allinea

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

Enabling life-changing scientific discoveries with Allinea's tools

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

Agenda for the day

- 08:30 – 09:00 : Arrival and Welcome
- 09:00 – 10:20 : Adding vectorization to linear algebra applications
- 10:20 – 10:40 : Coffee break
- 10:40 – 13:00 : Preparing applications for large scale
- 13:00 – 14:00 : Lunch Break
- 14:00 – 15:00 : Commercial presentation

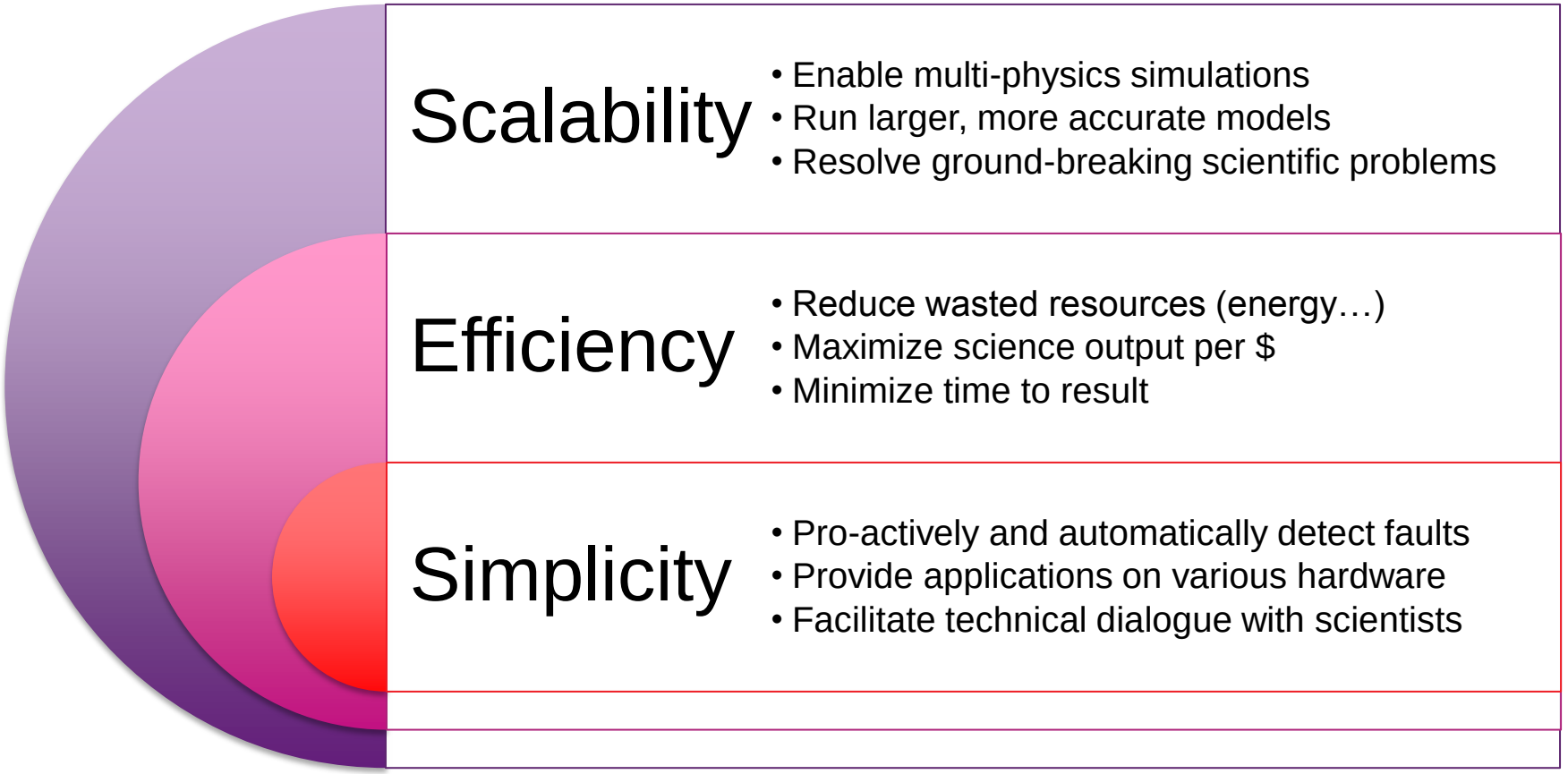
HPC Ultimate target

allinea

Example: Weather and Forecasting models

allinea

Building blocks for Exascale



Allinea's vision

- **Helping maximize HPC efficiency**
  - Reduce HPC systems operating costs
  - Resolve cutting-edge challenges
  - Promote Efficiency (as opposed to Utilization)
  - Transfer knowledge to HPC communities

- **Helping the HPC community design the best applications**
  - Reach highest levels of performance and scalability
  - Improve scientific code quality and accuracy

allinea

Where to find Allinea's tools

Over 65% of Top 100 HPC systems

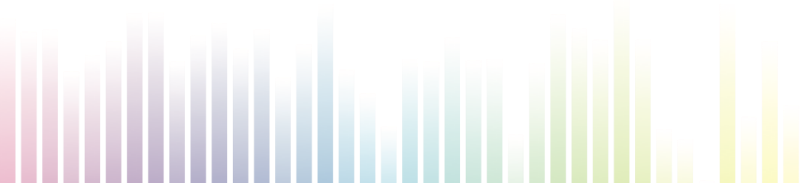
- From small to very large tools provision

8 of the Top 10 HPC systems

- From 1,000 to 700,000 core tools usage

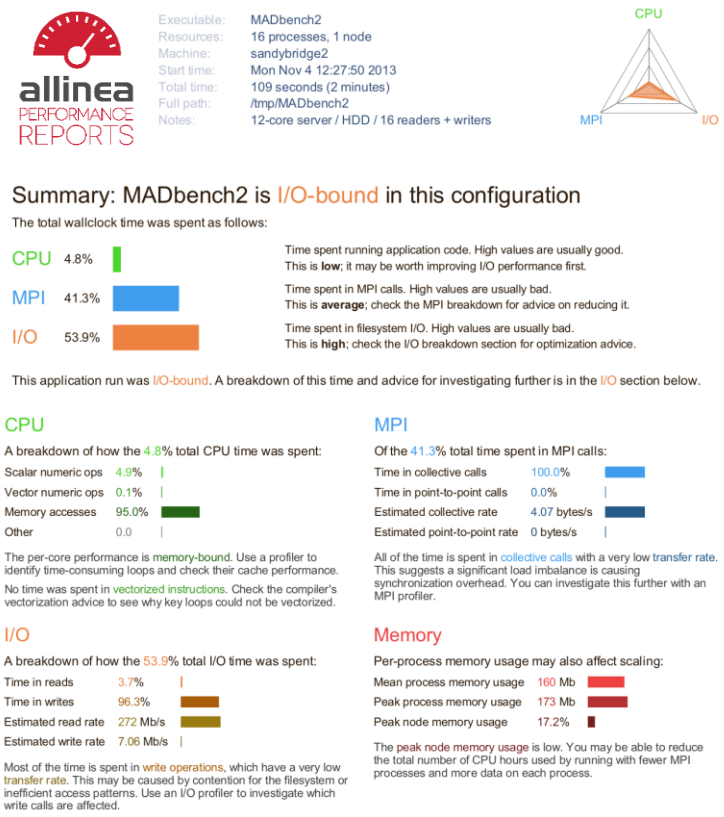
Future leadership systems

- Millions of cores usage



allinea

“Learn” with Allinea Performance Reports



allinea PERFORMANCE REPORTS

Executable: MADbench2
Resources: 16 processes, 1 node
Machine: sandybridge2
Start time: Mon Nov 4 12:27:50 2013
Total time: 109 seconds (2 minutes)
Full path: /tmp/MADbench2
Notes: 12-core server / HDD / 16 readers + writers

CPU
MPI I/O

Summary: MADbench2 is I/O-bound in this configuration

The total wallclock time was spent as follows:

CPU 4.8% — Time spent running application code. High values are usually good. This is **low**; it may be worth improving I/O performance first.

MPI 41.3% — Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.

I/O 53.9% — Time spent in filesystem I/O. High values are usually bad. This is **high**; check the I/O breakdown section for optimization advice.

This application run was I/O-bound. A breakdown of this time and advice for investigating further is in the I/O section below.

CPU

A breakdown of how the 4.8% total CPU time was spent:

Scalar numeric ops	4.9%
Vector numeric ops	0.1%
Memory accesses	95.0%
Other	0.0

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

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

MPI

Of the 41.3% total time spent in MPI calls:

Time in collective calls	100.0%
Time in point-to-point calls	0.0%
Estimated collective rate	4.07 bytes/s
Estimated point-to-point rate	0 bytes/s

All of the time is spent in collective calls with a very low transfer rate. This suggests a significant load imbalance is causing synchronization overhead. You can investigate this further with an MPI profiler.

I/O

A breakdown of how the 53.9% total I/O time was spent:

Time in reads	3.7%
Time in writes	96.3%
Estimated read rate	272 Mb/s
Estimated write rate	7.06 Mb/s

Most of the time is spent in write operations, which have a very low 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:

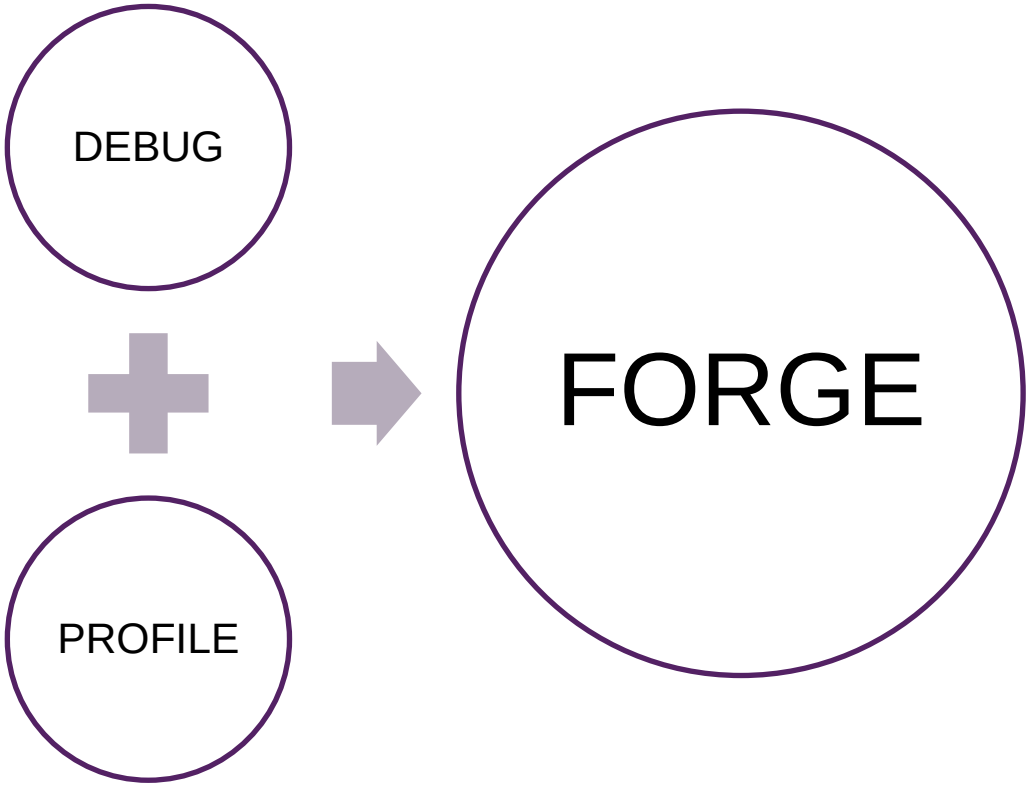
Mean process memory usage	160 Mb
Peak process memory usage	173 Mb
Peak node memory usage	17.2%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.

- Very simple start-up
- No source code needed
- Fully scalable, very low overhead
- Rich set of metrics
- Powerful data analysis

allinea

Allinea Forge: a toolkit to save developers' time



allinea

Allinea Forge: a toolkit to save developers' time

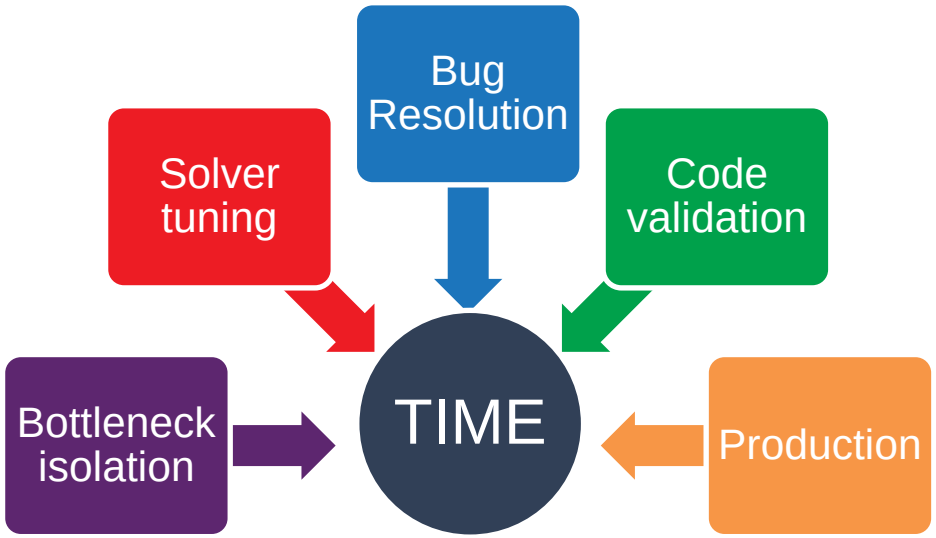
ACCESSIBLE
- Unique single interface
- Easy to start and use

POWERFUL
- State of the art features
- Fully scalable

INNOVATIVE
- Tackles new challenges
- For latest HPC systems

allinea

We are here today for a single reason.
Help you save your time.



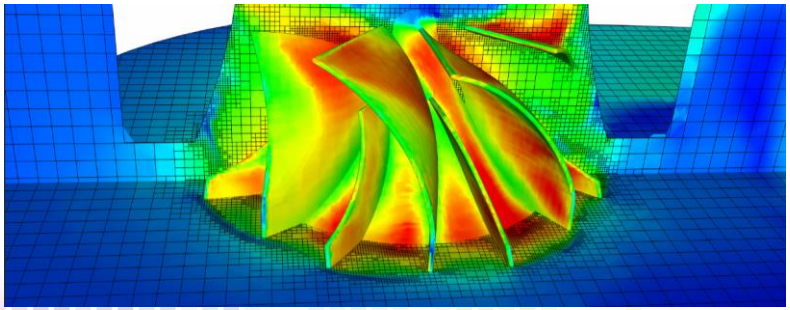
allinea

Convergent Science: developing faster software

- **Challenges**

  Developing faster and more capable software to meet the growing demands of Convergent Science clients for precision and performance

- **Huge speed up on CONVERGE: from 2h down to 4 sec**
  - Now possible to run jobs efficiently on hundreds of cores
  - Now possible to scale up from 2 million to 20 million nodes



Full case study:

http://www.allinea.com/news/201509/convergent-science-ignites-combustion-simulation-performance-allinea-forge

allinea

allinea

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

Questions?

Feel free to ask anything…!

allinea

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

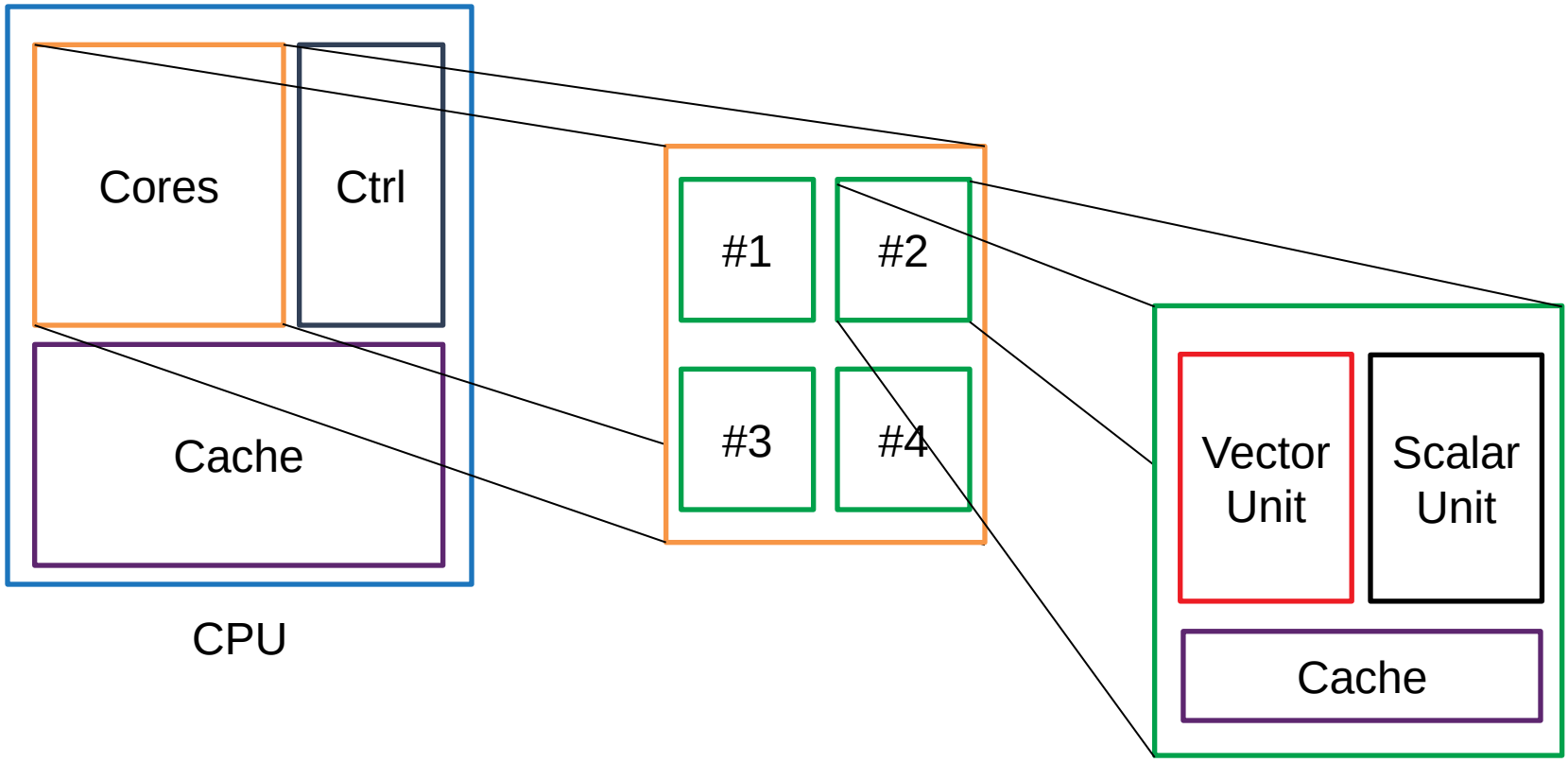
Technical session

Adding vectorization to linear algebra applications

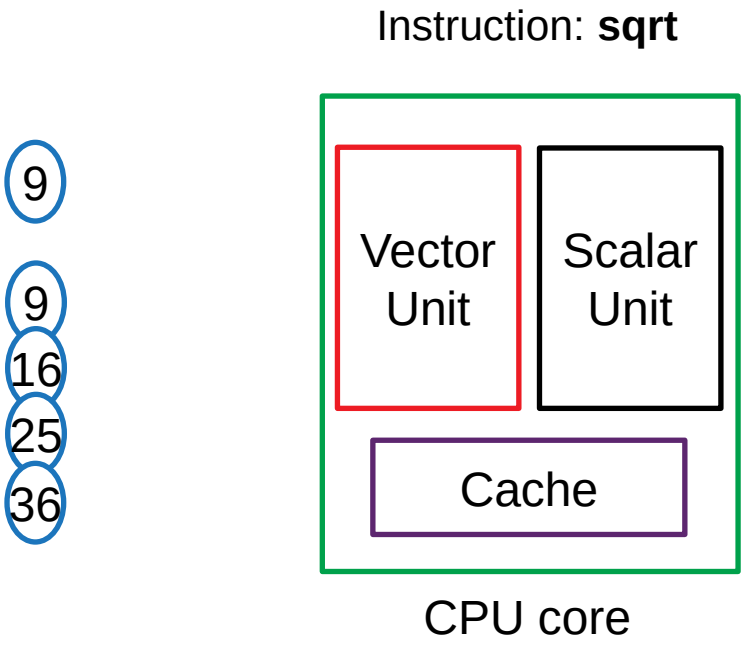
Agenda: Adding vectorization to an application

- What is vectorization?
- Why is vectorization important?
- When can my code use vectorization?
- How to actually enable vectorization?

What is vectorization?

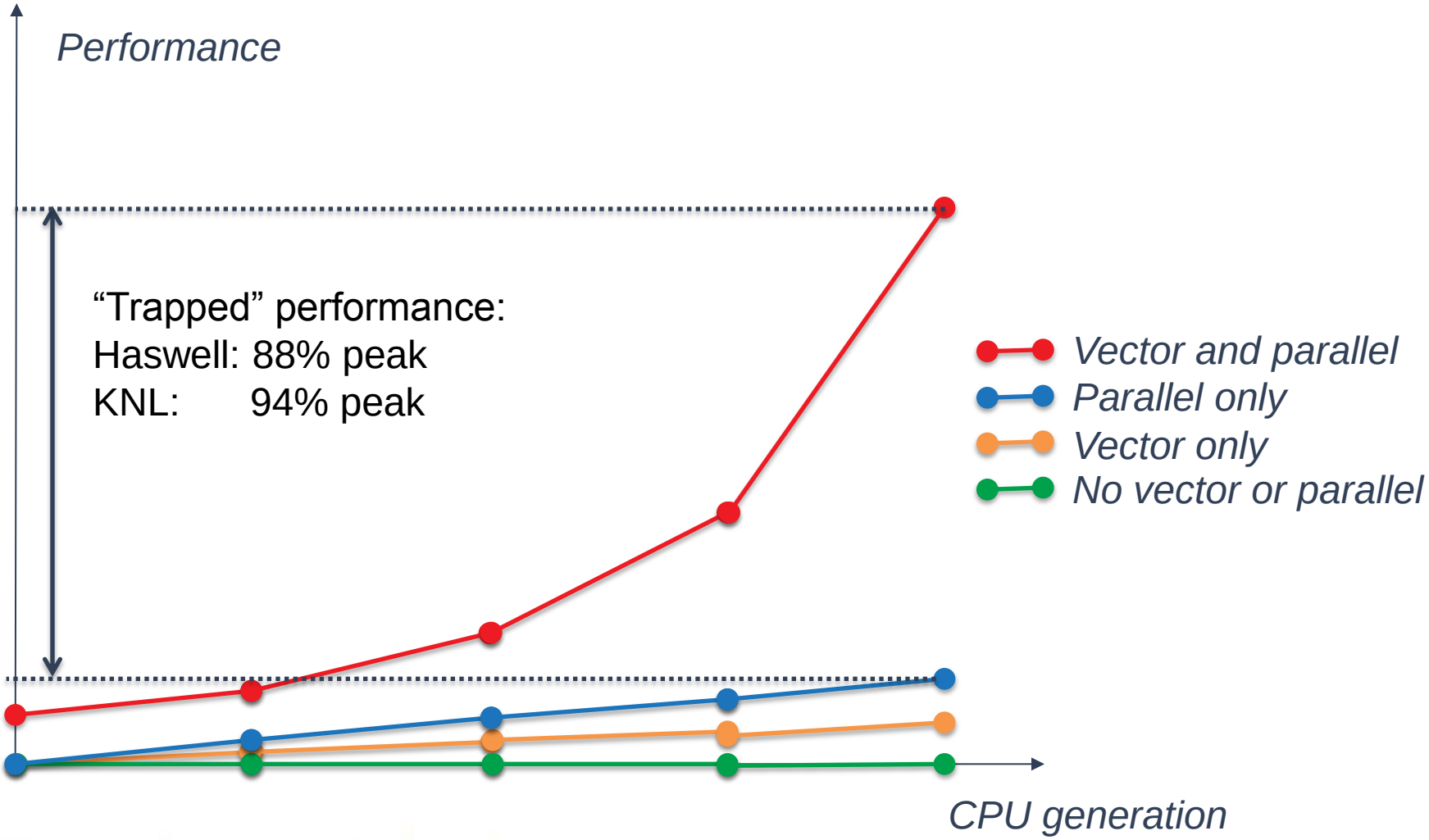


What is vectorization?



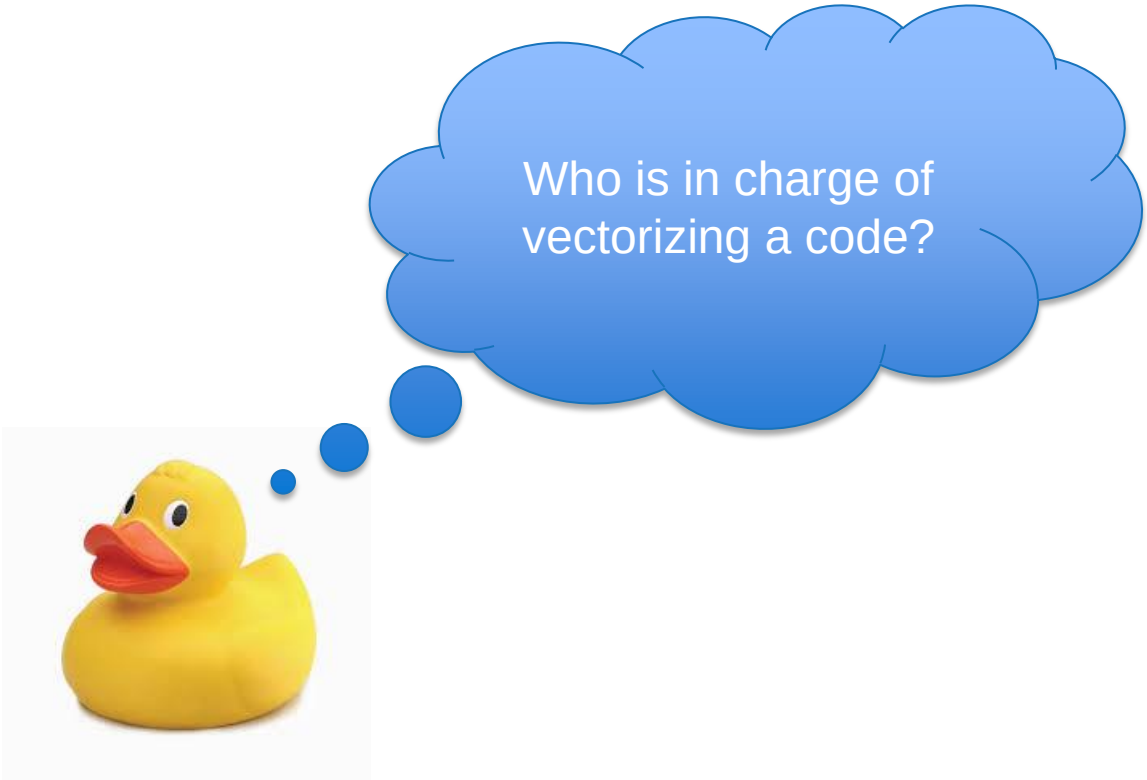
Intel Haswell: 256-bit vector unit ➔ 8 SP / 4 DP
Intel Knights Landing: 512-bit vector unit ➔ 16 SP / 8 DP

Why is vectorization important?



allinea

Question



When can my code use vectorization?

- **A compiler can vectorize a loop if:**
  - The loop is countable at runtime
  - There is a single control flow within the loop
  - The loop does not contain function calls

Is this enough? No!

- Data dependencies

  can stop vectorization

Consider the loop:

a= {0,1,2,3,4}

b = {5,6,7,8,9}

```
for( i=1; i<N; i++)
  a[i] = a[i-1] + b[i];
```

Applying each operation sequentially:

a[1] = a[0] + b[1] → a[1] = 0 + 6 → a[1] = 6

a[2] = a[1] + b[2] → a[2] = 6 + 7 → a[2] = 13

a[3] = a[2] + b[3] → a[3] = 13 + 8 → a[3] = 21

a[4] = a[3] + b[4] → a[4] = 21 + 9 → a[4] = 30

a = {0, 6, 13, 21, 30}

When can my code use vectorization?

- **A compiler can vectorize a loop if:**
  - The loop is countable at runtime
  - There is a single control flow within the loop
  - The loop

Is this enoug

- Data deper

  can stop vectorization

Now let's try vector operations:

a= {0,1,2,3,4}

b = {5,6,7,8,9}

```
for( i=1; i<N; i++)
  a[i] = a[i-1] + b[i];
```

Applying vector operations, i={1,2,3,4}:

a[i-1] = {0,1,2,3} (load)

b[i] = {6,7,8,9} (load)

{0,1,2,3} + {6,7,8,9} = {6, 8, 10, 12} (operate)

a[i] = {6, 8, 10, 12} (store)

```
i=1; i<N; i++)
i] = a[i-1] + b[i];
```

a[1] = a[0] + b[1] → a[1] = 0 + 6 → a[1] = 6

a[2] = a[1] + b[2] → a[2] = 6 + 7 → a[2] = 13

a[3] = a[2] + b[3] → a[3] = 13 + 8 → a[3] = 21

a[4] = a[3] + b[4] → a[4] = 21 + 9 → a[4] = 30

a = {0, 6, 13, 21, 30}

allinea

How to actually produce vectorized binaries?

- **Step 1: adopt a “vector-aware” coding methodology**
  - Make sure the code can be vectorized (see above)
  - Enable vectorization during compilation (e.g. –xhost, …)
  - Tell the compiler he can vectorize (#pragma ivdep)
  - Use optimized mathematical libraries as much as possible

- **Step 2: use tools to help you…**

… not just any tool. Use the Right Tools.

```
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(243): (col. 7) remark: loop was not vectorized: existence of vector depende
nce.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(753): (col. 13) remark: loop was not vectorized: vectorization possible but
 seems inefficient.
mg.f(762): (col. 13) remark: loop was not vectorized: vectorization possible but
 seems inefficient.
mg.f(749): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(746): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2255): (col. 16) remark: loop was not vectorized: existence of vector depen
dence.
mg.f(2254): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2251): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(2433): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(2433): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(527): (col. 7) remark: loop was not vectorized: nonstandard loop is not a v
ectorization candidate.
mg.f(552): (col. 7) remark: loop was not vectorized: nonstandard loop is not a v
ectorization candidate.
mg.f(1150): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1150): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1645): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1655): (col. 7) remark: loop was not vectorized: loop was transformed to me
```

… maybe?

allinea

allinea

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

Interactive demonstration

Adding vectorization to a linear algebra application

allinea

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

Coffee break

allinea

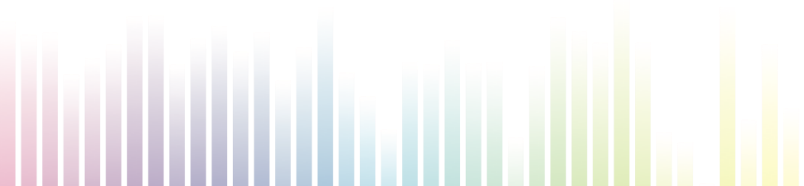
High performance tools to debug, profile, and analyze your applications

Technical session

Preparing applications for large scale

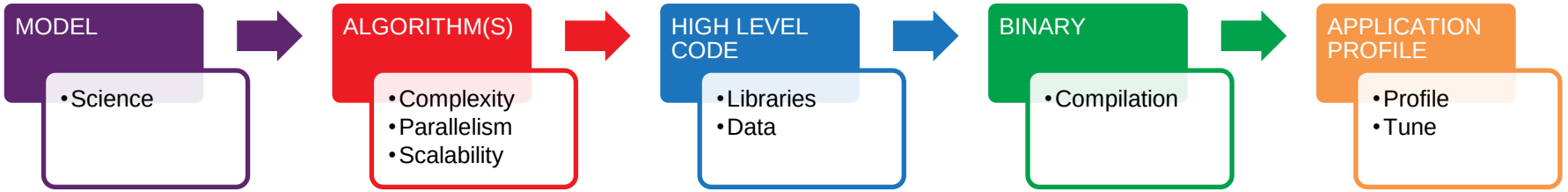
Agenda: Preparing applications for large scale

- Application design life-cycle
- Key concepts to address scalability issues
- Identifying the limiting factors

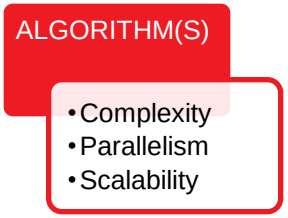


allinea

Building a scientific application



In your opinion, what is the most critical step?



Example: a sheet of paper (0.1 mm thick)

How many times do you need to fold a sheet of paper to make it as thick as the Eiffel Tower (300m)?

Answer: 22 times

Because: $0.1\text{mm} * 2^{22} = 419\text{m}$

allinea

Building a scientific application

MODEL
- Science

ALGORITHM(S
- Complexi
- Parallelis
- Scalability

npilation

APPLICATION PROFILE
- Profile
- Tune

In your op ical step?

Ex 1 mm thick)

Ho fold a sheet of paper

to r Tower (300m)?

An

Because: 0.1mm * 2^{22} = 419m

ALGORITHM(S)
- Complexity
- Parallelism
- Scalability

allinea

Algorithms Complexity Analysis

Definition:

> In computer science, the time complexity of an algorithm quantifies the amount of time taken by the algorithm to run,
> as a function of the length of the string representing the input.

Example: “Schoolbook Matrix Multiplication”

```
Input:    matrices A(n,m) and B(m,p)
Output:   matrice C(n,p)
For i from 1 to n:
     For j from 1 to p:
          Let sum = 0
          For k from 1 to m:
               Set sum ← sum + A_ik × B_kj
          Set C_ij ← sum
Return C
```

The calculation time is a function of

“n x p x m”

The time complexity of this algorithm is written $O(npm)$ or $O(n^3)$.

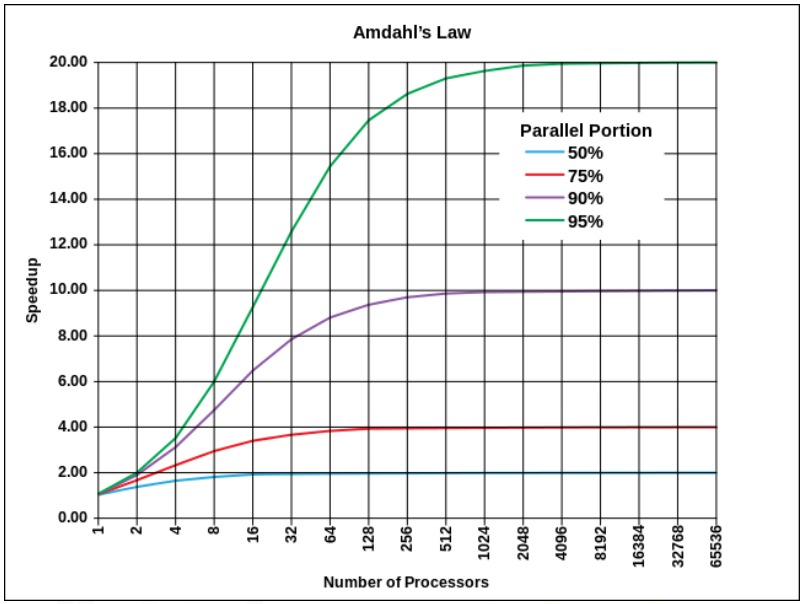
This is a **cubic complexity**...

… which is terrible!

Amdahl’s Law

Definition:

Amdahl's law gives the theoretical speedup of the execution of a task *at fixed workload* that can be expected of a system whose resources are improved.



Example:

A sequential programs runs in 20h.

A 1h-long part of this program cannot be parallelized (parallel portion = 95%)

Amdahl’s law says:

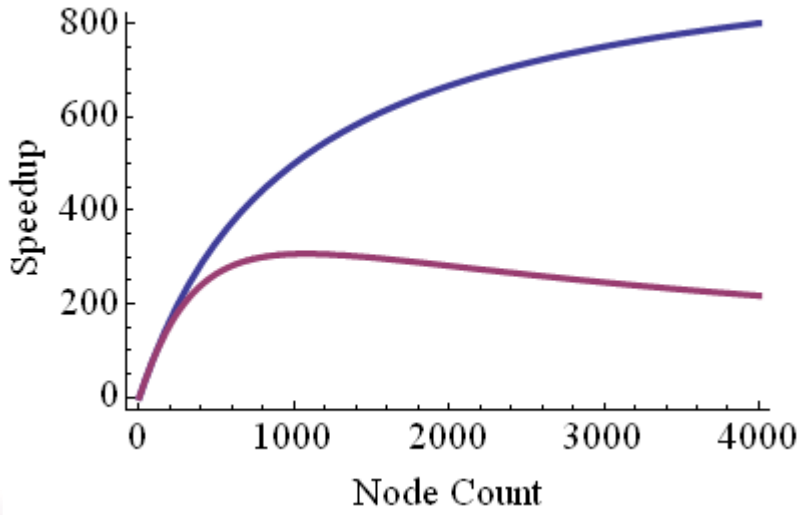
This program will never run in less than 1h (speedup of 20).

Scalability in practice

Total Time = Computation + Message Latency + Message Transfer + Noise

Computation = Serial time + (Parallel Time / N)

Message Latency = number of messages * latency
Message Transfer = size of messages / bandwidth
Noise = pre-empting processes, swapping, faulting pages…



Blue : Amdahl's law

Purple : Amdahl's law with messages

Tips for better scalability

Communication between neighbors as opposed to global

- All tasks communicate to all tasks: $O(n^2)$!
- Tasks communicate with their neighbors only: O(1)!

Try and avoid global communications!

Improve load balance to reduce synchronization

- When processes synchronize, they go as quickly as the slowest

Check the time spent in synchronization tasks (e.g. MPI_Barrier, etc.)

Monitor the supercomputer noise

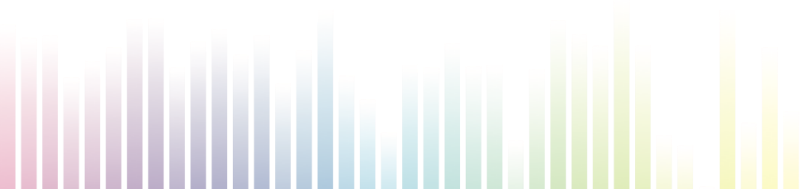
- Lost microseconds and milliseconds can turn into seconds

Pin your tasks, etc…

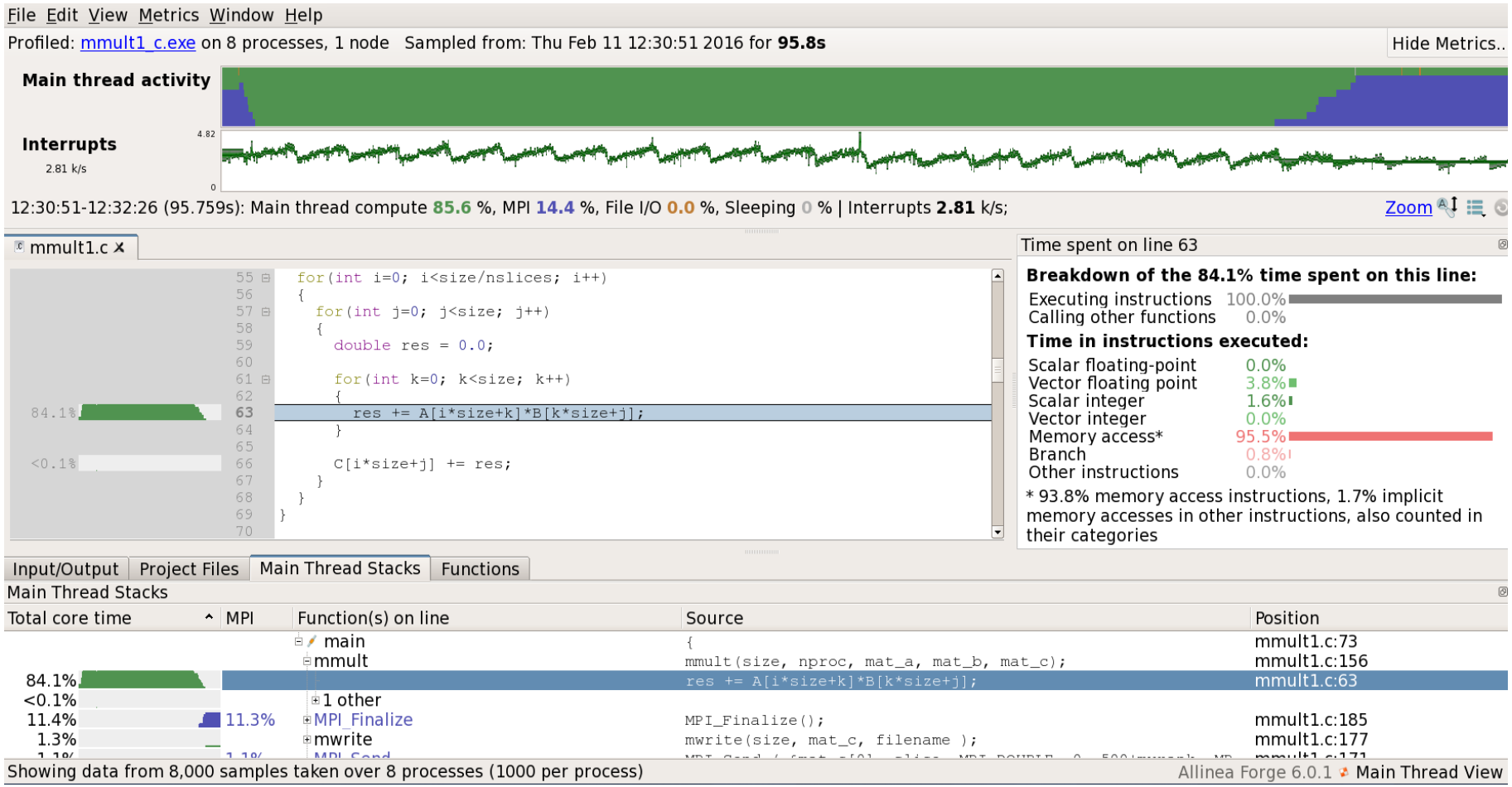
So… what is a scalable algorithm?

As a rule of thumb, an application will scale if:

- Its algorithms have a good complexity
- Its workload can be split into independent tasks
- Communications are infrequent or unnecessary
- Lots of calculations take place before messaging or I/O occurs
- The system is under control
- All the above remains true as the number of tasks grows.



Talking about system under control…



allinea

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

Interactive demonstration

Preparing an application for large scale

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

allinea

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

A real life example

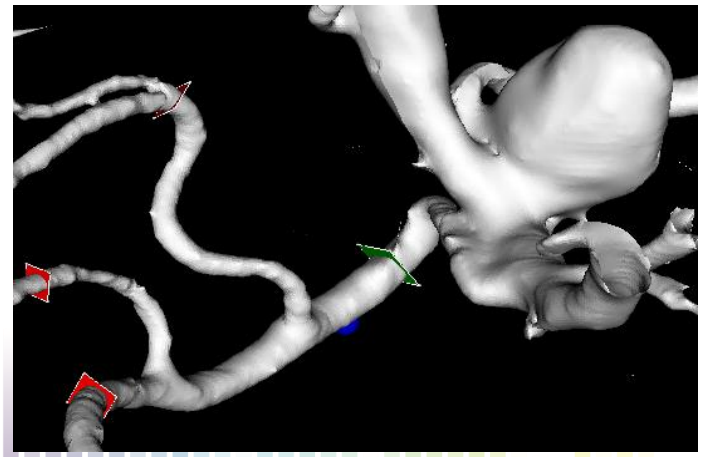
Bringing HemelB to Petascale with Allinea

UCL: HPC to fight aneurysms and save lives

- **Challenges**

  Make surgical decisions within minutes (instead of hours) following patients MRI scans

- **UCL got HemelB to perform and scale to above 50k cores**
  - Fixed limiting performance bottlenecks and crashes at scale
  - Study hemodynamics within the Circle of Willis for the 1st time

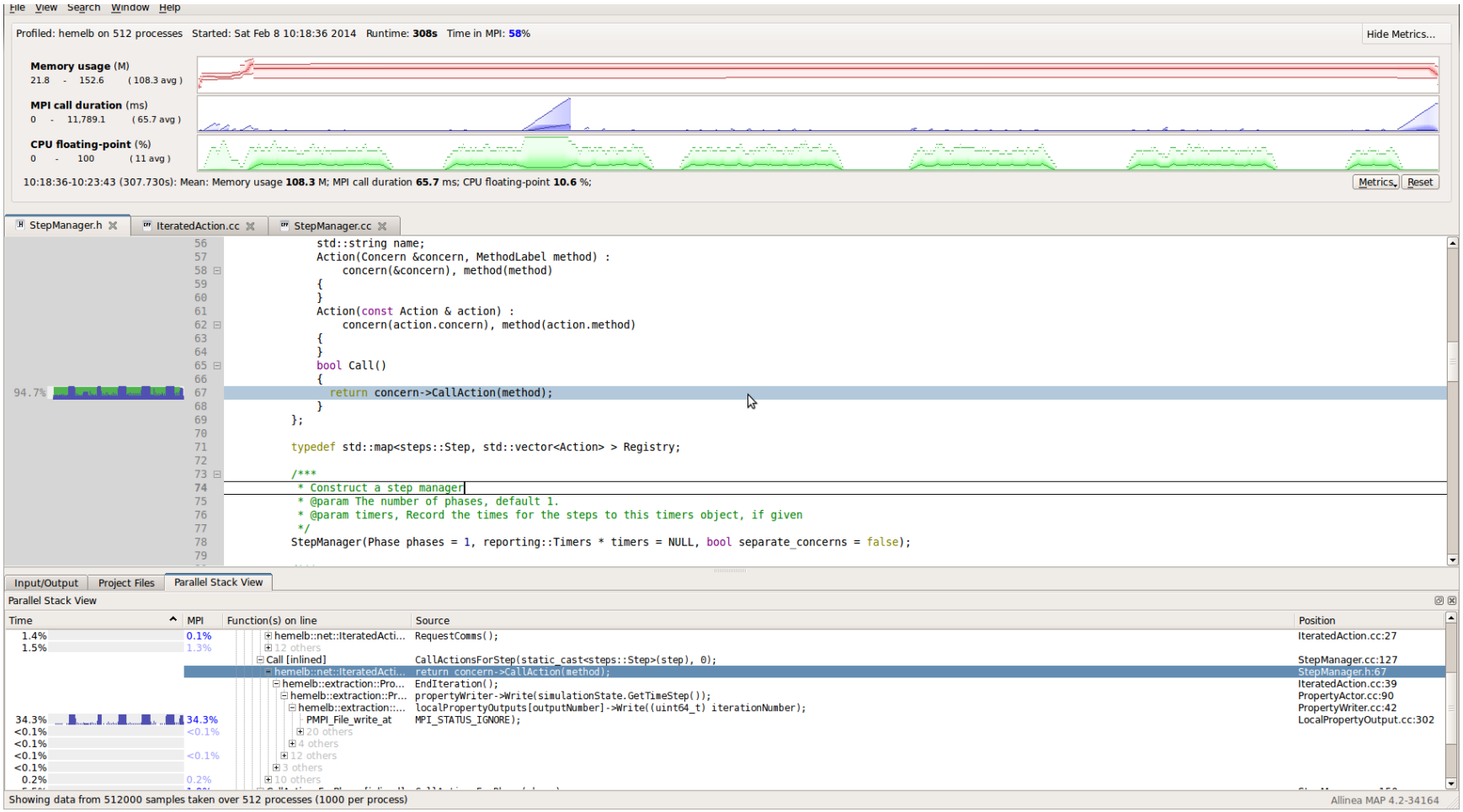


Full case study:

http://www.allinea.com/case-studies/education/application-soars-above-petascale-after-tools-collaboration

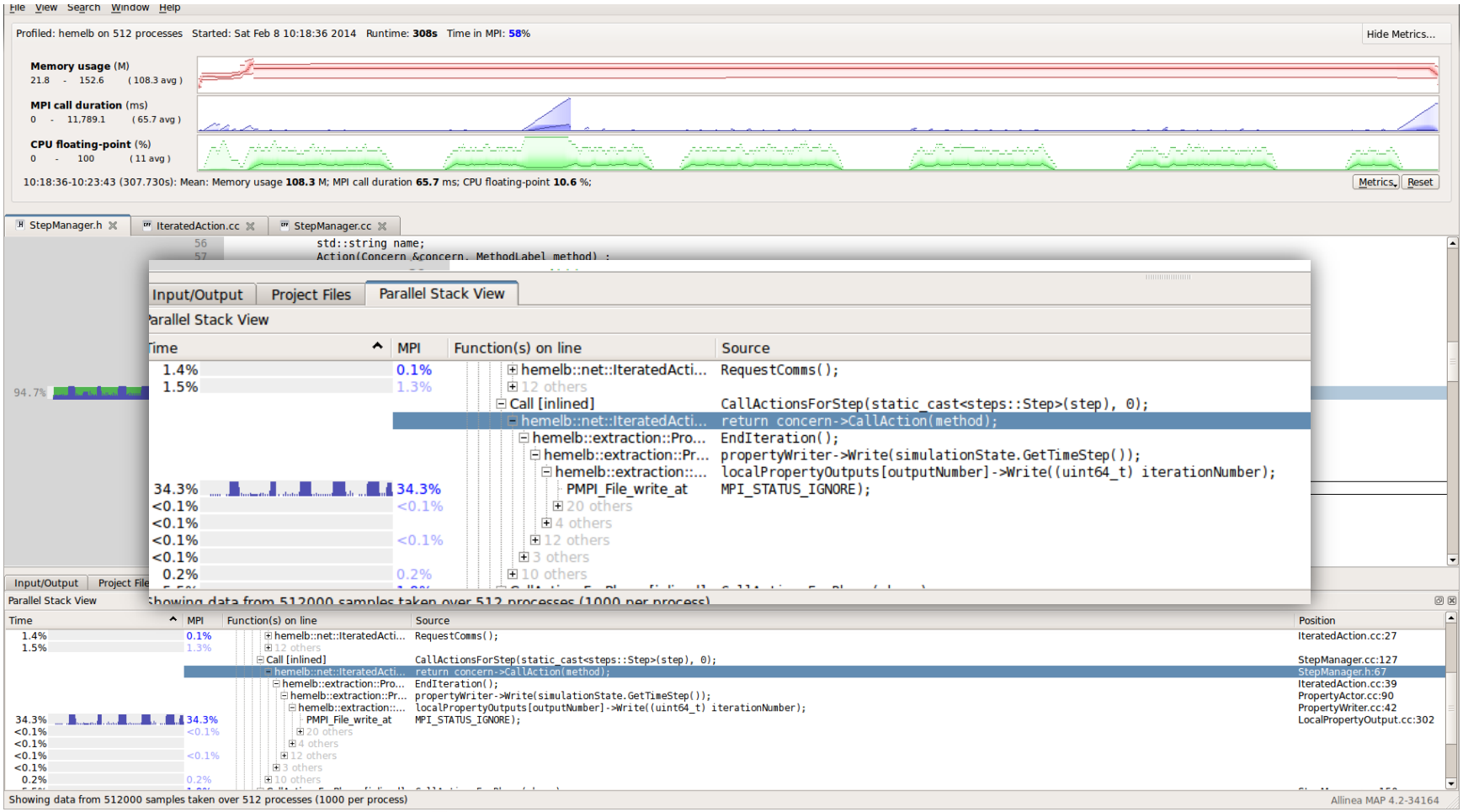
allinea

Scaling issue – at 512 processes



allinea

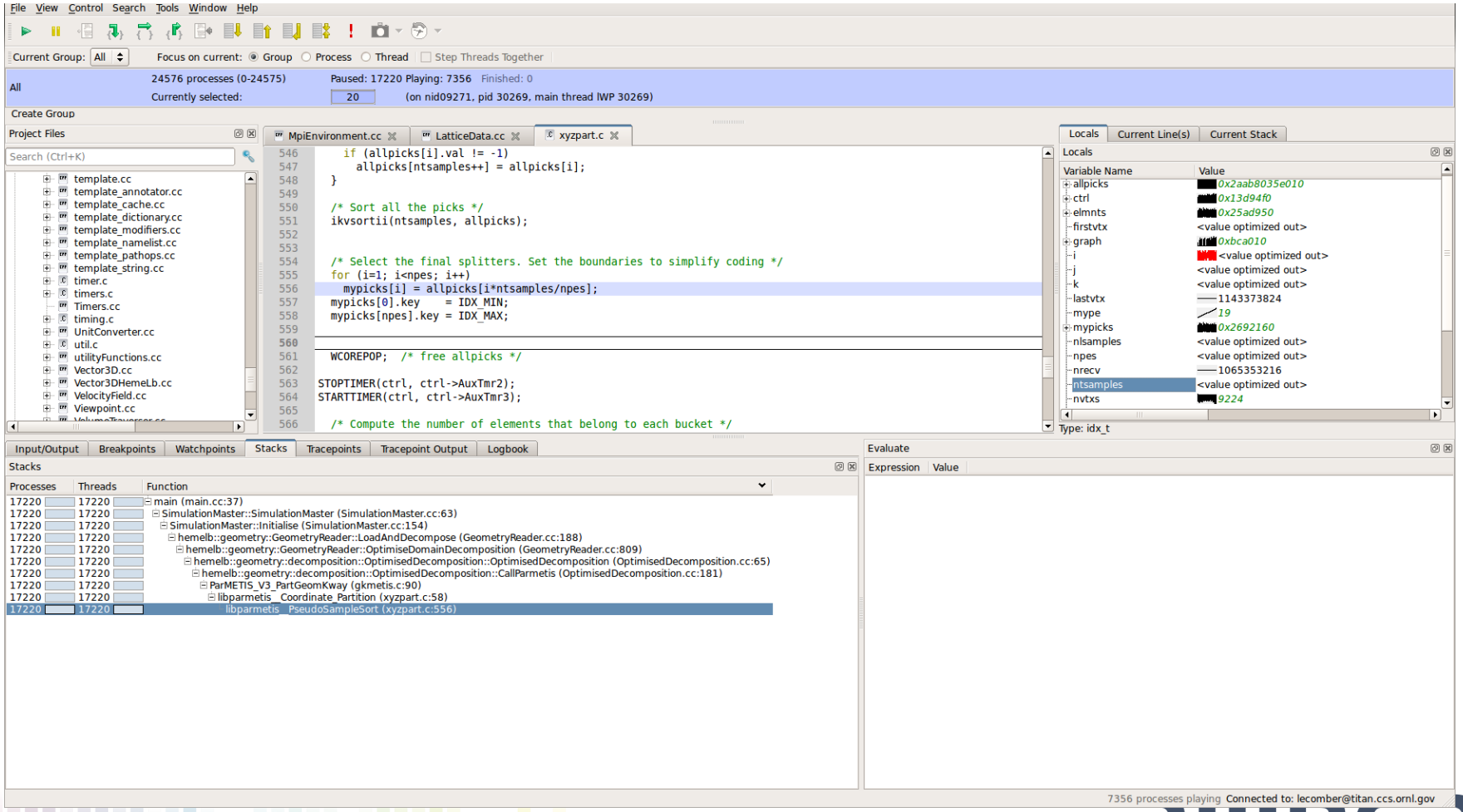
Scaling issue – at 512 processes



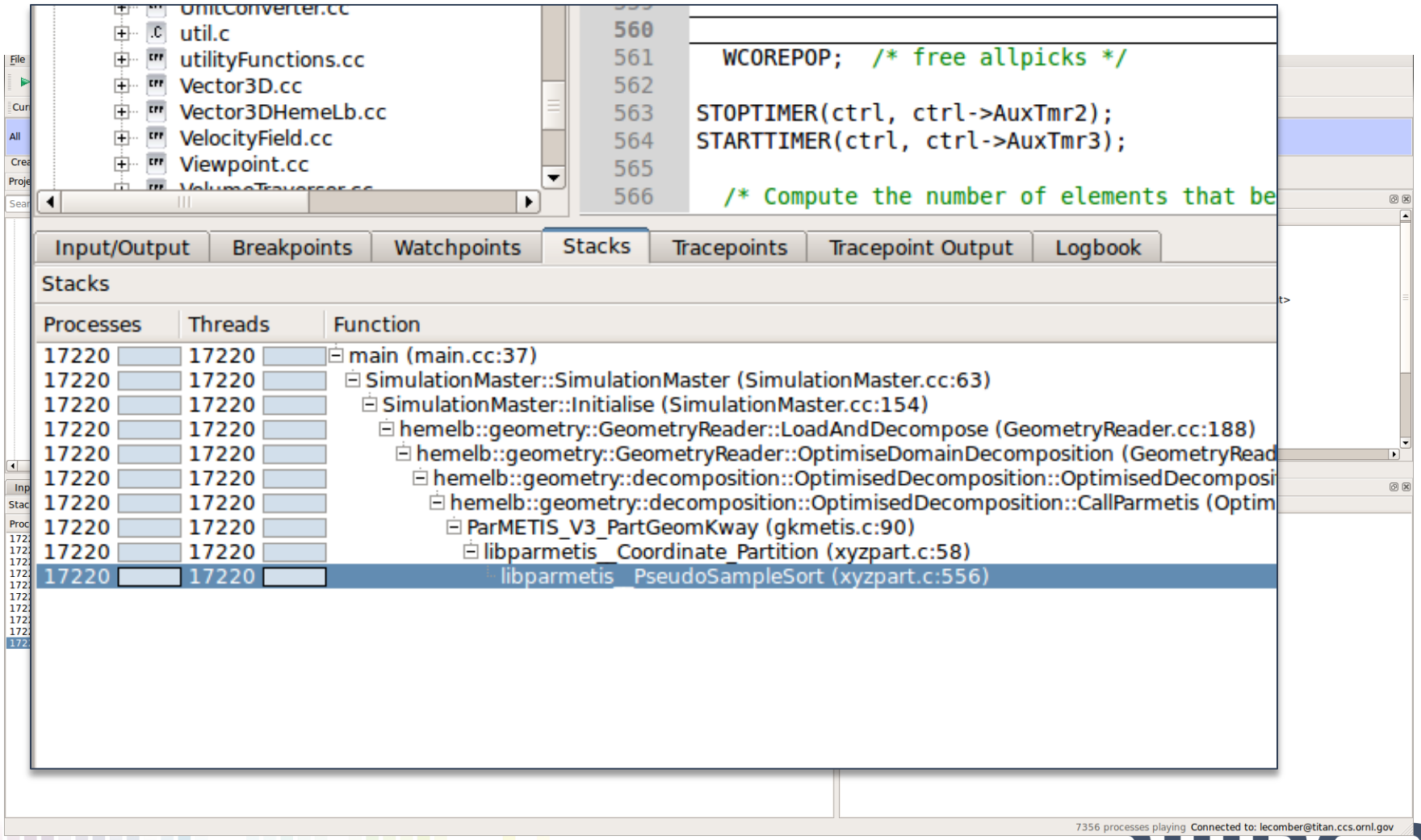
Simple fix… reduce periodicity of output

allinea

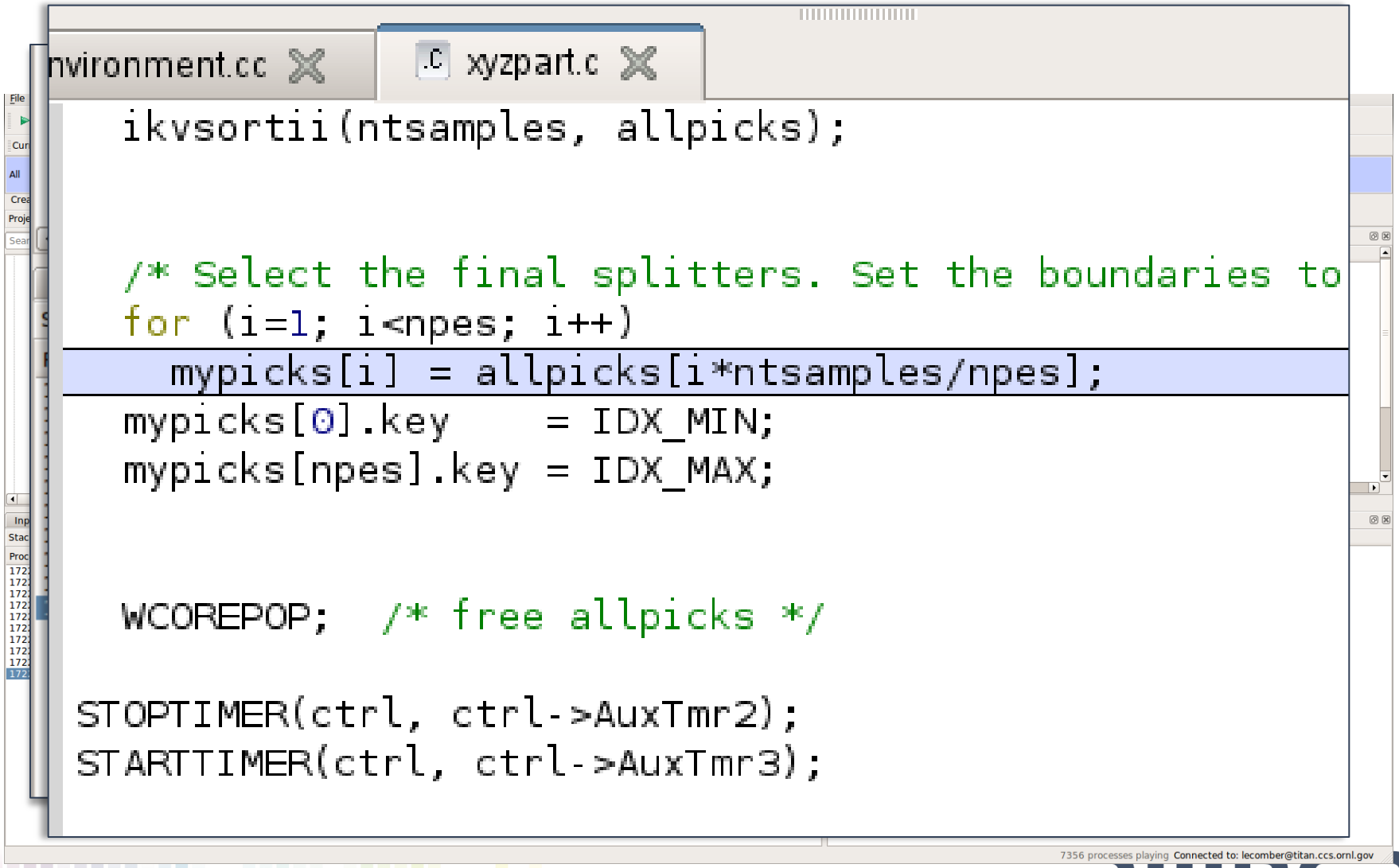
… leading to a bug!



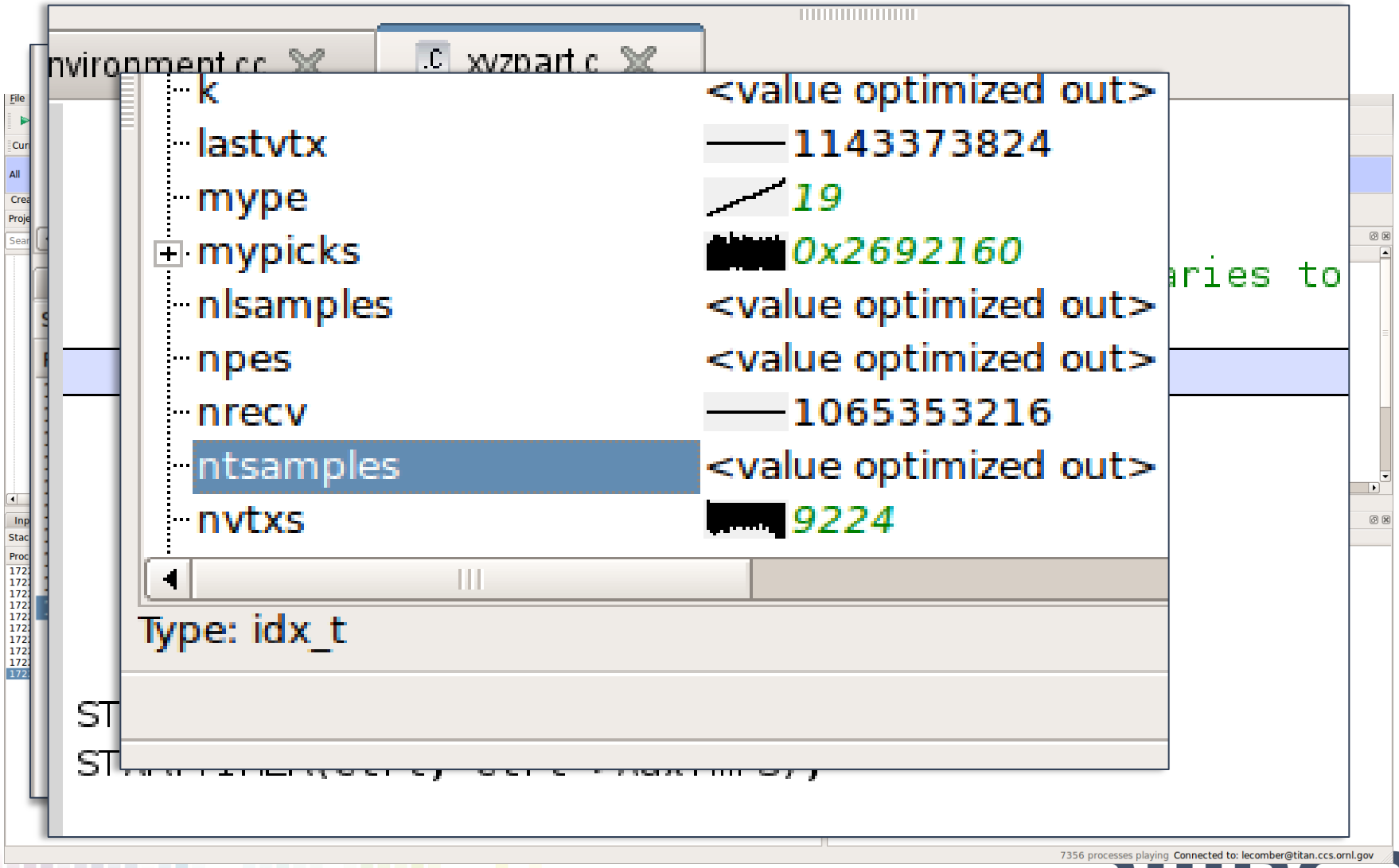
Looking at the bug location in the code



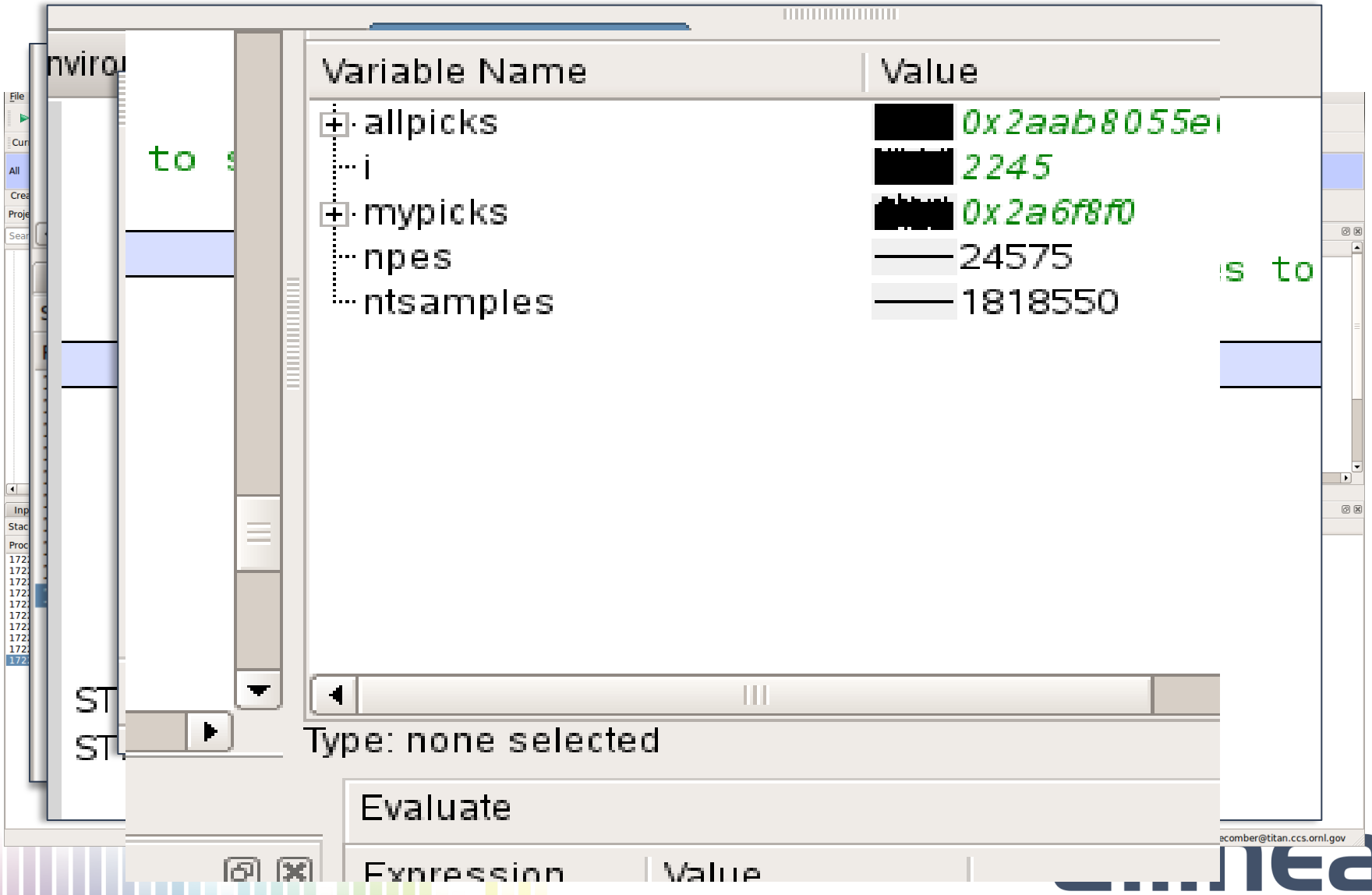
The stack highlights this particular line



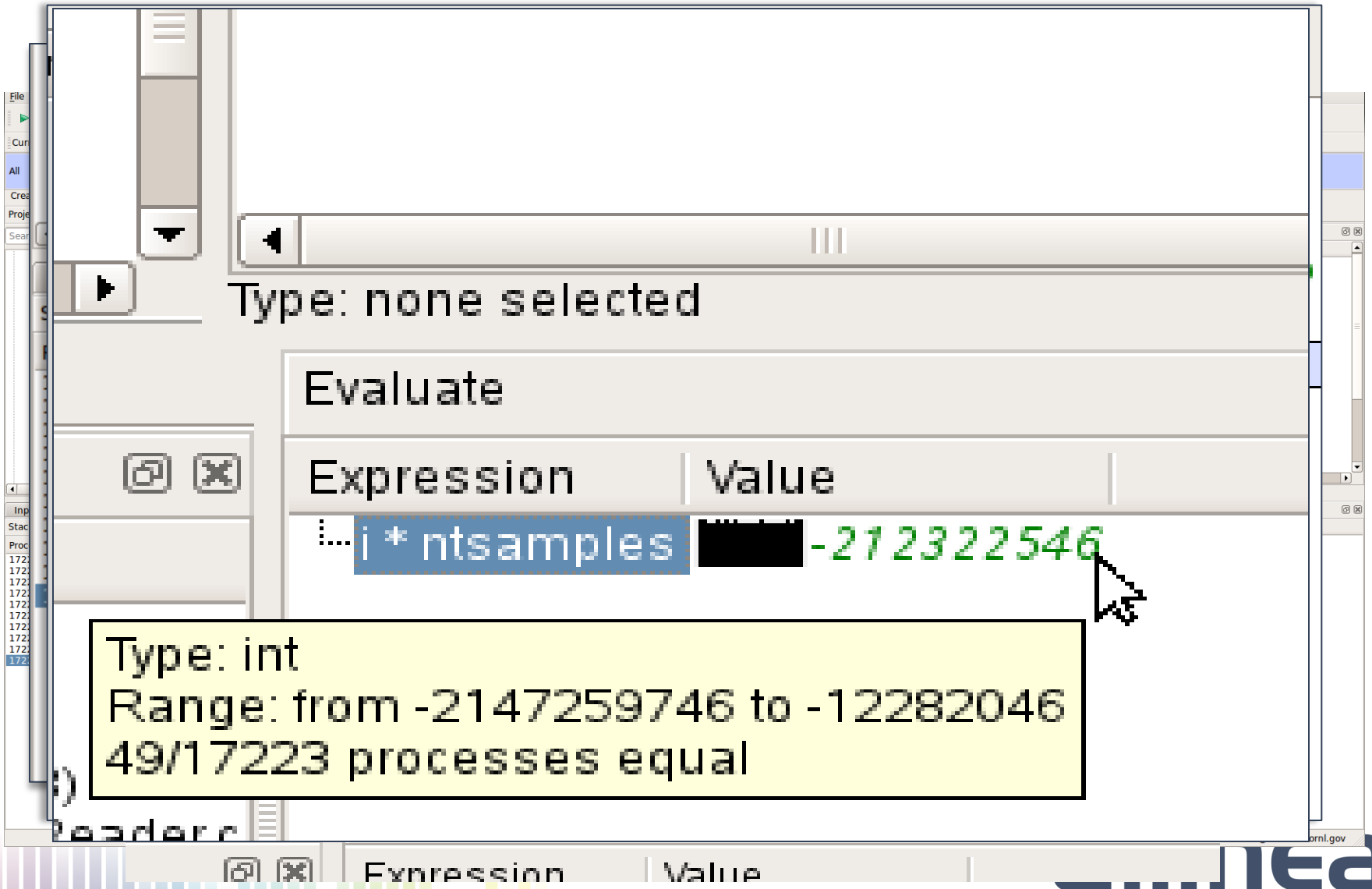
Here are the related variables… Optimized out by the compiler ! Need to compile with –O0!



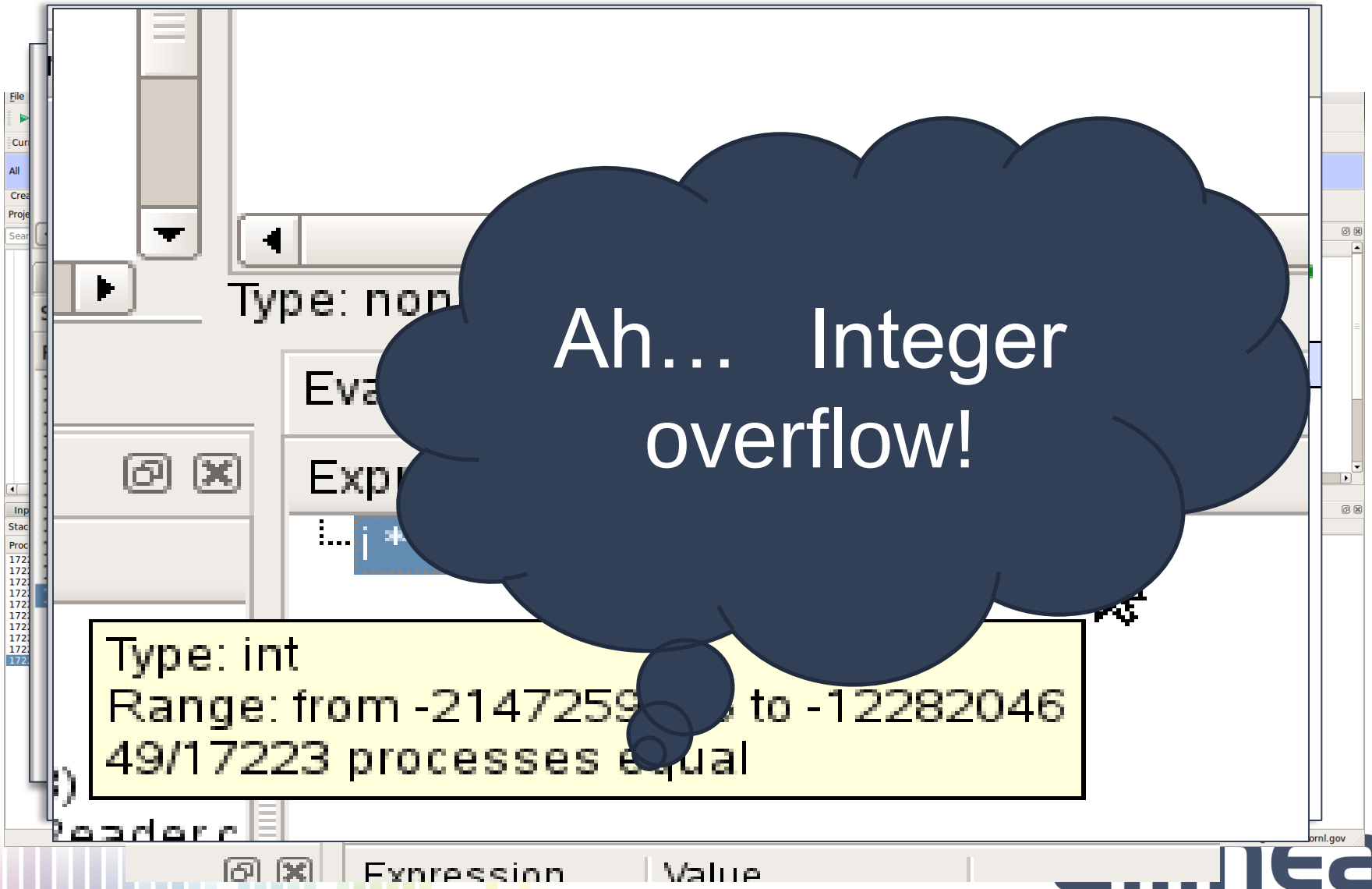
Variables are much clearer when compiled with low optimization options. Looking good!

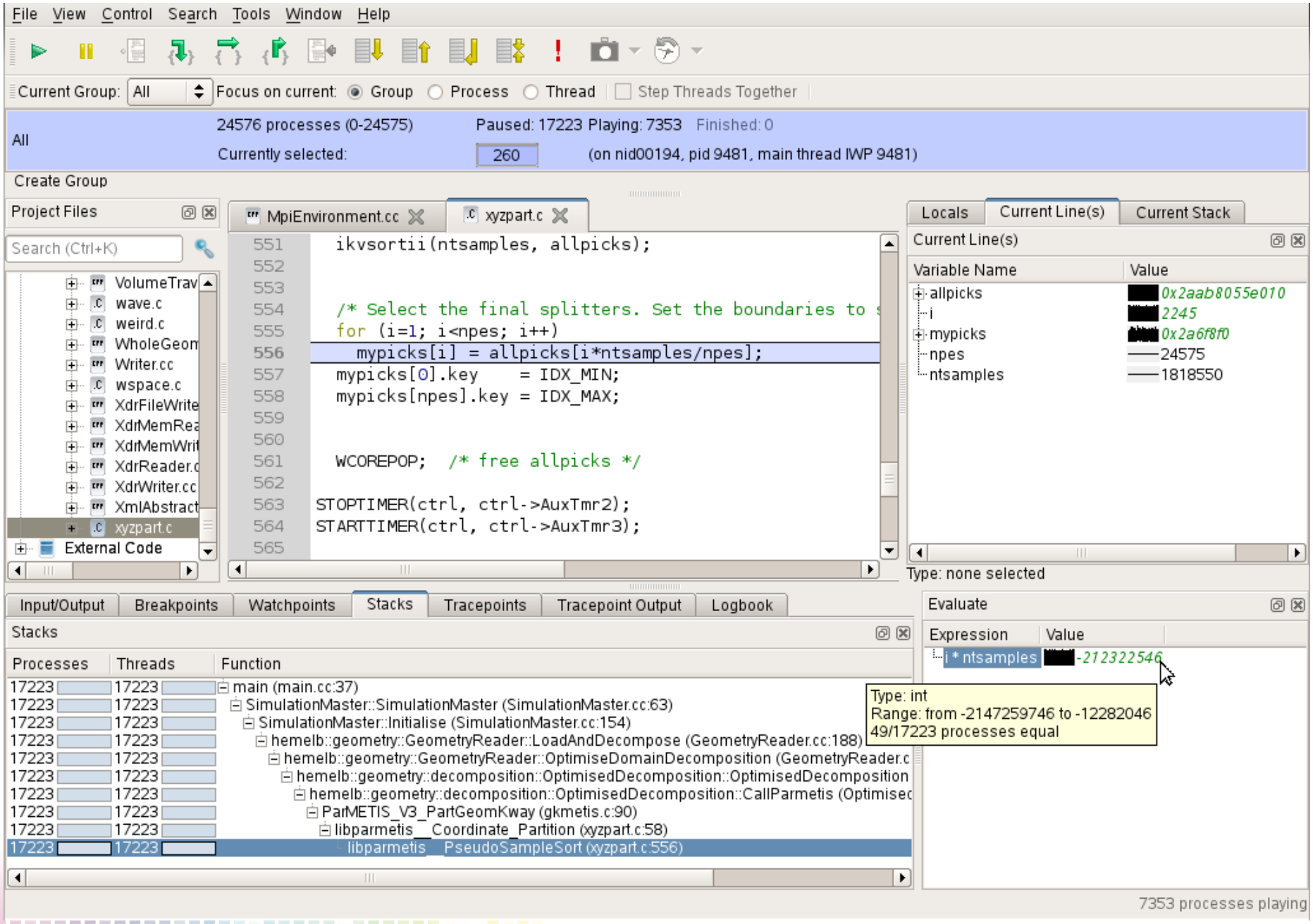


But the actual array index is not looking good at all… this shows…



...


Ah… Integer overflow!
Type: int
Range: from -2147259 to -12282046
49/17223 processes equal


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
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 (Optimised
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

allinea

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

Allinea new stuff and roadmap

More is available under non-disclosure agreement!

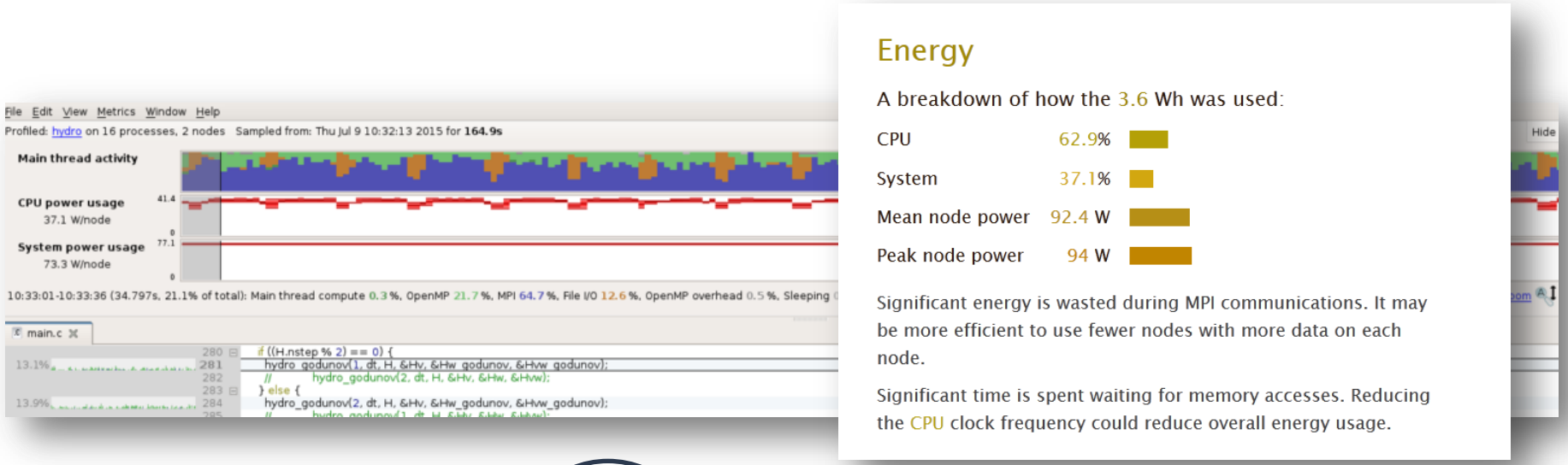
allinea FORGE

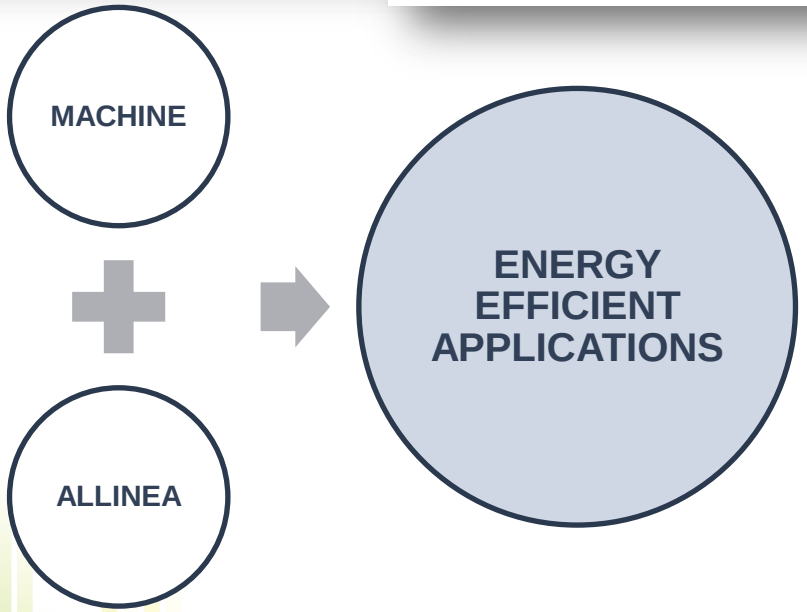
allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

Energy efficiency with Allinea's tools





allinea

Quantify gains immediately

CPU RUN

CPU

A breakdown of the 94.6% CPU time:

Scalar numeric ops	11.7%
Vector numeric ops	0.0%
Memory accesses	88.2%
Waiting for accelerators	0.0%

The per-core performance is memory-bound. Use a profiler to identify time-cons

No time is spent i
vectorization advi

Energy

A breakdown of how the 3.6 Wh was used:

CPU	62.9%
System	37.1%
Mean node power	92.4 W
Peak node power	94 W

Significant energy is wasted during MPI communications. It may be more efficient to use fewer nodes with more data on each node.

Significant time is spent waiting for memory accesses. Reducing the CPU clock frequency could reduce overall energy usage.

Accelerators

A breakdown of how accelerators were used:

GPU utilization	0.0%
Global memory accesses	0.0%
Mean GPU memory usage	0.0%
Peak GPU memory usage	0.0%

GPUs are available but are not used. Identify suitable hot loops with a profiler and try offloading them to the accelerator.

The peak device memory usage is low. It may be more efficient to offload a larger portion of the dataset to each device.

GPU RUN

CPU

A breakdown of the 70.2% CPU time:

Scalar numeric ops	2.5%
Vector numeric ops	0.0%
Memory accesses	39.7%
Waiting for accelerators	61.7%

Most of the time is spent waiting for accelerators. Use asynchronous calls to overlap CPU and accelerator workloads.

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

Energy

A breakdown of how the 2.84 Wh was used:

CPU	28.4%
System	71.6%
Mean node power	163 W
Peak node power	175.8 W

Energy usage appears to be optimal.

Accelerators

A breakdown of how accelerators were used:

GPU utilization	92.5%
Global memory accesses	40.4%
Mean GPU memory usage	9.6%
Peak GPU memory usage	15.2%

Significant time is spent in global memory accesses. Try modifying kernels to use shared memory instead and check for bad striding patterns.

The peak device memory usage is low. It may be more efficient to offload a larger portion of the dataset to each device.

allinea

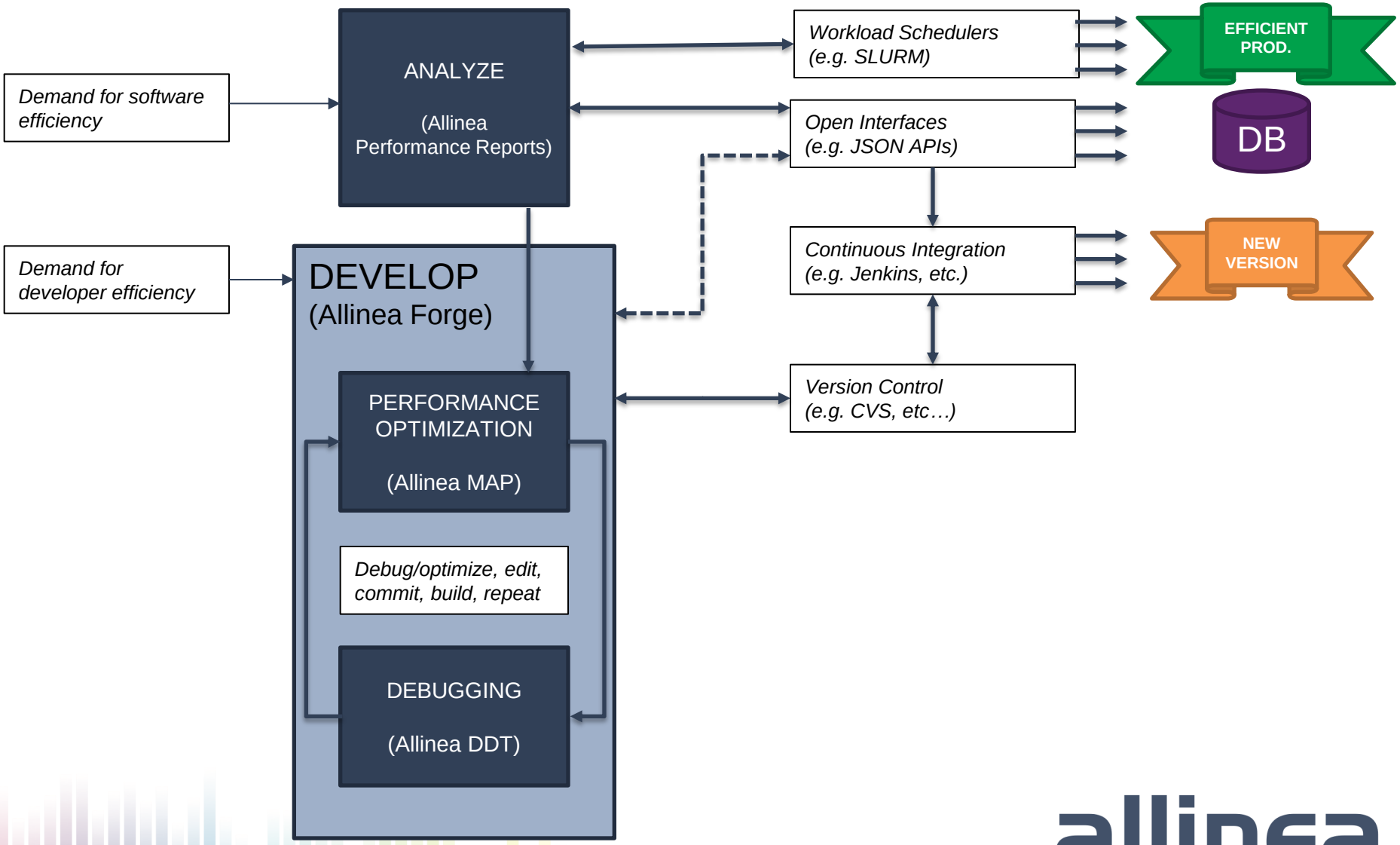
Allinea R&D Programs in preparation for Exascale

- **NRE Projects with HPC Centers**
  - ORNL: Application-level Trapped Capacity Reports
  - CEA : Providing a scalable interface to CEA profiler (MALP)
- **European projects**
  - Mont Blanc 2 : R&D on HPC systems using embedded technologies
  - Horizon 2020 : Towards Exascale computing
    - **ESIWACE** : Excellence in Simulation of Weather and Climate in Europe
    - **NextGenIO** : Next Generation IO for Exascale
    - **SAGE** : Percipient Storage for Exascale Data-Centric Computing
    - **ExaNest** : European Exascale System Interconnect and Storage
    - **ComPat** : Computing Patterns for High Performance Multiscale Computing
- **National Projects**
  - TSERO (UK) : Reducing energy consumption of HPC systems

Support for next generation systems

- **Intel KNL Support**
  - Support announced at ISC'2016 in Frankfurt (2 phases release)
- **ARMv8 Support (currently supported: Allinea Forge)**
  - Adding CPU metrics support for ARMv8 in Allinea MAP
  - Allinea Performance Reports for ARMv8 scheduled (H2020 ExaNest)
- **Nvidia GPUs Support (currently supported: CUDA 7.5)**
  - CUDA 8.0 expected as soon as Nvidia is ready
- **OpenPower Support (currently supported: Allinea Forge)**
  - Adding CPU metrics support for Power in Allinea MAP
  - Allinea Performance Reports for OpenPower in R&D

Our vision: HPC best practices



allinea

Summary

- Change is inevitable
- Reduce your costs
- Allinea's mission.

allinea

allinea

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

Thank you !

Technical Support team : support@allinea.com
Sales team : sales@allinea.com