

allinea

Now part of **ARM**

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

Accelerate HPC Development with Allinea Performance Tools

Olly Perks & Florent Lebeau

Olly.Perks@arm.com

Florent.Lebeau@arm.com



Agenda

- 09:00 – 09:15 Introduction to Alinea Tools
- 09:15 – 09:30 Analyse your Applications on ARCHER
- 09:30 – 09:45 Retrieve Hardware Counters' Data
- 09:45 – 10:15 Write Application-specific Metrics
- 10:15 – 10:30 Wrap-up and questions

- Afternoon: Hands-on coaching on your own codes

Age Group	Percentage
18-24	10%
25-34	15%
35-44	12%
45-54	18%
55-64	22%
65-74	25%
75-84	20%
85+	18%



Building blocks for better science



Scalability

- Enable multi-physics simulations
- Run larger, more accurate models
- Resolve ground-breaking scientific problems

Efficiency

- Reduce wasted resources (energy...)
- Maximize science output per \$
- Minimize time to result

Simplicity

- Pro-actively and automatically detect faults
- Provide applications on various hardware
- Facilitate technical dialogue with scientists

About Allinea

- Allinea: leading toolkit for HPC application developers
- As of December 2016 Allinea is now part of ARM
 - Objective: continue to be the trusted HPC Tools leader in tools across every platform
- This means:
 - The **same team** will continue to work with you, our customers and partners, and the wider HPC community
 - Being part of ARM gives us strength to **deliver on our roadmap faster**
 - We remain 100% committed to providing **cross-platform tools for HPC**
 - Our engineering roadmap is **aligned with upcoming architectures** from every vendor

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



Where to find Allinea tools

Over 65% of Top 100 HPC systems

- From small to very large tools provision

6 of the Top 10 HPC systems

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

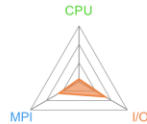
Future leadership systems

- Millions of cores usage

“Learn” with Allinea Performance Reports

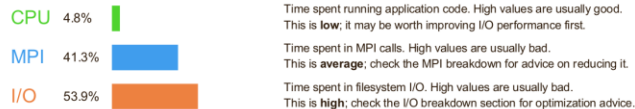


Executable: MADbench2
Resources: 16 processes, 1 node
Machine: sandybridge2
Start time: Mon Nov 4 12:27:50 2013
Total time: 109 seconds (2 minutes)
Full path: /tmp/MADbench2
Notes: 12-core server / HDD / 16 readers + writers



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

The total wallclock time was spent as follows:



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

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

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:



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.

I/O

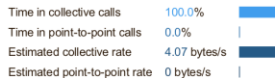
A breakdown of how the **53.9%** total I/O time was spent:



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.

MPI

Of the **41.3%** total time spent in MPI calls:



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.

Memory

Per-process memory usage may also affect scaling:



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

MPI

A breakdown of the 0.2% MPI time:

Time in collective calls

17.6%

Time in point-to-point calls

Effective process collective

Effective process point-to-p

Most of the time is spent in

transfer rate. Using larger n

communication and comput

transfer rate.

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 78.3%

70.9%

31.5%

38.7%

t in global

se shared n

table. Use

ance.

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.

CPU

A breakdown of the 99.8% CPU time:

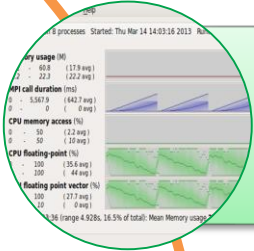
Scalar numeric ops 3.1%

Vector numeric ops 2.7%

Use a profiler to cache

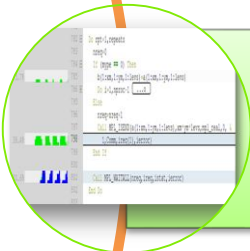
check the
oops could not be

Allinea MAP: Performance made easy



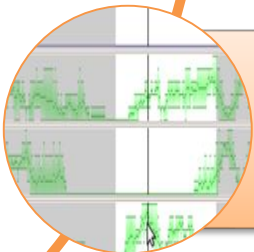
Low overhead measurement

- 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

Getting started on ARCHER: Set the environment

- Load the Allinea tools modules

```
$ module load allinea/7.0 allinea-reports/7.0
```

- Use the temporary licence for the workshop:

```
$ export ALLINEA_LICENCE_DIR=/fs4/y14/shared/allinea_licence
```

- Copy the NPB archive

```
$ cp /fs4/y14/shared/tutorial/NPB3.3-MZ-MPI.tar.gz .
```

```
$ tar xzvf NPB3.3-MZ-MPI.tar.gz
```

```
$ cd NPB3.3-MZ-MPI
```

Getting started: Prepare the code for profiling on Cray

- Compile the MAP wrapper libraries

```
$ mkdir allinea_libs  
$ cd allinea_libs  
$ make-profiler-libraries  
$ cd ..
```

The “make-profiler-libraries” command will display instructions to compile the application.

- Edit the config/make.def file and edit the following lines to add the extra parameters in bold

```
Line 11:  COMPILER = -Gfast -homp -N 255  
Line 62:  F_LIB = -dynamic -L/path/to/allinea_libs  
          \ -lmap-sampler-pmpi -lmap-sampler  
          \ -Wl,--eh-frame-hdr  
          \ -Wl,-rpath=/path/to/allinea_libs
```

- Compile bt-mz

```
$ make bt-mz CLASS=C NPROCS=24
```

Getting started: Submit the job to profile the application

- Copy a job script

```
$ cp /fs4/y14/shared/job.archer bin/  
$ cd bin/
```

- Edit “job.archer” to have the following lines instead of the execution command line 29

- To run Allinea Performance Reports:

```
module load allinea-reports/7.0  
export ALLINEA_LICENCE_DIR=/fs4/y14/shared/allinea_licence  
perf-report aprun -n 24 ./bt-mz_C.24
```

- Or to run Allinea MAP:

```
module load allinea/7.0  
export ALLINEA_LICENCE_DIR=/fs4/y14/shared/allinea_licence  
map --profile aprun -n 24 ./bt-mz_C.24
```

Default Metrics – Background

- Already covered the introduction to profiling
- Basic metrics for understanding performance

Activity Timeline

- What happens over time
- Breakdown of % by CPU / MPI / I/O / Overheads

CPU Instructions

- What type of instructions are being executed
- FP / Integer / Memory / Vector / Branch

I/O Breakdown

- Time spent in reads / writes
- Data for Posix I/O

MPI Time Breakdown

- Rate of calls (send / recv / collectives)
- Data communicated and time taken

Advanced Metrics

New Metrics:

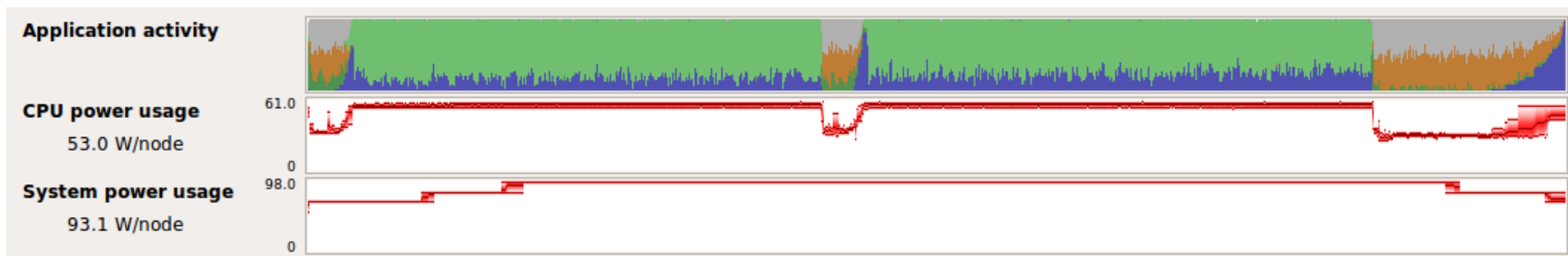
- Energy (RAPL + IPMI)
- Lustre I/O metrics
- PAPI hardware counters

New Capabilities:

- Custom metrics interface
- JSON export

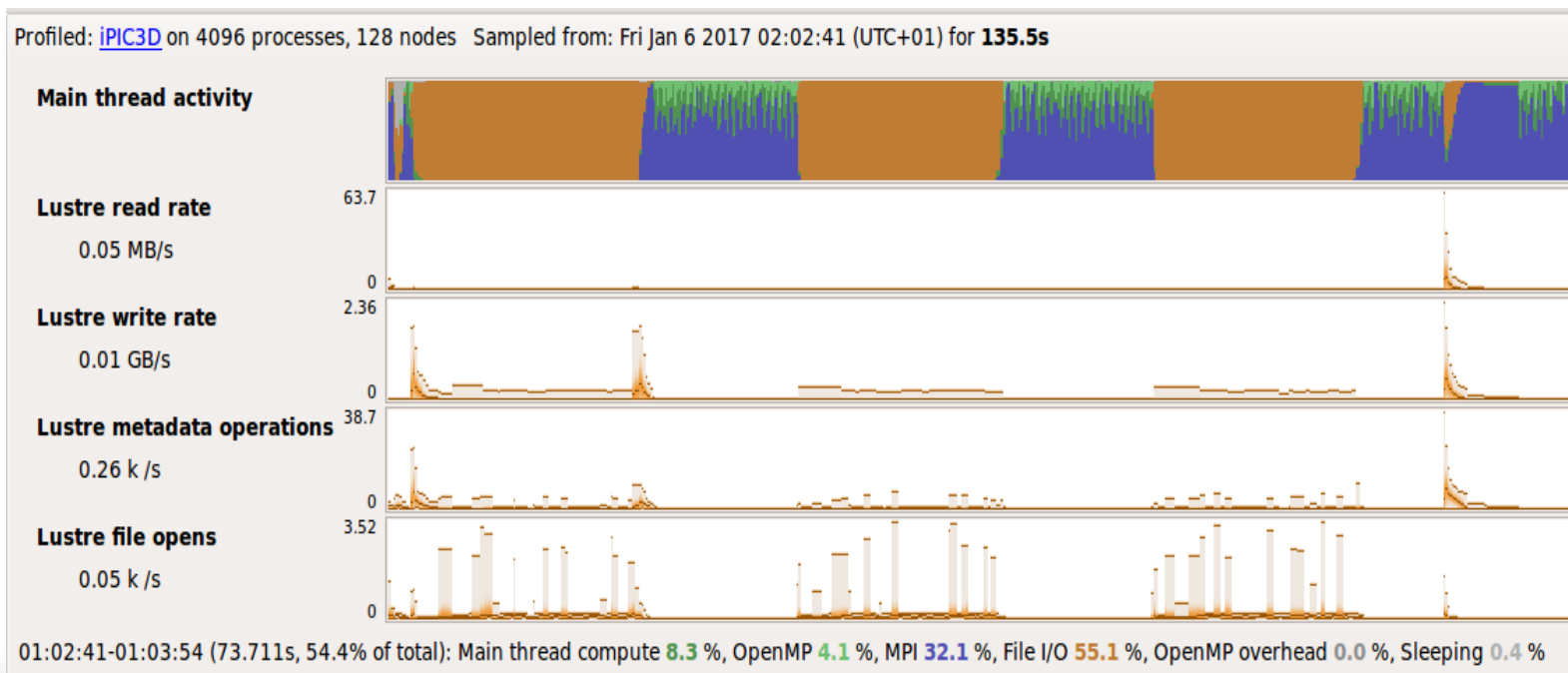
Energy Profiling

- Want to link energy consumption to application activity
- Two ways of collecting energy information
 - RAPL – CPU centric energy data on Intel CPUs (after Sandy Bridge)
 - IPMI – Node level energy collection on compatible hardware
- If available MAP will report this energy data



Lustre I/O Metrics

- Query the kernel for Lustre data activity
- Stores read / write rates, volume, file opens and metadata activity

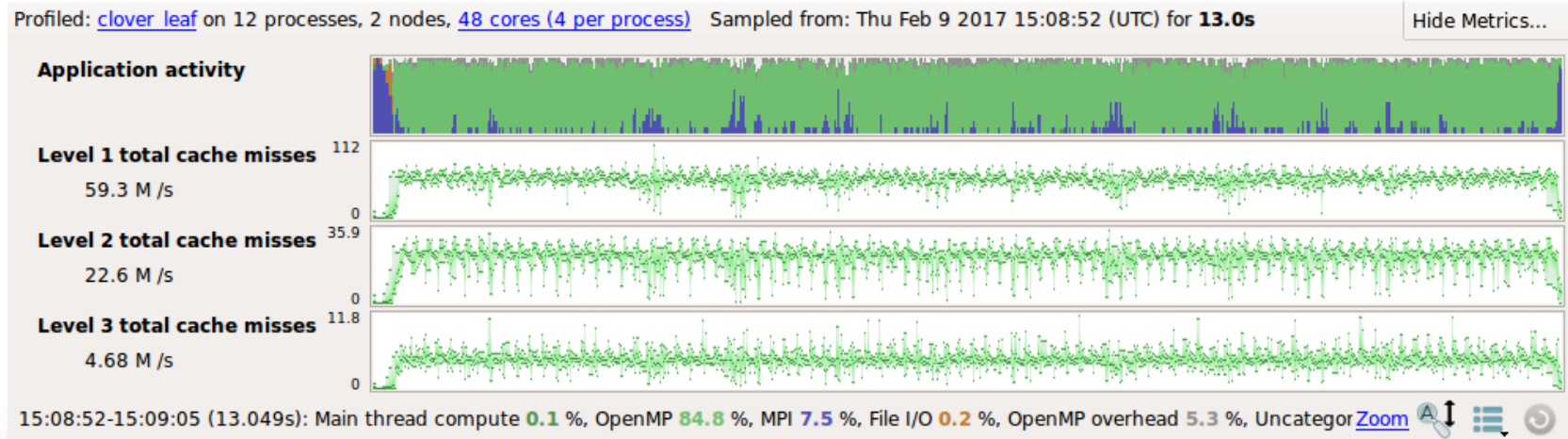
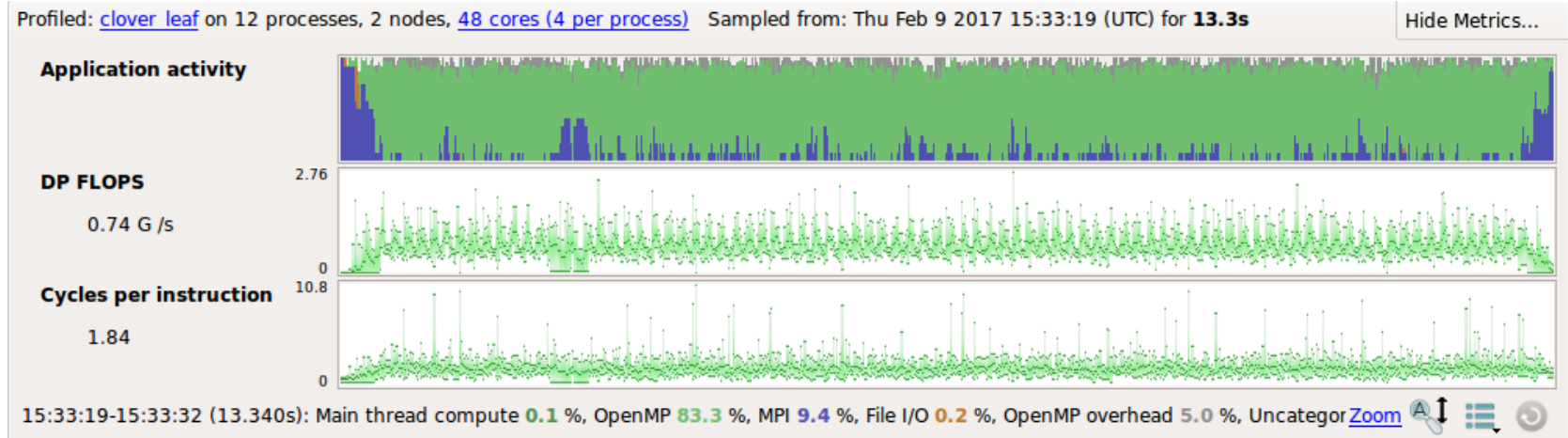


PAPI Hardware Counters

- Metric to interface with the PAPI library
- Limited to a subset of hardware counters
 - Select a PAPI 'profile' for the run
 - Configure in `.allinea/map/metrics/PAPI.config`

```
# Custom metric set
# Possible values are
#   Overview          : FLOPS and cycles per instruction
#   CacheMisses       : L1, L2, L3 total cache misses. Fallback of data
#                       cache misses if total cache misses unavailable.
#   BranchPrediction  : Total and mispredicted branch instructions
#   FloatingPoint     : Scalar and vector floating point instructions.
# Recommendation is Overview
set = Overview
```

PAPI Metrics: Overview & Cache Misses



How to use PAPI metrics on ARCHER

- Load the PAPI module

```
$ module load papi
```

- Install the Allinea PAPI library

```
$ cd /home/y07/y07/cse/allinea/forge/7.0
```

```
$ ./papi_install.sh
```

Should pick up the PAPI lib

Select 'P' for Personal install

By default, this will be installed in \$HOME/.allinea/map/metrics/ but \$HOME is not accessible on the compute nodes ... let's copy it in /work

```
$ cp $HOME/.allinea/ /work/y14/y14/$USER/
```

- Export the following environment variables in the job script

```
$ export ALLINEA_CONFIG_DIR=/work/y14/y14/$USER/.allinea/
```

```
$ export ALLINEA_PAPI_CONFIG=
```

```
\ /work/y14/y14/$USER/.allinea/map/metrics/PAPI.config
```

New Capabilities: Custom Metrics in MAP

Addition of a custom metrics interface

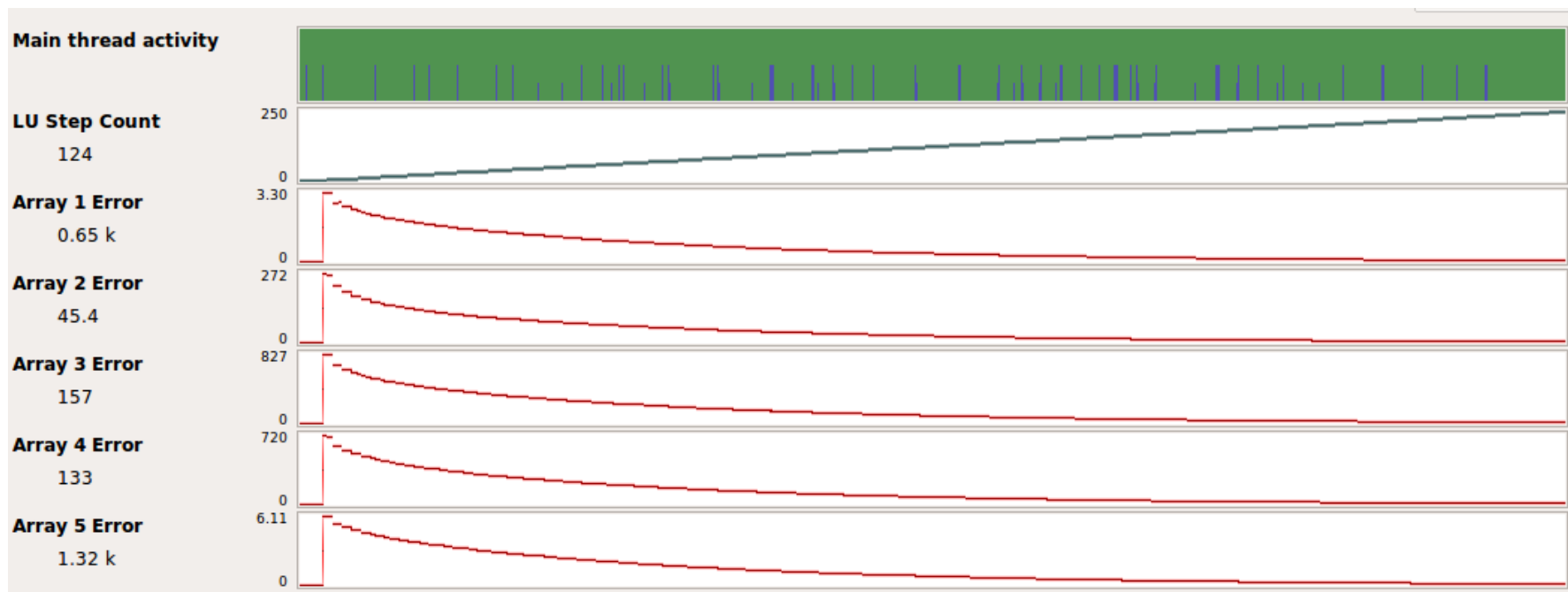
- Write your own metrics
- Must be async signal safe – quite limiting

Types of metric:

- System introspection – Specific counters / files
- Application introspection – Tracking application characteristics

Application Specific Custom Metric

- NPB-LU
 - Track the iteration number
 - Track the error term over time (for all 5 arrays)



Writing a Custom Metric

XML Description

- Define multiple variables
- Function to call to get variables
- Library where functions exist

Functions Library

- Async signal safe functions
- Limit flexibility
- Return data to MAP

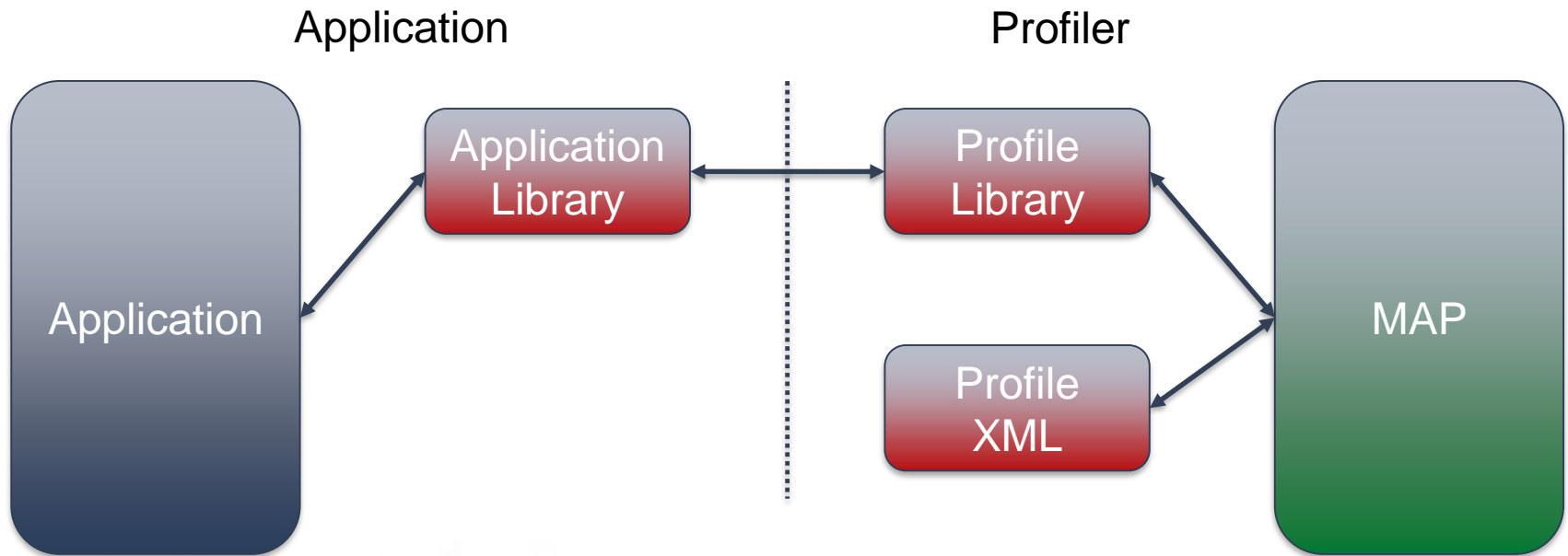
Makefile

- To build against MAP
- Install to metrics directory
- `$ALLINEA_CONFIG_DIR/map/metrics`

Writing a Custom Application Metric

Application
Library

- Library with application variables
- Called from profiling library
- Should maintain own data – with application getters / setters



Guide: LU Custom Metrics

- Install the LU Custom Metrics library

```
$ cp -r /fs4/y14/shared/allinea/custom_metric_lu .
```

```
$ cd custom_metric_lu
```

```
$ make install
```

Will install the metric in /work/y14/y14/\$USER/.allinea/map/metrics/

- Go to the NPB root directory and compile lu-mz

```
$ cd NPB3.3-MZ-MPI
```

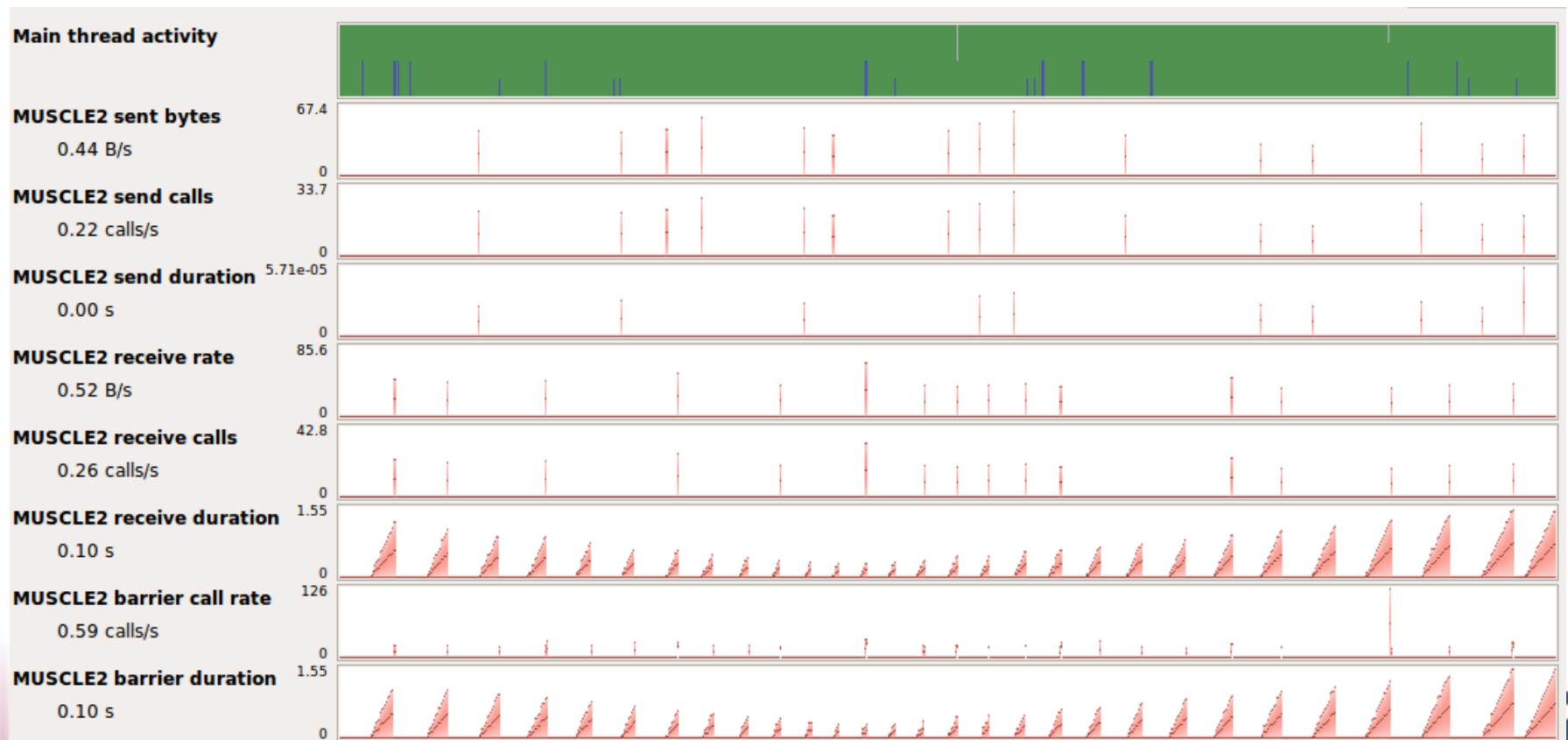
```
$ make lu-mz CLASS=C NPROCS=24
```

- Submit the job to profile in bin/

```
map --profile aprun -n 24 ./lu-mz_C.24
```

System Specific Custom Metric

- MUSCLE-2 Communication
 - Inter-MPI program communication library
 - Record communication performance data – Like MPI



JSON Export

- Export map profile data to JSON file
 - Command line or GUI
 - Provides meta data + samples

```
operks@eslogin001: /work/y14/y14/operks/CloverLeaf_ref> map --export=clover.json clover_leaf_6p_1n_4t_2017-02-09_12-18.map
Loading MAP file clover_leaf_6p_1n_4t_2017-02-09_12-18.map...
```

```
...done
Collecting samples...
```

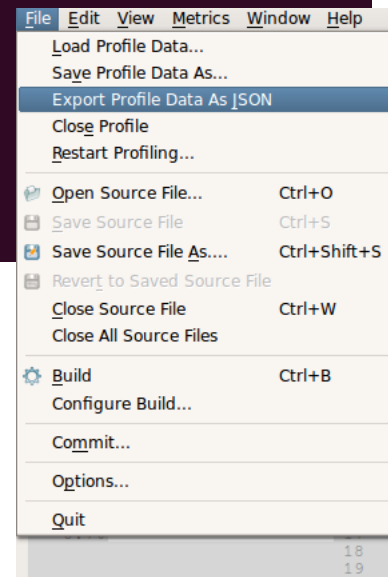
```
...done
Calculating...
```

```
...done
Collecting s...
```

```
...done
Calculating...
```

```
...done
MAP generate
```

```
{
  "info": {
    "command_line": "aprun -n 6 -N 6 -S 3 -d 4 ./clover_leaf",
    "create_version": "7.0",
    "machine": "mom5",
    "metrics": {
      "memory_per_node": {
        "max": 67658141696,
        "mean": 67658141696,
        "min": 67658141696,
        "sum": 67658141696,
        "var": 0
      },
      "num_cores_per_node": {
        "max": 48,
        "mean": 48,
        "min": 48,
        "sum": 288,
        "var": 0
      },
      "num_omp_threads_per_process": {
        "max": 3,
        "mean": 3,
        "min": 3,
        "sum": 18,
        "var": 0
      }
    }
  }
}
```



JSON Ideas

- JSON exposes the ~1000 sample data arrays
 - Min, Max, Mean, STD and sum
- Plot these with Python scripts
- Integrate with regression testing suite
 - Jenkins plugin developed
- Compatibility with Performance Reports JSON export
 - Multisource data visualisation / analytics

allinea

Now part of **ARM**

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

Thank you!

Questions?

Olly.Perks@arm.com

Florent.Lebeau@arm.com

Support:

support@allinea.com

