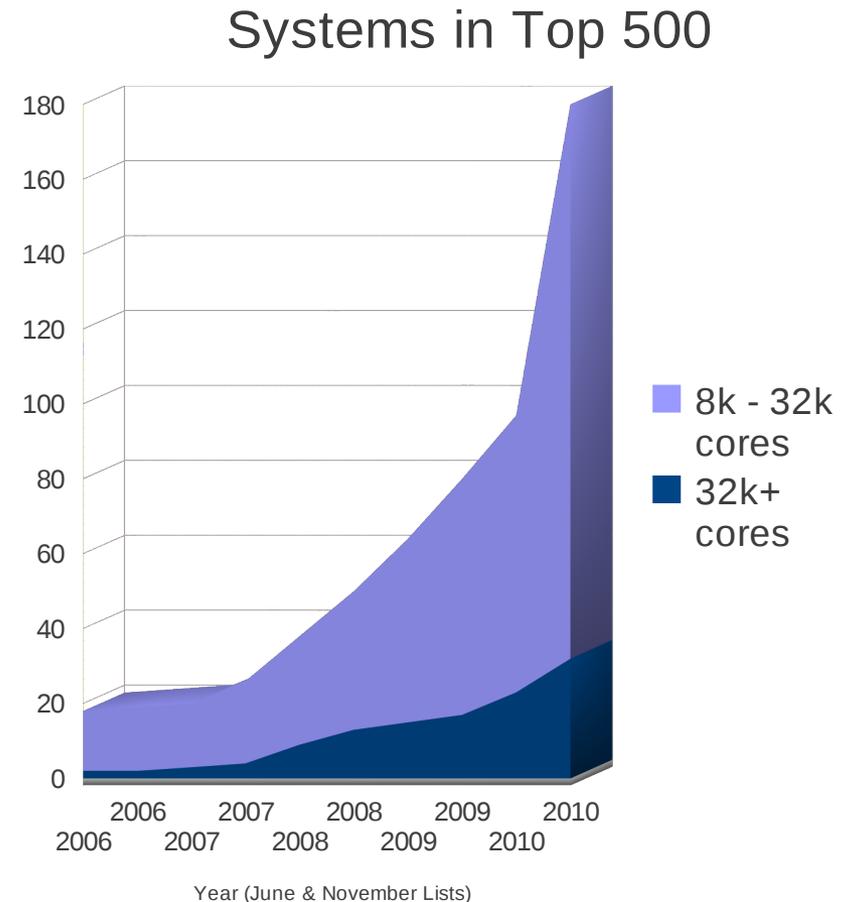




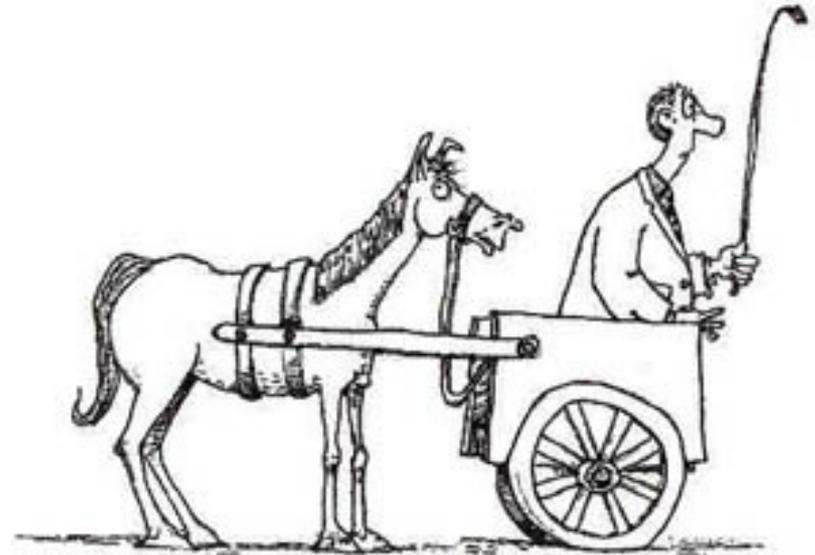
Debugging for Petascale

David Lecomber
david@allinea.com
CTO

- Processor counts growing rapidly
- GPUs entering HPC
- Large hybrid systems imminent
- But what happens when software doesn't work?



- *“Software has become the #1 roadblock ... Many applications will need a major redesign”* - IDC HPC Update, June 2010
 - Most ISV codes do not scale
 - High programming costs are delaying GPU usage
- Development tools are a vital part of the solution



- Developers of HPC tools since 2001
 - Modern and easy to use tools for HPC developers
 - Worldwide customer base from the smallest to the largest systems
- Allinea DDT
 - The market leading product for debugging
 - Designed for both new and experienced programmers
 - Scalable – the **only** Petascale debugger
 - Usable – debugging is easy at 1, or 100,000 cores

- Increasing job sizes leads to unanticipated errors
 - Regular bugs
 - Data issues from larger data sets - eg. garbage in..., overflow
 - Logic issues and control flow
 - Increasing probability of independent random error
 - Memory errors/exhaustion - “random” bugs!
 - Race conditions (particularly MPI timing)
 - System problems - MPI and operating system
 - Pushing coded boundaries
 - Algorithmic (performance)
 - Hard-wired limits (“magic numbers”)
 - Unknown unknowns
 -

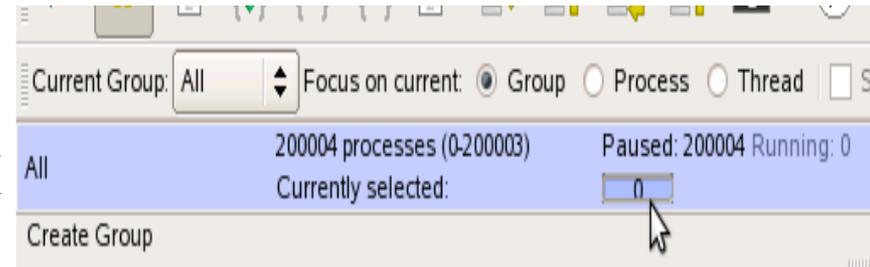
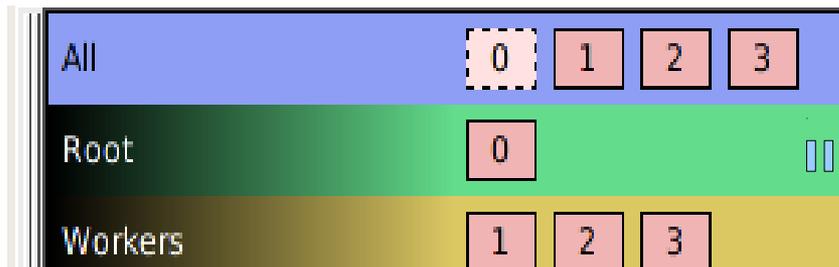
- Improved coding standards – unit tests, assertions
 - Good practice – but coverage is rarely perfect
 - Random/system issues – often missed
 - Combines well with debuggers
 - Find **why** a failure occurs not just a pass/fail
- Logging – printf and write
 - If you have good intuition into the problem
 - Edit code, insert print, recompile and re-run
 - **Slow and iterative**
 - Post-mortem analysis only
 - Hard establish real order of output of multiple processes
 - Rapid growth in log output size
 - **Unscalable**

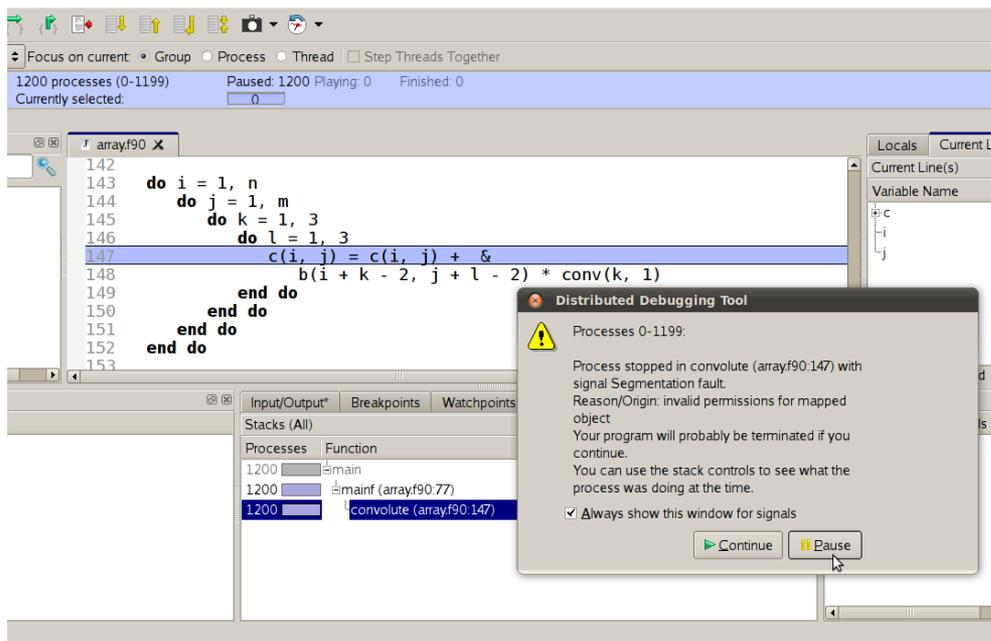
- Reproduce at a smaller scale
 - Attempt to make problem happen on fewer nodes
 - Often requires reduced data set – the large one may not fit
 - Smaller data set may not trigger the problem
 - Does the bug even exist on smaller problems?
 - Didn't you already try the code at small scale?
 - Is it a 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
 - What can a parallel debugger do to help?
 - Debug at the scale of the problem. **Now.**

- Many benefits to graphical parallel debuggers
 - Large feature sets for common bugs
 - Richness of user interface and real control of processes
- Historically **all** parallel debuggers hit scale problems
 - Bottleneck at the frontend: Direct GUI → nodes architectures
 - Linear performance in number of processes
 - Human factors limit – mouse fatigue and brain overload
- Are tools ready for the task?
 - Allinea DDT has changed the game

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)

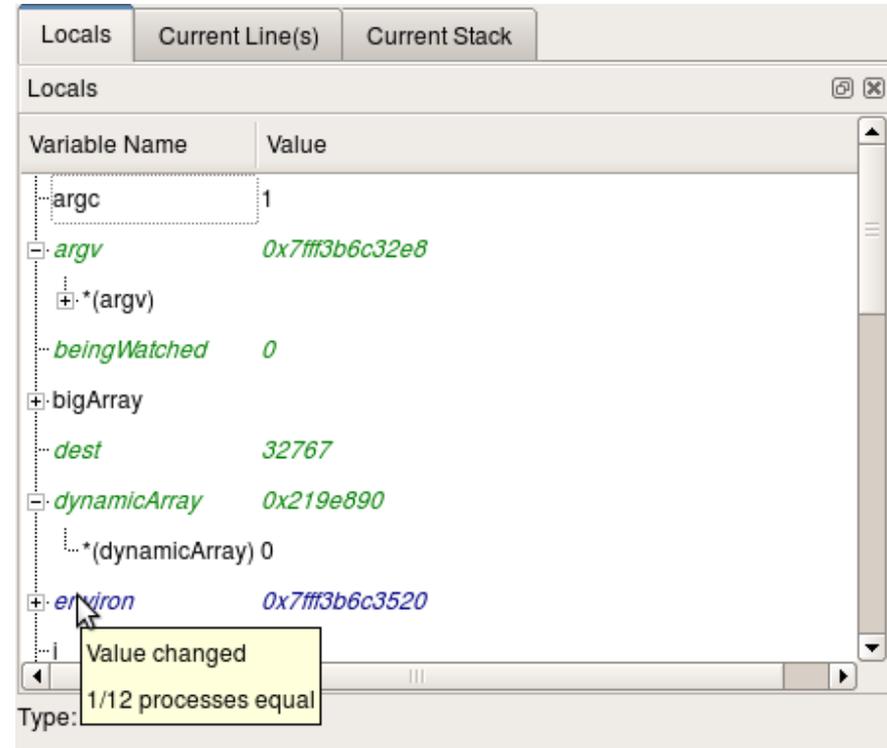
- Parallel Stack View
 - Finds rogue processes quickly
 - Identify classes of process behaviour
 - Rapid grouping of processes
- Control Processes by Groups
 - Set breakpoints, step, play, stop for groups
 - Scalable groups view: compact group display





- Immediate stop on crash
 - Segmentation fault, or other memory problems
 - Abort, exit, error handlers
 - CUDA errors
- Scalable handling of error messages
- Leaps to the problem
 - Source code highlighted
 - Affected processes shown
 - Process stacks displayed clearly in parallel

- Full class/structure browsing
 - Local variables and current line(s)
 - Show variables relevant to current position
 - Drag in the source code to see more
 - C, C++, F90: object members, static members and derived types
- Automatic comparison and change detection
 - Scalable and fast



Expression: `my_rank % 3`

Processes in current group (All, 189120 procs)

Limit comparison to s.f.

Filter:

Align stack frames

Value	Process(es)
0	0,3,6,9,12,15,18,21,24,27,30,...
1	1,4,7,10,13,16,19,22,25,28,31,...
2	2,5,8,11,14,17,20,23,26,29,32,...

Count: 189120

Filtered: 0

Errors: 0

Aggregate: 0

Numerical: 189120

Sum: 189120

Minimum: 0

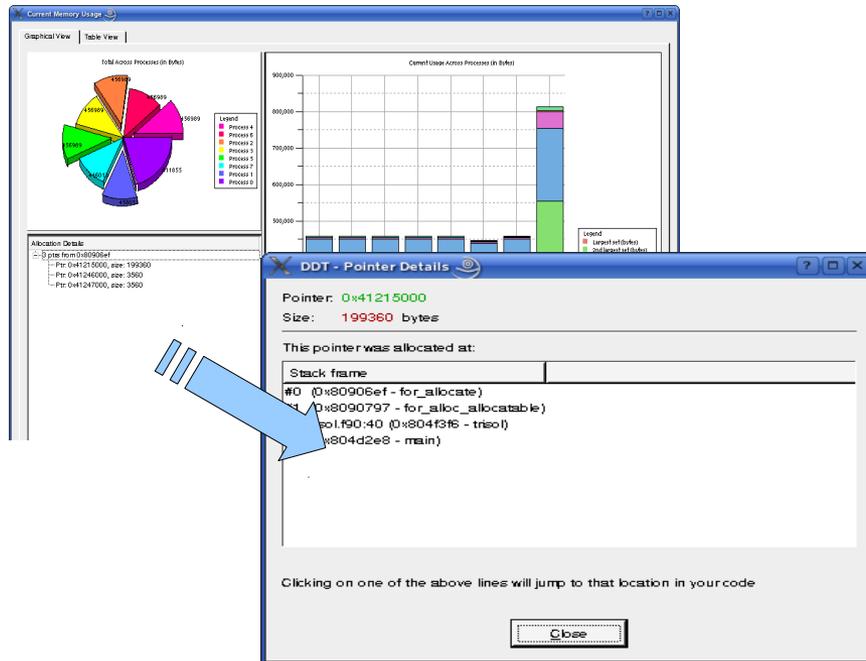
Maximum: 2

- Gathers data from every node and compares
 - Aggregated statistics
 - Shows max, min, count, averages, ...
 - Optimizal performance
 - Even in extreme differences cases
 - ~130ms total to compare scalar variable from 220,000 cores
 - Filter to find specific value ranges

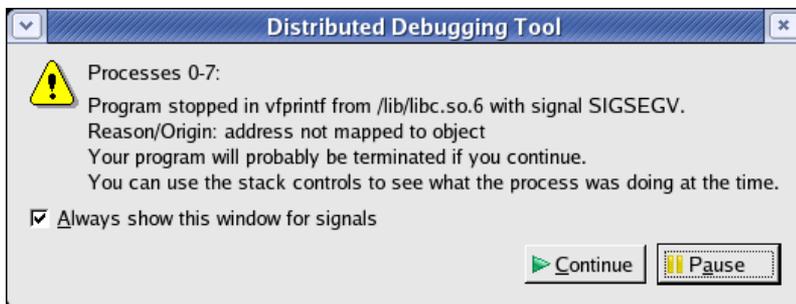
The screenshot shows the Allinea software interface for array configuration. The 'Array Expression' is set to 'bigArray[\$i]'. The 'Distributed Array Dimensions' is set to '1'. The 'Range of \$x (Distributed)' is set to 'From: 0' and 'To: 7'. The 'Range of \$i' is set to 'From: 0' and 'To: 9999'. The 'Display' options are 'Rows' and 'Columns'. The 'Auto-update' checkbox is unchecked. The 'Evaluate' and 'Cancel' buttons are visible. Below the configuration, there is a 'Data Table' tab and a 'Statistics' tab. The 'Data Table' tab is active, showing a table with columns labeled 'i' and values: 2444, 2733, 3011, 3185, 4704, 5343, 6795, 7881, 9108, 9467. The rows are labeled 'x 0' through 'x 6'. The table contains several '1' values, indicating data points.

	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 arrays from multiple processes
 - Search terabytes for rogue data - in parallel with [v3.0]

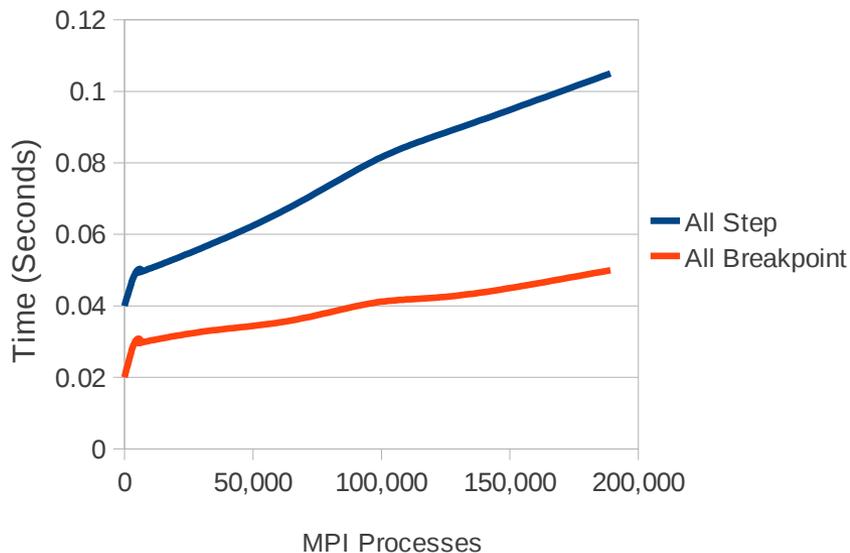


- Comprehensive memory debugging
 - Find memory leaks
 - Stop instantly on read/write beyond end of array
- Proven at scale
 - Where random memory bugs hurt!



DDT 3.0 Performance Figures

Jaguar XT5



- DDT delivers Petascale debugging **today**
 - Collaboration with ORNL on Jaguar Cray XT
- New tree architecture
 - The only logarithmic performance parallel debugger
 - Many operations faster at 220,000 than any other debugger at 1,000 cores
 - **~1/10th of a second¹** to step and gather all stacks at 220,000 cores

- Debuggers are recognized as the right tools to fix bugs quickly: other methods have limited success, and major issues at scale
- Debugging interfaces must scale to help the user understand what is happening
- Allinea DDT scales in performance and interface – breaking all records and making problems manageable

- The BG/P is a challenging platform for debugging at speed
 - Compute nodes won't support a debugger
 - IO nodes must look after the debugging for the compute nodes - all 256 with Intrepid! (512 at Juelich!)
 - .. Memory usage of debuggers needs to be low per process
 - .. Performance is hit - too much work and communication for one process
 - Allinea worked with ANL to improve DDT memory usage within each IO node: done!
 - DDT now supports up to 256 processes per IO node
 - ... across as many IO nodes as you need!

- Worked examples
 - Handout with worked examples and exercises
 - Handful of simple codes with some bugs
 - Listen out for where to find the code and the materials
- Hands on with own code
 - Once you're confident in using DDT for simple problems