Accelerate HPC Development with Allinea Performance Tools

08 July 2016
VI-HPS, Cambridge

Florent Lebeau
flebeau@allinea.com

Agenda

- 11:00 – 11:15 Introduction
- 11:15 – 11:45 Understand application behaviour with Performance Reports
- 11:45 – 12:15 Profiling: dive in the code with Allinea MAP
- 12:15 – 12:30 Wrap-up and questions
- Afternoon: Hands-on coaching on your own codes

Introduction

Allinea : an expanding company since 2004

- **Based in Warwick (UK), leader in HPC software tools**
  - Subsidiaries in USA, Japan

- **Strong R&D investment to drive Innovation**
  - Significant part of the revenue is spent on R&D yearly
  - Founder and board member of HPC consortiums
  - Strong technological collaborations

- **Strong references all around the world**
  - The main supercomputing centres in the world are using Allinea tools
    - Over 65% of Top 100 HPC systems
    - 7 of the Top 10 HPC systems

They trust Allinea

HLRN

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

JÜLICH
FORSCHUNGSZENTRUM

IT4Innovations
national
supercomputing
center

epcc

lrz
Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

OAK RIDGE
National Laboratory

TACC

BERKELEY LAB
Lawrence Berkeley National Laboratory

SNIC

HLRS
High-Performance Computing Center | Stuttgart

cea

TECHNISCHE
UNIVERSITÄT
DRESDEN

Allinea's vision

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

allinea PERFORMANCE REPORTS

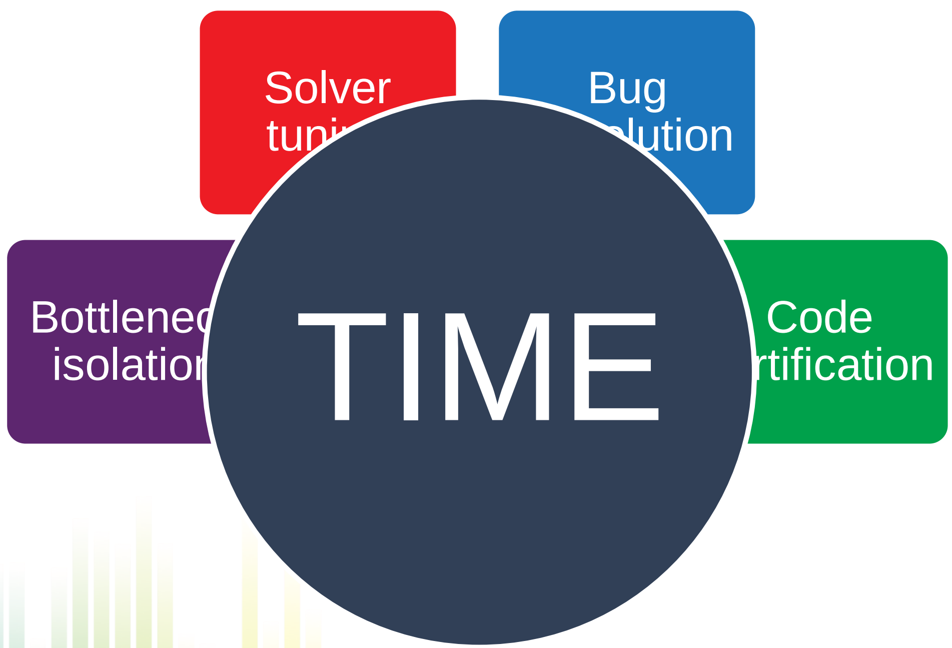
- **Helping the HPC community design the best applications**
  - Reach highest levels of performance and scalability
  - Improve scientific code quality and accuracy

allinea FORGE DDT + MAP

Improvements create pressure on developers

- New generation application are more complex
  - Rely on MPI, OpenMP, TBB, CUDA, OpenACC...
  - Several types of hardware: x86_64, ARM, GPUs, co-processors…

Allinea can help **save time** on multiple tasks

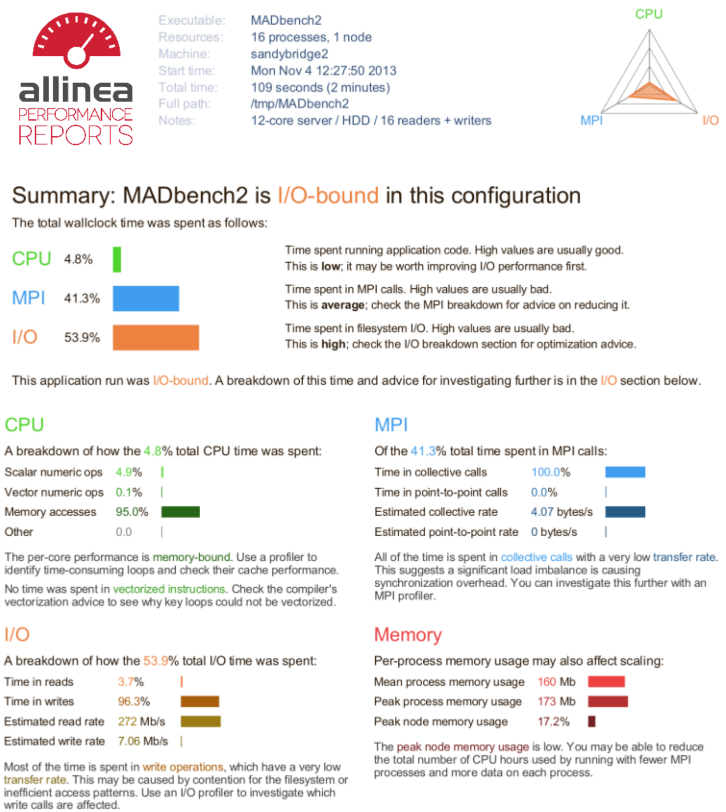


Understand Application Behaviour with Performance Reports

Define your scope

- Before starting to optimise an application, it is important to define the scope
  - Objectives, target speedup
  - Candidate technologies / hardware
  - Development time
- To archive this, developer have to:
  - Understand the application behaviour
  - Know its limitations
  - What if they don’t know the source code?
- Prior to modifying the code, they need to:
  - Define the best candidate versions
  - Select reference and meaningful test cases
  - Know the aspects of the code to refactor and corresponding effort

“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

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

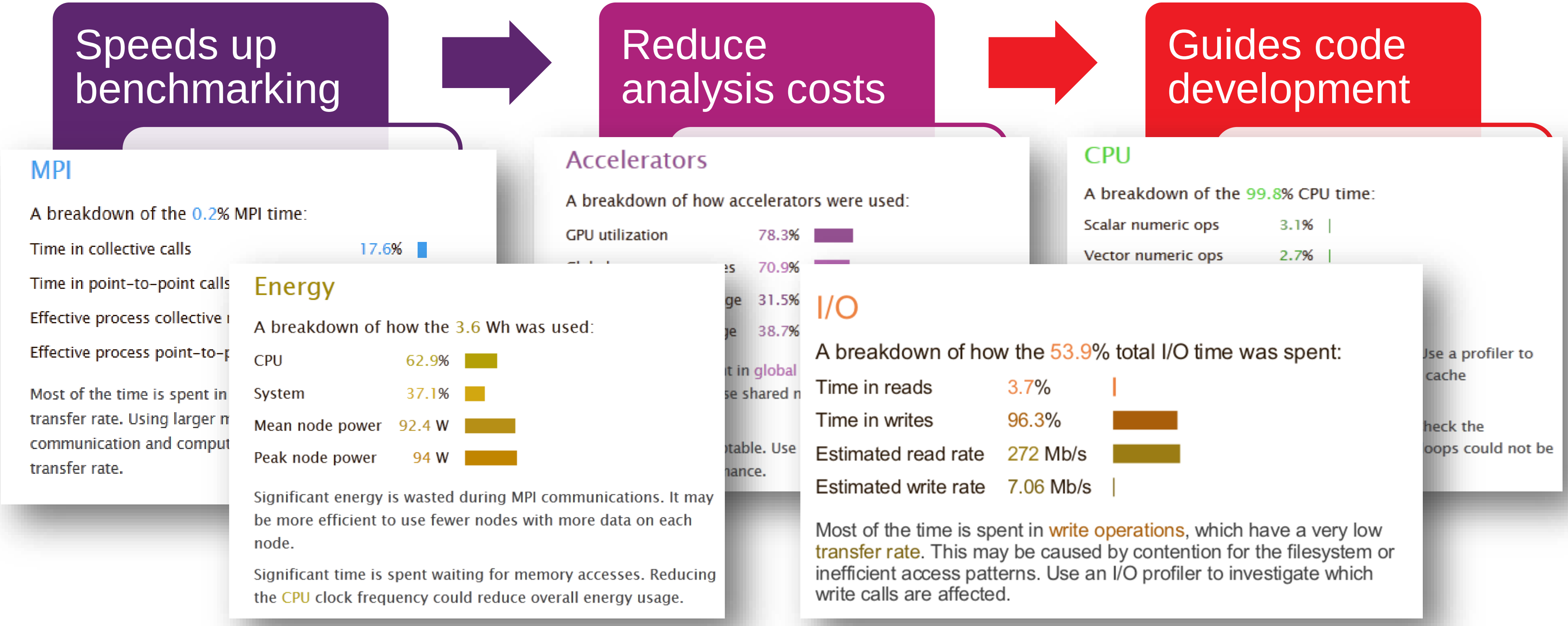
Per-process memory usage may also affect scaling:

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

Maintaining a high efficiency production



Getting started on Darwin with NPB

1- Connect to Darwin using X forwarding “ssh -X”

```
$ ssh -X <username>@login.hpc.cam.ac.uk
```

2- Retrieve labs

```
$ cd ${HOME}/scratch
$ cp -r /home/hpcwyli1/tutorial/NPB3.3-MZ-MPI.tar.gz .
$ tar xvzf NPB3.3-MZ-MPI.tar.gz
```

3- Configure your environment

```
$ module switch default-impi default-impi-LATEST
$ module load allinea/reports/6.0.6
$ perf-report -v
```

Generate your first report on NPB

1- Compile the application

```
$ cd NPB-3.3-MZ-MPI/
$ make bt-mz CLASS=B NPROCS=8
```

2- Edit the job script

```
$ cd bin/
$ cp ../jobscript/darwin/reference.sbatch perf-report.sbatch
```

And change the following lines in perf-report.sbatch:

```
line 17:  module load default-impi-LATEST
          ➔ module load default-impi-LATEST allinea/reports/6.0.6
line 27:  mpirun –np $PROCS $EXE
          ➔ perf-report mpirun –np $PROCS $EXE
```

3- Submit the job script

```
$ sbatch perf-report.sbatch
```

4- Analyse the results

```
$ cat bt-mz_B_8p_4t_2016-07-05_17-00.txt
$ firefox bt-mz_B_8p_4t_2016-07-05_17-00.html
```

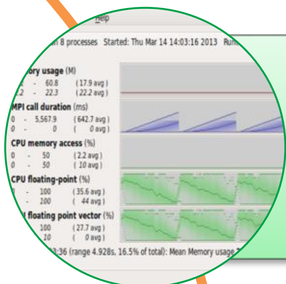
Profiling: Dive in the code with Allinea MAP

The quest for the Holy Performance

Code optimisation can be time-consuming.

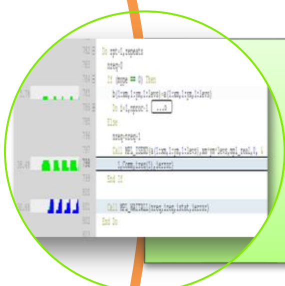
Efficient tools can help you focus on the most important bottlenecks.

Allinea MAP: Performance made easy



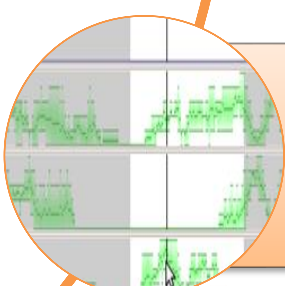
Low overhead measurement

- Accurate, non-intrusive application performance profiling
- Seamless – no instrumentation required



Easy to use

- Source code viewer pinpoints bottleneck locations
- Zoom in to explore iterations, functions and loops
- Re-compile with “-g” and profile by adding “map” to the mpirun command



Deep

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

Allinea MAP and tracing tools: 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

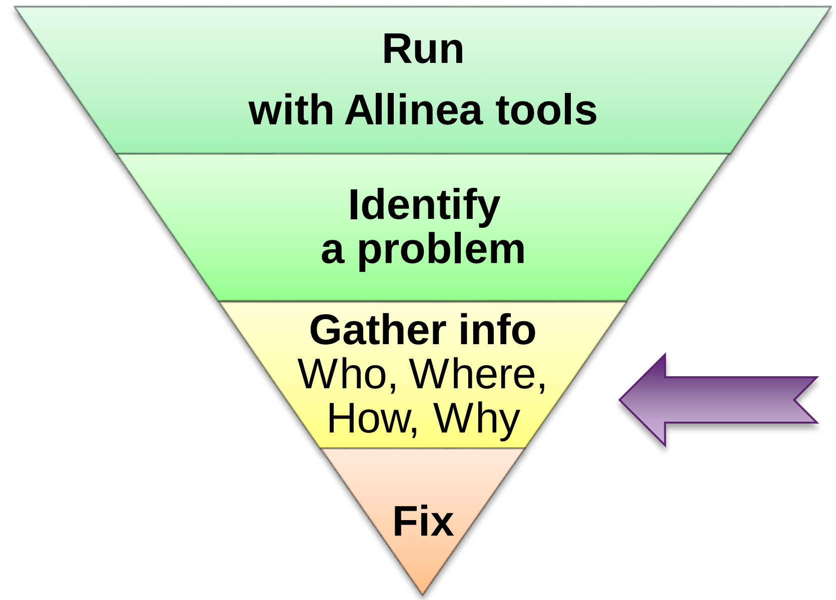
- Identify loop(s) to instrument
- Identify performance counter(s) to record
- Document performance issues to communicate to profiling experts

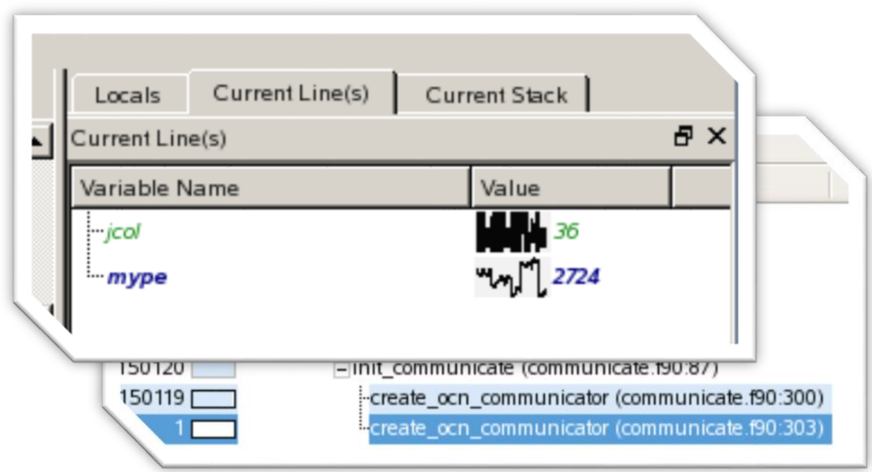
Fine tune the code with tracing tool

- Retrieve low-level details using traces
- Fix up CPU usage to make the code fly

Check your code with Allinea DDT

- ***Who*** **had a rogue behaviour ?**
  - 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





Getting started with profiling on Darwin with NPB (1)

1- Configure your environment

```
$ module load allinea/forge/6.0.6
$ map -v
```

2- Prepare the application

```
$ cd ${HOME}/scratch/NPB-3.3-MZ-MPI/
```

And change the following line in config/make.def:

```
line 53:    FFLAGS = -O3 $(COMPFLAGS)
            ➔ FFLAGS = -O3 –g $(COMPFLAGS)
$ make bt-mz CLASS=B NPROCS=8
```

Getting started with profiling on Darwin with NPB (2)

3- Edit the job script

```
$ cd bin/
$ cp ../jobscript/darwin/reference.sbatch map.sbatch
```

And change the following lines in map.sbatch:

```
line 17   module load default-impi-LATEST
          ➔ module load default-impi-LATEST allinea/forge/6.0.6
line 27   mpirun –np $PROCS $EXE
          ➔ map --profile mpirun –np $PROCS $EXE
```

4- Submit the job script

```
$ sbatch map.sbatch
```

5- Analyse the results

```
$ map bt-mz_B_8p_4t_2016-07-05_17-00.map
```

Analyse the results: using the remote client

- Install the Allinea Remote Client

  Go to : http://www.allinea.com/products/downloads

- Copy the *.map file on your laptop

- View the results locally

Modify sampling

- By default: 1 sample every 20 ms at start up

- The sampling rate automatically adapts to the length of the run:
  - This ensures only a few megabytes of data are collected

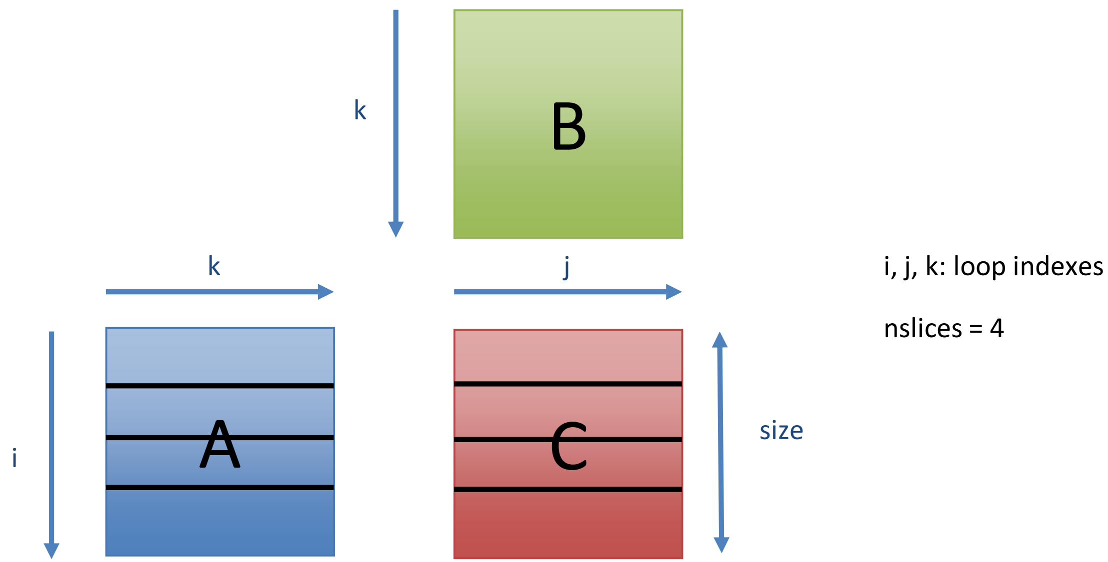
- By default: 1000 samples are collected per process

- Use the following env variables to change these settings:
  - ALLINEA_SAMPLER_INTERVAL (specify a value in ms)
  - ALLINEA_SAMPLER_NUM_SAMPLES (specify a number of samples to collect per process)

Profiling sections of an application with MAP

- Command-line options:
  - map --start-after=TIME
  - map --stop-after=TIME
  - And the profiler will start/stop sampling TIME seconds after the program starts
- In the source code:
  - Include headers “mpi_sampler_mpi.h” in /path/to/allinea/map/wrapper
  - Link with libmap-sampler in /path/to/allinea/lib/64
  - Use the following API functions
    - allinea_start_sampling
    - allinea_stop_sampling
  - Export ALLINEA_SAMPLER_DELAY_START=1 before starting MAP

Tutorial: Matrix Multiplication C = A x B + C



Algorithm

1- Master initialises matrices A, B & C

2- Master slices the matrices A & C, sends them to slaves

3- Master and Slaves perform the multiplication

4- Slaves send their results back to Master

5- Master writes the result Matrix C in an output file

Tutorial: Matrix Multiplication C = A x B + C

- Retrieve source codes:

```
$ cp -r /home/hpclebe1/allinea_workshop.tar.gz .
$ tar xzvf allinea_workshop.tar.gz
```

- Version 1:
  - Identify hotspot and CPU performance issue with Allinea MAP
- Version 2:
  - Check for memory leaks in the optimised version
- Version 3:
  - Resolve load imbalance and fix IO bottleneck
- Version 4:
  - Increase vectorisation and parallelisation

- Compile

```
$ make
```

- Submit

```
$ sbatch job.scratch
```

allinea

Wrap-up and questions

Summary

- Increase job efficiency with Allinea Performance Reports
  - Squeeze more jobs within a given time frame
  - Increase research by freeing machine time without hardware investment
  - Helps focus on the right issues: configuration or source-code related

- Reach your performance goals with Allinea MAP
  - Easily profile your applications at scale
  - Quickly find bottlenecks
  - Investigate performance issues and save development time

Thank you

Your contacts :

- Questions? *flebeau@allinea.com*
- Sales team: *sales@allinea.com*

allinea