Allinea Tools Workshop

Performance and Reporting Tools

23 February 2015
VI-HPS Workshop
HLRS, Stuttgart

Florent Lebeau
flebeau@allinea.com

Agenda

11:15 – 11:30: Introduction to Allinea tools and latest changes

11:30 – 11:45: Getting started with Allinea Forge

11:45 – 12:30: Profile and Optimise with Allinea MAP

12:30 – 12:45: Allinea Performance Reports

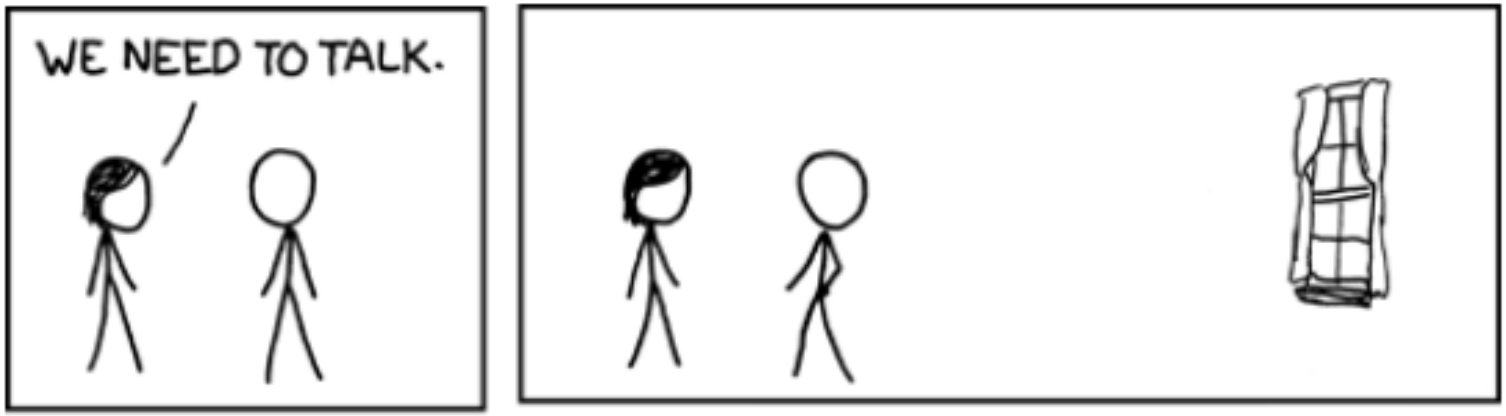
12:45 – 13:45 : Lunch break

13:45 – 17:00: Hands-on session

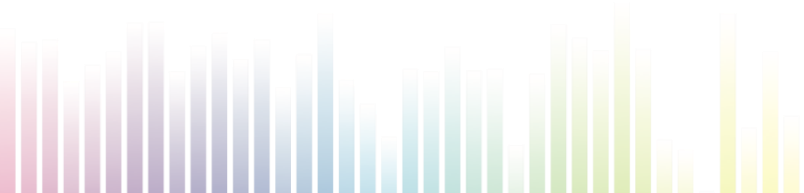
17:00 – 17:30: Wrap-Up and questions

And now…

Let's talk about us!

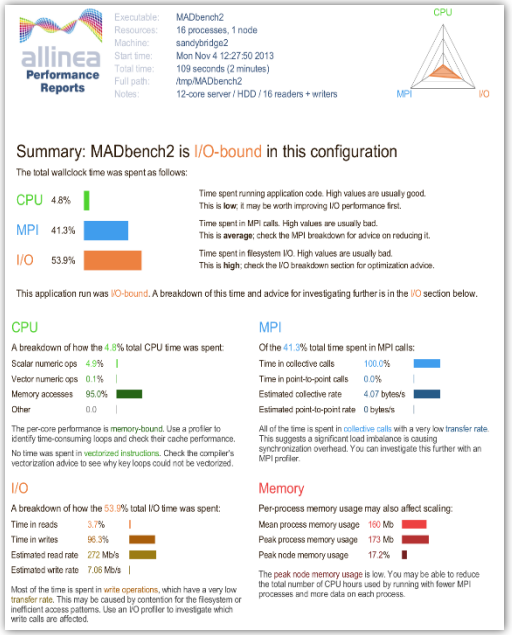


Introduction to Allinea Tools and latest changes

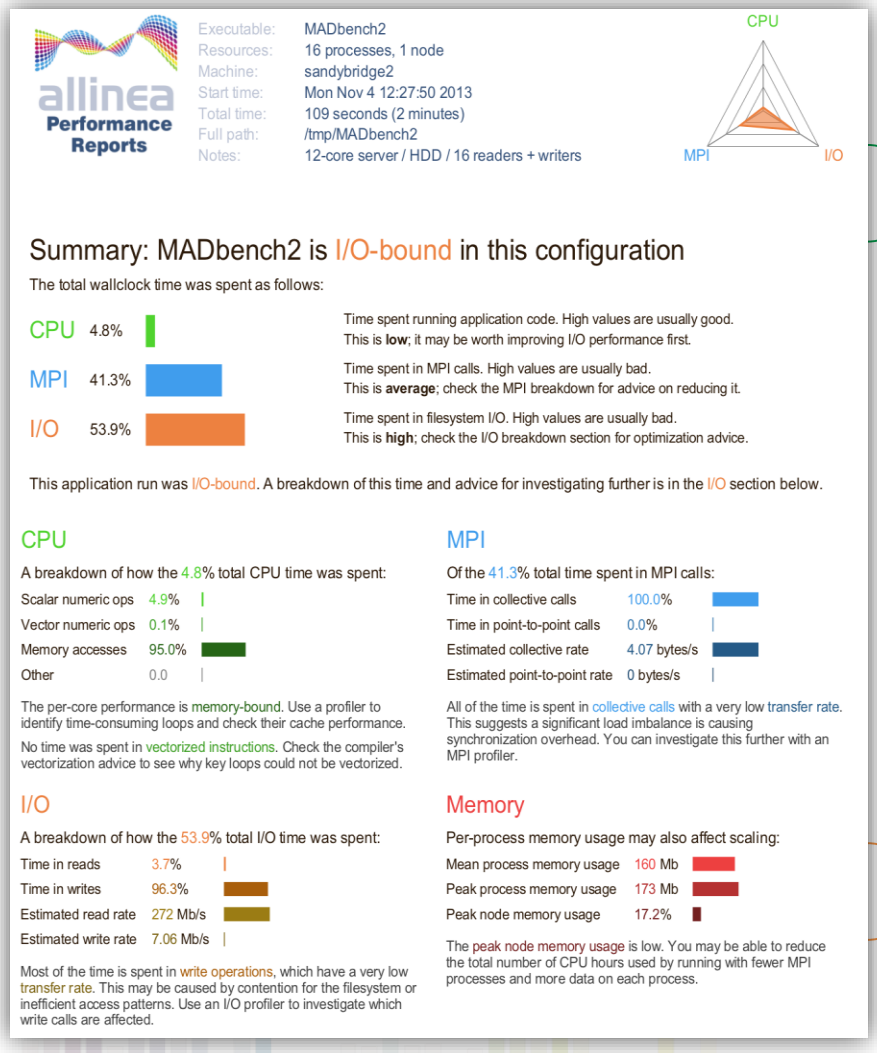


Use Allinea Performance Report to increase cluster efficiency

- **Focus support teams' expertise**
  - Optimising each application would spend ages
  - Need to focus on the ones that are flawed
- **Find candidates for optimisation**
  - How to retrieve relevant metrics?
  - How to minimize the number of benchmarks?
  - How to automate benchmarks on several applications?
- **Generate effortless one-touch reports with allinea**
  - Explicit and readable reports with metrics and explanations
  - Understand optimized HPC applications effortlessly
- **Available to you**
  - Allinea performance report – 3072 tokens



Designed for better runs, quickly


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.

- No instrumentation needed
- No source code needed
- No recompilation needed
- Less than 5% runtime overhead
- Fully scalable
- Run regularly – or in regression tests
- Explicit and usable output

Need to dive into the code ?

- **Allinea Forge: a modern integrated environment for HPC developers**
  - Rebranding of Allinea Unified (Allinea DDT + Allinea MAP)
- **Supporting the lifecycle of application development and improvement**
  - Productively debug code with Allinea DDT
  - Enhance application performance with Allinea MAP
- **Designed for productivity**
  - Consistent easy to use tools
  - Fewer failed jobs
- **Available to you**
  - Allinea forge – 3072 tokens

Allinea Forge
One Unified Solution

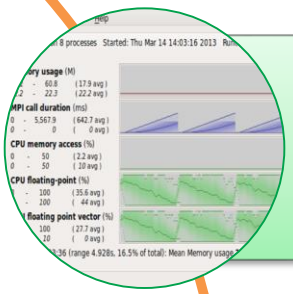
Use Allinea MAP to find a bottleneck

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

Flick to Allinea DDT
Common interface and settings files

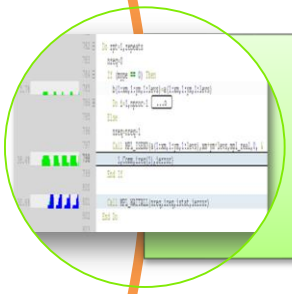
Observe and debug your code step by step

Allinea MAP to find bottlenecks



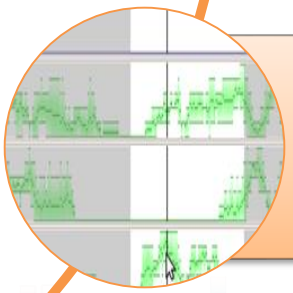
Low overhead measurement

- Accurate, non-intrusive application performance profiling
- Seamless – no recompilation or relinking required



Easy to use

- Source code viewer pinpoints bottleneck locations
- Zoom in to explore iterations, functions and loops



Deep

- Measures CPU, communication, I/O and memory to identify problem causes
- Identifies vectorization and cache performance

Allinea MAP and tracers: a great synergy

Simple optimization with Allinea MAP

- Characterize performance at-scale with a lightweight tool
- See which lines of code are hotspots
- Identify common problems at once

Prepare optimization strategy with Allinea MAP

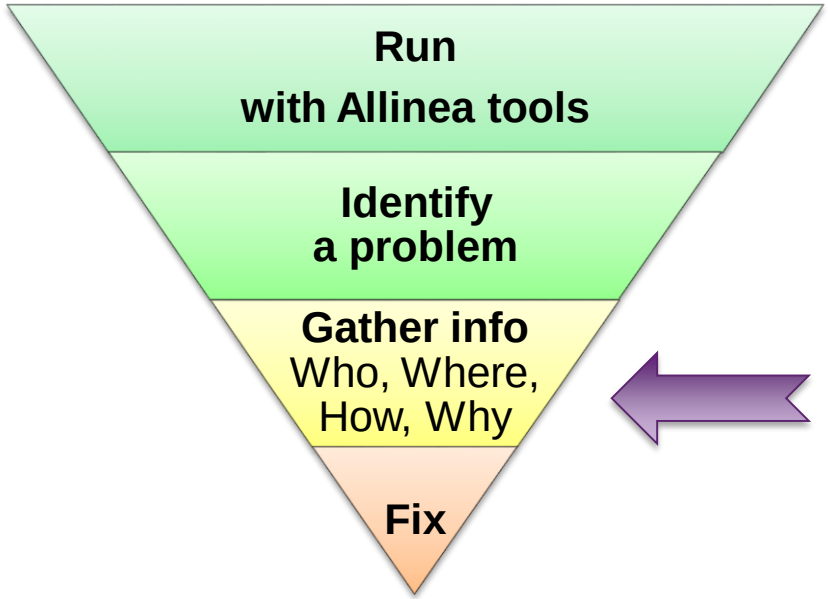
- Identify loop(s) to instrument
- Identify performance counter(s) to record
- Document performance issues to communicate to optimisation experts

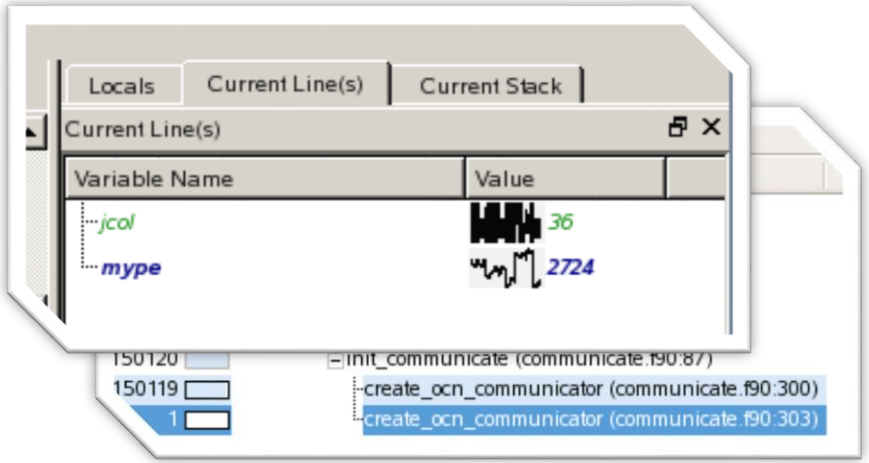
Fine tune the code with tracer

- Retrieve low-level details with tracer
- Fix up CPU usage to make the code fly

Debug your code with Allinea DDT

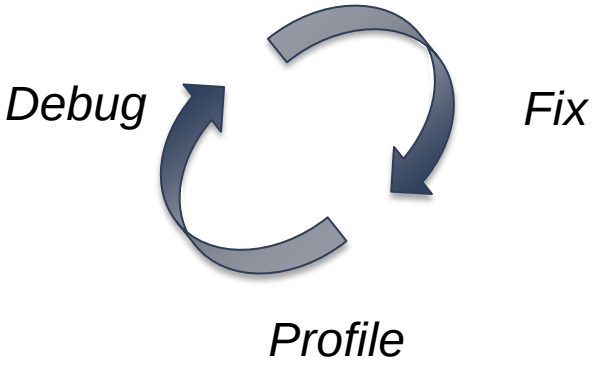
- ***Who* had a rogue behavior?**
  - Merges stacks from processes and threads
- ***Where* did it happen?**
  - Allinea DDT leaps to source automatically
- ***How* did it happen?**
  - Detailed error message given to the user
  - Some faults evident instantly from source
- ***Why* did it happen?**
  - Unique “Smart Highlighting”
  - Sparklines comparing data across processes

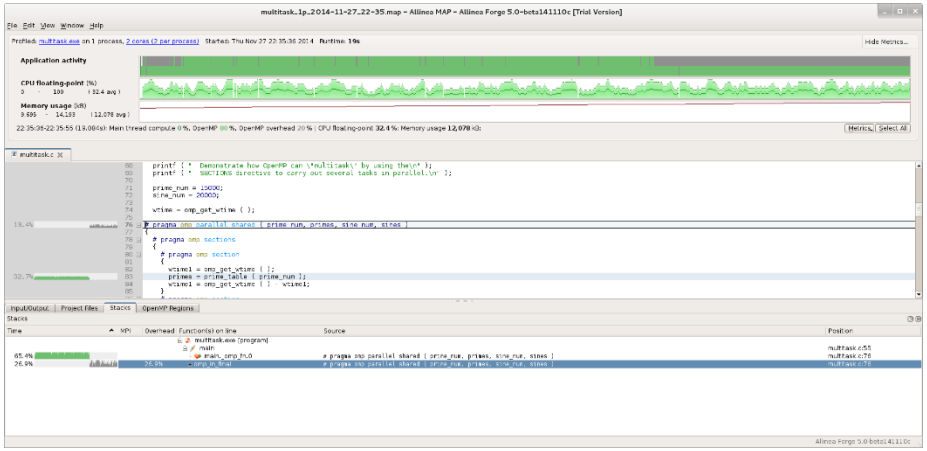




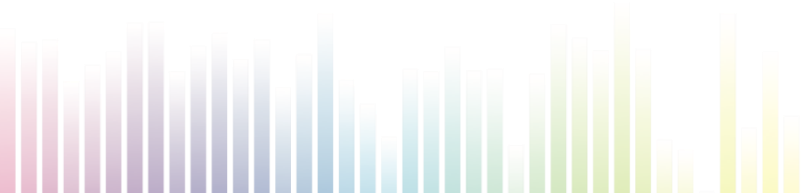
Allinea Forge 5.0 released

- New features:
  - CODE EDITOR:
    - Debug
    - Fix the code
    - Compile and run
    - Profile
    - Optimise the code
    - Compile and run
  - OPENMP PROFILING





Getting Started with Allinea Forge



Get started

1- Connect to HLRS

2- Configure your environment

```
$ . /zhome/academic/HLRS/xhp/xhpfl/env.sh
$ map
```

“Submit through queue” ➔ Configure

Select /zhome/academic/HLRS/xhp/xhpfl/qtf/hornet.qtf in “Submission Template File”

3- Retrieve labs

```
$ cp –r /zhome/academic/HLRS/xhp/xhpfl/allinea_workshop.gz ~
$ tar xzvf allinea_workshop.tar.gz
```

When this is done, please wait for the others

Profile and Optimise with Allinea MAP

Code optimisation is time-consuming.

Relevant metrics help you focus on your application bottleneck.

Start Allinea MAP

- Compile MPI wrapper

```
$ make-profile-libraries --platform=cray --lib-type=static
```

- Prepare the code

```
$ cc –O3 –g map.c –o myapp.exe -Wl,@$(PWD)/allinea-profiler.ld
```

- Start Allinea MAP in interactive mode

```
$ map -n 16 ./myapp.exe arg1 arg2
```

- Start Allinea MAP in profile mode

```
$ map –profile -n 16 ./myapp.exe arg1 arg2
```

Exercise 1: slow

Objectives

- Compile Allinea MAP MPI wrapper libraries
- Compile the code in order to use Allinea MAP
- Submit the job through the queue
- Discover Allinea MAP interface and features

Content

- handout_slow.pdf: instructions
- slow.f90: the example code
- Makefile
- slow.sub: queue submission file without Allinea MAP
- slow.map: a profile example of the application

Key commands

```
$ make
$ qsub slow.sub
$ checkjob $JOB_ID
```

Exercise 2: sqrtmax

Objectives

- Find the application bottleneck using Allinea MAP
- Optimise and improve application speedup

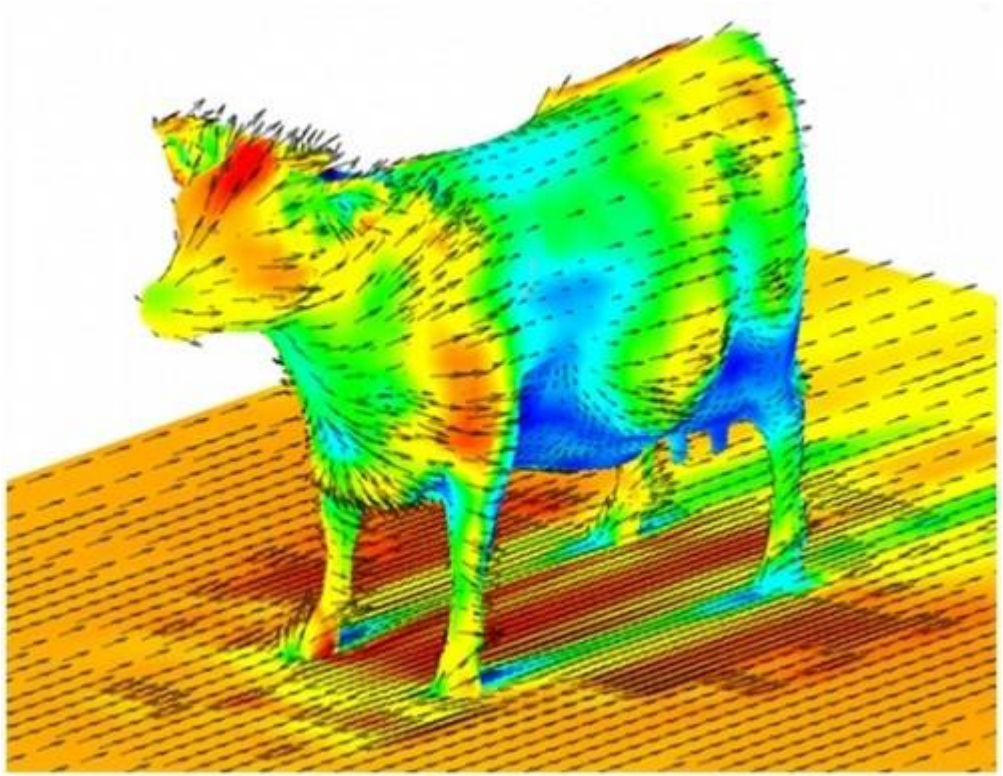
Content

- handout_sqrtmax.pdf: instructions
- problem/
  - sqrtmax.c: the example code
  - Makefile
  - sqrtmax.sub: queue submission file without Allinea MAP
  - sqrtmax_4p.map: a profile example of the application
- solution/

Key commands

```
$ make
$ qsub sqrtmax.sub
$ checkjob $JOB_ID
```

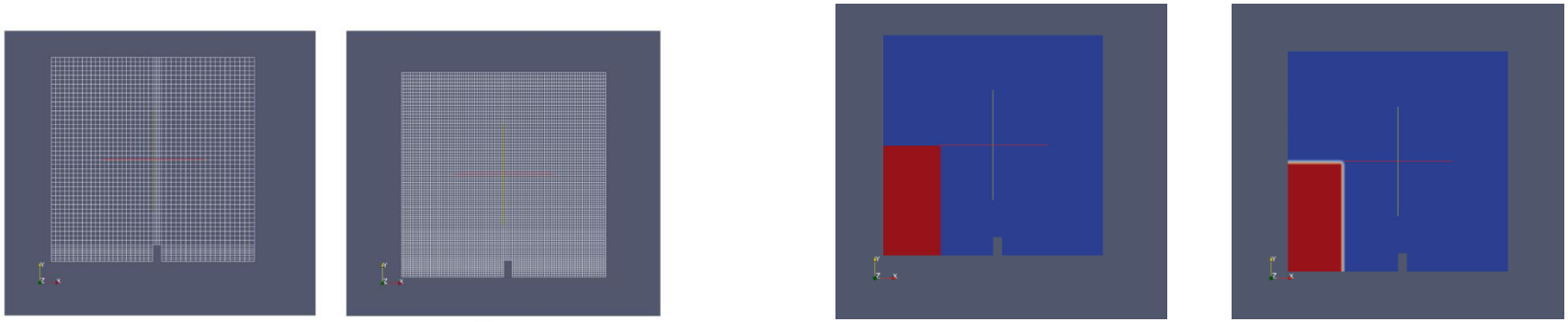
Allinea Performance Reports



One example with a CFD application

Allinea Performance-Reports and OpenFOAM

- How to make sure OpenFOAM is using the best of a system?
- Example from the OpenFOAM tutorial
  - http://www.openfoam.org/docs/user/damBreak.php



Agenda

11:15 – 11:30: Introduction to Allinea tools and latest changes

11:30 – 11:45: Getting started with Allinea Forge

11:45 – 12:30: Profile and Optimise with Allinea MAP

12:30 – 12:45: Allinea Performance Reports

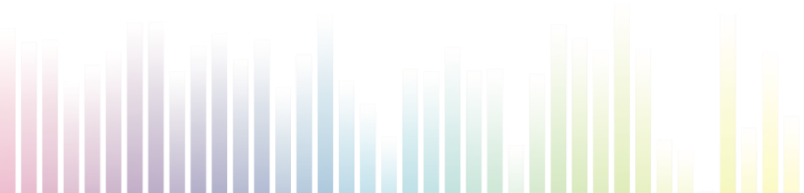
12:45 – 13:45 : Lunch break

13:45 – 17:00: Hands-on session

17:00 – 17:30: Wrap-Up and questions

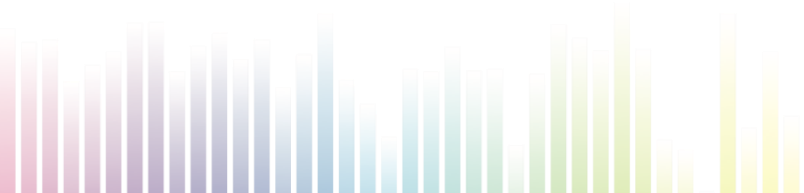
Hands-on Session

On your own codes



Conclusion

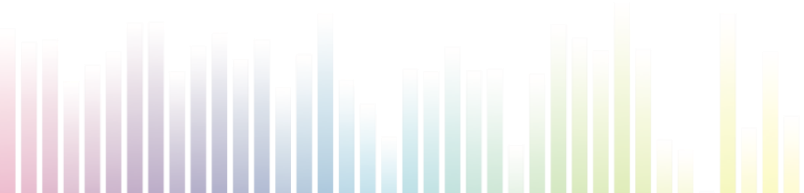
- Enhance application development with Allinea Forge
  - PROFILING WITH ALLINEA MAP
  - DEBUGGING WITH ALLINEA DDT
- Improve resource usage with Allinea Performance Reports



Thank you

Your contacts :

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



allinea