allinea

High performance tools to debug, profile, and analyze your applications

Debugging and Profiling your HPC Applications

David Lecomber, CEO and Co-founder

david@allinea.com

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

About this talk

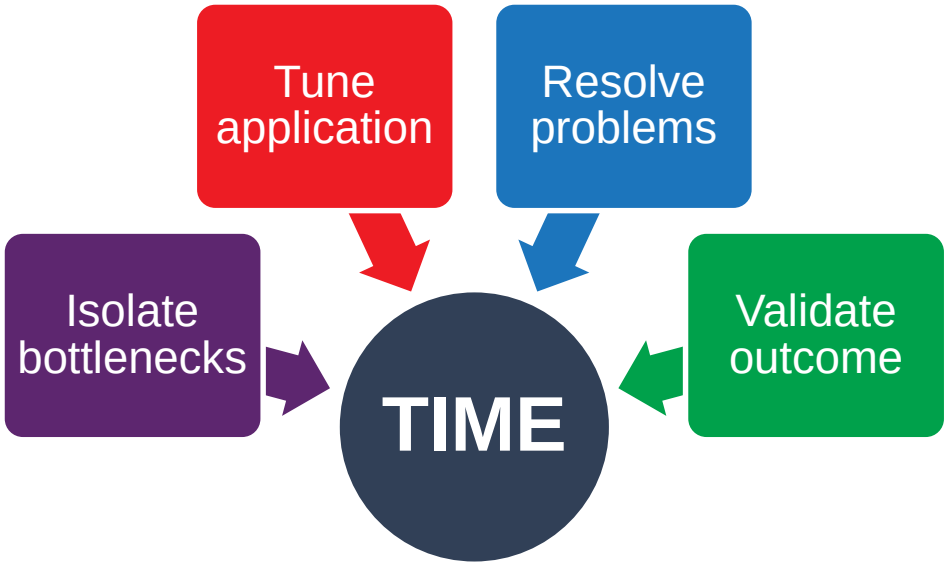
- Learn how to debug and profile your code
  - Techniques to take home
- Tools we will use: Allinea Forge
  - Debugging with Allinea DDT
  - Profiling with Allinea MAP
  - NB. Allinea MAP is not supported on BG/Q
- Where to find Allinea's tools
  - > 70% of Top 500 have at least one Allinea tool

Motivation

- HPC systems are finite
  - Limited lifetime to achieve most science possible
  - Sharing a precious resource means your limited allocation needs to be used well
- Your time is finite
  - PhD to submit
  - Project to complete
  - Paper to write
  - Career to develop
- Doing good things with HPC means creating better software, faster
  - Unrivaled productive and easy-to-use development environment…
  - … To help reach the highest level of performance and scalability
- High performance parallel code needs tools designed for the challenge

allinea

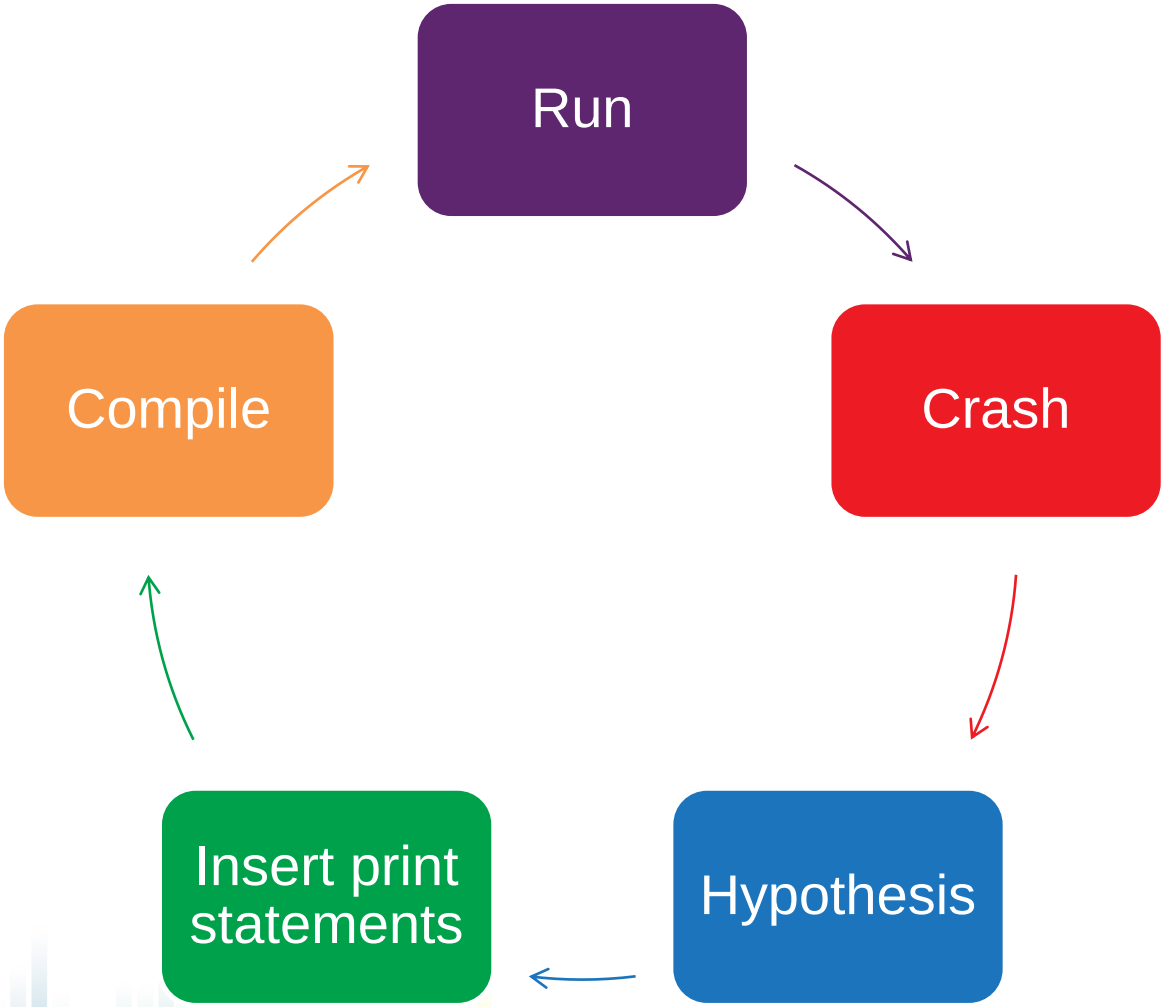
Use the right software tool to be faster



- What parts of the code would benefit most from being rewritten?
- How should I modify a code to make it better (or work at all)?

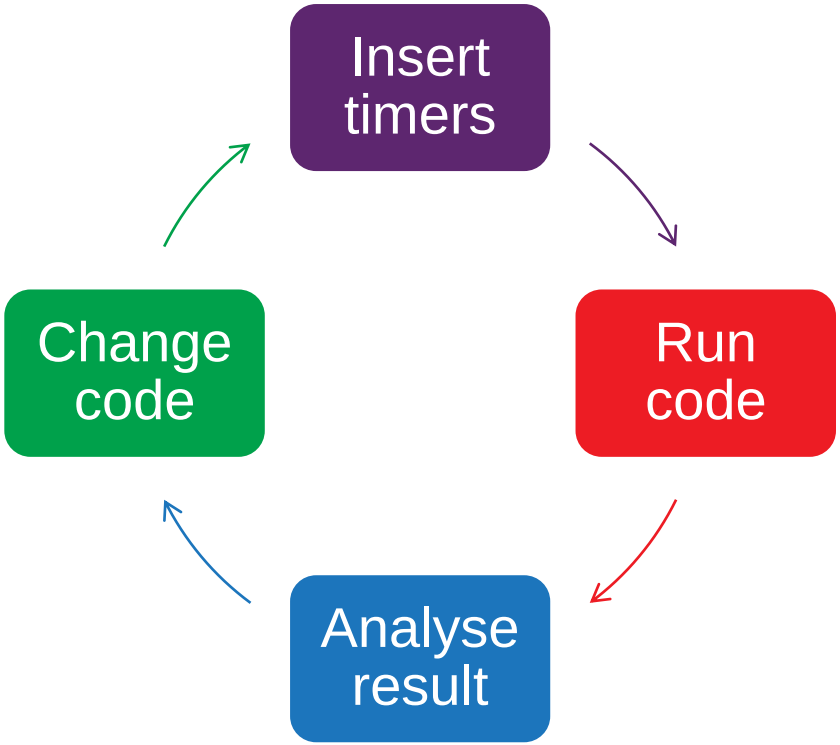
allinea

Debugging in practice…



allinea

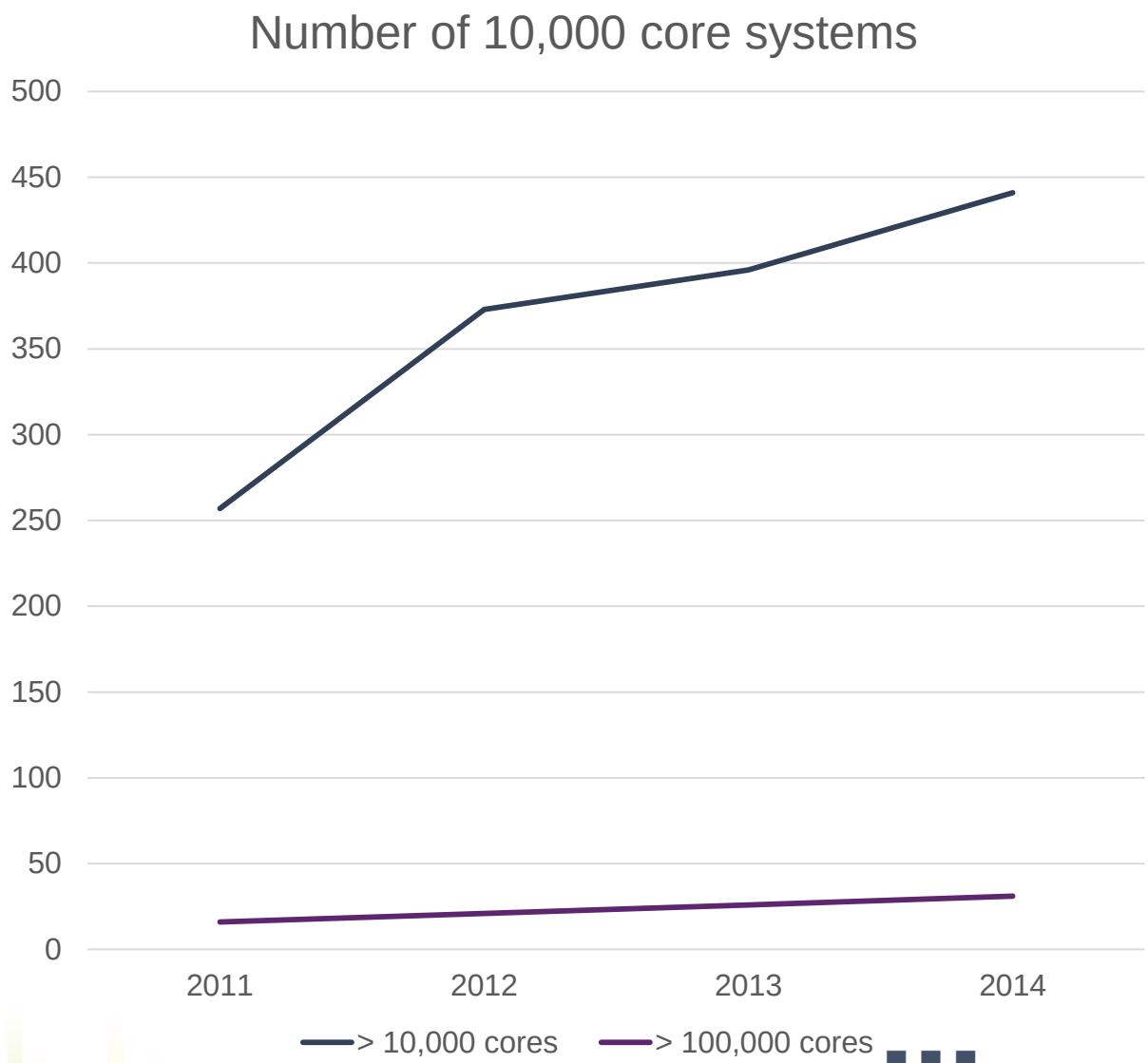
Optimization in Practice



Motivation

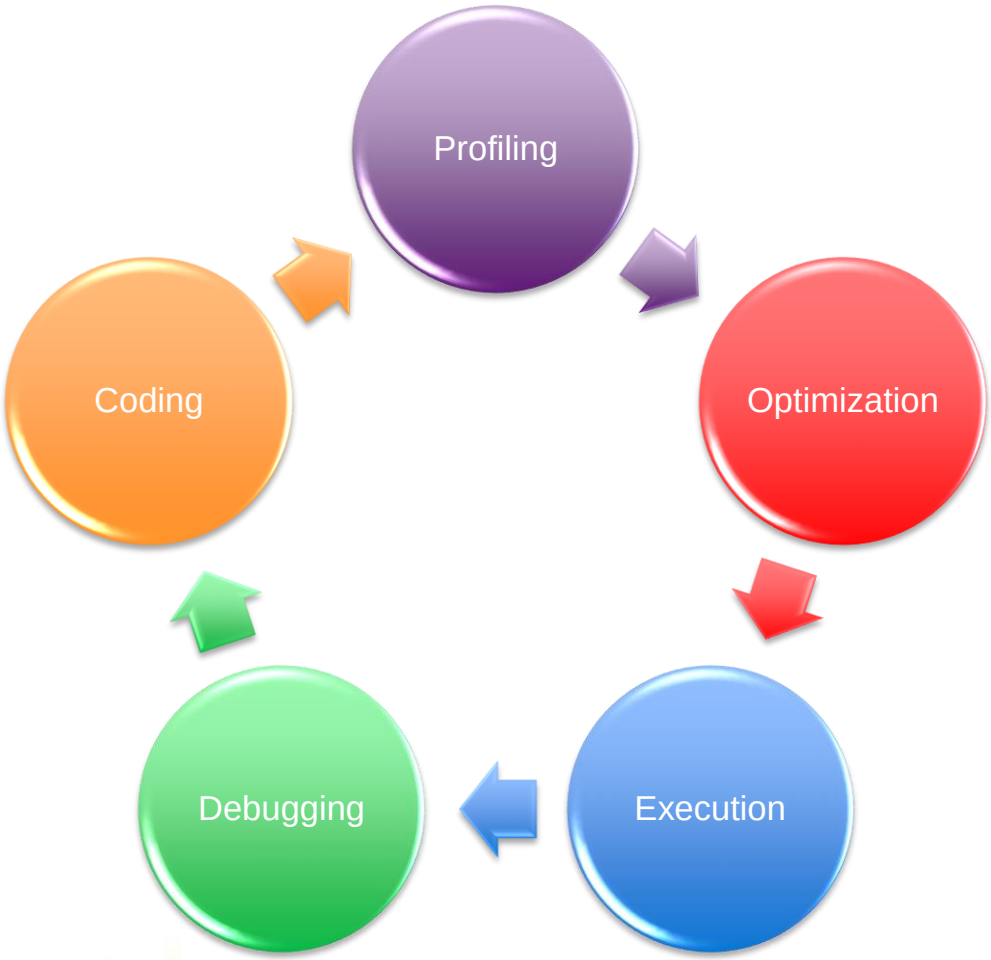
- "Without capable highly parallel software, large supercomputers are less useful"
  - Council on Competitiveness

- "1% of HPC application codes can exploit 10,000 cores"
  - IDC, 2011



allinea

Application Development Workflow



allinea

Hello Allinea Forge!

Allinea MAP to find performance bottleneck

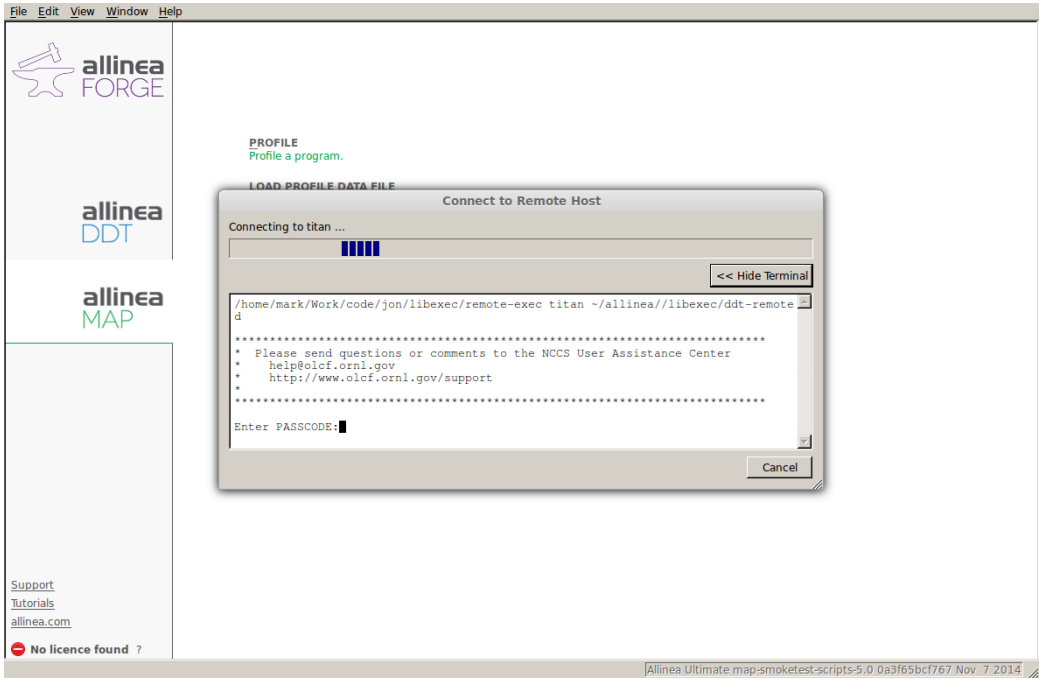
Increasing memory usage? *Memory leak!*
Workload imbalance? *Possible partitioner bug!*

Flick to Allinea DDT
Common interface and settings files

Observe and debug your code step by step

HPC means being productive on remote machines

- Linux
- OS/X
- Windows
- Multiple hop SSH
- RSA + Cryptocard
- Uses server license



Profiling for performance

- Code optimisation can be time-consuming…

  – (image courtesy of xkcd.com)

6 steps to improve performance

Get a realistic test case

- Performance on real data matters
- Keep the test case for reference and re-use

Profile your code

- Add “-g” flag to your compilation
- Run “map –profile mpirun –np 4 application.exe”

Look for the significant

- Which part/phase of the code dominates time?
- Is there any unexpected significant time use?

What is the nature of the problem?

- Compute? I/O? MPI? Thread synchronization?
- Display the metrics that show the problem best

Apply brain to solve

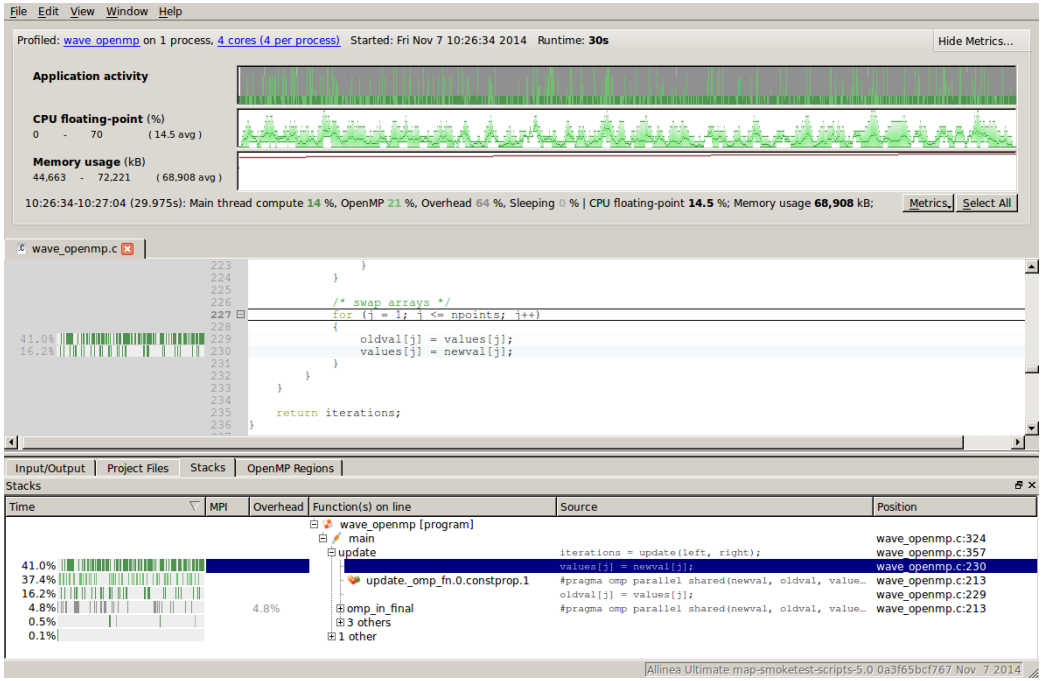
- MPI – can you balance the work better?
- Compute – is memory time dominant – can you improve layout?

Think of the future

- Try larger process or thread counts to watch for scalability problems
- Keep the profile (.map file) for future comparison

MAP in a nutshell

- Small data files
- <5% slowdown
- No instrumentation
- No recompilation



How Allinea MAP is different

Adaptive sampling	Sample frequency decreases over time	Data never grows too much	**Run for as long as you want**
Scalable	Same scalable infrastructure as Allinea DDT	Merges sample data at end of job	**Handles very high core counts, fast**
Instruction analysis	Categorizes instructions sampled	Knows where processor spends time	**Shows vectorization and memory bandwidth**
Thread profiling	Core-time not thread-time profiling	Identifies lost compute time	**Detects OpenMP issues**
Integrated	Part of Forge tool suite	Zoom and drill into profile	**Profiling within your code**

Above all…

- Aimed at any performance problem that matters
  - MAP focuses on time
- Does not prejudge the problem
  - Doesn't assume it's MPI messages, threads or I/O
- If there's a problem..
  - MAP shows you it, next to your code

Scaling issue – 512 processes

hemelb_512p_2014-02-08_10-18.map - Allinea MAP 4.2-34164 [Trial Version]

File View Search Window Help

Profiled: hemelb on 512 processes Started: Sat Feb 8 10:18:36 2014 Runtime: **308s** Time in MPI: **58%**

Hide Metrics...

Memory usage (M) 21.8 - 152.6 (108.3 avg)

MPI call duration (ms) 0 - 11,789.1 (65.7 avg)

CPU floating-point (%) 0 - 100 (11 avg)

10:18:36-10:23:43 (307.730s): Mean: Memory usage **108.3** M; MPI call duration **65.7** ms; CPU floating-point **10.6** %;

Metrics... Reset

StepManager.h | IteratedAction.cc | StepManager.cc

```
56          std::string name;
57          Action(Concern &concern, MethodLabel method) :
```

Input/Output | Project Files | Parallel Stack View

Parallel Stack View

Time	MPI	Function(s) on line	Source
1.4%	0.1%	hemelb::net::IteratedActi...	RequestComms();
1.5%	1.3%	12 others	
		Call [inlined]	CallActionsForStep(static_cast<steps::Step>(step), 0);
		hemelb::net::IteratedActi...	return concern->CallAction(method);
		hemelb::extraction::Pro...	EndIteration();
		hemelb::extraction::Pr...	propertyWriter->Write(simulationState.GetTimeStep());
		hemelb::extraction::...	localPropertyOutputs[outputNumber]->Write((uint64_t) iterationNumber);
34.3%	34.3%	PMPI_File_write_at	MPI_STATUS_IGNORE);
<0.1%	<0.1%	20 others	
<0.1%		4 others	
<0.1%	<0.1%	12 others	
<0.1%		3 others	
0.2%	0.2%	10 others	

Showing data from 512000 samples taken over 512 processes (1000 per process)

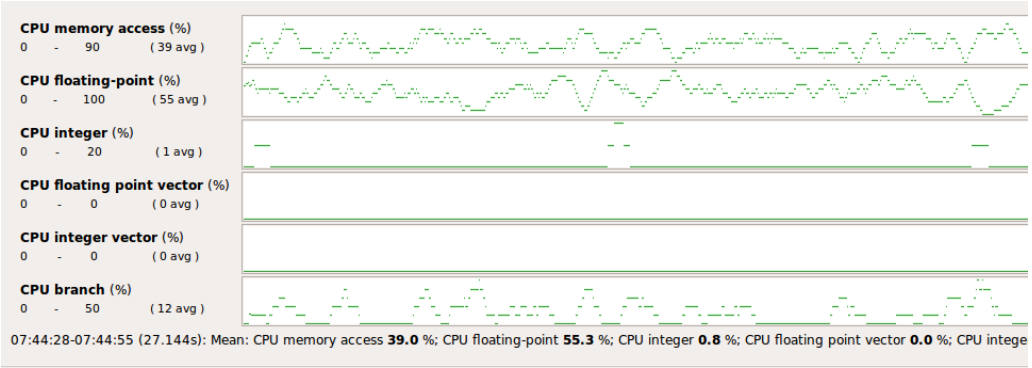
Time	MPI	Function(s) on line	Source	Position
1.4%	0.1%	hemelb::net::IteratedActi...	RequestComms();	IteratedAction.cc:27
1.5%	1.3%	12 others		
		Call [inlined]	CallActionsForStep(static_cast<steps::Step>(step), 0);	StepManager.cc:127
		hemelb::net::IteratedActi...	return concern->CallAction(method);	StepManager.h:67
		hemelb::extraction::Pro...	EndIteration();	IteratedAction.cc:39
		hemelb::extraction::Pr...	propertyWriter->Write(simulationState.GetTimeStep());	PropertyActor.cc:90
		hemelb::extraction::...	localPropertyOutputs[outputNumber]->Write((uint64_t) iterationNumber);	PropertyWriter.cc:42
34.3%	34.3%	PMPI_File_write_at	MPI_STATUS_IGNORE);	LocalPropertyOutput.cc:302
<0.1%	<0.1%	20 others		
<0.1%		4 others		
<0.1%	<0.1%	12 others		
<0.1%		3 others		
0.2%	0.2%	10 others		

Showing data from 512000 samples taken over 512 processes (1000 per process) Allinea MAP 4.2-34164

Simple fix… reduce periodicity of output

Deeper insight into CPU usage

- Runtime of application still unusually slow



- Allinea MAP identifies vectorization close to zero

- Why? Time to switch to a debugger!

Some types of bug

Bohrbug	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

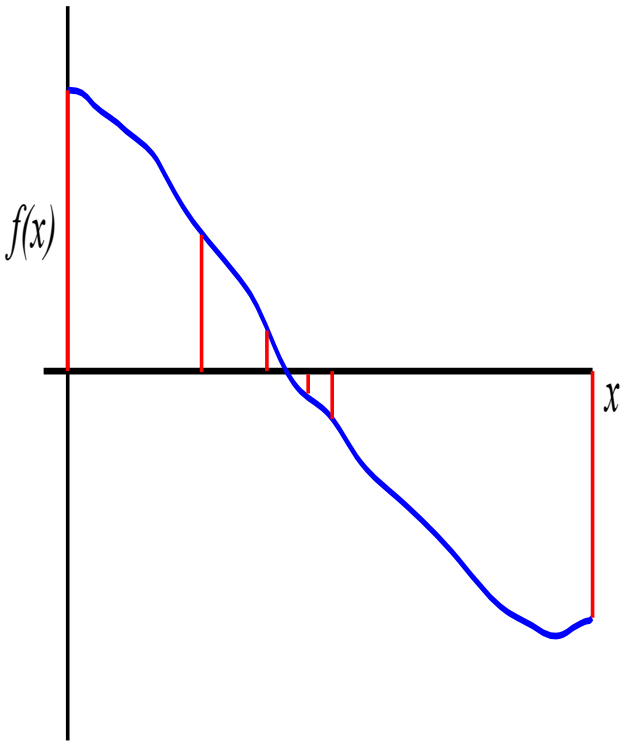
Debugging

- 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

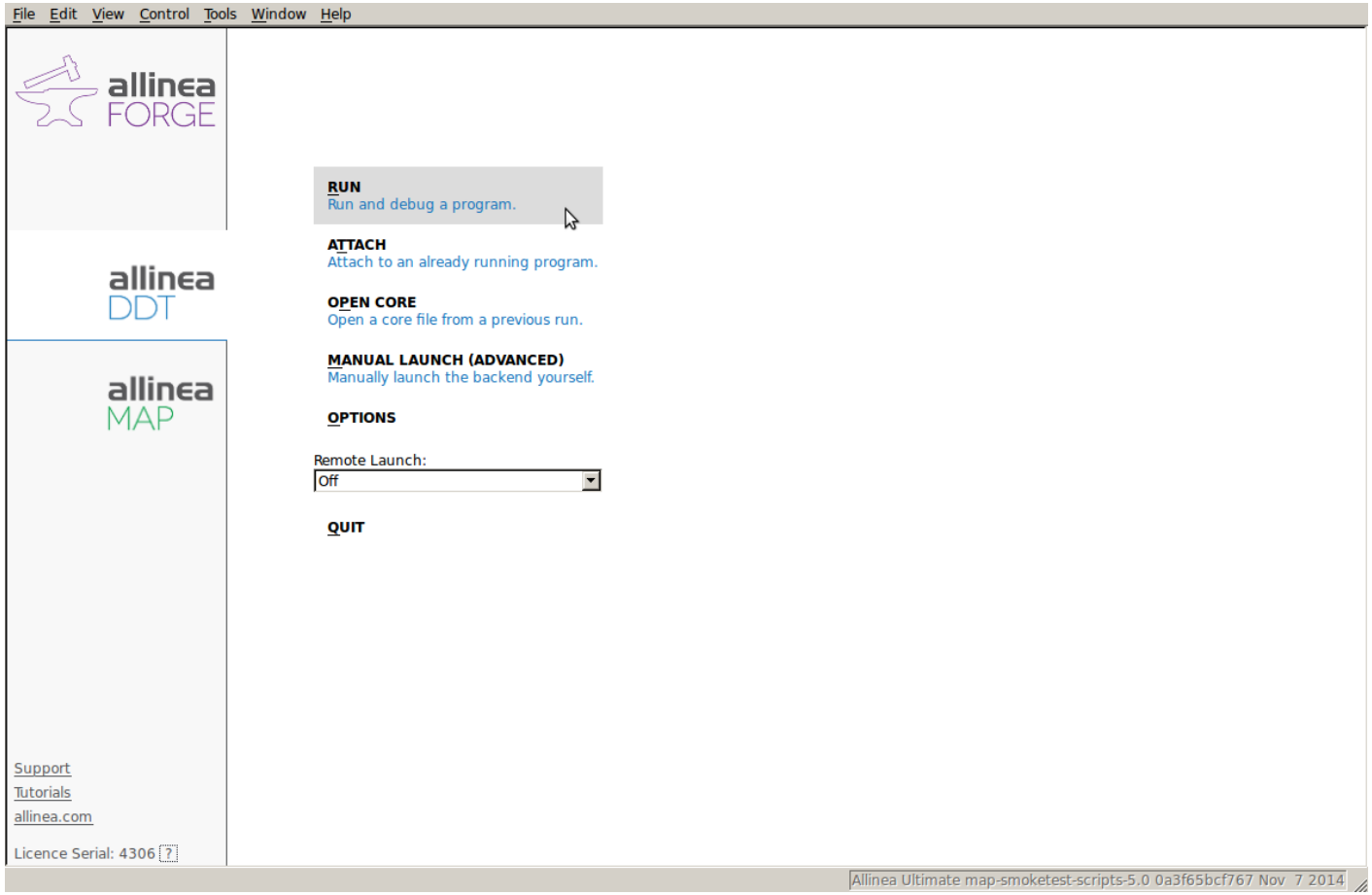
- Suggested Reading:
  - Zeller A., “Why Programs Fail”, 2nd Edition, 2009
  - Zen and the Art of Motorcycle Maintenance, Robert M. Pirsig

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
- Broken at modest scale
  - Too much output – too many log files



While still connected to the server we switch to the debugger



It’s already configured to reproduce the profiling run

File Edit View Control Tools Window Help

allinea FORGE

allinea DDT

allinea MAP

RUN
Run and debug a program.

ATTACH
Attach to an already running program.

OPEN CORE
Open a core file from a previous run.

MANUAL LAUNCH (ADVANCED)
Manually launch the backend yourself.

OPTIONS

Remote Launch:
Off

QUIT

Support
Tutorials
allinea.com

Licence Serial: 4306 ?

Run

Application: /home/mark/wave_openmp/wave_openmp — Details

Application: /home/mark/wave_openmp/wave_openmp

Arguments:

stdin file: /home/mark/Work/r/bench.R

Working Directory:

MPI: 1 process, Open MPI (Compatibility) — Details

Number of MPC Tasks: 1

Number of Processes: 1

Processes per Node 1

Implementation: Open MPI (Compatibility) Change...

mpirun Arguments:

OpenMP: 4 threads — Details

Number of OpenMP threads: 4

CUDA — Details

Memory Debugging — Details...

Submit to Queue — Configure... Parameters...

Environment Variables: none — Details

Plugins: none — Details

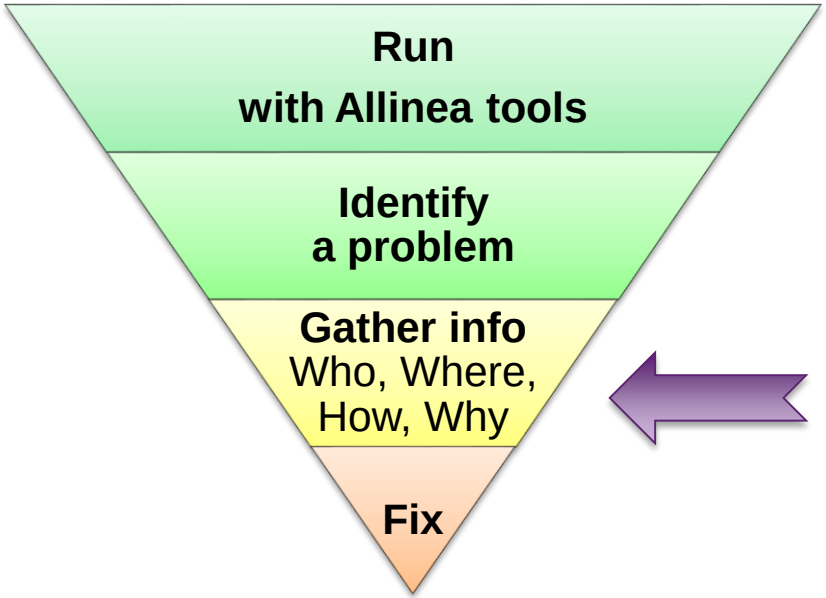
Help Options Run Cancel

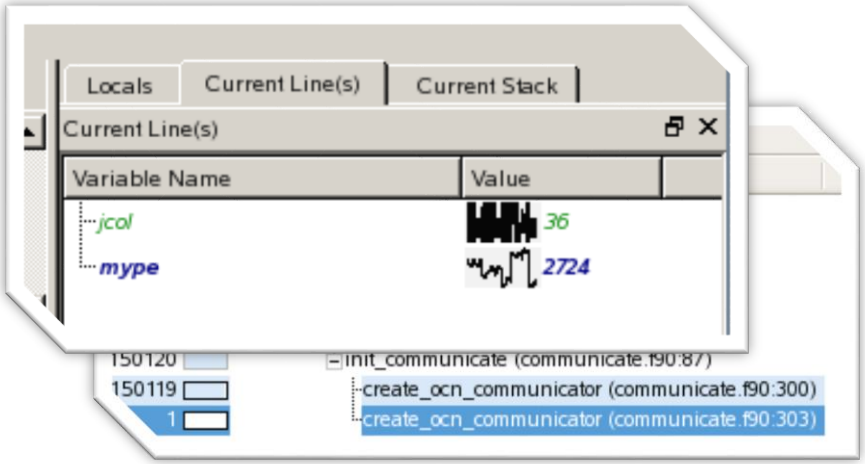
Allinea Ultimate map-smoketest-scripts-5.0 0a3f65bcf767 Nov 7 2014

allinea

Solving Software Defects

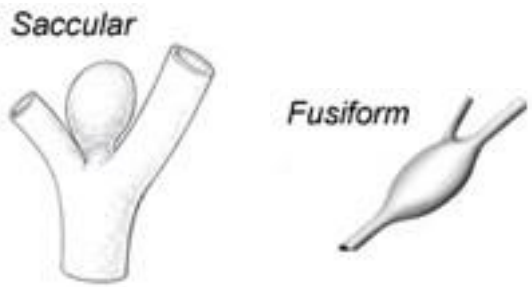
- Who had a rogue behavior ?
  - Merges stacks from processes and threads
- Where did it happen?
  - leaps to source
- How did it happen?
  - Diagnostic messages
  - Some faults evident instantly from source
- Why did it happen?
  - Unique “Smart Highlighting”
  - Sparklines comparing data across processes

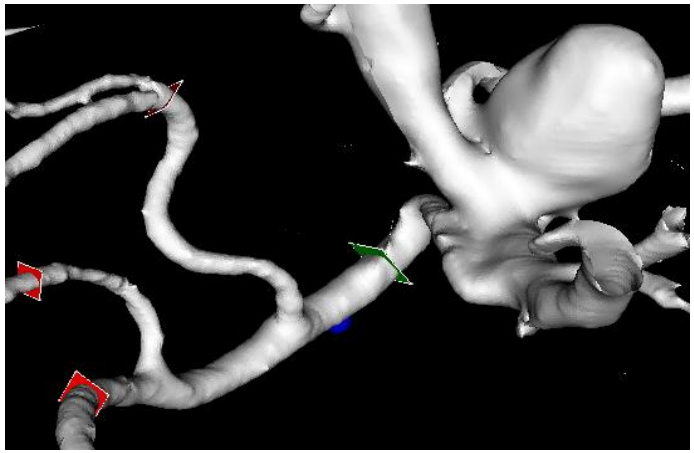




HPC could be brain surgery

- Brain aneurysms
  - 2-5% of population – most are undiagnosed
  - 30,000 rupture in US each year – 40% fatal
  - Early discovery and treatment increases survival rates
- Neurosurgery as HPC
  - MRI provides the blood vessel structure
  - Intra-cranial blood flow and pressures is just complex CFD
  - Full brain 3D model is 2-10GB geometry
- Individualized HPC
  - Patient's MRI scan enables surgical decision: whether to operate, how to operate, …
  - Circle of Willis requires super-Petascale ~~machine~~ software
  - Need answer in minutes or hours
- … but it crashes at 49152 cores





Allinea DDT 4.2.1-36484

File View Control Search Tools Window Help

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

All | 24576 processes (0-24575) | Paused: 17223 Playing: 7353 Finished: 0

Currently selected: 260 (on nid00194, pid 9481, main thread IWP 9481)

Create Group

Project Files

Search (Ctrl+K)

- VolumeTrav
- wave.c
- weird.c
- WholeGeom
- Writer.cc
- wspace.c
- XdrFileWrite
- XdrMemRea
- XdrMemWrit
- XdrReader.c
- XdrWriter.cc
- XmlAbstract
- xyzpart.c
- External Code

MpiEnvironment.cc | xyzpart.c

```
551   ikvsortii(ntsamples, allpicks);
552
553
554   /* Select the final splitters. Set the boundaries to s
555   for (i=1; i<npes; i++)
556     mypicks[i] = allpicks[i*ntsamples/npes];
557   mypicks[0].key    = IDX_MIN;
558   mypicks[npes].key = IDX_MAX;
559
560
561   WCOREPOP;  /* free allpicks */
562
563 STOPTIMER(ctrl, ctrl->AuxTmr2);
564 STARTTIMER(ctrl, ctrl->AuxTmr3);
565
```

Locals | Current Line(s) | Current Stack

Current Line(s)

Variable Name	Value
allpicks	0x2aab8055e010
i	2245
mypicks	0x2a6f8f0
npes	24575
ntsamples	1818550

Type: none selected

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

Stacks

Processes	Threads	Function
17223	17223	main (main.cc:37)
17223	17223	SimulationMaster::SimulationMaster (SimulationMaster.cc:63)
17223	17223	SimulationMaster::Initialise (SimulationMaster.cc:154)
17223	17223	hemelb::geometry::GeometryReader::LoadAndDecompose (GeometryReader.cc:188)
17223	17223	hemelb::geometry::GeometryReader::OptimiseDomainDecomposition (GeometryReader.c
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::OptimisedDecomposition
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::CallParmetis (Optimisec
17223	17223	ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223	17223	libparmetis__Coordinate_Partition (xyzpart.c:58)
17223	17223	libparmetis__PseudoSampleSort (xyzpart.c:556)

Evaluate

Expression	Value
i * ntsamples	-212322546

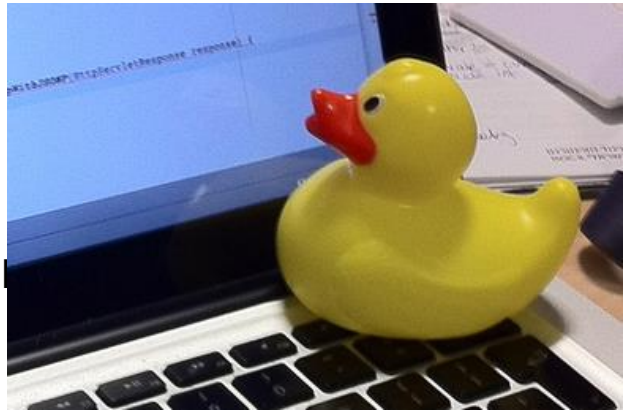
Type: int
Range: from -2147259746 to -12282046
49/17223 processes equal

7353 processes playing

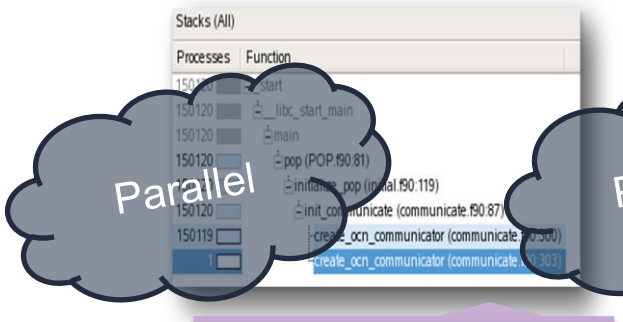
Computer | titan-ext7 | Allinea DDT 4.2.1-36484 | Sun Aug 10, 7:50 PM

Debugging by Inspiration

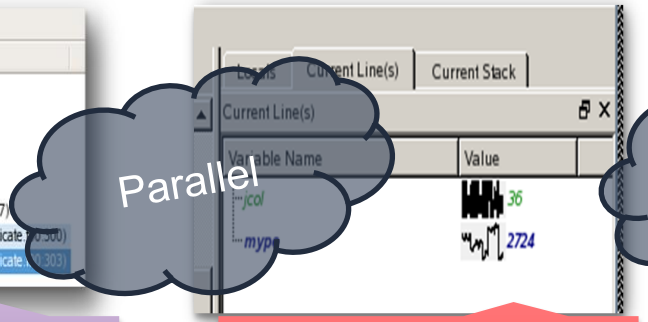
- Some errors are harder than others to diagnose
  - A bug that occurs 1% of the time is many times harder to fix than one that occurs 100% of the time
  - Often caused by incorrect memory usage
- Get help from Allinea DDT's memory debugging
  - Checks double frees
  - Checks use of dangling (freed) pointers
  - Can force O/S to check for read/write beyond bounds
    - Make some random bugs deterministic and occur 100% of the time ("Guard Pages")
  - showing memory leaks (group by allocation point)
- Other tips – get inspiration from a colleague
  - try explaining your code
    - The very act of explaining your thinking in the code can
  - don't have a colleague to hand?
    - Follow http://www.rubberduckdebugging.com



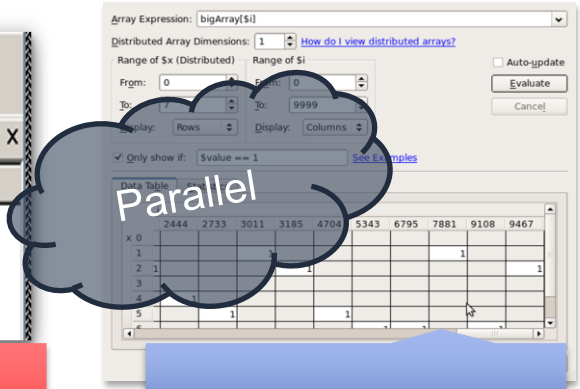
Favorite Allinea DDT Features for Scale



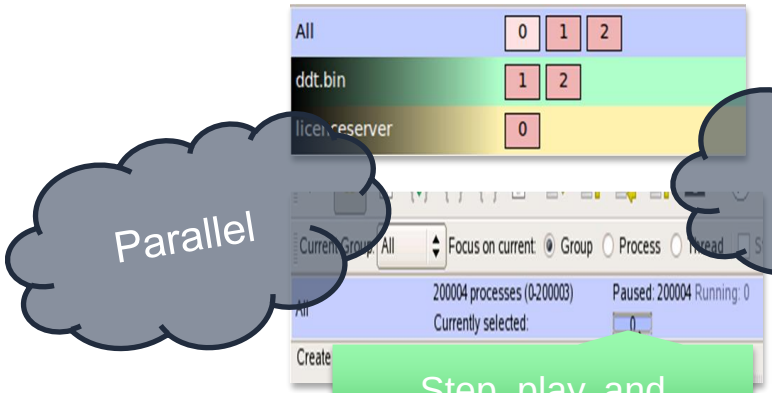
Parallel stack view



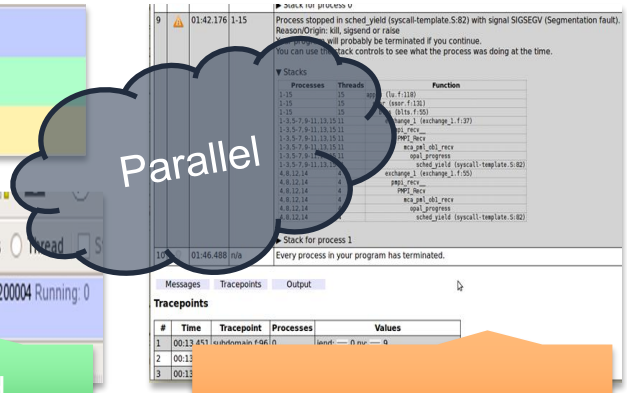
Automated data comparison: sparklines



Parallel array searching



Step, play, and breakpoints

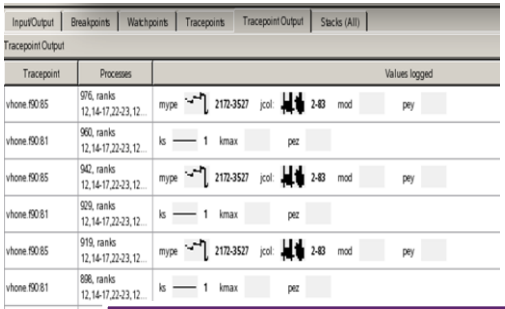


Offline debugging

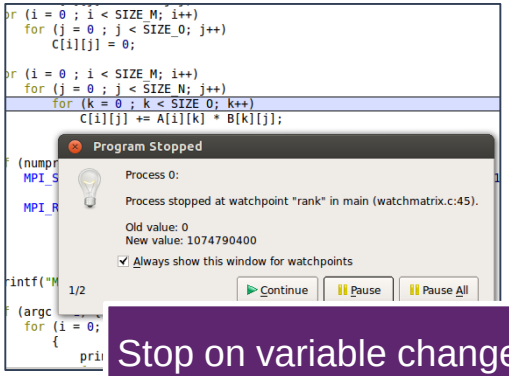
Today's Status on Scalability

- Debugging and profiling
  - Active users at 100,000+ cores debugging
  - 50,000 cores is largest profiling tried to date (and was Very successful)
  - … and active users with just 1 process too
- Deployed on
  - ORNL's Titan, NCSA Blue Waters, ANL Mira etc.
  - Hundreds of much smaller systems – academic, research, oil and gas, genomics, etc.
- Tools help the full range of programmer ambition
  - Very small slow down with either tool (< 5%)

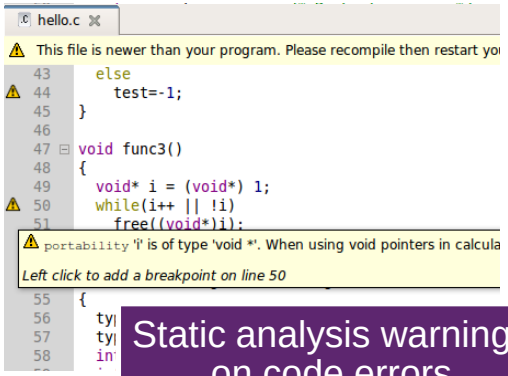
Five great things to try with Allinea DDT



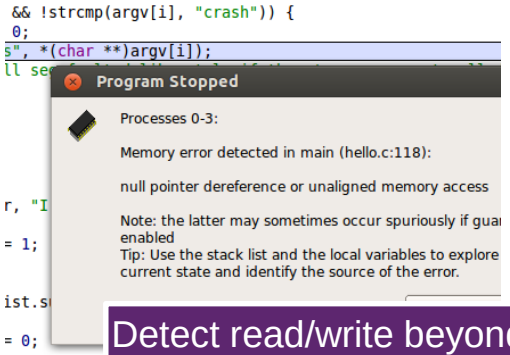
The scalable print alternative



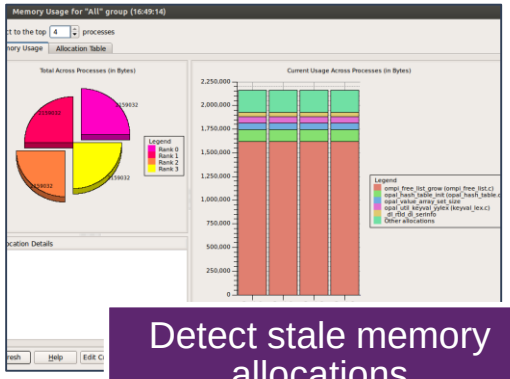
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds

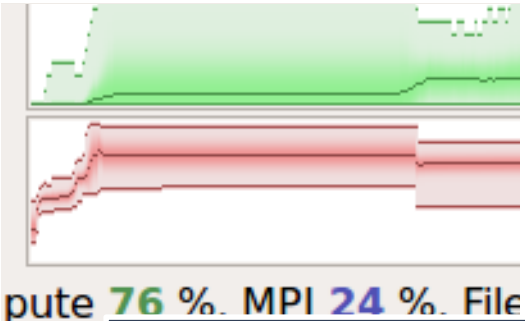


Detect stale memory allocations

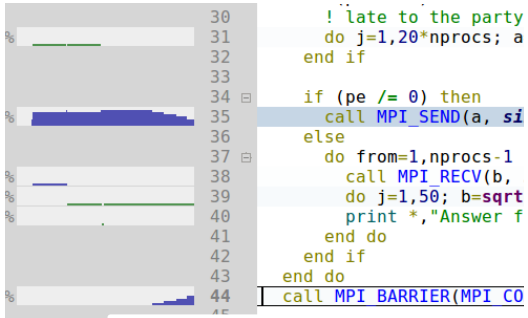
allinea MAP

Six Great Things to Try with Allinea MAP

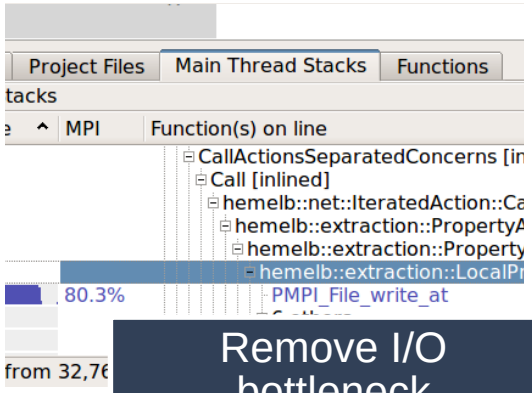
allinea FORGE



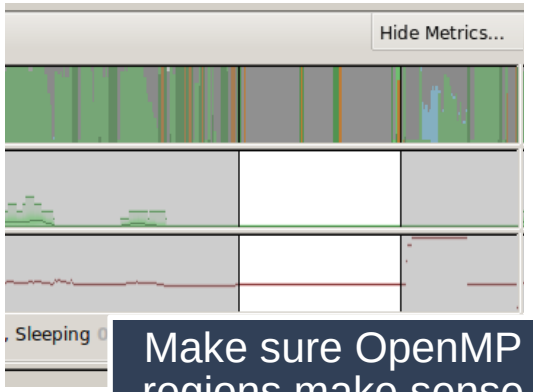
Find the peak memory use



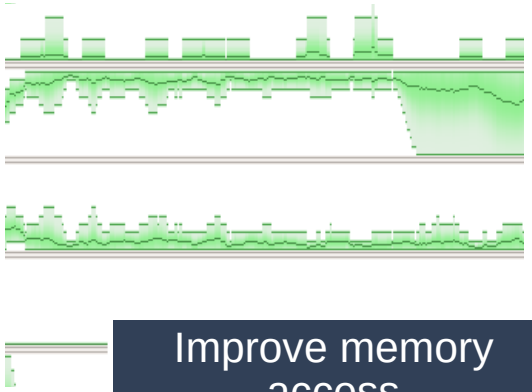
Fix an MPI imbalance



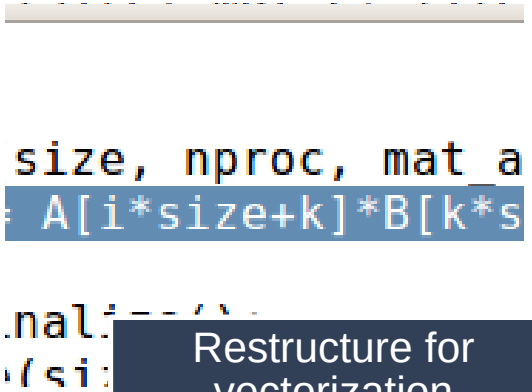
Remove I/O bottleneck



Make sure OpenMP regions make sense



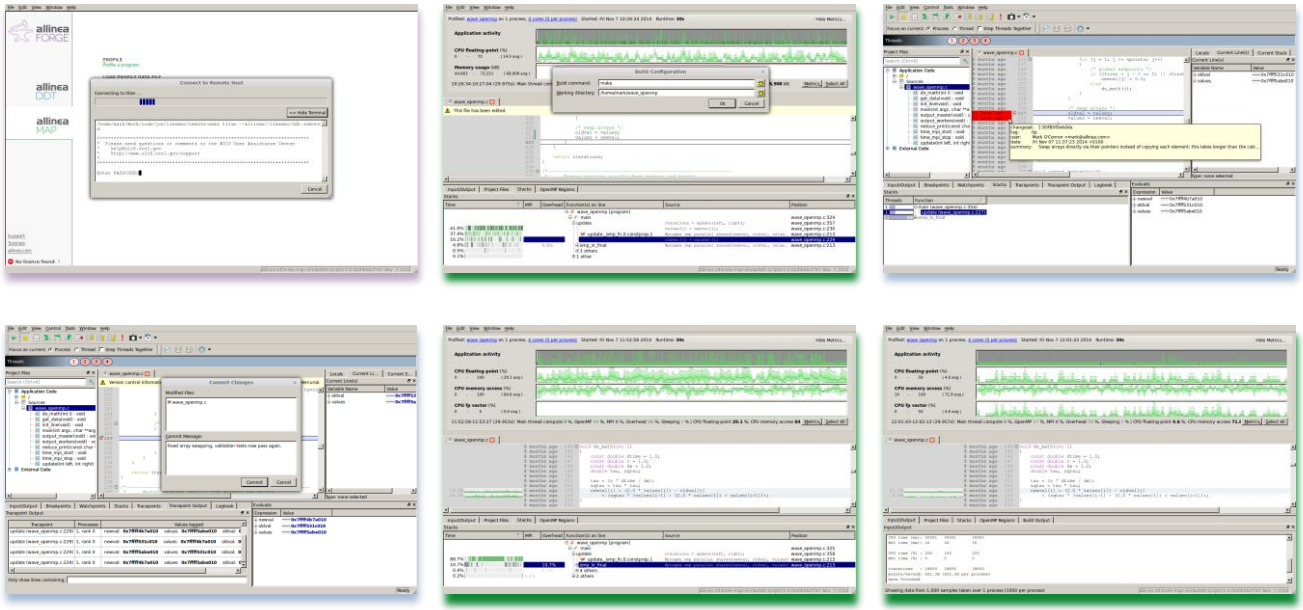
Improve memory access



Restructure for vectorization

allinea

A Productive HPC Development Workflow



allinea FORGE
DDT + MAP

allinea

Tour/Demo of Forge

- Debugging with DDT
  - Walkthrough example – cstartmpi.exe
- Profiling with MAP
  - Matrix-multiplication

allinea

Hands on Session

- Use Allinea DDT on your favorite system to debug your code – or example codes
- Use Allinea MAP on NERSC Edison or ANL Cooley to see your code performance
- Use Allinea DDT and Allinea MAP together to improve our test code
- Can you beat a 50% speed up?

Getting started on Mira/Cooley

- Install local client on your laptop
  - www.allinea.com/products/forge/downloads
    - Linux – installs full set of tools
    - Windows, Mac – just a remote client to the remote system
  - Run the installation and software
  - “Connect to remote host”
  - Hostname:
    - username@cetus.alcf.anl.gov
    - username@cooley.alcf.anl.gov
  - Remote installation directory: /soft/debuggers/ddt
  - Click Test
- Congratulations you are now ready to debug on Mira/Vesta/Cetus – or debug and profile on Cooley.

allinea

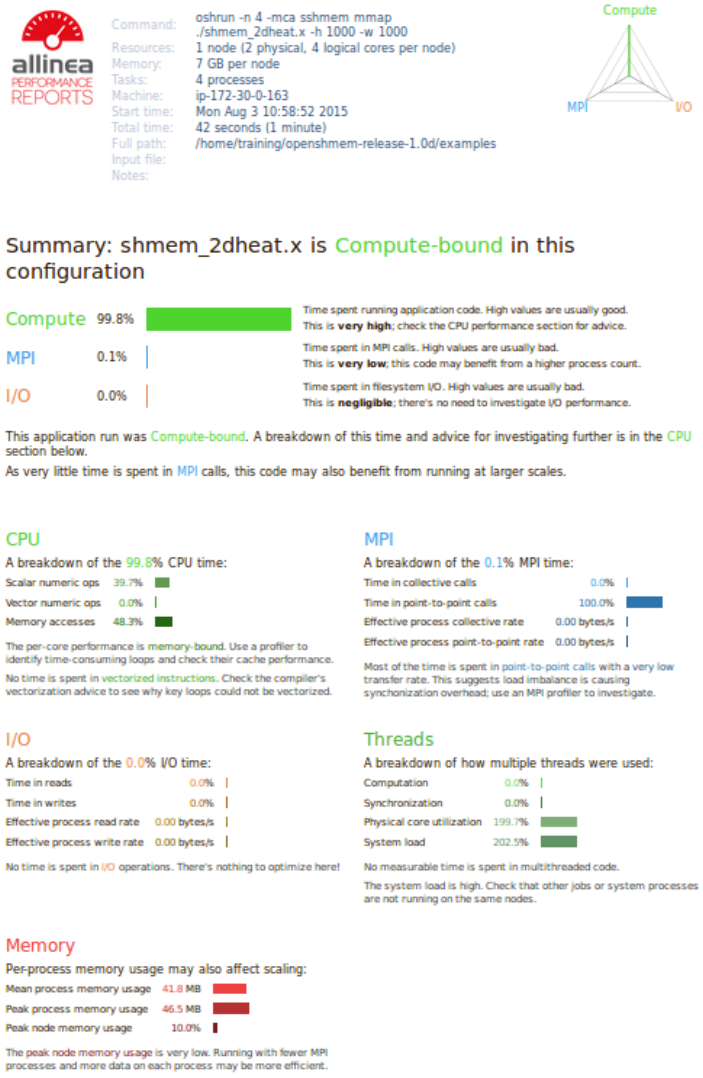
Performance and debugging challenge

- Obtain the code:
  - git clone https://github.com/estrabd/2d-heat.git
- The challenge
  - Improve run time by 50%
  - Parameters: –h 1000 –w 1000 set the dimensions
- Submit/run
    - mpirun –np 16 2d-heat.x –h 1000 –w 1000

- Use MAP to find the lowest hanging fruits
- Use DDT to look at behaviour - confirm what is safe to change
- Edit and build in the GUI: File/Configure Build/ and re-run

allinea

Initial Performance Report



allinea PERFORMANCE REPORTS

Command: oshrun -n 4 -mca sshmem mmap ./shmem_2dheat.x -h 1000 -w 1000
Resources: 1 node (2 physical, 4 logical cores per node)
Memory: 7 GB per node
Tasks: 4 processes
Machine: ip-172-30-0-163
Start time: Mon Aug 3 10:58:52 2015
Total time: 42 seconds (1 minute)
Full path: /home/training/openshmem-release-1.0d/examples
Input file:
Notes:

Compute
MPI
I/O

Summary: shmem_2dheat.x is Compute-bound in this configuration

Compute	99.8%	Time spent running application code. High values are usually good. This is **very high**; check the CPU performance section for advice.
MPI	0.1%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process count.
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance.

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below.
As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU
A breakdown of the 99.8% CPU time:

Scalar numeric ops	39.7%
Vector numeric ops	0.0%
Memory accesses	48.3%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.
No time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI
A breakdown of the 0.1% MPI time:

Time in collective calls	0.0%
Time in point-to-point calls	100.0%
Effective process collective rate	0.00 bytes/s
Effective process point-to-point rate	0.00 bytes/s

Most of the time is spent in point-to-point calls with a very low transfer rate. This suggests load imbalance is causing synchonization overhead; use an MPI profiler to investigate.

I/O
A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

Threads
A breakdown of how multiple threads were used:

Computation	0.0%
Synchronization	0.0%
Physical core utilization	199.7%
System load	202.5%

No measurable time is spent in multithreaded code.
The system load is high. Check that other jobs or system processes are not running on the same nodes.

Memory
Per-process memory usage may also affect scaling:

Mean process memory usage	41.8 MB
Peak process memory usage	46.5 MB
Peak node memory usage	10.0%

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

- Using Forge – profile, debug, edit, build, repeat!
  - profiler to find targets
  - debugger to reveal usage
  - Make changes
  - Configure the Build command
    - make CFLAGS='-O3 –g'
    - ***NB: single quotes..***
  - Build, repeat!

allinea