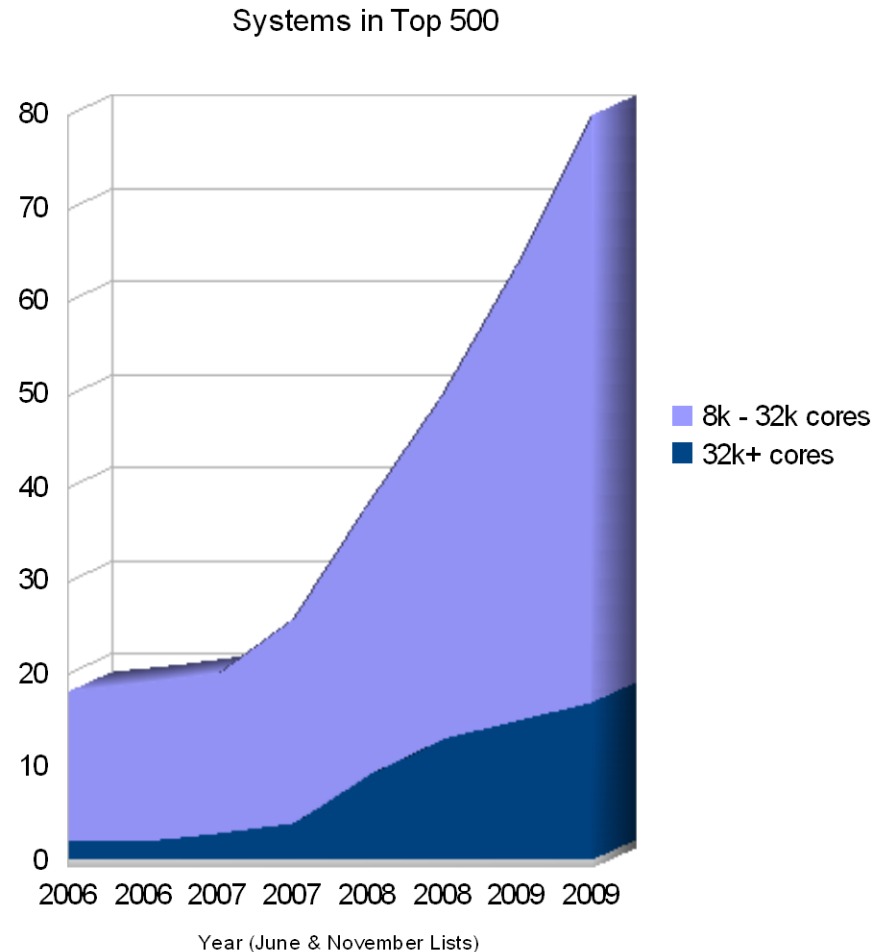




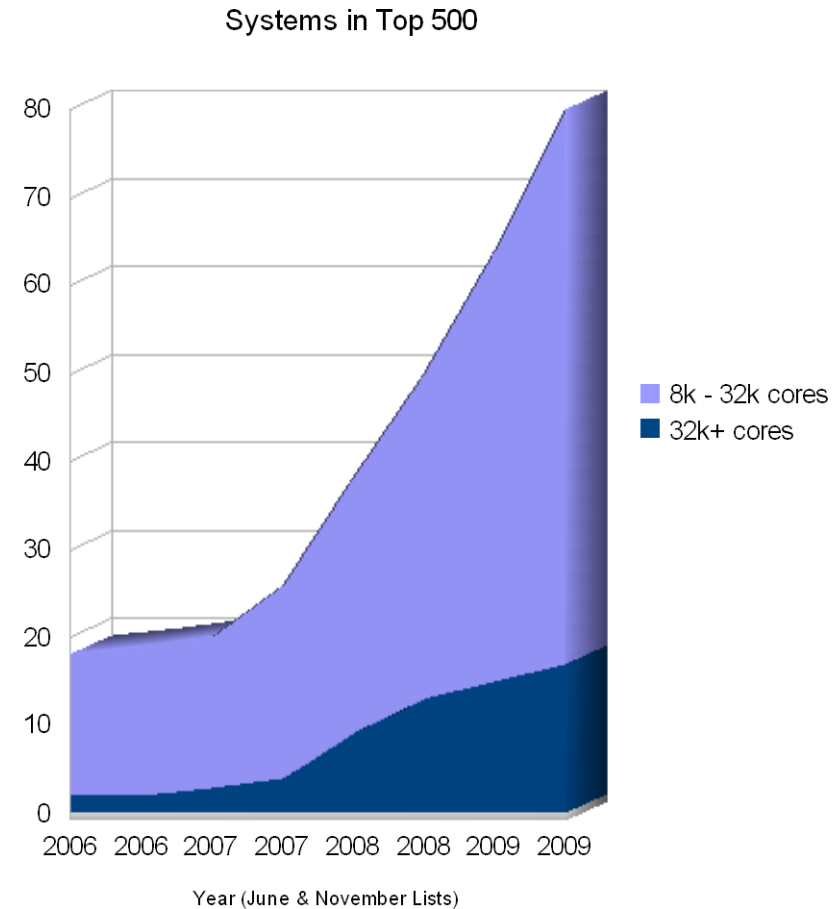
Petascale Debugging with Allinea DDT

HPC ADVISORY COUNCIL WORKSHOP 2010

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

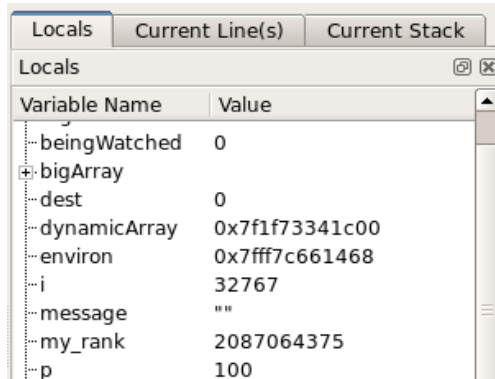


- Debuggability
 - A subjective measure of the ability to be debugged
- Linear tool architectures
 - Linear (or worse) bottlenecks
 - Pain threshold varies: 1 second, 1 minute, 1 hour?
- A major problem
 - Previously exclusive to big labs
 - Now everyone is joining in the fun



- Ignore the problem
 - Pretend bugs at scale do not happen
- Best programming practices
 - Consistency checking and self-diagnosis within code
 - Still frustrated by some types of bug
- Lightweight debugging
 - STAT (LLNL) identifies equivalent processes using stacks
 - STAT calls Allinea DDT (or TTV) to debug representatives
 - Other work is promising
- But what about full-strength debuggers?

- 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



- Scalar features

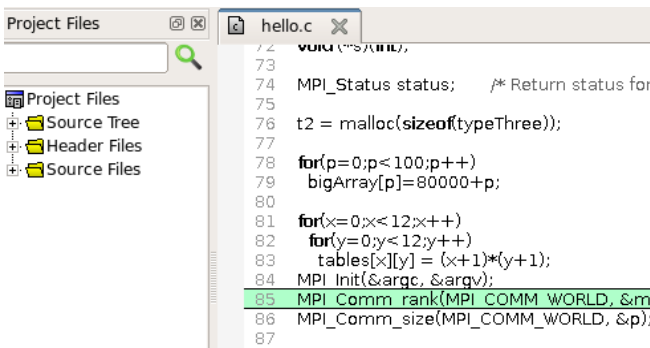
- Advanced C++ and STL
- Fortran 90, 95 and 2003: modules, allocatable data, pointers, derived types
- Memory debugging

- Multithreading & OpenMP features

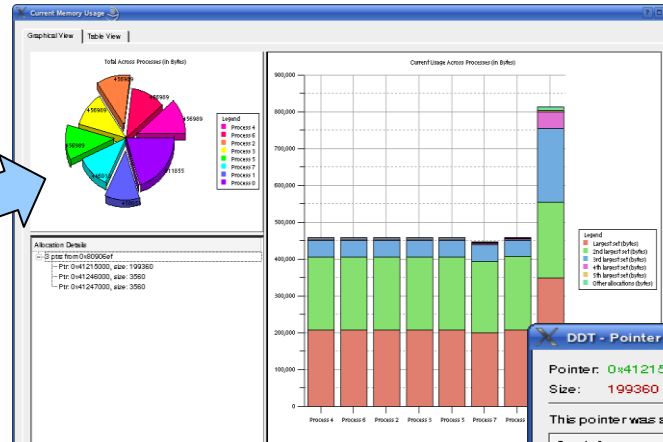
- Step, breakpoint etc. one or all threads

- MPI features

- Easy to manage groups
- Control processes by groups
- Compare data
- Visualize message queues



- Find memory leaks



DDT - Pointer Details

Pointer: 0x41215000
Size: 199360 bytes

This pointer was allocated at:

```
Stack frame
#0 (0x80906ef - for_allocate)
#1 (0x8090797 - for_alloc_allocatable)
#2 triso1.f90:40 (0x804f3f6 - triso1)
#3 (0x804d2e8 - main)
```

Clicking on one of the above lines will jump to that location in your code

Close

- Or stop on read/write beyond end of array

Distributed Debugging Tool

Processes 0-7:
Program stopped in vfprintf from /lib/libc.so.6 with signal SIGSEGV.
Reason/Origin: address not mapped to object
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.

Always show this window for signals

Continue Pause

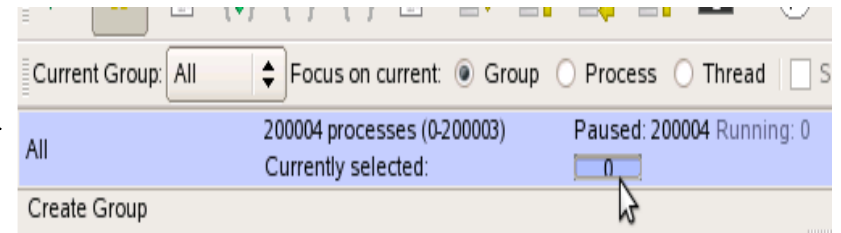
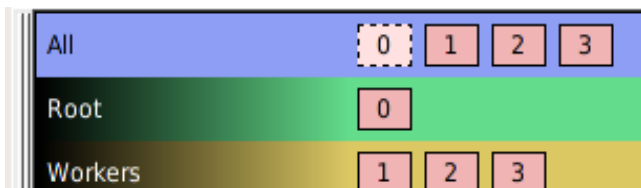
- Run the code
 - Browse source
 - Set breakpoints
 - Stop at a line of CUDA code
 - Stops once for each scheduled collection of blocks
- Select a CUDA thread
 - Examine variables and shared memory
 - Step a warp

```
18 {1, 2, 1},
19 {0, 0, 0},
20 {-1, -2, -1}
21 };
22
23 /// 2D convolution filter using global memory
24 __global__ void conv2d_global(uchar4* in, uchar4* out, int
25 {
26     int x = threadIdx.x + blockIdx.x * BLOCK_SIZE;
27     int y = threadIdx.y + blockIdx.y * BLOCK_SIZE;
28     int index = (y * width + x);
29
30     float4 output = make_float4(0.0f, 0.0f, 0.0f, in[index].w
31
32     #pragma unroll
33     for(int cy=-1; cy<=1; ++cy)
34     {
35         #pragma unroll
36         for(int cx=-1; cx<=1; ++cx)
37     {
```

Threads	Function
1	main (edge.cu:75)
32	conv2d_global (edge.cu:35)
480	conv2d_global (edge.cu:39)

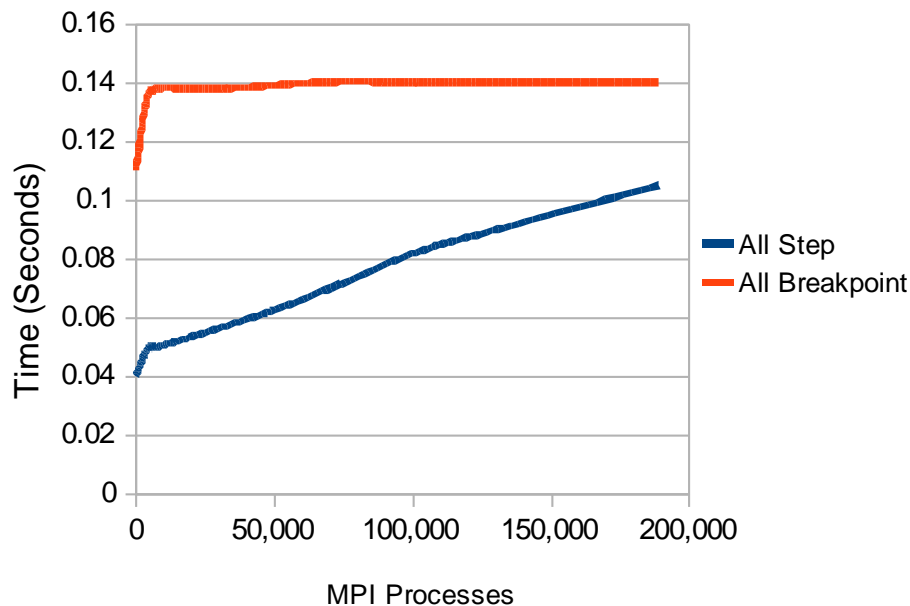
Stacks (All)	
Processes	Function
150120	_start
150120	__libc_start_main
150120	main
150120	pop (POP.f90:81)
150120	initialize_pop (initial.f90:119)
150120	init_communicate (communicate.f90:87)
150119	create_ocn_communicator (communicate.f90:300)
1	create_ocn_communicator (communicate.f90:303)

- Parallel Stack View
 - Finds rogue processes faster
 - Identifies classes of process behaviour
 - Allows rapid grouping of processes
- Control Processes by Groups
 - Set breakpoints, step, play, stop etc. using user-defined groups
 - Mutates to scalable groups view
 - Compact group representations

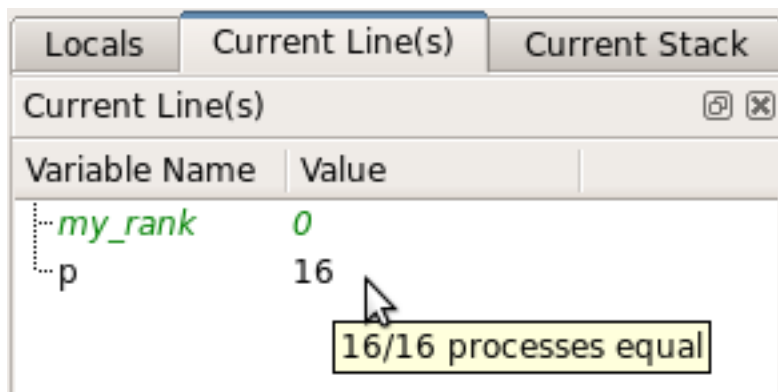


DDT 3.0 Performance Figures

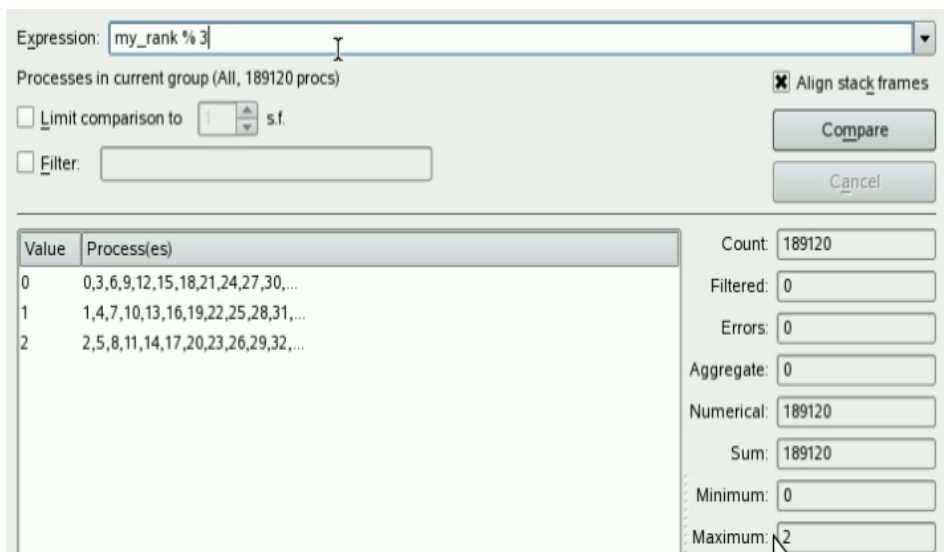
Jaguar XT5



- Allinea DDT is delivering petascale debugging **today**
 - Collaboration with ORNL on Jaguar Cray XT
 - Tree architecture – logarithmic performance
 - Many operations now faster at 220,000 than previously at 1,000 cores
 - **~1/10th of a second** to step and gather all stacks at 220,000 cores

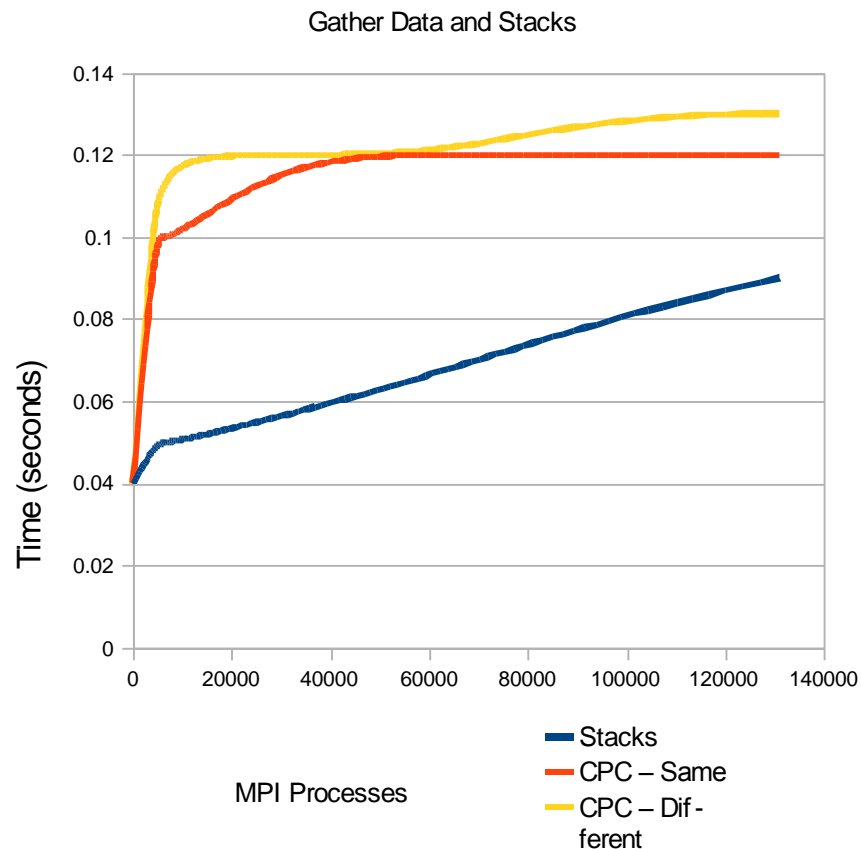


- Gather from every node
 - Potentially costly – if all data different
 - Easy if data mostly same
 - New ideas
 - Aggregated statistics
 - Probabilistic algorithms optimize performance – even in pathological case



- Watch this space!
 - With a fast and scalable architecture, new things become possible

- Benchmarked on five codes on Jaguar XT
 - Stacks gathering mileage can vary: default install at ORNL has full debug info deep into MPI
 - Cross Process Comparison
 - Of equal variable
 - Of MPI rank (a bad case!)



- Most features now scale
 - Attach, run, process control and breakpoints
 - Process stacks
 - Data comparison
 - Memory debugging – out-of-bound array access, leaks, etc.
 - Import/export – stacks (XML/CSV), arrays, compared data
 - Tested at 220k cores on XT; 8k on Blue Gene P (SMP mode) – more timings soon; Ranger (Linux IB cluster)
 - New distributed array features
 - New grow/shrink attached-set - in addition to existing subset capabilities

- Lessons learnt
 - The scalable tree has really delivered!
 - More optimizations still possible
 - Even if you're quick, it's still all about the GUI
 - Present sensibly to the user – parallel stacks, data comparison
 - ... but some machines don't encourage full power of debugging due to their architecture
 - MPI spec probably never meant debuggers to scale!
 - Still linear things in there.. eg. MPIR_proctable
 - It's hard to debug a debugger without a debugger

- Logarithmic performance should last for many years
 - Any linear factors will eventually dominate
 - Must eradicate them all over time
 - Any memory usage on per-process basis
 - More intelligence can be pushed down the tree as need arises
 - Predict core operations on 1M or 10M cores will be under the pain threshold
 - SIMD/almost-SIMD GPUs fit within current approach (as threads, not individual processes)
- ... but bugs can still be hard to find

- Collaboration opportunity
 - No single organization has the resources to do everything
 - Plenty of opportunity for everyone in debugging
 - We use tools independently – but using together is more compelling
 - Examples:
 - MPI correctness checking – Marmot, Intel MPI Checker
 - Library specific sanity checkers for data
 - Comparative debugging
 - Ideal scenario: easy to prototype new bug finding ideas
 - Not tied to a particular product – but tied to an open API/scripting language
 - Single process or built from the top (drive a full debugger, or eg. combination of Wisconsin tools)

**Come and see us at ISC10
Booth #745**

Thank you