



Leaders in parallel software development tools

Debugging HPC Applications

David Lecomber

CTO, Allinea Software

david@allinea.com



Agenda

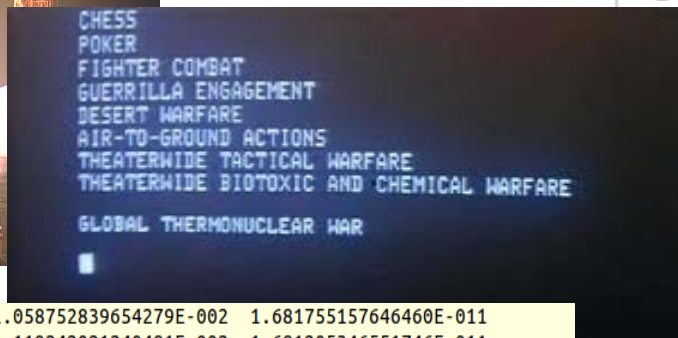
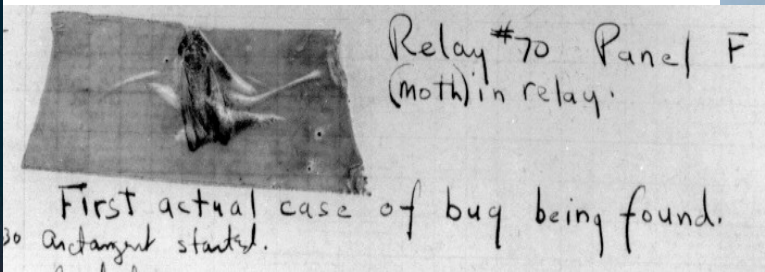
- Bugs and Debugging
- Debugging parallel applications
- Debugging OpenACC and other hybrid codes
- Debugging for Petascale (and beyond)

About Allinea



- HPC development tools company
 - Flagship product Allinea DDT
 - Now the leading debugger in parallel computing
 - The scalable debugger
 - Record holder for debugging software on largest machines
 - Production use at extreme scale – and desktop
 - Wide customer base
 - Blue-chip engineering, government and academic research
 - Strong collaborative relationships with customers and partners

Bugs in Practice



Country:

* United Kingdom ▼

ice Phone:

4.42E+11

Industry:

Please Select ▼



```
124395.444928040 1.058752839654279E-002 1.681755157646460E-011
124395.444323148 1.119242021240481E-002 1.681205346551746E-011
124395.443701451 1.181411574161518E-002 1.680444969505865E-011
124395.443062951 1.245261508079283E-002 1.679731384893576E-011
124395.442407647 1.310791832922166E-002 1.679052894606482E-011
124395.441735539 1.378002558885051E-002 1.678304215668999E-011
^forrtl: error (79): process quit (SIGQUIT)
Image PC Routine Line Source
omp-break 0000000000405400 Unknown Unknown Unknown
omp-break 0000000000404B23 Unknown Unknown Unknown
libomp5.so 00007F6E3A7C6B93 Unknown Unknown Unknown
Aborted (core dumped)
```

Some types of bug



- Some Terminology
 - Bohr bug
 - Steady, dependable bug
 - Heisenbug
 - Vanishes when you try to debug (observe)
 - Mandelbug
 - Complexity and obscurity of the cause is so great that it appears chaotic
 - Schroedinbug
 - First occurs after someone reads the source file and deduces that it never worked, after which the program ceases to work

Debugging



- Transforming a broken program to a working one
- **How?**
 - Track the problem
 - **Reproduce**
 - **Automate** - (and simplify) the test case
 - **Find origins** – where could the “infection” be from?
 - *Focus* – *examine the origins*
 - *Isolate* – *narrow down the origins*
 - *Correct* – *fix and verify the testcase is successful*
- TRAFFIC
- Suggested Reading:
 - Zeller A., “Why Programs Fail”, 2nd Edition, 2009

How to focus and isolate

- A scientific process?
 - Hypothesis, trial and observation, ...
- Requires the ability to understand what a program is doing
 - Printf
 - Command line debuggers
 - Graphical debuggers
- Other options
 - Static analysis
 - Race detection
 - Valgrind
 - Manual source code review

What are debuggers?

- Tools to inspect the insides of an application whilst it is running
 - Ability to inspect process state
 - Inspect process registers, and memory
 - Inspect variables and stacktraces (nesting of function calls)
 - Step line by line, function by function through an execution
 - Stop at a line or function (breakpoint)
 - Stop if a memory location changes
 - Ideal to watch how a program is executed
 - Less intrusive on the code than printf
 - See exact line of crash – unlike printf
 - Test more hypotheses at a time



Debugging Parallel Applications

Debugging Parallel Applications

- Scalar bugs can be challenging: parallel even more so!
- The same need: observation, control
 - Complex environment – with complex problems
 - More processes, more data
 - More Heisenbugs – MPI communication library introduces potential non-determinism
 - Fewer options
 - Printf or command line debuggers are not quick enough

First example



- Typical problem scenario: application ends abruptly
 - Example potential causes
 - Segmentation fault
 - Early termination due to invalid parameters
- Where do we start?

Print statement debugging

- The first debugger: print statements

- Each process prints a message or value at defined locations
- Diagnose the problem from evidence and intuition

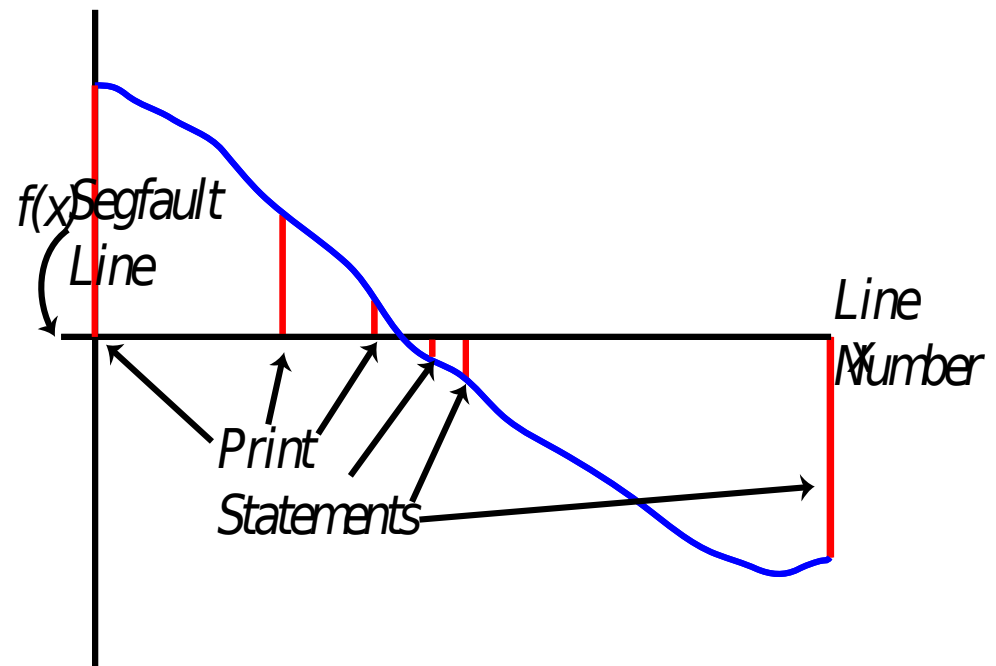
- A long slow process

- Analogous to bisection root finding

Thanks to Rebecca Hartman-Baker of ORNL for the analogy and animation

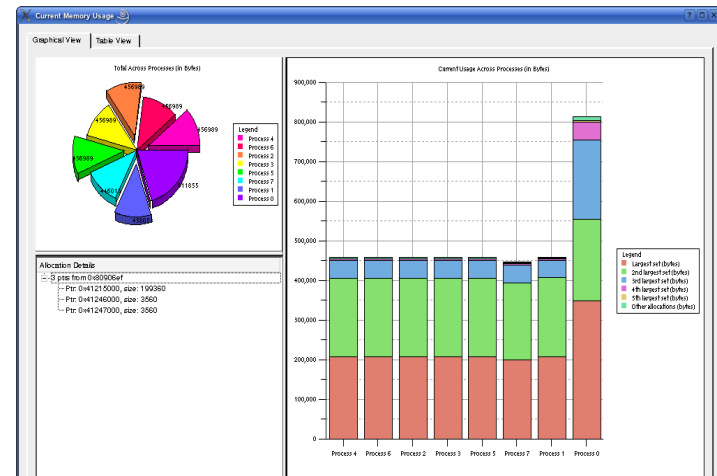
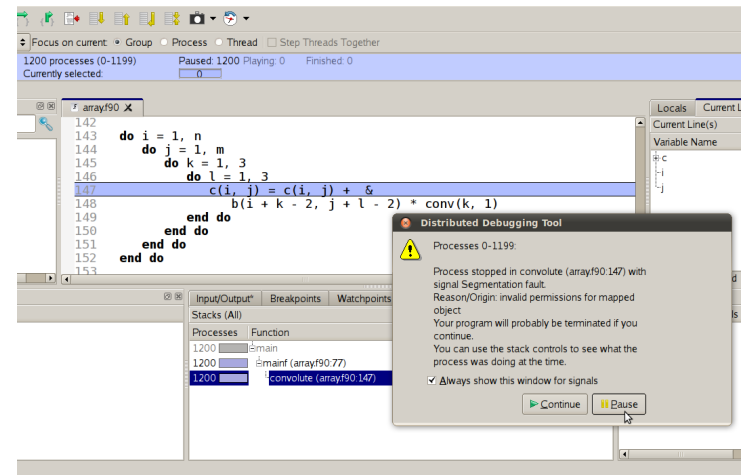
- Broken at modest scale

- Too much output – too many log files



Alinea DDT in a nutshell

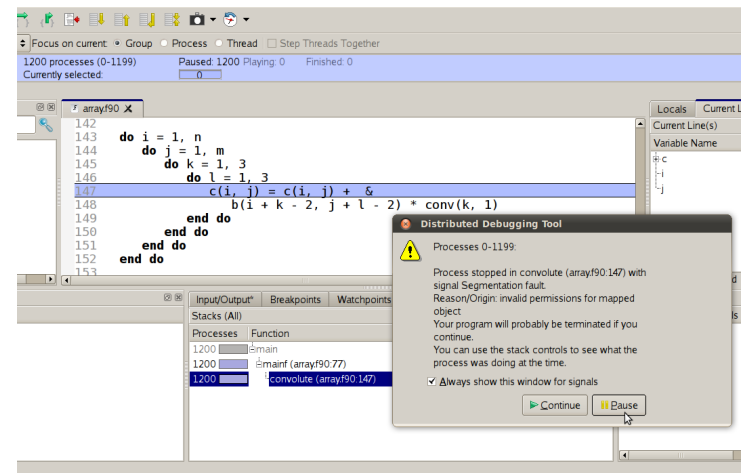
- Graphical source level debugger for
 - Parallel, multi-threaded, scalar or hybrid code
 - C, C++, F90, Co-Array Fortran, UPC
- Strong feature set
 - Memory debugging
 - Data analysis
- Managing concurrency
 - Emphasizing differences
 - Collective control



Fixing the everyday crash

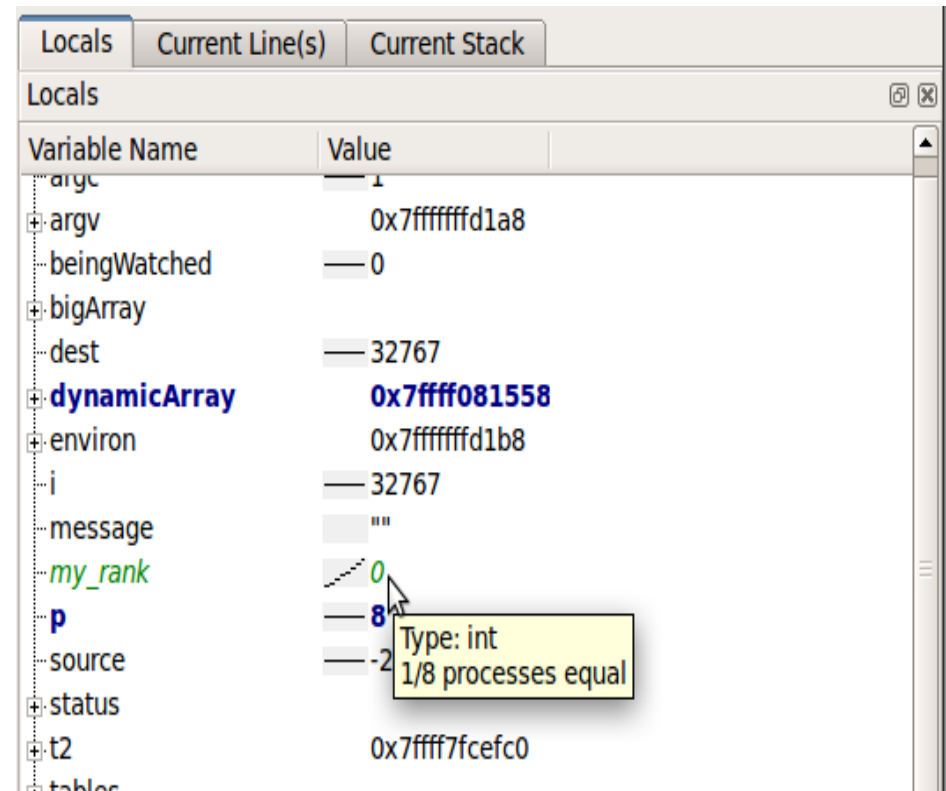
- The typical application crash or early exit:
 - Run your program in the debugger
ddt {application} {parameters}
 - Application crashes or starts to exit
- **Where** did it happen?
 - Allinea DDT merges stacks from processes and threads into a tree
 - Leaps to source automatically
- **Why** did it happen?
 - Some faults evident instantly from location(s)
 - But for others we need to look further – at variables

Stacks (All)	
Processes	Function
150120	__start
150120	__libc_start_main
150120	main
150120	pop (POP.f90:81)
150120	initialize_pop (initial.f90:119)
150120	init_communicate (communicate.f90:87)
150119	create_ocn_communicator (communicate.f90:300)
1	create_ocn_communicator (communicate.f90:303)



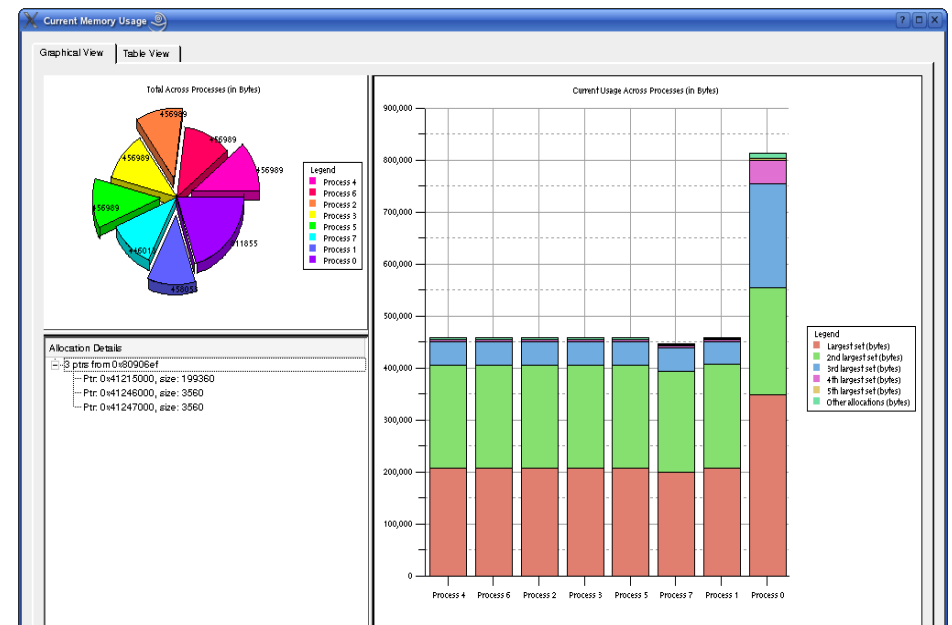
Simplifying data divergence

- Need to understand the data
 - Too many variables to trawl manually
 - Allinea DDT compares data automatically
- Smart highlighting
 - Subtle hints for differences and changes
 - With sparklines!
- More detailed analysis
 - Full cross process comparison
 - Historical values via tracepoints



Memory debugging

- Random errors are the worst kind
 - You can't fix a bug that doesn't repeat - memory debugging can force the bug
 - Better to crash every time, than only during product demos
- Allinea DDT helps eliminate random memory bugs
 - Enable memory debugging by ticking an option
 - Monitors usage: detects memory leaks
 - Automatically protects ends of arrays
 - Trigger instant stop on touching invalid memory
 - Also with CUDA support





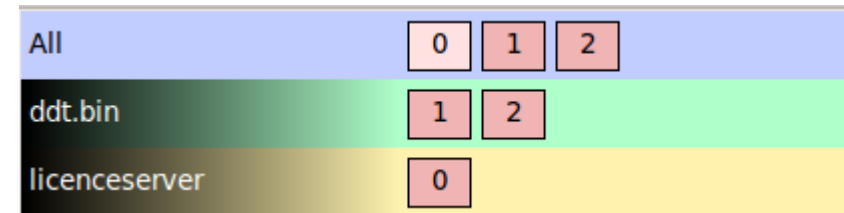
Interlude: Fixing a simple MPI bug

Second example

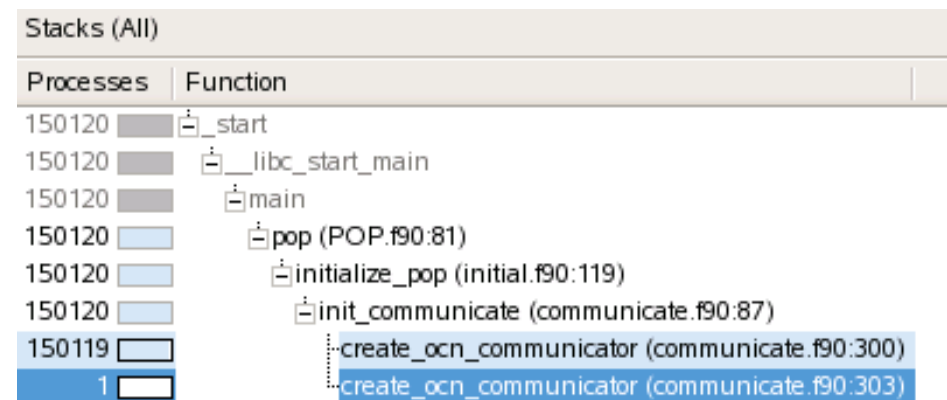
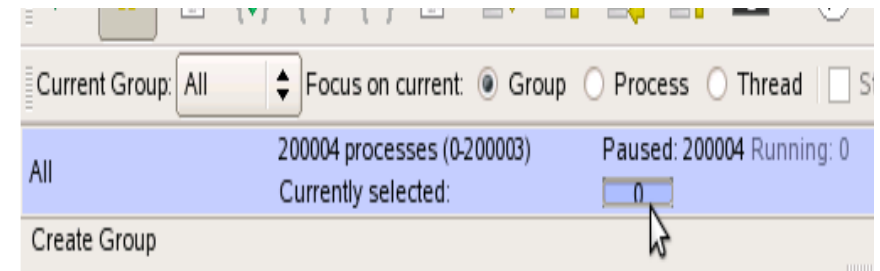
- More subtle issue
 - Not an immediately obvious crash
 - Crash occurs in self-evidently correct code
 - Something went wrong – somewhere else!
- How can a debugger help here?
 - Observation: we can watch things go bad

Controlling execution

- Observe application behaviour by controlling execution
 - Step, play or run to line based on groups
 - Change interleaving order by stepping/playing selectively
 - Set breakpoints
 - Set data watchpoints
- Examine data and progress at each point
- Group creation is easy
 - Integrated throughout Allinea DDT - eg. stack and data views



All	<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2
ddt.bin	<input type="checkbox"/> 1	<input type="checkbox"/> 2	
licenceserver	<input type="checkbox"/> 0		



Stacks (All)	
Processes	Function
150120	_start
150120	__libc_start_main
150120	main
150120	pop (POP.f90:81)
150120	initialize_pop (initial.f90:119)
150120	init_communicate (communicate.f90:87)
150119	create_ocn_communicator (communicate.f90:300)
1	create_ocn_communicator (communicate.f90:303)

Exploring large arrays

Array Expression:

Distributed Array Dimensions: [How do I view distributed arrays?](#)

Range of \$x (Distributed) Range of \$i

From: From:

To: To:

Display: Display:

☐ Auto-update

☒ Only show if: [See Examples](#)

Data Table

	i	2444	2733	3011	3185	4704	5343	6795	7881	9108	9467
x 0											
1				1					1		
2	1				1						1
3											
4		1									
5			1			1					
6											













- Browse arrays
 - 1, 2, 3, ... dimensions
 - Table view
- Filtering
 - Look for an outlier
- Export
 - Save to a spreadsheet
- View and search arrays from multiple processes
 - Search terabytes for rogue data – in parallel

Static analysis

- Analyzes source code
 - Detects some common errors (eg.)
 - Memory leaks
 - Buffer overflow
 - Unused variables
 - Not exhaustive – but a useful hint for debugging

```
29
30 threads = calloc(sizeof(pthread_t), nthreads);
31 ids = calloc(sizeof(int), nthreads);
32
33 init_mutex();
34
35 pthread_mutex_lock(mutley);
36 for (i = 0; i < nthreads; ++i) {
37     ids[i] = i;
38     pthread_create (threads + i, NULL, &thread,
39 }
40 pthread_mutex_unlock(mutley);
41 for (i = 0; i < nthreads; ++i)
42     pthread_join (threads[i], NULL);
43
44 return 0;
45 }
error Memory leak: threads
error Memory leak: ids
void *q)
46 {
49 volatile int busy = 0;
50 volatile int locker = 0; /* to be amended by
51 int i, j;
52 double k = 1;
53 int tid = *(int*) q;
54
55 usleep(rand() % 31);
56
```

Tracepoints

Input/Output	Breakpoints	Watchpoints	Tracepoints	Tracepoint Output	Stacks (All)
Tracepoint Output					
Tracepoint	Processes	Values logged			
vhone.f90:85	976, ranks 12, 14-17, 22-23, 12...	mype 	2172-3527	jcol:  2-83	mod <input type="checkbox"/> pey <input type="checkbox"/>
vhone.f90:81	960, ranks 12, 14-17, 22-23, 12...	ks 	1 kmax	pez <input type="checkbox"/>	
vhone.f90:85	942, ranks 12, 14-17, 22-23, 12...	mype 	2172-3527	jcol:  2-83	mod <input type="checkbox"/> pey <input type="checkbox"/>
vhone.f90:81	929, ranks 12, 14-17, 22-23, 12...	ks 	1 kmax	pez <input type="checkbox"/>	
vhone.f90:85	919, ranks 12, 14-17, 22-23, 12...	mype 	2172-3527	jcol:  2-83	mod <input type="checkbox"/> pey <input type="checkbox"/>
vhone.f90:81	898, ranks 12, 14-17, 22-23, 12...	ks 	1 kmax	pez <input type="checkbox"/>	
vhone.f90:85	884, ranks 12, 14-17, 22-23, 12...	mype 	2172-3527	jcol:  2-83	mod <input type="checkbox"/> pey <input type="checkbox"/>
vhone.f90:81	880, ranks 12, 14-17, 22-23, 12...	ks 	1 kmax	pez <input type="checkbox"/>	

- A scalable print alternative
 - Merged print – with a sparkline graph showing distribution
 - Change at runtime – no recompilation required

Offline debugging

- Machine access can be a problem
 - New offline mode
 - Set breakpoints, tracepoints from command line
 - Memory debugging
 - Record variables, stacks – crashes and breakpoints
 - Submit and forget
 - Post-mortem analysis
 - HTML/plain text
 - Debug while you sleep

Stack for process 0

Process stopped in sched_yield (syscall-template.S:82) with signal SIGSEGV (Segmentation fault). Reason/Origin: kill, sigsend or raise. Your program will probably be terminated if you continue. You can use the stack controls to see what the process was doing at the time.

▼ Stacks

Processes	Threads	Function
1-15	15	aplu (lu.f:118)
1-15	15	ssor (ssor.f:131)
1-15	15	blts (blts.f:55)
1-3,5-7,9-11,13,15,11		exchange_1 (exchange_1.f:37)
1-3,5-7,9-11,13,15,11		pmpi_recv
1-3,5-7,9-11,13,15,11		PMPi_Recv
1-3,5-7,9-11,13,15,11		mca_pml_obl_recv
1-3,5-7,9-11,13,15,11		opal_progress
1-3,5-7,9-11,13,15,11		sched_yield (syscall-template.S:82)
4,8,12,14	4	exchange_1 (exchange_1.f:55)
4,8,12,14	4	pmpi_recv
4,8,12,14	4	PMPi_Recv
4,8,12,14	4	mca_pml_obl_recv
4,8,12,14	4	opal_progress
4,8,12,14	4	sched_yield (syscall-template.S:82)

Stack for process 1

Every process in your program has terminated.

Messages Tracepoints Output

Tracepoints

#	Time	Tracepoint	Processes	Values
1	00:13.451	subdomain.f:96	0	jend: 0 ny: 9
2	00:13.453	ssor.f:177	0-15	delunm(5): 0.25
3	00:13.455	ssor.f:177	0-15	delunm(5): 0.25



Interlude: Fixing a more unusual bug

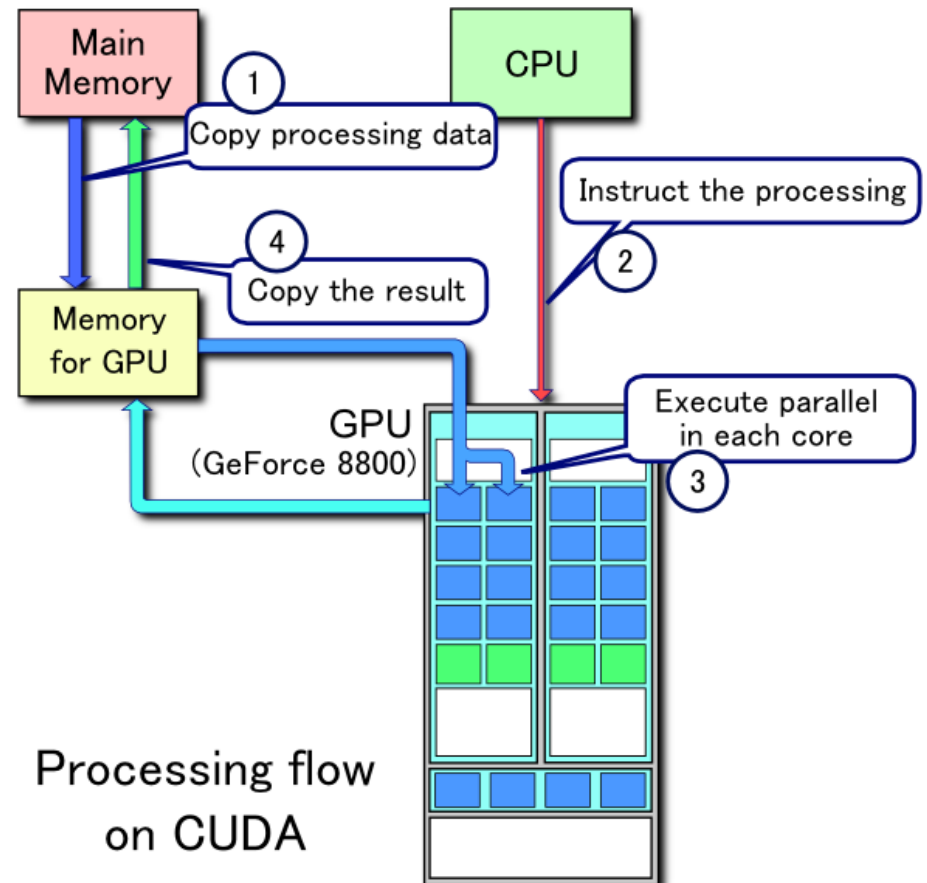
Hands on: A first exercise with Allinea DDT



Debugging OpenACC and other hybrid codes

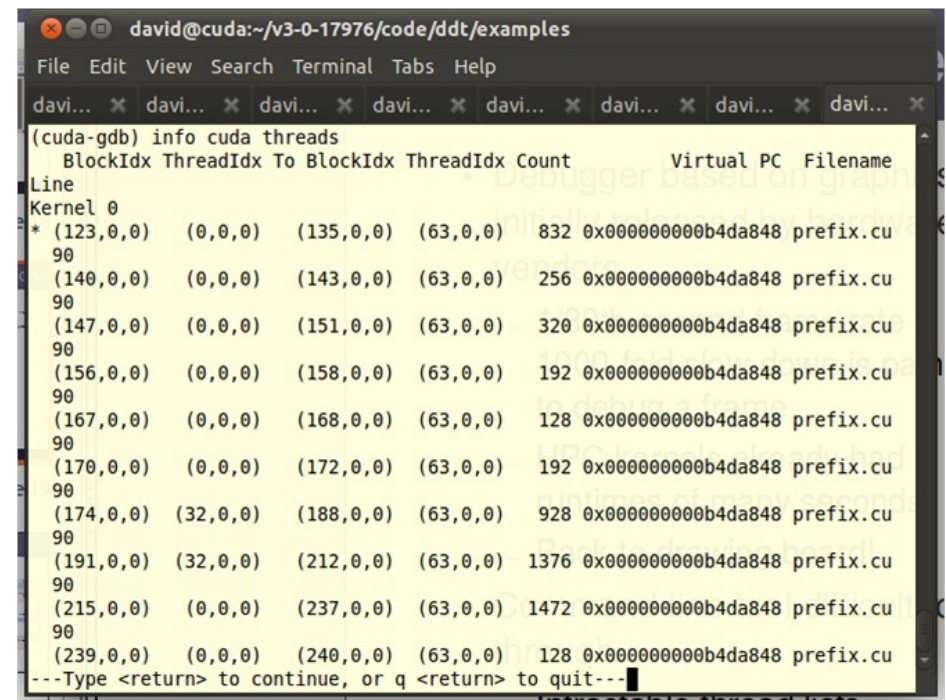
HPC's current challenge

- GPUs – a rival to traditional processors
 - AMD and NVIDIA
 - OpenCL, CUDA
- New languages, compilers, standards
- Great bang-for-bucks ratios
- A big challenge for HPC developers
 - Data transfer
 - Several memory levels
 - Grid/block layout and thread scheduling
 - Synchronization
- Bugs are **inevitable**



How do we fix GPU bugs?

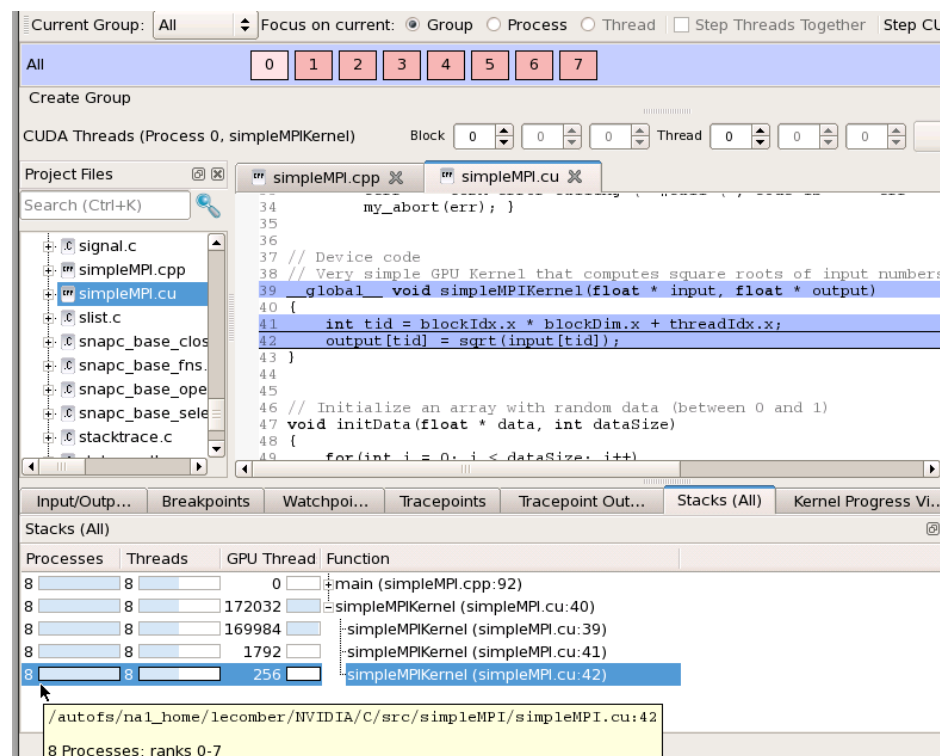
- Print statements
 - Too intrusive
- Command line debugger?
 - A good start:
 - Variables, source code
 - Large thread counts overwhelming
 - Too complex
- A graphical debugger...



```
david@cuda:~/v3-0-17976/code/ddt/examples
File Edit View Search Terminal Tabs Help
davi... x davi... x davi... x davi... x davi... x davi... x davi... x davi... x
(cuda-gdb) info cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count Virtual PC Filename
Line
Kernel 0
* (123,0,0) (0,0,0) (135,0,0) (63,0,0) 832 0x00000000b4da848 prefix.cu
90
(140,0,0) (0,0,0) (143,0,0) (63,0,0) 256 0x00000000b4da848 prefix.cu
90
(147,0,0) (0,0,0) (151,0,0) (63,0,0) 320 0x00000000b4da848 prefix.cu
90
(156,0,0) (0,0,0) (158,0,0) (63,0,0) 192 0x00000000b4da848 prefix.cu
90
(167,0,0) (0,0,0) (168,0,0) (63,0,0) 128 0x00000000b4da848 prefix.cu
90
(170,0,0) (0,0,0) (172,0,0) (63,0,0) 192 0x00000000b4da848 prefix.cu
90
(174,0,0) (32,0,0) (188,0,0) (63,0,0) 928 0x00000000b4da848 prefix.cu
90
(191,0,0) (32,0,0) (212,0,0) (63,0,0) 1376 0x00000000b4da848 prefix.cu
90
(215,0,0) (0,0,0) (237,0,0) (63,0,0) 1472 0x00000000b4da848 prefix.cu
90
(239,0,0) (0,0,0) (240,0,0) (63,0,0) 128 0x00000000b4da848 prefix.cu
---Type <return> to continue, or q <return> to quit---
```

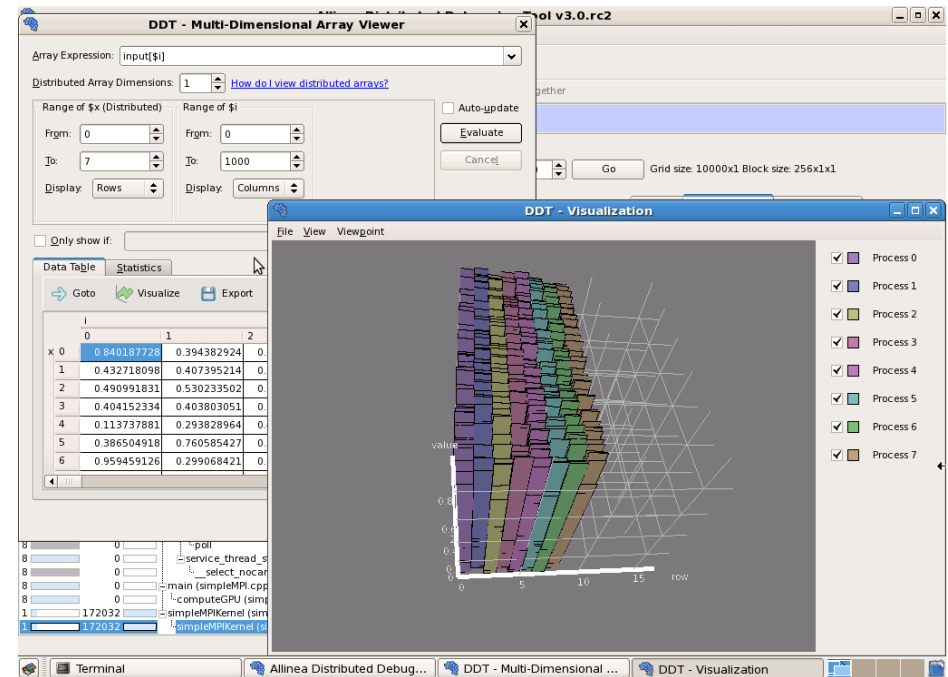
GPU Debugging

- Almost like debugging a CPU – we can still:
 - Run through to a crash
 - Step through and observe
- CPU-like debugging features
 - Double click to set breakpoints
 - Hover the mouse for more information
 - Step a warp, block or kernel
 - Follow threads through the kernel
- Simultaneously debugs CPU code
- CUDA Memcheck feature detects read/write errors

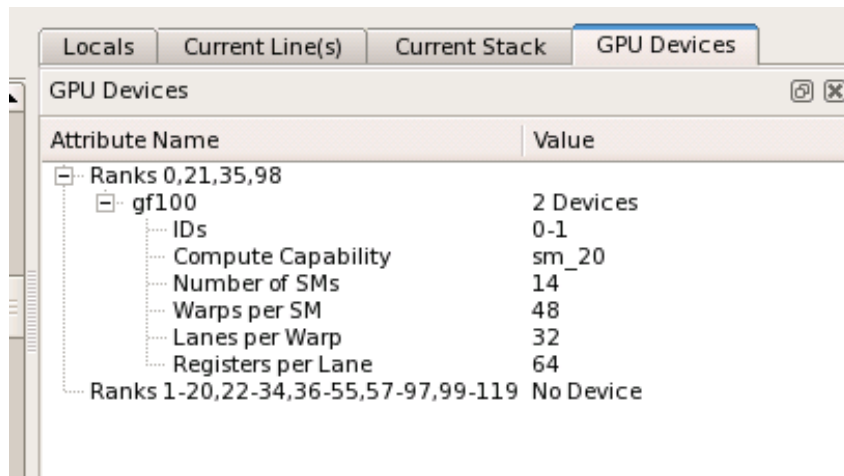


Examining GPU data

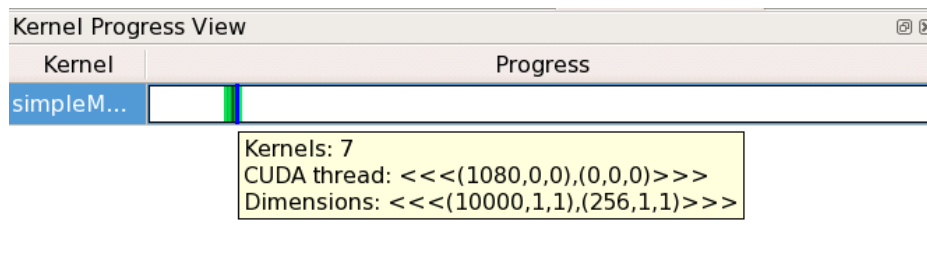
- Debugger reads host and device memory
 - Shows all memory classes: shared, constant, local, global, register..
 - Able to examine variables
 - ... or plot larger arrays directly from device memory



New overviews of GPUs



- Device overview shows system properties
 - Helps optimize grid sizes
 - Handy for bug fixing – and detecting hardware failure!
- Kernel progress view
 - Shows progress through kernels
 - Click to select a thread



Debugging for Directives

```
#pragma acc region
{
  for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
  /* compute on the host to compare */
  for( i = 0; i < n; ++i ) e[i] = a[i]
  /* check the results */
  for( i = 0; i < n; ++i )
    assert( r[i] == e[i] );
}
```

On this line:
1 Process: rank 0
1 Thread (Process 0): #1

PGI

- Supporting the environments that you use for hybrid development

```
5 # 29 "basic.c"
6 void hmpp_codelet_myFunc(int n, int A[n], const int B[n])
7 {
8 #pragma hmppcg gridify(i)
9 # 7 "<preprocessor>"
10 # 32 "basic.c"
11 for (int i = 0; i < n; ++i)
12 {
13   A[i] += B[i] + 1;
14 }
15 }
16
17
18 /* end of extracted source code
19
```

On this line:
1 Process: rank 0
Kernel 1: 32 GPU threads
<<<(0,0),(0,0,0)>>> ... <<<(0,0),(31,0,0)>>> (32 threads)

CAPS
Innovative software for manycore paradigms

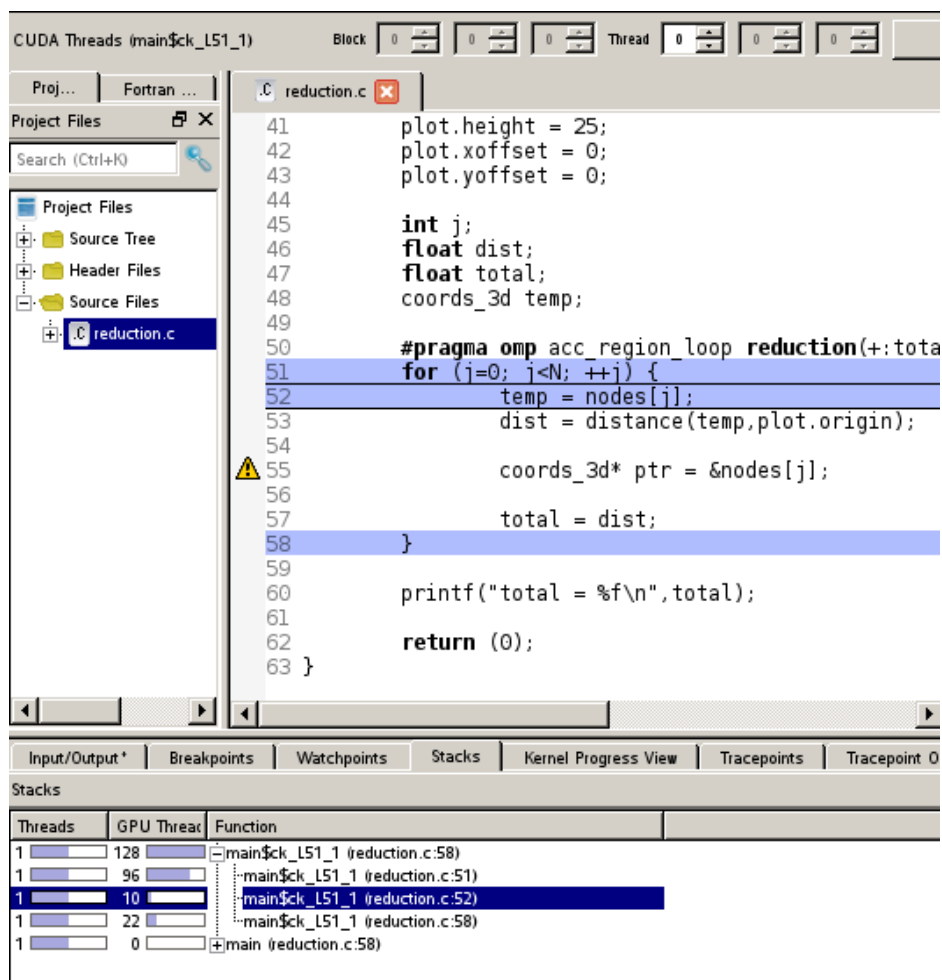
```
1 program main
2 integer, parameter :: n = 1000
3 real,dimension(n) :: input
4 real,dimension(n) :: result
5 integer :: i
6 do i = 1,n
7 input(i) = i*4.0
8 enddo
9 !$omp acc_region
10 !$omp acc loop
11 do i = 1,n
12 result(i) = input(i) * 4.0
13 enddo
14 !$omp
15 !$omp
16 print
17 end program
```

On this line:
1 Process: rank 0
Kernel 1: 32 GPU threads
<<<(0,0),(0,0,0)>>> ... <<<(0,0),(31,0,0)>>> (32 threads)

CRAY
THE SUPERCOMPUTER COMPANY

allinea
www.allinea.com

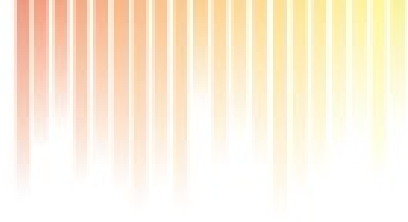
OpenACC debugging



- OpenACC for C and F90
 - Straightforward CUDA compute power
 - Getting code onto the GPU quickly
 - Optimization may still be required
- On device debugging with Allinea DDT
 - Variables – arrays, pointers, full F90 and C support
 - Set breakpoints and step warps and blocks
- Requires Cray compiler for on device debugging
 - Other compilers to follow



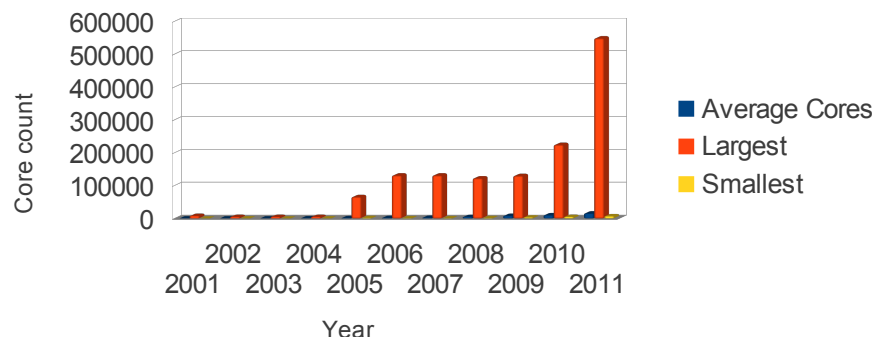
Interlude: CUDA Debugging



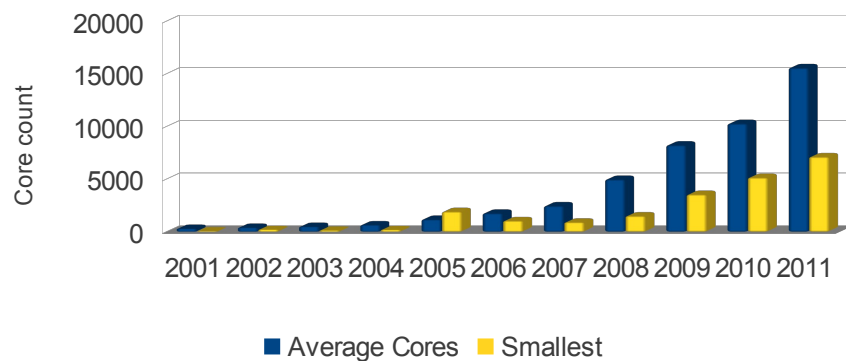
Debugging for Petascale

Extreme machine sizes

Growth in HPC core counts



HPC core counts



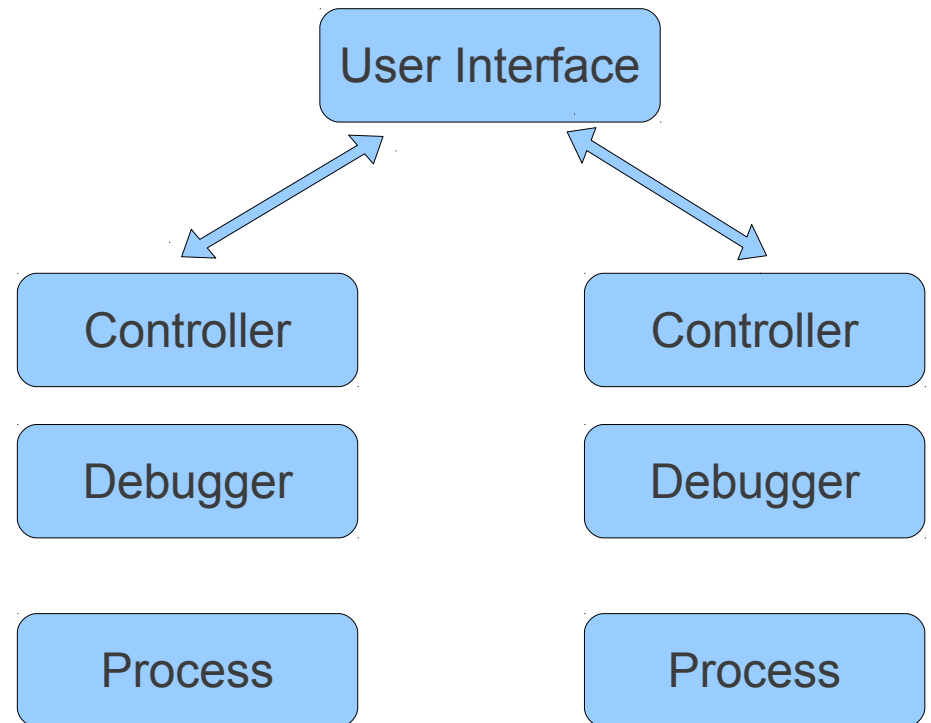
- Progress requires ever more CPU hours
 - Machine sizes are exploding
 - Skewed by largest machines
 - ... but a common trend everywhere else
 - Software is changing to exploit the machines

Bug fixing as scale increases

- Can we reproduce at a smaller scale?
 - Attempt to make problem happen on fewer nodes
 - Often requires reduced data set – the large one may not fit
 - Smaller data set may not trigger the problem
 - Does the bug even exist on smaller problems?
 - Didn't you already try the code at small scale?
 - Is it a system issue – eg. an MPI problem?
 - Is probability stacking up against you?
 - Unlikely to spot on smaller runs – without many many runs
 - But near guaranteed to see it on a many-thousand core run
- Debugging at extreme scale is a necessity

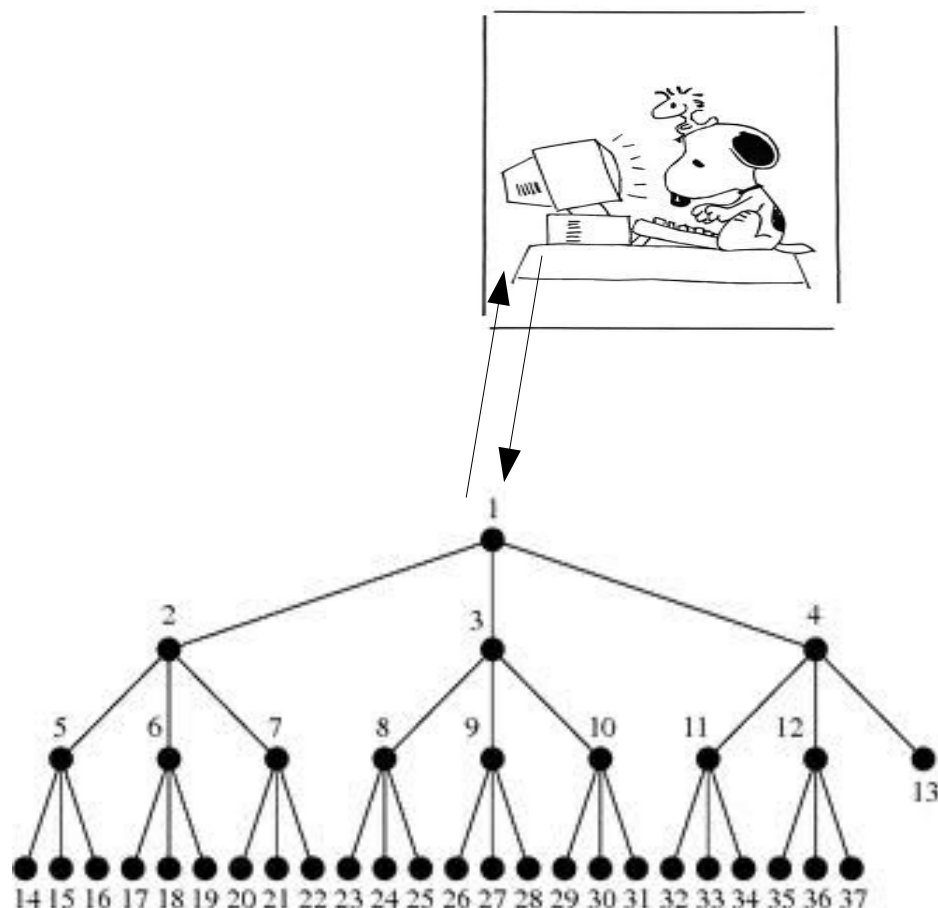
A simple parallel debugger

- A basic parallel debugger
 - Aggregate scalar debuggers and control asynchronously
 - Implement support for many platforms and MPI implementations
 - Develop user interface: simplify control and state display
- Initial architecture
 - Scalar debuggers connect to user interface
 - Eventual scalability bottlenecks
 - Operating system limitations: file handles, threads, processes
 - I/O limitations, memory and computation limitations
 - Machines still getting bigger...



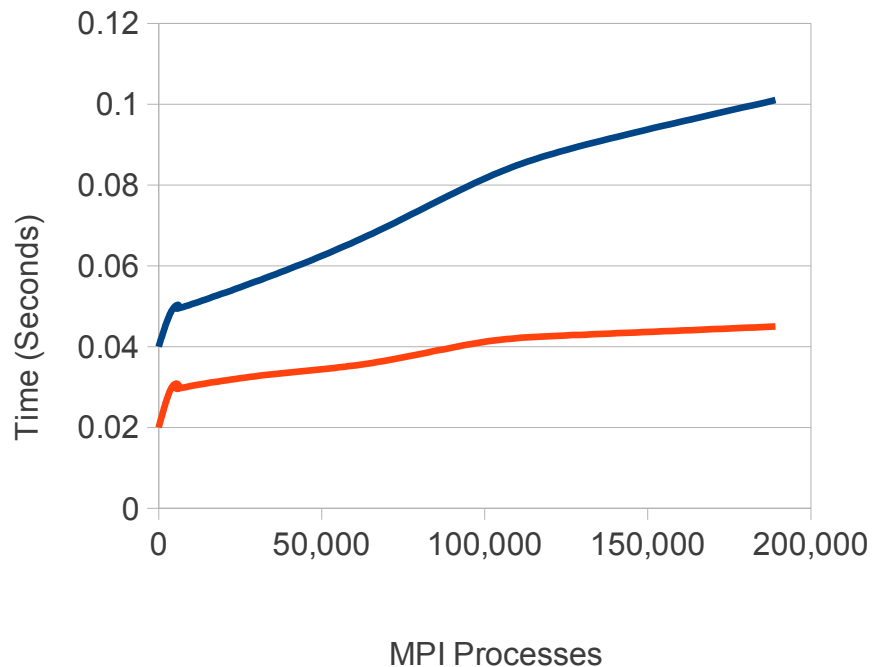
How to make a Petascale debugger

- A control tree is the solution
 - Ability to send bulk commands and merge responses
 - 100,000 processes in a depth 3 tree
 - Compact data type to represent sets of processes
 - eg. For message envelopes
 - An ordered tree of intervals?
 - Or a bitmap?
 - Develop aggregations
 - Merge operations are key
 - Not everything can merge losslessly
 - Maintain the essence of the information
 - eg. min, max, distribution



For Petascale and beyond

DDT 3.0 Performance Figures

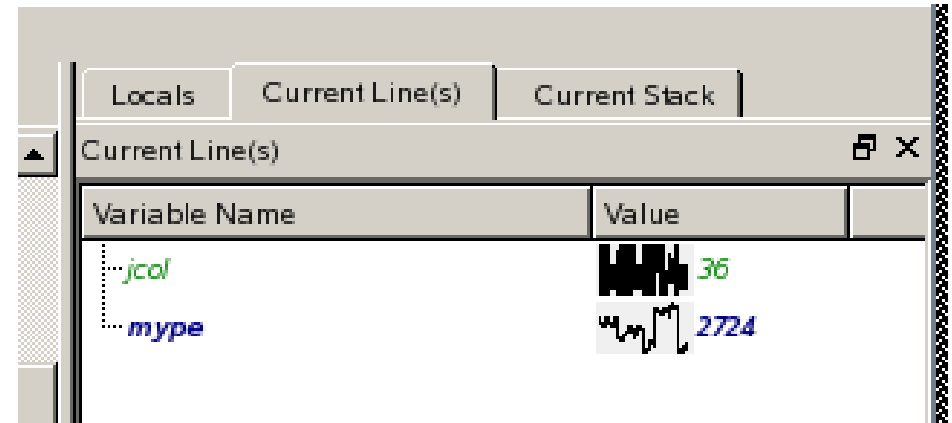


— All Step
— All Breakpoint

- Scale doesn't have to be hard
 - 100,000 cores should be as easy as 100 cores
 - The user interface is vital to success
- Scale doesn't have to be slow
 - High performance debugging - even at 200,000 cores
 - Step all and display stacks: 0.1 seconds
 - Logarithmic performance
- Stable and in production use
 - Routinely used by end users at over 100,000 cores

Key features at scale

- Top 5 features at scale
 - Parallel stack view
 - Ideal for divergence or deadlock
 - Automated data comparison: sparklines
 - Rogue data is easily seen
 - Parallel array searching
 - Data is too large to examine manually
 - Process control with step, play, and breakpoints
 - Still essential
 - Offline debugging
 - Access to machine may be hard – try offline debugging instead





Interlude: Petascale demonstration