ▲

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

☑ **MPI**: 2 processes, OpenMPI Details ▲

Number of processes: 2

Implementation: OpenMPI, no queue Change...

mpirun arguments

☑ **OpenMP:** 4 threads Details ▲

Number of OpenMP threads: 4

☐ **CUDA** Details

☑ **Memory Debugging:** Minimal, No guard pages, Backtraces, Preload Details...

iff3d

Details ▲

ff3d

Details ▲

Details ▲

Details ▼

es, Backtraces, Preload

Details...

Memory Debugging Options

☑ Preload the memory debugging library:

Note: Preloading only works for programs link
program is statically linked, you must relink it a

Heap Debugging

- ◉ Minimal (fewest tests, picks up invalid poi
- ○ Runtime (fast, basic tests including fence-p
- ○ Low (adds minimal heap checking, overwri
- ○ Medium (adds full heap checking, always r
- ○ High (adds checking for arguments to com
- ○ Custom:

Heap Overflow/Underflow Detection

☐ Add guard pages to detect out of bounds

Guard pages: 1 Add guard pages:

Advanced

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:

Heap Overflow/Underflow Detection

☐ Add guard pages to detect out of bounds heap access

Guard pages: 1 Add guard pages: After

Advanced

Run and Debug a Program (session setup)

DDT - Run (on gpc-f102n084)

Application: /home/s/scinet/rzon/Code/diff3d/diff3d — Details

Application: /home/s/scinet/rzon/Code/diff3d/diff3d

Arguments:

Input File:

Working Directory:

MPI: 2 processes, OpenMPI — Details

Number of processes: 2

Implementation: OpenMPI, no queue — Change...

mpirun arguments

OpenMP: 4 threads — Details

Number of OpenMP threads: 4

CUDA — Details

Memory Debugging: Minimal, No guard pages, Backtraces, Preload — Details...

Environment Variables: none — Details

Plugins: none — Details

Run — Cancel

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:

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%

OK — Cancel

SciNet compute • calcul CANADA

User interface (1)

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

Search (Ctrl+K)

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

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 << "nthreads=" << nthrds << 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

Stacks

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

Evaluate

Expression	Value

Ready

compute • calcul CANADA

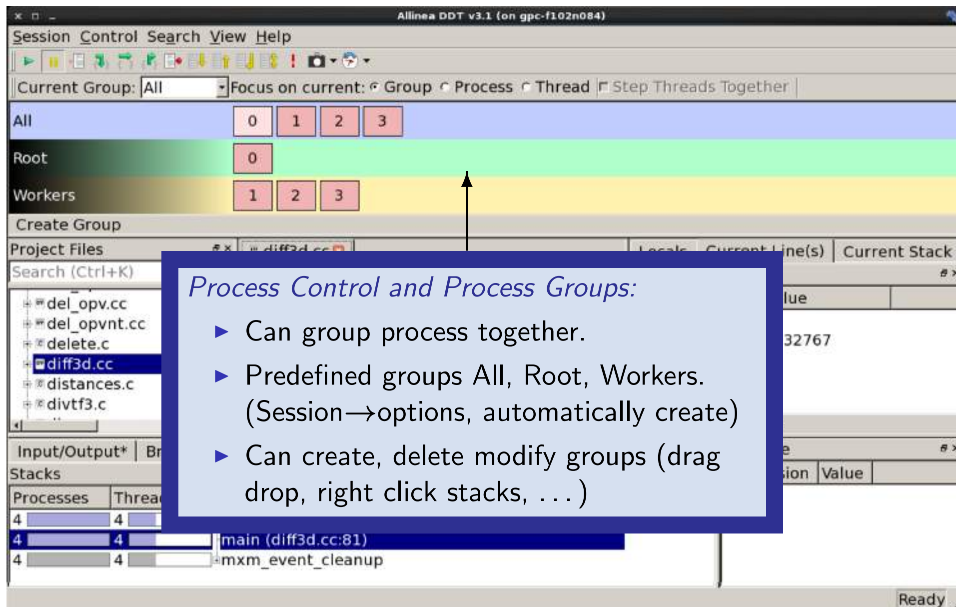
User interface (2)

DDT uses a tabbed-document interface.

User interface (3)

When the session begins, DDT automatically finds source code from information compiled in the executable.

User interface (4)



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)

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

Current Group: All

All

Root

Workers 1 2 3

Create Group

Different colour coding for each group's current source line.

Project Files

Search (Ctrl+K)

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

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 << "nthreads=" << nthrds << 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

Stacks

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

Evaluate

Expression	Value

Ready

compute • calcul CANADA

User interface (6)

Session Control Dialog:

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

User interface (7)

Breakpoints Tab

Can suspend, jump to, delete, load, save

User interface (8)

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

Create Group

Project Files

Search (Ctrl+K)

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

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

Current Line(s) | Current Stack

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

Type: none selected

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

Stacks

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

Evaluate

Expression	Value

Ready

Focus:
Choose between Group, process or thread

compute • calcul CANADA

User interface (9)

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

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

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

Locals | Current Line(s) | Current Stack

Current Line(s)

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

Type: none selected

diff3d.cc
distances.c
divtf3.c

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

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

Stacks

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

Evaluate

Expression	Value

Ready

compute • calcul CANADA

User interface (9)

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

User interface (10)

Current line variables

Allinea 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

Search (Ctrl+K)

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

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 << "nthreads=" << nthrds << 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

Stacks

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

Evaluate

Expression	Value

Ready

compute • calcul CANADA

User interface (11)

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

Current Group: All

Local variables for process

All

Root 0

Workers 1 2 3

Create Group

Project Files

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

diff3d.cc

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

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

Stacks

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

Evaluate

Expression	Value

Ready

compute • calcul CANADA

User interface (12)

Allinea DDT v3.1 (on gpc-f102n084)

Session Control Search View Help

Current Group: All

Evaluate window

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

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 << "nthreads=" << nthrds << 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

Stacks

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

Evaluate

Expression	Value

Ready

First Demonstration DDT

```
$ cd $SCRATCH/debug/code
$ source setup
$ cd bugexample
$ make
$ ddt bugexample
```


SciNet compute • calcul CANADA

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!

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

Other features of DDT (3)

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

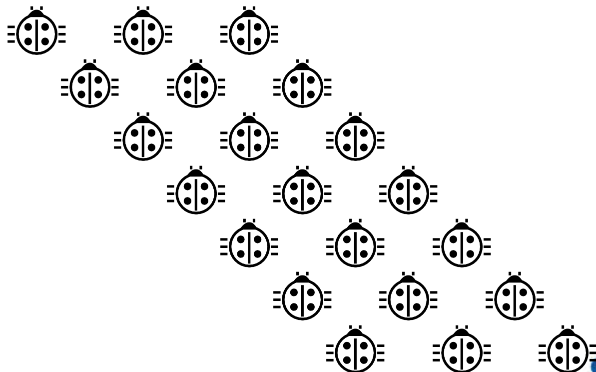
SciNet compute • calcul CANADA

Demonstration Memory Debugging with DDT

```
$ cd $SCRATCH/debug/code
$ source setup
$ cd ex4
$ make
$ ddt ex4
```


SciNet compute • calcul CANADA

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>`**

SciNet compute • calcul CANADA

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)

SciNet compute • calcul CANADA

Shared memory debugging with DDT

Or use DDT:

Thread debugging example

```
$ cd $SCRATCH/debug/code
$ source setup
$ cd ex5
$ make
$ ddt ex5
```


SciNet compute • calcul CANADA

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.

SciNet compute • calcul CANADA

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

SciNet compute • calcul CANADA

Demonstration MPI Debugging with DDT

```
$ cd $SCRATCH/debug/code
$ source setup
$ cd ex2
$ make
$ ddt ex2
```


SciNet compute • calcul CANADA

Detecting deadlock with DDT

Message Queue

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

Demonstration MPI Message Queue in DDT

```
$ cd $SCRATCH/debug/code
$ source setup
$ cd ex3
$ make
$ ddt ex3
```

SciNet compute • calcul CANADA

Useful references

- N Matloff and PJ Salzman
  The Art of Debugging with GDB, DDD and Eclipse
- *GDB:* sources.redhat.com/gdb
- *DDT:* www.allinea.com/knowledge-center/tutorials
- *SciNet Wiki:* wiki.scinethpc.ca: Tutorials & Manuals