



### Debugging and Profiling your HPC Applications Srinath Vadlamani, Field Application Engineer srinath.vadlamani@arm.com



### **About this talk**

Techniques not tools

Learn ways to debug and profile your code

Use tools to apply techniques

- Debugging with Allinea DDT
- Benchmarking with Allinea Performance Reports
- Profiling with Allinea MAP
- Go to <u>www.allinea.com/trials</u>

Tools are available on the ATPESC machines

### **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

- Being smart about what you're doing
- Using the tools that help you apply smart techniques

### **Real-world example**

le Edit View Metrics Window					File Edit View Metric	Window Help	
ofiled: Discovar on 1 process, 1 no	de, <u>36 cores (</u>	36 per process) Sampled from: Fri Jul 24 16:16:36 2015	for 300.0s	Hide Metrics	Profiled: hait on 1 proce	ss, 1 node, 24 cores (24 per process) Sampled from: Tue	Jun 28 13:41:48 2016 for 1,200.1s
Application activity					Missing Debugging Info	mation I THTensorMath.c THTensorDimA	
CPU floating-point 100							
1.2 %			<u>AAAAAA_</u>			48 while(!TH_TENSOR_DIM_APPL 49 { \ 50 CODE \	Y_hasFinished) \
Memory usage 3.91				~~		51 \ 52 if (TENSOR1->nDimension 53 break; \	1) \
۰ ا						54 \ 55 for ITH TEXSOR DIM APPLY	i = 0; TH TENSOR DIM APPLY i < TENSOR
J:16:36-16:21:36 (300.013s): Mai	n thread comp	oute 11.9 %, Pthreads 9.5 %, OpenMP 55.7 %, File I/O 1.5	%, Synchronisation 0.2%, OpenMP overhead 21.0%, Sleeping 0%   CPU float	ing-point 1.2 %; Memory usage 2.19 (Zoom 🖏 🇮 💿		56 { \ 57 if(TH_TENSOR_DIM_APPL	
LoadCorrectCore.cc						58 ( ) 59 if (TH TENSOR DIM AP	PLY_i == TERSOR1->nDimension-1) \
0.3%		q(j) = Min(q(j), (int) cqu for ( int j = 0; j < (int) cquals[id] cquals[id][j] = q[j]; } )	ls[id][l] ); } size(); ;++ )	-		60 (\ 61 TH_TENSOR_DIM_APP 62 break; \	LY_hasFinished = 1; \
		d CorrectionSuite( const String& TMP, const const long_logging& logc, const long_loggi vecbasevector& creads, VecEFasta& correcte	g_control6 log_control,			63 } \ 64 continue; \ 65 } \	
	445 446 🖂	<pre>vec<pairing_info>6 cpartner, const uint NU const double clock, bool useOldLRPMethod )</pairing_info></pre>	_THREADS, const String& EXIT,				ounter[TH_TENSOR_DIM_APPLY_i]++; \ SOR1->stride[TH_TENSOR_DIM_APPLY_i]; \
64.78 M A		LongProtoTmpDirManager tmp_mgr(TMP);	strol, creads, corrected, cid, cpartner, NUM THREADS, EXIT,	clock useOldtPDNethod ):		69 TENSOR2## data += TEN	SOR2->stride[TH_TENSOR_DIM_APPLY_i]; \ SOR3->stride[TH_TENSOR_DIM_APPLY_i]; \
	450 1 2		the second second and second the second				Y counter[TH TENSOR DIM APPLY 1] == TEN
	452 void 453	I CorrectionSuite( LongProtoTmpDirManager& t const long_logging& logc, const long_loggi	p_mgr, const long_heuristicsé heur, g_controlé log_control,			73 { \	PLY_i == TENSOR1->nDimension-1) \
		<pre>vecbasevector&amp; creads, VecEFasta&amp; corrected, vec<int>&amp; cid, vec<p const uint NUM_THREADS, const String&amp; EXIT</p </int></pre>	iring_info>6 cpartner, const double clock,			75 {\ 76 TH_TENSOR_DIM_APP 77 breaki \	LY_hasFinished = 1; \
	457 E	bool useOldLRPMethod )				78 ) \ 79 else \	
		// Run Correct1.				80 ( X	
	461	vec <int> trace ids, precorrect seg;</int>		¥		82 TENSOR2##_data -=	TH_TENSOR_DIM_APPLY_counter[TH_TENSOR_ TH_TENSOR_DIM_APPLY_counter[TH_TENSOR]
	penMP Stacks	OpenMP Regions Functions				83 TENSOR 3##_data -= 84 TH TENSOR DIM APP	TH_TENSOR_DIM_APPLY_counter[TH_TENSOR LY counter[TH_TENSOR_DIM_APPLY_1] = 0;
ienMP Stacks				e ×		85 ) X	
tal core time 🛛 💎		unction(s) on line	Source	Position			
		Ø Discovar [program]	1	Discovar.cc:26			
		B CorrectionSuite(FeudalString <char, <c<="" std::char="" td="" traits=""><td>cpartner, NUM_THREADS, **, clock, USE_OLD_LRP_METHOD );</td><td>Discovar.cc:137</td><td>Input/Output Projec</td><td>Files OpenMP Stacks OpenMP Regions Function</td><td>5</td></char,>	cpartner, NUM_THREADS, **, clock, USE_OLD_LRP_METHOD );	Discovar.cc:137	Input/Output Projec	Files OpenMP Stacks OpenMP Regions Function	5
37.1%	0.3%		iii CorrectionSuite( tmp_mgr, heur, logo, log_control, creads, corr matrix, to_edit, trim_to, trace_ids, logo, heur ); }	ected, cid, cpartne LoadCorrectCore.cc:449 LoadCorrectCore.cc:604	OpenMP Stacks		
17.9%	<0.1%		har trace_ids, heur, log_control, logc, corrected );	LoadCorrectCore.cc:752	Total core time	V Overhead Function(s) on line	Source
2.7%		⊕ CorrectPairs1(FeudalString < char, std::char_traits <)	har trace_ids, heur2, log_control, logc, corrected );	LoadCorrectCore.cc:791		🖯 🌾 luajit (program)	
6.9% la i i i 1.9% i		B 13 others B ImproveLongHyper(FeudalString <char, p="" std::char="" trait)<=""></char,>	· International Action of the Manual Action	Discovar.cc:149		© THThread_main	[OpenMD worker thread] (no debug
3.6%	0.379	B ImproveLongHyper(HeudalString <char, std::char_trait<br="">B 4 others</char,>	<pre>&lt;</pre>	Discovar.cc:149		B lua_pcall Bli BC FUNCC	THThread_main (no debug info) lus scall (no debug info)
		OpenMP worker threads]		-			trafforers (DD 08003 1810)
howing data from 1.000 samples t	aken over 1 pr	ocess (1000 per process)	Allines Forge v5-1-BRANCH Co	nnected to: ec2-user@52.18.49.160 💝 OpenMP View 🖉	Showing data from 1.00	i samples taken over 1 process (1000 per process)	

#### 🛪 🕴 🕨 🕴 🚺 🖢 🖄 🕶 💬 🕶 cess C Thread 🔽 Step Threads Together 2 B B 🕸 🗸 2)(3)(4) # × 🛛 🗴 wave\_openmp.c 🔀 Locals Current Line(s) Current Stack Current Line(s) /\* global endpoints ' if ((first + j - 1 == newval[j] = 0.0; Variable Name Application Cod -0x7ffff531c010 \_\_\_\_0x7ffff5abe010 do\_math(j); get\_data(void) init line(void) : void main(int argc, char iii output\_master(void) ii output\_workers(void) reduce print(const ch time\_mpi\_start : void time\_mpi\_stop : void update(int left, int right) Mark O'Connor <mark@allinea.com> Fri Nov 07 11:37:23 2014 +0100 yr. Swapa arrays directly via their pointers instead of copying each element; this takes longer than the cal External Code Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook Expression Value newval 0x7fff4b7a010 oldval 0x7fff531c010 values 0x7fff5abe010 Eunction Ready

### **Bioinformatics**

**Discover Assembly** 

3x speedup

EC2

### **Deep Learning** Torch + DeepMind 5.3x speedup Intel Xeon Phi (KNL)

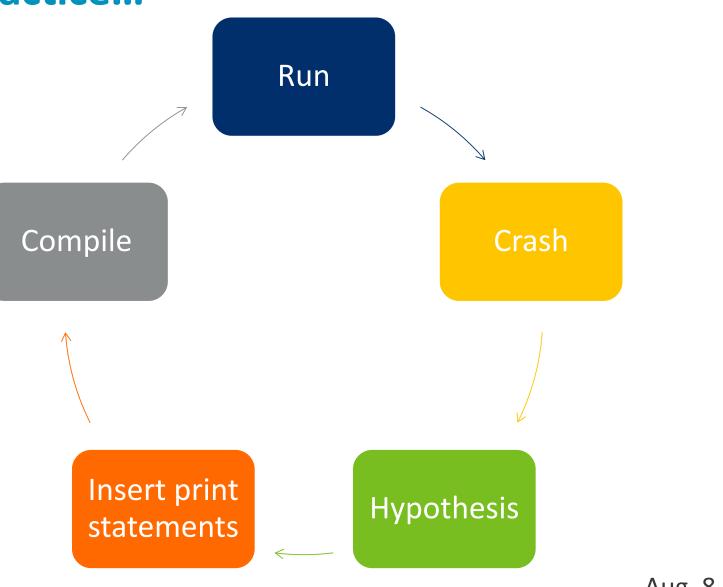
Show Metrics.

8)

**Fluid Dynamics** HemeLB blood flow 16.8x capability boost 50k core crash fixed

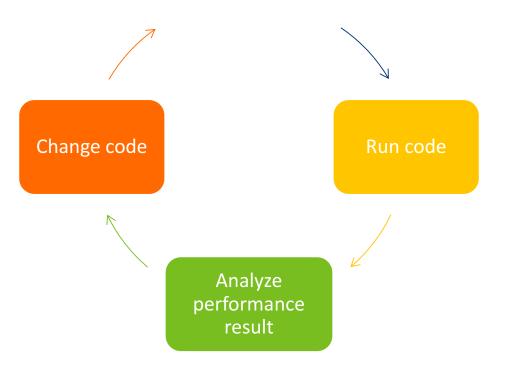


### **Debugging in practice...**





### **Optimization in Practice**



### **About those techniques...**

"No-one cares how quickly you can compute the wrong answer"

• Old saying of HPC performance experts

## Let's start with debugging then...

### 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 the code should have never worked, after which the program ceases to work until fixed

## Debugging

The art of transforming a broken program to a working one:

- Debugging requires thought and discipline:
  - 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:

• Andreas Zeller, "Why Programs Fail", 2nd Edition, 2009

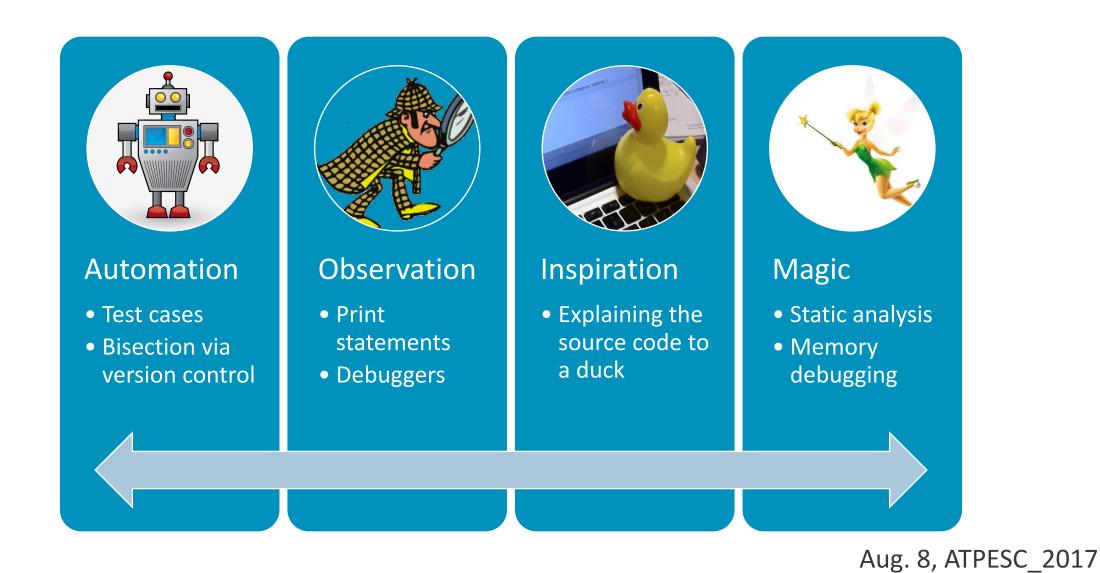
### What you will read:

• Crowd sources like *stack overflow* 

### **Popular techniques**



arm



10 © 2017 Arm Limited

## **Solving Software Defects**



### Who had a rogue behavior ?

Merges stacks from processes and threads

### Where did it happen?

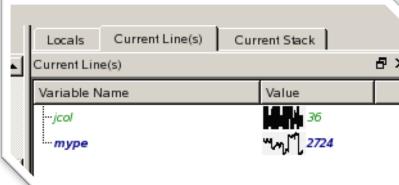
leaps to source

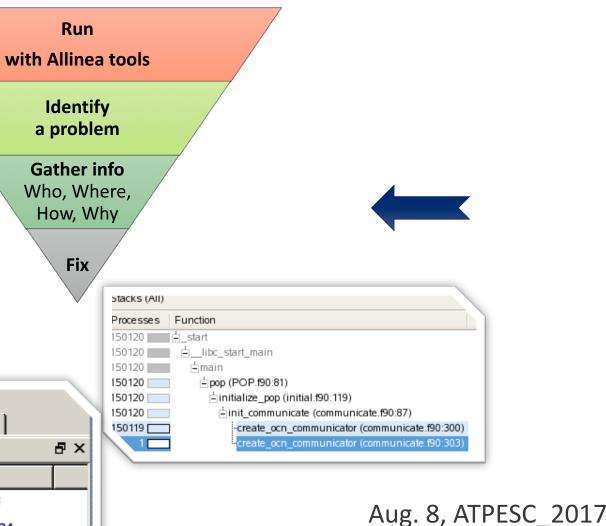
### How did it happen?

- Diagnostic messages
- Some faults evident instantly from source

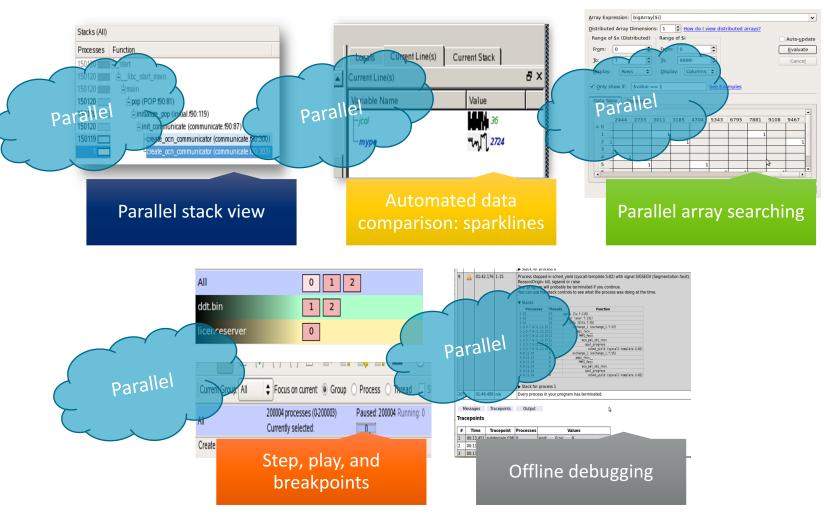
### Why did it happen?

- Unique "Smart Highlighting"
- Sparklines comparing data across processes





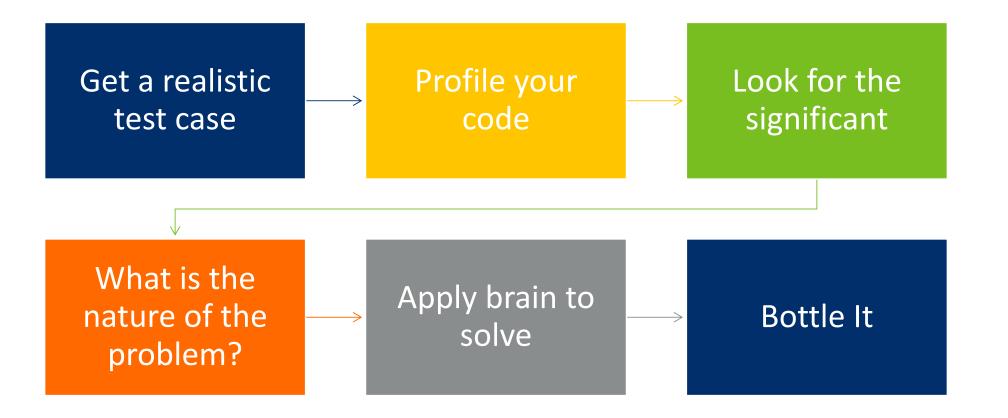
## **Favorite Allinea DDT Features for Scale**



Aug. 8, ATPESC\_2017

allinea

## 6 steps to help improve performance



Logging like an experiment is useful.

Aug. 8, ATPESC\_2017

allinea

ΜΑΡ

13 © 2017 Arm Limited

## Bottling it...



- Lock in performance once you have won it
- Save your nightly performance
- Tie your performance results to your continuous integration server

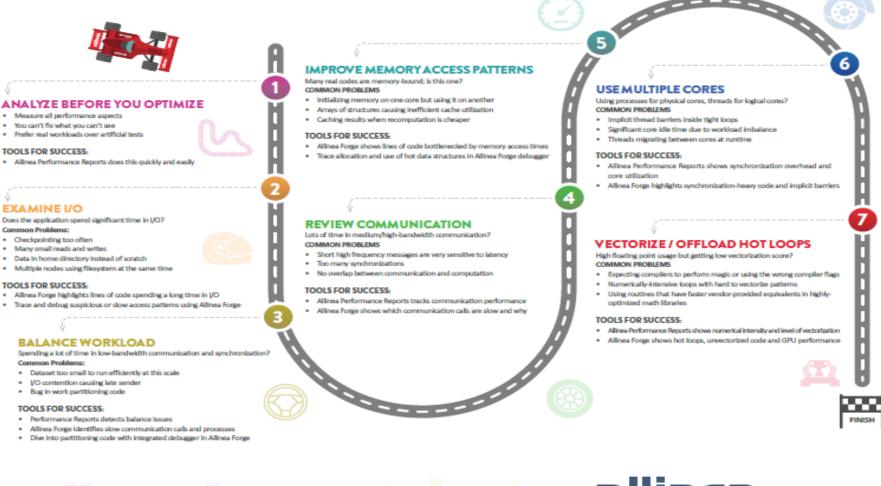
- Lock in the bug fixes
- Save the test cases
- Tie the test cases to your continuous integration server

• Regression tests do help you from regressing!!!



## **PERFORMANCE ROADMAP**

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster. This pragmatic, step by step guide will help you to identify and focus on bottlenecks and optimizations one at a time with an emphasis on measuring and understanding before rewriting.



### allinea

#### Contact up

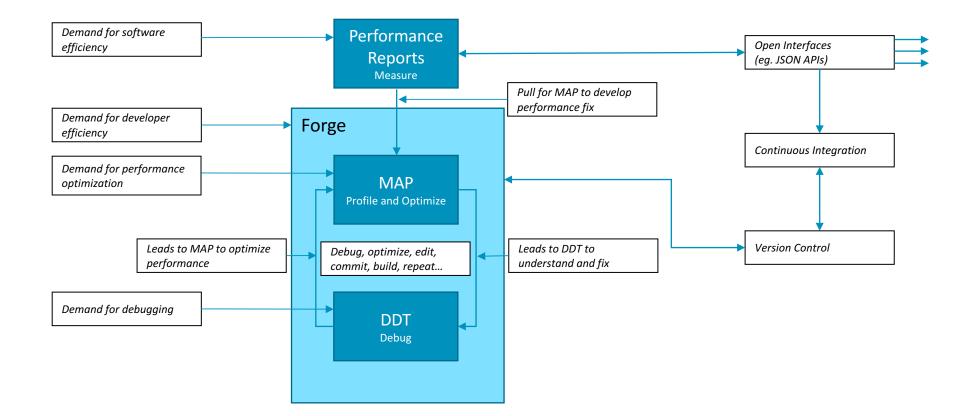
w: www.allinea.com t +1 (400) 600 2700 e: sales⊚ellinea.com Allinea.Software Inc. 2033 Gateway Mace. Suite 500. San Jose. CA 95110



### How The Tools Fit...







## How to help scientific developers best?



### You **can** teach a man to fish But first he must realize **he is hungry**

Image © <u>Kanani</u> CC-BY



## **Communicate the benefits of optimization**

Show, don't tell...

### CPU



A breakdown of the 84.4% CPU time:

Scalar numeric ops	27.4%	
Vector numeric ops	0.0%	
Memory accesses	72.6%	
Waiting for accelerators	0.0%	

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.

... this is your code on "--O0", ie. no optimizations

### Show performance they understand



### CPU

A breakdown of the 88.5% CPU time:

Single-core code	100.0%	
Scalar numeric ops	22.4%	
Vector numeric ops	0.0%	1
Memory accesses	77.6%	

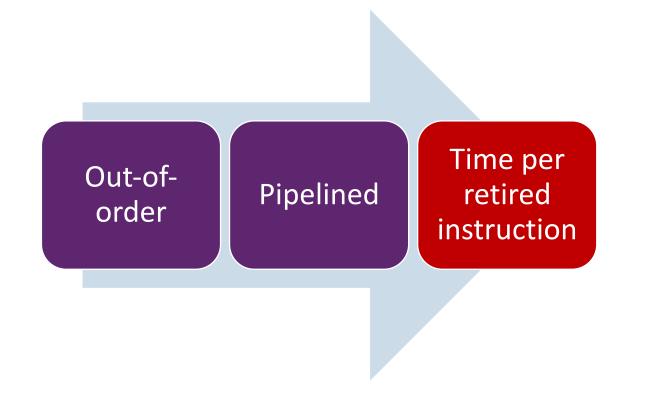
The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.



## **Communicating at the right level**







# Explaining performance at the right level CPU



A breakdown of the 88.5% CPU time:

Single-core code	100.0%	
Scalar numeric ops	22.4%	
Vector numeric ops	0.0%	1
Memory accesses	77.6%	

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.

+ simple, actionable advice

Compiler advice is your friend.



## Vectorization, MPI, I/O, memory, energy...



nd: mpiexec -n 4 ./wave\_c 8000 4 processes, 1 node (4 physical, 8 logical cores per node) kaze Fri Oct 17 17:00:27 2014 30 seconds (1 minute)

2.1 Ghz CPU frequency

#### Summary: wave\_c is CPU-bound in this configuration

CPU 88.5%	Time spent running application code. High values are usually good. This is <b>high</b> ; check the CPU performance section for optimization advice.
MPI 11.4%	Time spent in MPI calls. High values are usually bad. This is <b>very low</b> ; this code may benefit from increasing the process count.
I/O 0.0%	Time spent in filesystem I/O. High values are usually bad. This is <b>negligible</b> ; there's no need to investigate I/O performance.

This application run was CPU-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 88.5% CPU time:

Single-core code	100.0%	
Scalar numeric ops	22.4%	
Vector numeric ops	0.0%	1
Memory accesses	77.6%	

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.

#### I/O

A breakdown of the 0.0% I/O time:				
Time in reads	0.0%	1		
Time in writes	0.0%	1		
Effective process read rate	0.00 bytes/s	1		
Effective process write rate	0.00 bytes/s	1		

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

#### Memory

Per-process memory usage may also affect scaling:					
Mean process memory usage	49.7 MB				
Peak process memory usage	53.6 MB				
Peak node memory usage	24.0%				

The peak node memory usage is very low. You may be able to reduce the amount of allocation time used by running with fewer MPI processes and more data on each process.

#### MPI A breakdown of the 11.4% MPI time

A breakdown of the 11.4% MPT time.				
Time in collective calls	3.1%	1		
Time in point-to-point calls	96.9%			
Effective process collective rate	31.7 kB/s	1 - C		
Effective process point-to-point rate	269 kB/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 further.

#### Threads

A breakdown of how multiple threads	were use	ed:
Computation	0.0%	1
Synchronization	0.0%	1
Physical core utilization	100.0%	
Involuntary context switches per second	1.8	1
No measurable time is spent in multithrea	aded code.	

#### Energy

A breakdowr	n of how th	ne total 588 J energy was spent:
CPU	<b>100.0%</b>	
Accelerators	0.0%	1
Peak power	23.00 W	
Mean power	19.80 W	

The CPU is responsible for all measured energy usage. Check the CPU breakdown section to see if it is being well-used. Note: system-level measurements were not available on this run.

### CPU

A breakdown of the 88.5% CPU time:

Single-core code	100.0%	
Scalar numeric ops	22.4%	
Vector numeric ops	0.0%	1
Memory accesses	77.6%	



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.

### Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	49.7 MB
Peak process memory usage	53.6 MB
Peak node memory usage	24.0%



The **peak node memory usage** is very low. You may be able to reduce the amount of allocation time used by running with fewer MPI processes and more data on each process.



### Accelerator support...



/scratch/mark/miniMD\_OpenCL/miniMD\_nvidia -t 128 2 processes, 1 node (12 physical, 24 logical cores per node) kaze Mon Nov 3 11:52:14 2014 110 seconds (2 minutes)

## Compute



#### Summary: miniMD nvidia is Compute-bound in this configuration



Time spent running application code. High values are usually good. This is high; check the CPU and accelerator sections for optimization advice. Time spent in MPI calls. High values are usually bad.

This is very low; this code may benefit from increasing the process count.

Time spent in filesystem I/O. High values are usually bad. This is very low; however single-process I/O often causes large MPI wait times.

This application run was compute-bound. Investigate further with the CPU and accelerator sections below. As very little time is spent in MPI calls, this code may also benefit from running at larger scales

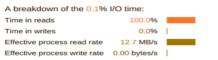
#### CPU



Waiting for accelerators 31.5% 15.6% Scalar numeric ops Vector numeric ops 0.0% Memory accesses 62.0%

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.

#### I/O



Most of the time is spent in read operations with a low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected

#### Memory

Per-process memory usage may also affect scaling:



The peak node memory usage is very low. You may be able to reduce the amount of allocation time used by running with fewer MPI processes and more data on each process.



Most of the time is spent in point-to-point calls with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

The collective transfer rate is very low. This suggests load imbalance is causing synchonization overhead; use an MPI profiler to investigate further.

#### Threads

#### A breakdown of how multiple threads were used:

Computation 68.1% Synchronization 31.9% Physical core utilization 16.7% 7.1 Involuntary context switches per second

Physical core utilization is low. Try increasing the number of threads or processes to improve performance

Significant time is spent synchronizing threads. Check which locks cause the most overhead with a profiler.

#### Accelerators

А	breakdown	of	how	accelerators	were	used
---	-----------	----	-----	--------------	------	------

Compute utilization	93.1%	
Memory utilization	20.4%	

Device memory used 9.8 %

High compute and low memory utilization suggests compute throughput is limiting GPU peformance.

Use a profiler to find the hottest kernels and check them for divergent branches and over-subscribed function units.

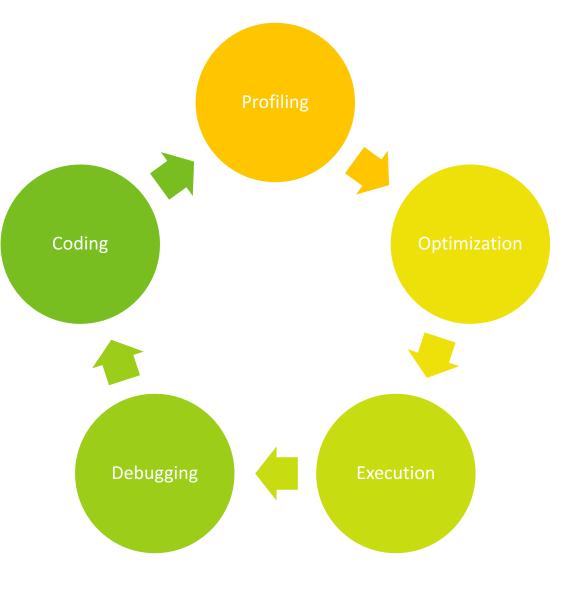


### **Application Development Workflow**



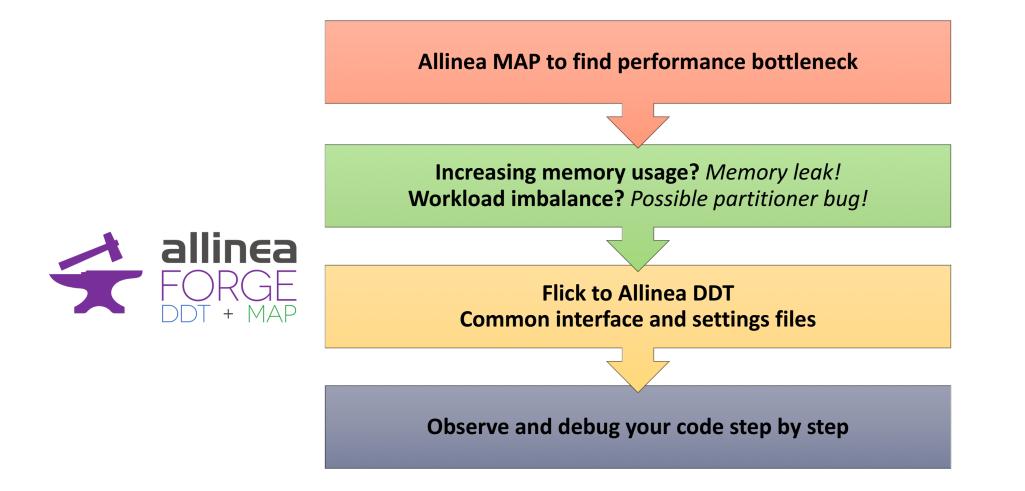
Aug. 8, ATPESC\_2017

arm



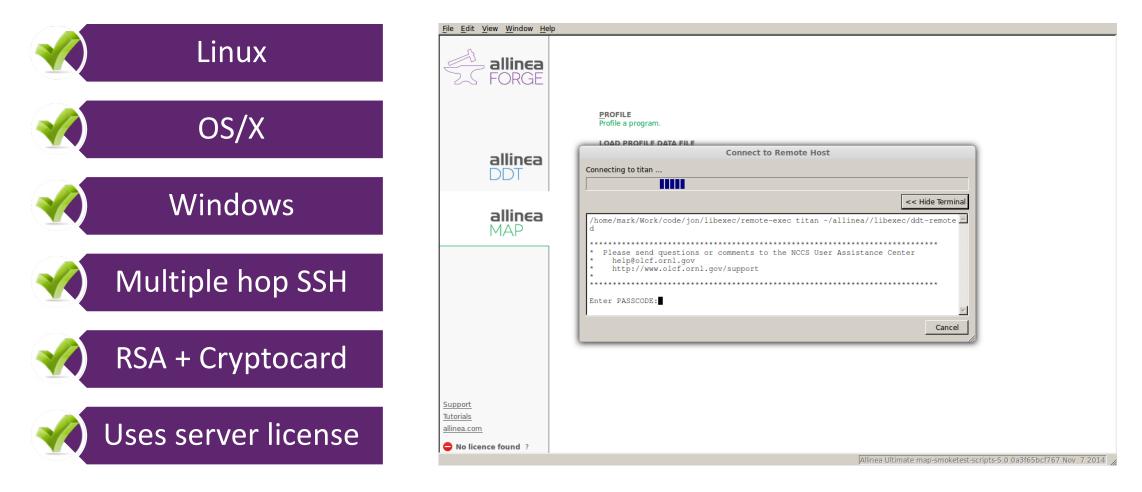
## **Hello Allinea Forge!**





## HPC means being productive on remote machines

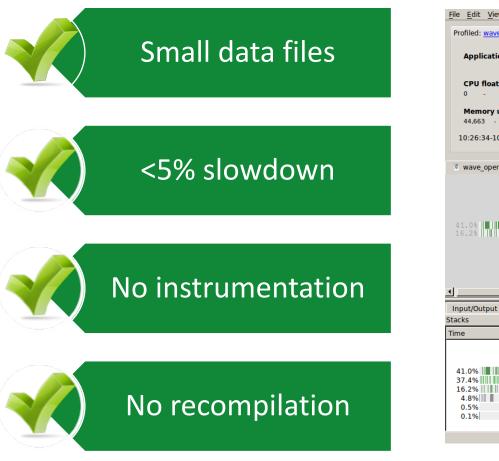


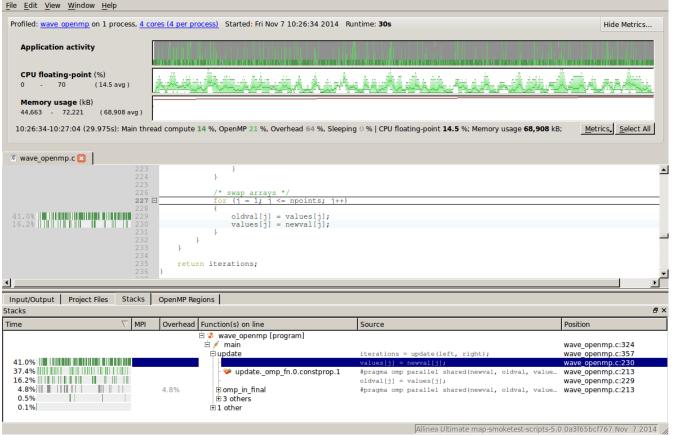




### **MAP in a nutshell**







## 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



Aug. 8, ATPESC\_2017

arm

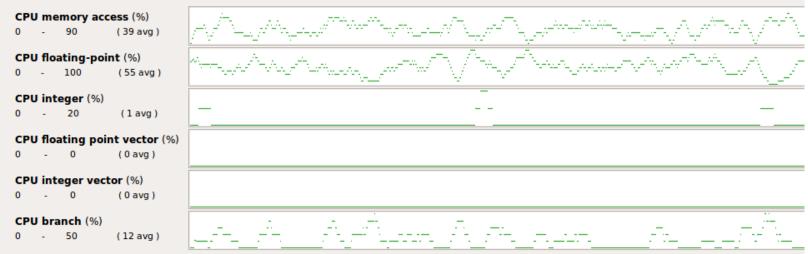
	08_10-18.map - Allinea MAP 4.2-34164	[Trial Version]					
<u>File View Search Window H</u> elp							
Profiled: hemelb on 512 processes	Started: Sat Feb 8 10:18:36 2014 Runtime	:: 308s Time in MPI: 589	6				Hide Metrics
Memory usage (M)	-1						
21.8 - 152.6 (108.3 avg)							
MPI call duration (ms)							
0 - 11,789.1 (65.7 avg)	Lan		<u></u>				
CPU floating-point (%) 0 - 100 (11 avg)							
10:18:36-10:23:43 (307.730s): Me	an: Memory usage <b>108.3</b> M; MPI call duration	on <b>65.7</b> ms; CPU floating-	point <b>10.6</b> %;				Metrics, Reset
🗷 StepManager.h 💥 🏾 Iterated	Action.cc 🕱 🦉 StepManager.cc 🕱						
	56 std::string na		hal anthad)				<u> </u>
		&concern, MethodLa oncern), method(met					
	59 {						
	60 } 61						
	62 🗉	c'i Devellat Ch	al Mari			unununun	
	63 Input/Output Project	Files Parallel Sta					
	65						=
	66 67 ime	A MPI	Function(s) on line	Source			
	68 <b>1.4%</b>	0.1%	hemelb::net::	IteratedActi RequestComms(	();		
	69 <b>1.5%</b> 70	1.3%	12 others	c-111-1-1-5			
	71		□ Call [inlined]		orStep(static_cast <steps::ste rn-&gt;CallAction(method);</steps::ste 	ep>(step), 0);	
	72			traction::Pro EndIteration(			
	73 🗆			xtraction::Pr propertyWrite	er->Write(simulationState.Get		
	75					e((uint64_t) iterationNumber);	
	76 34.3%			ile_write_at MPI_STATUS_IC	GNORE);		
	77 <0.1% 78 <0.1%	<0.1%	± 20 oth     ± 4 others				
	79 <0.1%	<0.1%	12 others				-
Input/Output Project Files Para	llel Stack Vi <0.1%	-0.170					
Parallel Stack View	0.2%	0.2%	10 others				0 8
	IPI Funcibry princeta from 5120	00 samples taken o	ver 512 processes (100		81 ( 1 )		
	.1% Demolority of the						IteratedAction.cc:27
	.3% 🗈 12 others						
	Call [inlined]	<pre>CallActionsForStep(s return concern-&gt;Call</pre>	<pre>tatic_cast<steps::step>(ste Action(method):</steps::step></pre>	p), 0);			StepManager.cc:127 StepManager.h:67
	hemelb::extraction::Pro	EndIteration();					IteratedAction.cc:39
			e(simulationState.GetTimeSt [outputNumber]->Write((uint				PropertyActor.cc:90 PropertyWriter.cc:42
34.3%	4.3% PMPI_File_write_at						LocalPropertyOutput.cc:302
<0.1%	:0.1%						
<0.1% <	0.1% 🖻 12 others						
<0.1% 0.2%	.2% ① 3 others						
	aken over 512 processes (1000 per process	)					Allinea MAP 4.2-34164
showing data nom size oo samples to	and over one processes (noos bei process	,					Allifica MAL 4.2-34104

Simple fix... reduce periodicity of output

## **Deeper insight into CPU usage**



Runtime of application still unusually slow



07:44:28-07:44:55 (27.144s): Mean: CPU memory access 39.0 %; CPU floating-point 55.3 %; CPU integer 0.8 %; CPU floating point vector 0.0 %; CPU integer

Allinea MAP identifies vectorization close to zero

Why? Time to switch to a debugger!

### While still connected to the server we switch to the debugger

<u>File Edit View Control To</u>	ols <u>W</u> indow <u>H</u> elp
S FORGE	
	Run and debug a program.
allinea	ATTACH Attach to an already running program.
DDT	OPEN CORE Open a core file from a previous run.
<b>allinea</b> MAP	MANUAL LAUNCH (ADVANCED) Manually launch the backend yourself. OPTIONS
	Remote Launch: Off
	ουιτ
<u>Support</u> Tutorials	
allinea.com	
Licence Serial: 4306 ?	
	Allinea Ultimate map-smoketest-scripts-5.0 0a3f65bcf767 Nov 7 2014



allinea

## It's already configured to reproduce the profiling run

<u>File E</u> dit <u>V</u> iew	<u>C</u> ontrol <u>T</u> ool	ls <u>W</u> indow <u>H</u> elp			
A			Run	×	
ر میں اور			Application: /home/mark/wave_openmp/wave_openmp	Details	
	JRGE		Application: /home/mark/wave_openmp/wave_openmp		
		RUN	Arguments:	•	
		Run and debug a program.	stdin file: //home/mark/Work/r/bench.R	-	
_		<b>ATTACH</b> Attach to an already running program.	Working Directory:	-	
<b>allinea</b> DDT	OPEN CORE	MPI: 1 process, Open MPI (Compatibility)			
	Open a core file from a previous run.	Number of MPC Tasks: 1			
	MANUAL LAUNCH (ADVANCED) Manually launch the backend yourself.	Number of Processes: 1			
allinea			Processes per Node 1		
M,	AP	<u>O</u> PTIONS	Implementation: Open MPI (Compatibility) Change		
		Remote Launch:	mpirun Arguments:	•	
		Off	OpenMP: 4 threads	Details	
		<u>Q</u> UIT	Number of OpenMP threads: 4		
			CUDA	Details	
			Memory Debugging	Details	
			☐ Submit to Queue Configure F	Parameters	
			Environment Variables: none	Details	
			Plugins: none	Details	
<u>Support</u> <u>Tutorials</u> allinea.com			Help Options Run	Cancel	
Licence Serial: 43	306 ?			v	

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

Aug. 8, ATPESC\_2017

allinea

## **Today's Status on Scalability**

### Debugging and profiling

- Active users at 100,000+ cores debugging
- 50,000 cores was largest profiling tried to date (and was Very Successful)
- ... and active users with just 1 process too

Deployed on

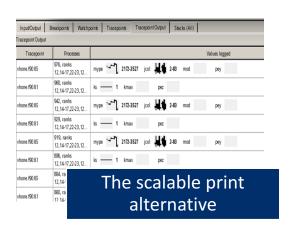
- NERSC Cori, ORNL's Titan, NCSA Blue Waters, ANL Mira etc.
- Hundreds of much smaller systems academic, research, oil and gas, genomics, etc.

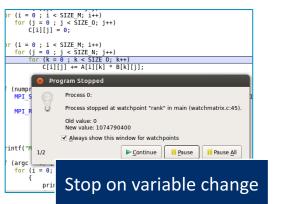
### Tools help the full range of programmer ambition

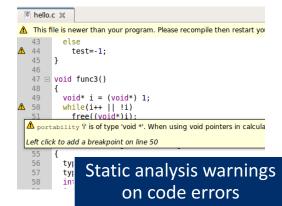
• Very small slow down with either tool (< 5%)

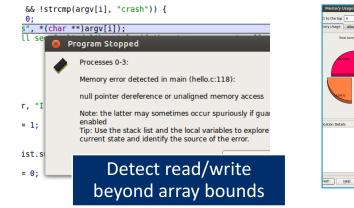
### Five great things to try with Allinea DDT

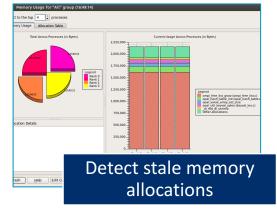








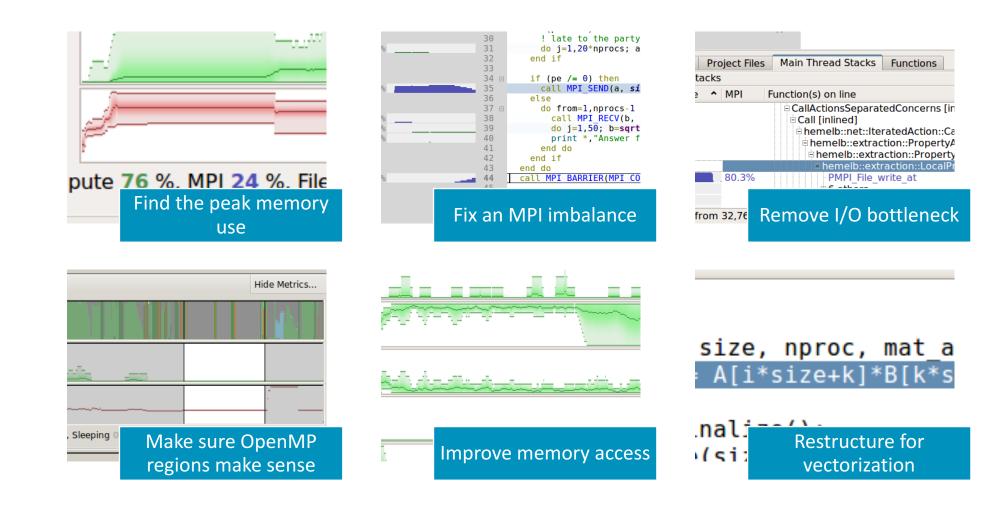






## Six Great Things to Try with Allinea MAP





## **Getting started on Theta**

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:
  - <u>username@theta.alcf.anl.gov</u>
- Remote installation directory: /soft/debuggers/forge-7.0.6-2017-08-07/
- Click Test

Congratulations you are now ready to debug Theta.





### Hands on Session

Use Allinea DDT on your favorite system to debug your code – or example codes

Use Allinea MAP or Performance Reports on Cooley to see your code performance

Use Allinea DDT and Allinea MAP together to improve our test code

• Download examples from <u>www.allinea.com</u> - Trials menu, Resources – "trial guide"



## Thank you for your attention!

Contact:

- <u>support@allinea.com</u>
- <u>support@arm.com</u>

### Download a trial for ATPESC (or later)

http://www.allinea.com/trials

