# Accelerate HPC Development with Allinea Performance Tools

19 April 2016
VI-HPS, LRZ

Florent Lebeau / Ryan Hulguin
*flebeau@allinea.com / rhulguin@allinea.com*

# Agenda

- 09:00 – 09:15 Introduction
- 09:15 – 09:45 Understand application behaviour with Performance Reports
- 09:45 – 09:50 Dive in the code with Allinea MAP
- 09:50 – 09:20 Profile and Optimise
- 10:20 – 10:30 Wrap-up and questions
- Afternoon: Hands-on session

# 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

# They trust Allinea

HLRN

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

JÜLICH
FORSCHUNGSZENTRUM

IT4Innovations
national01$#&0
supercomputing
center@#01%101

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

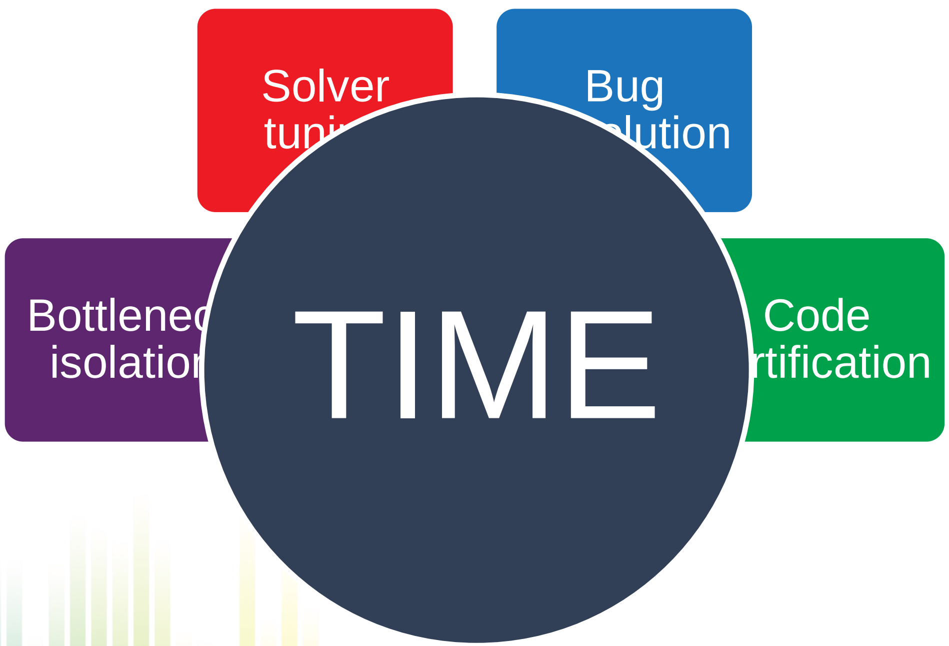
- **Helping the HPC community design the best applications**
  - Reach highest levels of performance and scalability
  - Improve scientific code quality and accuracy

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

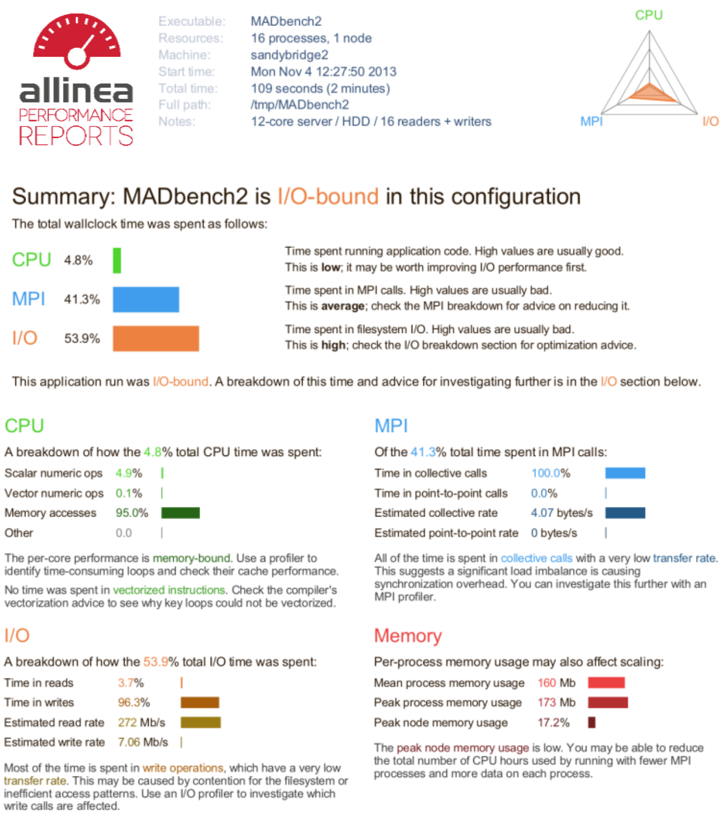
# Define your scope

- Before starting to optimize an application, it is important to define the scope
  - Objectives, target speedup
  - Candidate technologies / hardware
  - Development time

- To achieve 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

# Matrix Multiplication Example

- With Allinea Performance Reports, it is not important for the user to know exactly what an application is doing
- The focus is on how well does the application perform on a given system.
- In the next few slides we will generate and analyze Performance Reports for an application that multiplies two separate matrices

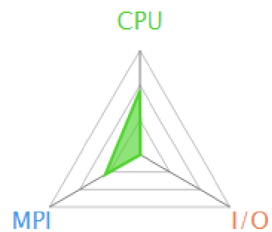
# Usage and Select Options

```
Usage: perf-report [OPTION...] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] MAP_FILE

  --cwd=DIRECTORY     configure the current working directory for the target
  --debug             include extra debug information in log file
  -h, --help          display this help and exit
  --input=FILE        pass the contents of FILE to the target's stdin
  --log=FILE          writes a log to FILE
  --nompi, --no-mpi   run without MPI support
  --mpiargs=ARGUMENTS command line arguments to pass to mpirun
  --nodes=NUMNODES    configure the number of nodes for MPI jobs
  --openmp-threads=NUMTHREADS
                      configure the number of OpenMP threads for the target
  -o, --output=FILE   writes the Performance Report to FILE
  -n, --np, --processes=NUMPROCS
                      specify the number of MPI processes
  --procs-per-node=PROCS
                      configure the number of processes per node for MPI jobs
  --verbose           print extra information before and after report generation
  -v, -V, --version   display version information then exit
```

# Report Summary – Compute-bound

| | |
|---|---|
| Command: | mpirun -n 12 ./mmult3_c.exe 4608 |
| Resources: | 1 node (12 physical, 24 logical cores per node) |
| Tasks: | 12 processes |
| Machine: | mic2 |
| Start time: | Fri Feb 20 21:31:21 2015 |
| Total time: | 39 seconds (1 minute) |
| Full path: | /scratch/allinea/mmult/3_fix |
| Input file: | |
| Notes: | |




## Summary: mmult3_c.exe is CPU-bound in this configuration

CPU 61.4% — Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for optimization advice.

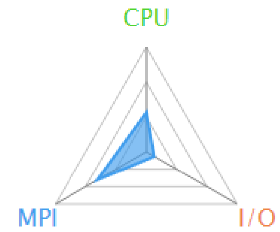
MPI 38.1% — Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.

I/O 0.5% — 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 CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

# Report Summary – MPI-bound

| | |
|---|---|
| Command: | mpirun -n 6 --bind-to-core ./mmult3_c.exe 4608 |
| Resources: | 1 node (12 physical, 24 logical cores per node) |
| Tasks: | 6 processes |
| Machine: | mic2 |
| Start time: | Fri Feb 20 21:46:04 2015 |
| Total time: | 118 seconds (2 minutes) |
| Full path: | /home/allinea/mmult/3_fix |
| Input file: | |
| Notes: | |




## Summary: mmult3_c.exe is MPI-bound in this configuration

| | | |
|---|---|---|
| CPU | 37.5% | Time spent running application code. High values are usually good. This is **low**; it may be worth improving MPI or I/O performance first. |
| MPI | 53.7% | Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it. |
| I/O | 8.8% | Time spent in filesystem I/O. High values are usually bad. This is **low**; check the I/O breakdown section for optimization advice. |

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

# CPU

## CPU

A breakdown of the 61.4% CPU time:

| | |
|---|---|
| Scalar numeric ops | 16.1% |
| Vector numeric ops | 10.1% |
| Memory accesses | 73.8% |

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

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

# MPI

## MPI

A breakdown of the 53.7% MPI time:

| | |
|---|---|
| Time in collective calls | 97.5% |
| Time in point-to-point calls | 2.5% |
| Effective process collective rate | 0.00 bytes/s |
| Effective process point-to-point rate | 462 MB/s |

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

# Memory

## Memory

Per-process memory usage may also affect scaling:

| | | |
|---|---|---|
| Mean process memory usage | 198 MB | |
| Peak process memory usage | 555 MB | |
| Peak node memory usage | 14.0% | |

There is significant variation between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak.

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

# I/O – Home directory

## I/O

A breakdown of the 8.8% I/O time:

| | |
|---|---|
| Time in reads | 0.0% |
| Time in writes | 100.0% |
| Effective process read rate | 0.00 bytes/s |
| Effective process write rate | 4.07 MB/s |

Most of the time is spent in write operations with a very 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.

# I/O – Scratch Filesystem

## I/O

A breakdown of the 0.5% I/O time:

| | |
|---|---|
| Time in reads | 0.0% |
| Time in writes | 100.0% |
| Effective process read rate | 0.00 bytes/s |
| Effective process write rate | 112 MB/s |

Most of the time is spent in write operations with an average effective transfer rate. It may be possible to achieve faster effective transfer rates using asynchronous file operations.

# Getting started on UV2

1- Connect to UV2 using X forwarding “ssh –X”

```
$ ssh –X <username>@lxlogin1.lrz.de
$ ssh –X <username>@ssh -Y ice1-login
```

2- Retrieve labs

```
$ scp /home/hpc/a2c06/lu23vow/allinea/allinea_workshop.tar.gz ~
$ tar xzvf allinea_workshop.tar.gz
```

3- Configure your environment

```
$ cd allinea_workshop
$ . env.sh
$ perf-report -v
```

# Usage Instructions for the UV2 System

- Compile code with Intel MPI Library

```
module unload mpi/mpt
module load mpi/intel
```

- Use the thread safe MPI version

```
CFLAGS = -mt_mpi
FFLAGS = -mt_mpi
```

# UV2 Instructions continued

## Submission script modifications:

```
source /etc/profile.d/modules.sh
module use /home/hpc/a2c06/lu23vow/allinea/modulefiles
module load perf-reports-6.0.3
module unload mpi.mpt
module load mpi.intel
export
ALLINEA_MPI_WRAPPER=/home/hpc/a2c06/lu23vow/.allinea/wrapper/
libmap-sampler-pmpi-uv2-intel.so
```

# UV2 Instructions continued

Change the MPI launch line from

```
srun_ps -n NUMPROCS -t NUMTHREADS ./prog.exe
```

to

```
perf-report -np=NUMPROCS --openmp-threads=NUMTHREADS
./prog.exe
```

# Hydro Application

- Hydro solves the compressible Euler equations of hydrodynamics
- It is a simplified version of a real HPC code named RAMSES to study large scale structure and galaxy formation
- More details of this application can be found in the article below located at www.prace.ri-eu

Porting and optimizing HYDRO to new platforms and programming paradigms – lessons learnt

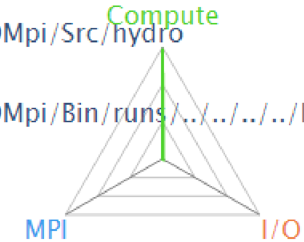
Pierre-François Lavallée[a], Guillaume Colin de Verdière[b], Philippe Wautelet[a], Dimitri Lecas[a], Jean-Michel Dupays[a]

[a]IDRIS/CNRS, Campus universitaire d'Orsay, rue John Von Neumann, Bâtiment 506, F-91403 Orsay, France

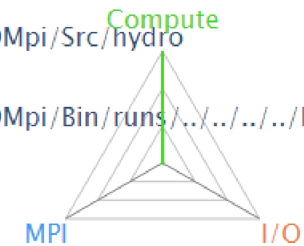
[b]CEA,Centre DAM Ile-de-France, Bruyères-le-Châtel,F-91227 Arpajon, France

# Hydro Reports on UV2

| | |
|---|---|
| Command: | /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Src/hydro -i /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Bin/runs/../../../../In |
| Resources: | 1 node (960 physical, 1920 logical cores per node) |
| Memory: | 3028 GB per node |
| Tasks: | 1 process, OMP_NUM_THREADS was 64 |
| Machine: | uv2 |
| Start time: | Sun Apr 17 19:01:58 2016 |
| Total time: | 2098 seconds |
| Full path: | /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Src |




| | |
|---|---|
| Command: | /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Src/hydro -i /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Bin/runs/../../../../In |
| Resources: | 1 node (960 physical, 1920 logical cores per node) |
| Memory: | 3028 GB per node |
| Tasks: | 1 process, OMP_NUM_THREADS was 128 |
| Machine: | uv2 |
| Start time: | Sun Apr 17 19:02:26 2016 |
| Total time: | 4473 seconds |
| Full path: | /home/hpc/a2c06/lu23vow/ryan/Hydro/HydroC/HydroC99_2DMpi/Src |




- Reports are available at `allinea_workshop/0_reports` from the `allinea_workshop.tar.gz` file

allinea

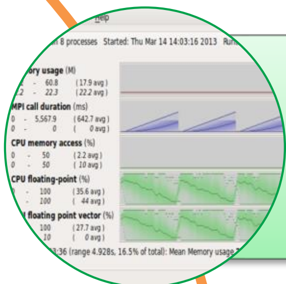
# Dive in the code with Allinea MAP

# The quest for the Holy Performance


Code optimisation can be time-consuming.

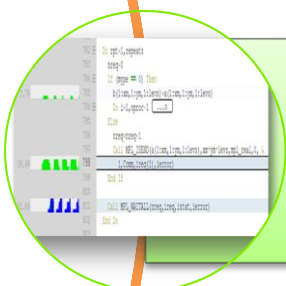
Efficient tools can help you focus on the most important bottlenecks.

# Allinea MAP: Performance made easy



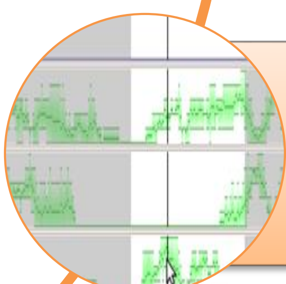
## Low overhead measurement

- 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**

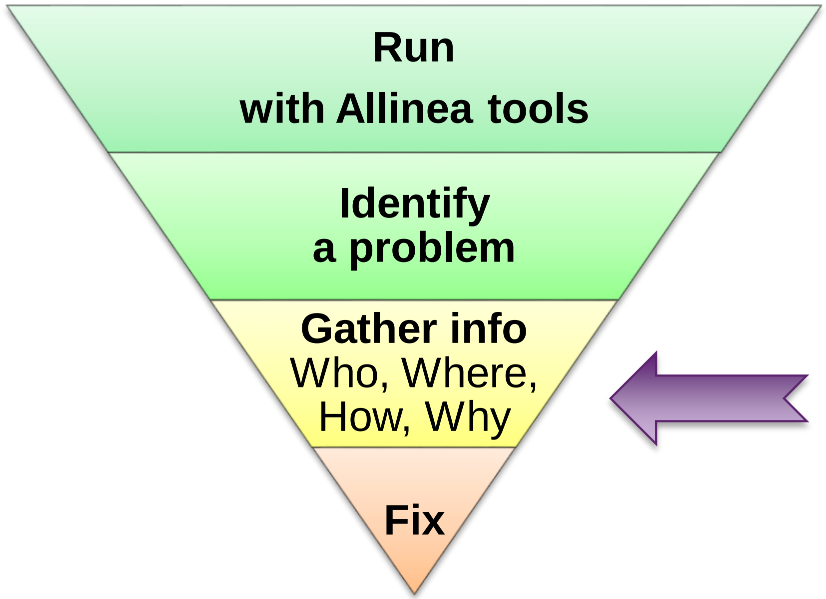
- Identify loop(s) to instrument
- Identify performance counter(s) to record
- Document performance issues to communicate to profiling experts

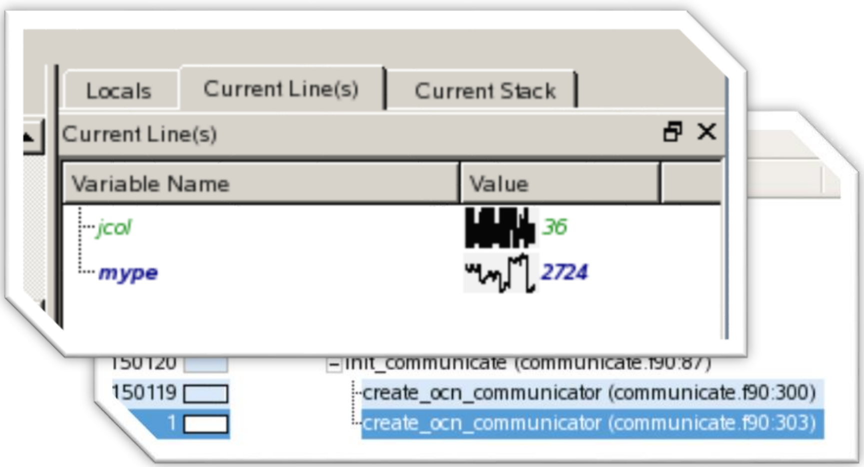
**Fine tune the code with tracing tool**

- 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



# Profile and Optimise

# How to use Allinea MAP on UV2

- Prepare the code

```
$ mpicc –O3 -g myapp.c –o myapp
```

- Profile the application with Allinea MAP

```
$ make-profiler-libraries
```

Follow the instructions displayed in the terminal

Edit your job script to replace the srun_ps command by:

```
map --profile -n 8 ./myapp arg1 arg2
```

- Open the results in the GUI afterwards

```
$ map ./myapp_8p_YYYY-MM-DD_HH-MM.map
```

# Using the remote client

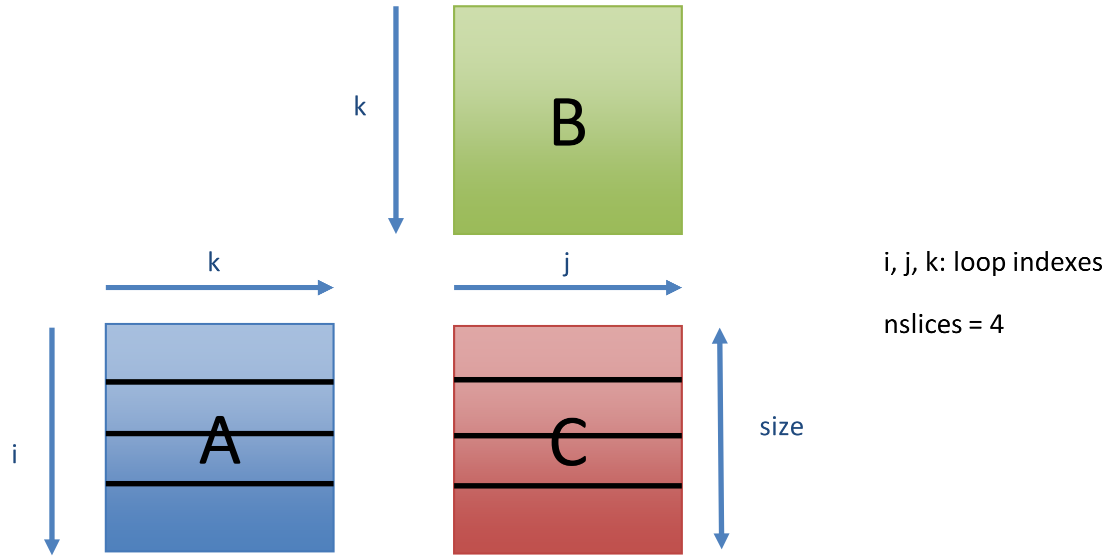
- Install the Allinea Remote Client

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

- Copy the *.map file on your laptop

- View the results locally

# Exercise:
# 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:

**Exercise objectives:**

- Load Allinea Forge environment
- Compile a code for allinea MAP
- Submit the job through the queue
- Discover allinea MAP interface and features
- Analyse the application and identify vectorisation metrics

**Content**

- Source code in C and F90 + Makefile
- Submission script

**Useful commands:**

```
$ make

$ make clean

$ sbatch job.sub

$ squeue –cluster=uv2

$ scancel JOB_ID
```

# Wrap-up and questions

# Thank you

Your contacts :

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