UL HPC School 2017

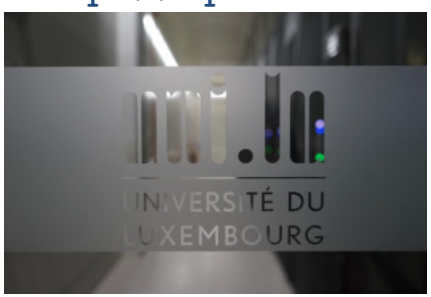
PS6: Debugging, profiling and performance analysis

UL High Performance Computing (HPC) Team

V. Plugaru

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu



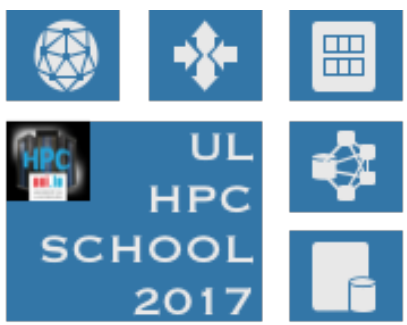
Latest versions available on Github:

UL HPC tutorials: https://github.com/ULHPC/tutorials

UL HPC School: http://hpc.uni.lu/hpc-school/

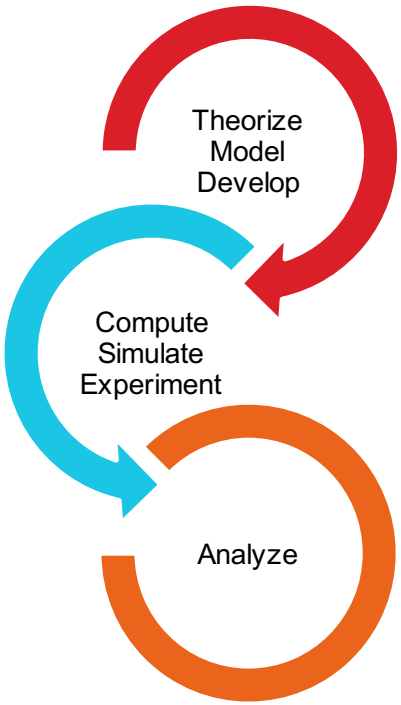
PS6 tutorial sources:

https://github.com/ULHPC/tutorials/tree/devel/advanced/debugging_profiling



Summary

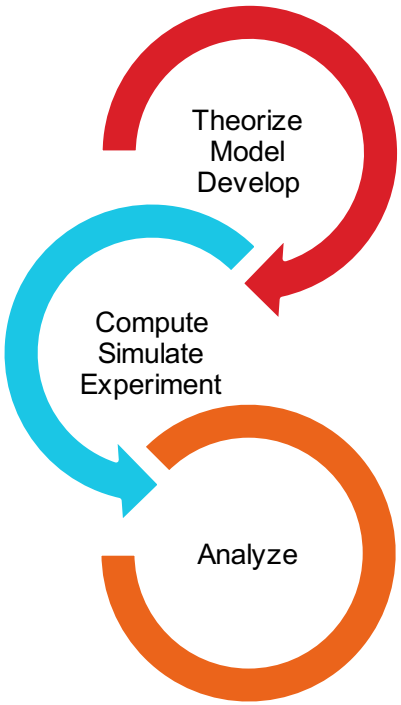
Main Objectives of this Session



This session is meant to show you some of the various tools you have at your disposal on the UL HPC platform to:

understand + solve development & runtime problems

Main Objectives of this Session



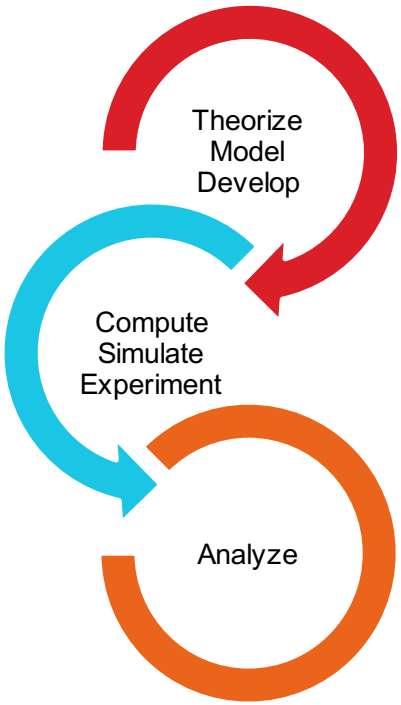
This session is meant to show you some of the various tools you have at your disposal on the UL HPC platform to:

understand + solve development & runtime problems

During the session we will:

- discuss what happens when an application runs **out of memory** and how to discover how much memory it actually requires.
- see **debugging tools** that help you understand **why your code is crashing**.
- see **profiling tools** that show the **bottlenecks of your code** - and **how to improve** it.

Main Objectives of this Session



This session is meant to show you some of the various tools you have at your disposal on the UL HPC platform to:

understand + solve development & runtime problems

During the session we will:

- discuss what happens when an application runs **out of memory** and how to discover how much memory it actually requires.
- see **debugging tools** that help you understand **why your code is crashing**.
- see **profiling tools** that show the **bottlenecks of your code** - and **how to improve** it.

Knowing what to do when you experience a problem is half the battle.

Summary

Tools at your disposal (I)

Common tools used to understand problems

- Do you know what time it is?
  ↪ **/usr/bin/time -v** is just magic sometimes
- Don't remember where you put things?
  ↪ **Valgrind** can help with your memory issues
- Is your application firing on all cylinders?
  ↪ with **htop** green means go! (red is bad)
- Got stuck?
  ↪ **strace** can tell you where you are and how you got there

Some times simple tools help you solve big issues.

Tools at your disposal (II)

HPC specific tools - Allinea

- Allinea DDT (part of Allinea Forge)
  - ↪ Visual debugger for C, C++ and Fortran threaded and // code
- Allinea MAP (part of Allinea Forge)
  - ↪ Visual C/C++/Fortran profiler for high performance Linux code
- Allinea Performance Reports
  - ↪ Application characterization tool

Tools at your disposal (II)

HPC specific tools - Allinea

- Allinea DDT (part of Allinea Forge)
  - ↪ Visual debugger for C, C++ and Fortran threaded and // code
- Allinea MAP (part of Allinea Forge)
  - ↪ Visual C/C++/Fortran profiler for high performance Linux code
- Allinea Performance Reports
  - ↪ Application characterization tool

Allinea tools are licensed

Make sure enough tokens available to profile/debug your code in the requested configuration (#cores)!

- ↪ license check can be integrated in common RJMS (is in SLURM)
- ↪ ... so your jobs are able to wait for tokens to be available

Tools at your disposal (III)

HPC specific tools - Intel

- Intel Advisor
  - ↪ Vectorization + threading advisor: check blockers and opport.
- Intel Inspector
  - ↪ Memory and thread debugger: check leaks/corrupt., data races
- Intel Trace Analyzer and Collector
  - ↪ MPI communications profiler and analyzer: evaluate patterns
- Intel VTune Amplifier
  - ↪ Performance profiler: CPU/FPU data, mem. + storage accesses

Tools at your disposal (III)

HPC specific tools - Intel

- Intel Advisor
  - ↪ Vectorization + threading advisor: check blockers and opport.
- Intel Inspector
  - ↪ Memory and thread debugger: check leaks/corrupt., data races
- Intel Trace Analyzer and Collector
  - ↪ MPI communications profiler and analyzer: evaluate patterns
- Intel VTune Amplifier
  - ↪ Performance profiler: CPU/FPU data, mem. + storage accesses

Intel tools are licensed

All come as part of Intel Parallel Studio XE - Cluster edition!

Tools at your disposal (IV)

HPC specific tools - Scalasca & friends

- Scalasca
  - ↪ Study behavior of // apps. & identify optimization opport.
- Score-P
  - ↪ Instrumentation tool for profiling, event tracing, online analysis.
- Extra-P
  - ↪ Automatic performance modeling tool for // apps.

Tools at your disposal (IV)

HPC specific tools - Scalasca & friends

- Scalasca
  - ↪ Study behavior of // apps. & identify optimization opport.
- Score-P
  - ↪ Instrumentation tool for profiling, event tracing, online analysis.
- Extra-P
  - ↪ Automatic performance modeling tool for // apps.

Free and Open Source!
See other awesome tools at `http://www.vi-hps.org/tools`

Allinea DDT - highlights

DDT features

- **Parallel debugger**: threads, OpenMP, MPI support
- Controls processes and threads
  - ↪ step code, stop on var. changes, errors, breakpoints
- Deep **memory debugging**
  - ↪ find memory leaks, dangling pointers, beyond-bounds access
- C++ debugging – including STL
- Fortran – including F90/F95/F2008 features
- See vars/arrays **across multiple processes**
- Integrated editing, building and **VCS integration**
- Offline mode for **non-interactive debugging**
  - ↪ record application behavior and state

Full details at `allinea.com/products/ddt/features`

Allinea DDT - on ULHPC

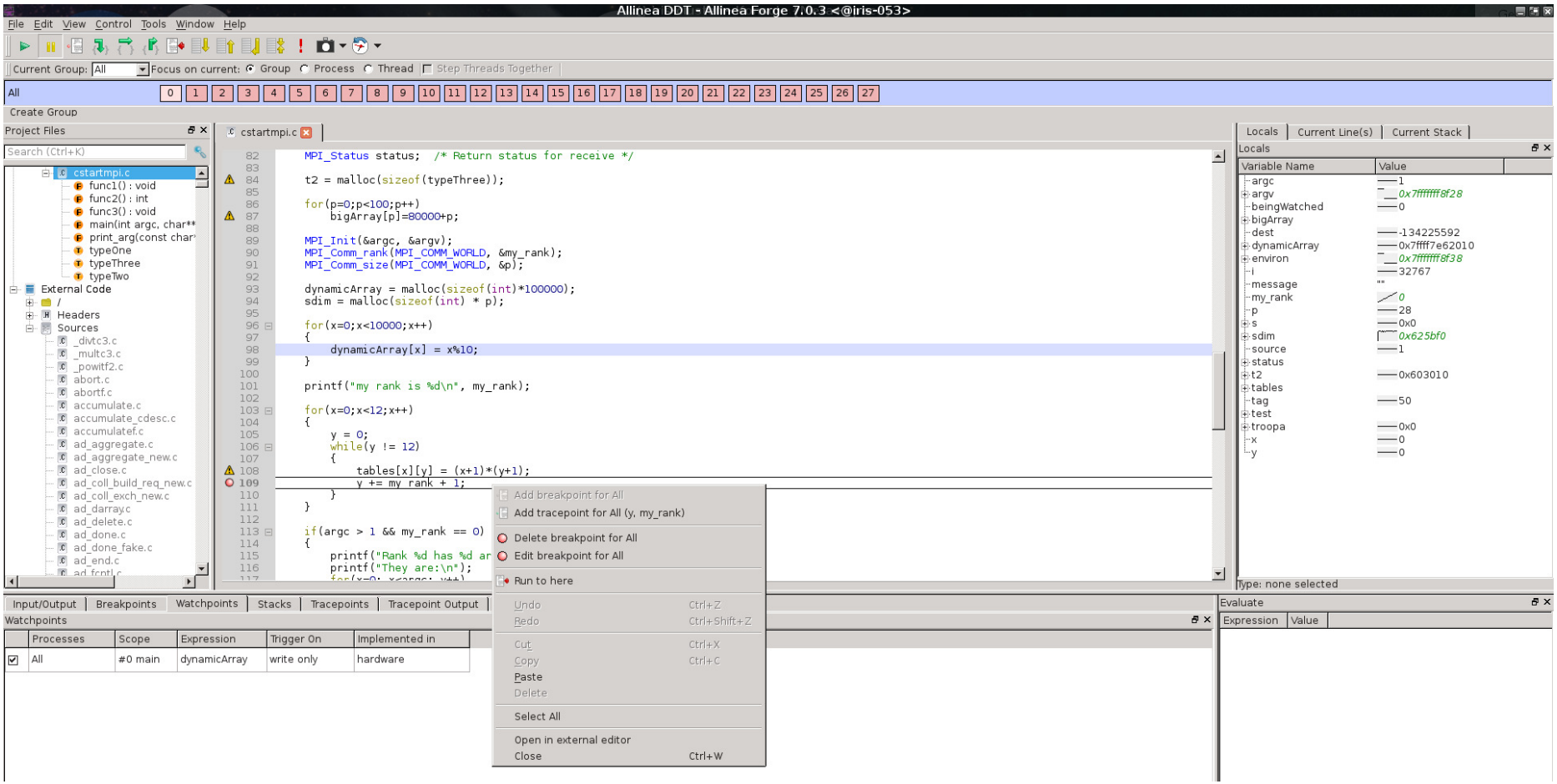
Modules

- On all clusters: `module load tools/AllineaForge`
- Caution! May behave differently between:
  - ↪ Debian+OAR (Gaia, Chaos) and CentOS+SLURM (Iris)

Debugging with DDT

1. Load toolchain, e.g. (for Intel C/C++/Fortran, MPI, MKL):
   - ↪ `module load toolchain/intel`
2. Compile your code, e.g. `mpiicc $code.c -o $app`
3. Run your code through DDT (GUI version)
   - ↪ iris: `ddt srun ./$app`
   - ↪ gaia/chaos: `ddt mpirun -hostfile $OAR_NODEFILE ./$app`
4. Run DDT in batch mode (no GUI, just report):
   - ↪ `ddt --offline -o report.html --mem-debug=thorough ./$app`

Allinea DDT - interface



Allinea MAP - highlights

MAP features

- Meant to show developers **where&why code is losing perf.**
- **Parallel profiler**, especially made for MPI applications
- Effortless profiling
  - ↪ no code modifications needed, may not even need to recompile
- Clear **view of bottlenecks**
  - ↪ in I/O, compute, thread or multi-process activity
- Deep insight in **CPU instructions affecting perf.**
  - ↪ vectorization and memory bandwidth
- **Memory usage over time** – see changes in memory footprint
- Integrated editing and building as for DDT

Full details at `allinea.com/products/map/features`

Allinea MAP - on ULHPC

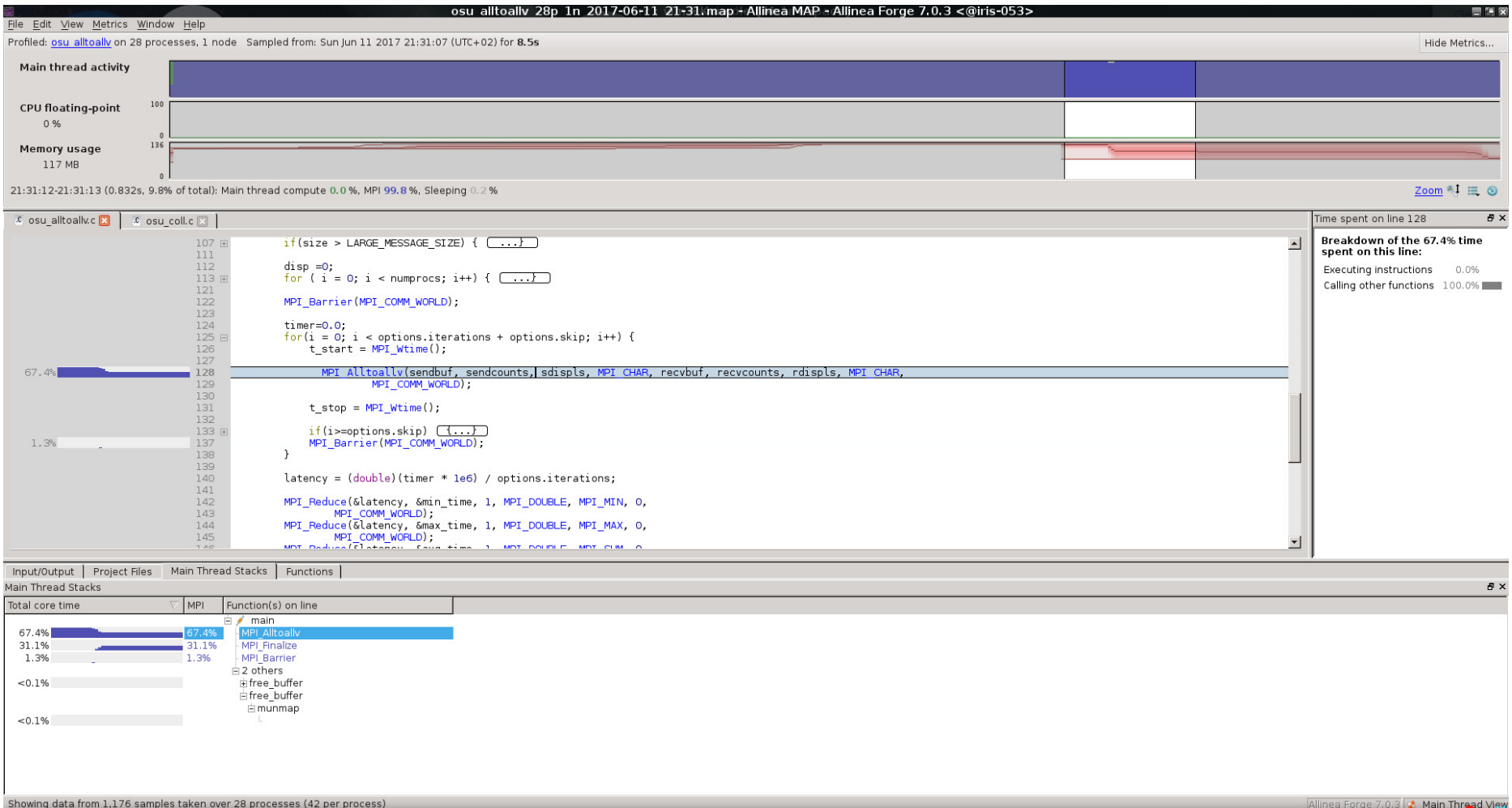
Modules

- On all clusters: `module load tools/AllineaForge`
- Caution! May behave differently between:
  - ↪ Debian+OAR (Gaia, Chaos) and CentOS+SLURM (Iris)

Profiling with MAP

1. Load toolchain that built your app., e.g.
   - ↪ `module load toolchain/intel`
2. Run your code through MAP (attached, GUI version)
   - ↪ iris: `map srun ./$app`
   - ↪ gaia/chaos: `map mpirun -hostfile $OAR_NODEFILE ./$app`
3. Run MAP in batch mode (no GUI, create .map file):
   - ↪ iris: `map --profile srun ./$app`

Allinea MAP - interface



Allinea Perf. Reports - highlights

Performance Reports features

- Meant to answer **How well do your apps. exploit your hw.?**
- Easy to use, on unmodified applications
  - ↪ outputs HTML, text, CSV, JSON reports
- One-glance view if application is:
  - ↪ **well-optimized** for the underlying hardware
  - ↪ running **optimally at** the given **scale**
  - ↪ **affected by** I/O, networking or threading **bottlenecks**
- Easy to integrate with continuous testing
  - ↪ programatically improve performance by continuous profiling
- **Energy metric** integrated
  - ↪ using RAPL (CPU) for now on iris
  - ↪ IPMI-based monitoring may be added later

Full details at `allinea.com/products/allinea-performance-reports`

Allinea Perf. Reports - on ULHPC

Modules

- On all clusters: `module load tools/AllineaReports`
- Caution! May behave differently between:
  - ↪ Debian+OAR (Gaia, Chaos) and CentOS+SLURM (Iris)
  - ↪ Gaia: can collect GPU metrics
  - ↪ Iris: can collect energy metrics

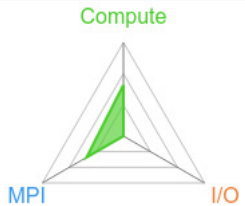
Using Performance Reports

1. Load toolchain that you run your app. with, e.g.
   - ↪ `module load toolchain/intel`
2. Run your application through Perf. Reports
   - ↪ iris: `perf-report srun ./$app`
   - ↪ gaia/chaos: `perf-report mpirun -hostfile $OAR_NODEFILE ./$app`
3. Analysis by default in .html and .txt indicating also run config.

Allinea Perf. Reports - output (I)

allinea PERFORMANCE REPORTS

Command:	srun gmx_mpi mdrun -s bench_rnase_cubic.tpr -nsteps 10000
Resources:	1 node (28 physical, 28 logical cores per node)
Memory:	126 GiB per node
Tasks:	28 processes, OMP_NUM_THREADS was 0
Machine:	iris-053
Start time:	Sun Jun 11 2017 20:13:59 (UTC+02)
Total time:	19 seconds
Full path:	/mnt/irisgpfs/apps/resif/data/production/v0.1-20170602/default/software/bio/GROMACS/2016.3-intel-2017a-hybrid/bin



Summary: gmx_mpi is Compute-bound in this configuration

Compute	54.6%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	45.4%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

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

CPU

A breakdown of the 54.6% CPU time:

Single-core code	5.5%
OpenMP regions	94.5%
Scalar numeric ops	5.2%
Vector numeric ops	44.2%
Memory accesses	50.6%

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

MPI

A breakdown of the 45.4% MPI time:

Time in collective calls	33.5%
Time in point-to-point calls	66.5%
Effective process collective rate	426 MB/s
Effective process point-to-point rate	419 MB/s

Most of the time is spent in point-to-point calls with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

Allinea Perf. Reports - output (II)

CPU

A breakdown of the 54.6% CPU time:

Single-core code	5.5%
OpenMP regions	94.5%
Scalar numeric ops	5.2%
Vector numeric ops	44.2%
Memory accesses	50.6%

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

MPI

A breakdown of the 45.4% MPI time:

Time in collective calls	33.5%
Time in point-to-point calls	66.5%
Effective process collective rate	426 MB/s
Effective process point-to-point rate	419 MB/s

Most of the time is spent in point-to-point calls with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

OpenMP

A breakdown of the 94.5% time in OpenMP regions:

Computation	99.5%
Synchronization	0.5%
Physical core utilization	100.0%
System load	101.9%

OpenMP thread performance looks good. Check the CPU breakdown for advice on improving code efficiency.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	75.6 MiB
Peak process memory usage	86.6 MiB
Peak node memory usage	11.0%

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

Energy

A breakdown of how the 0.899 Wh was used:

CPU	100.0%
System	not supported %
Mean node power	not supported W
Peak node power	not supported W

The whole system energy has been calculated using the CPU energy usage.

System power metrics: No Allinea IPMI Energy Agent config file found in (null). Did you start the Allinea IPMI Energy Agent?

Intel Advisor - highlights

Advisor features

- Vectorization Optimization and Thread Prototyping
- Analyze vectorization opportunities
  - ↪ for code compiled either with Intel and GNU compilers
  - ↪ SIMD, AVX* (incl. AVX-512) instructions
- Multiple data collection possibilities
  - ↪ loop iteration statistics
  - ↪ data dependencies
  - ↪ memory access patterns
- Suitability report - predict max. speed-up
  - ↪ based on app. modeling

Full details at `software.intel.com/en-us/intel-advisor-xe`

Intel Advisor - on ULHPC

Modules

- On iris/gaia/chaos: `module load perf/Advisor`

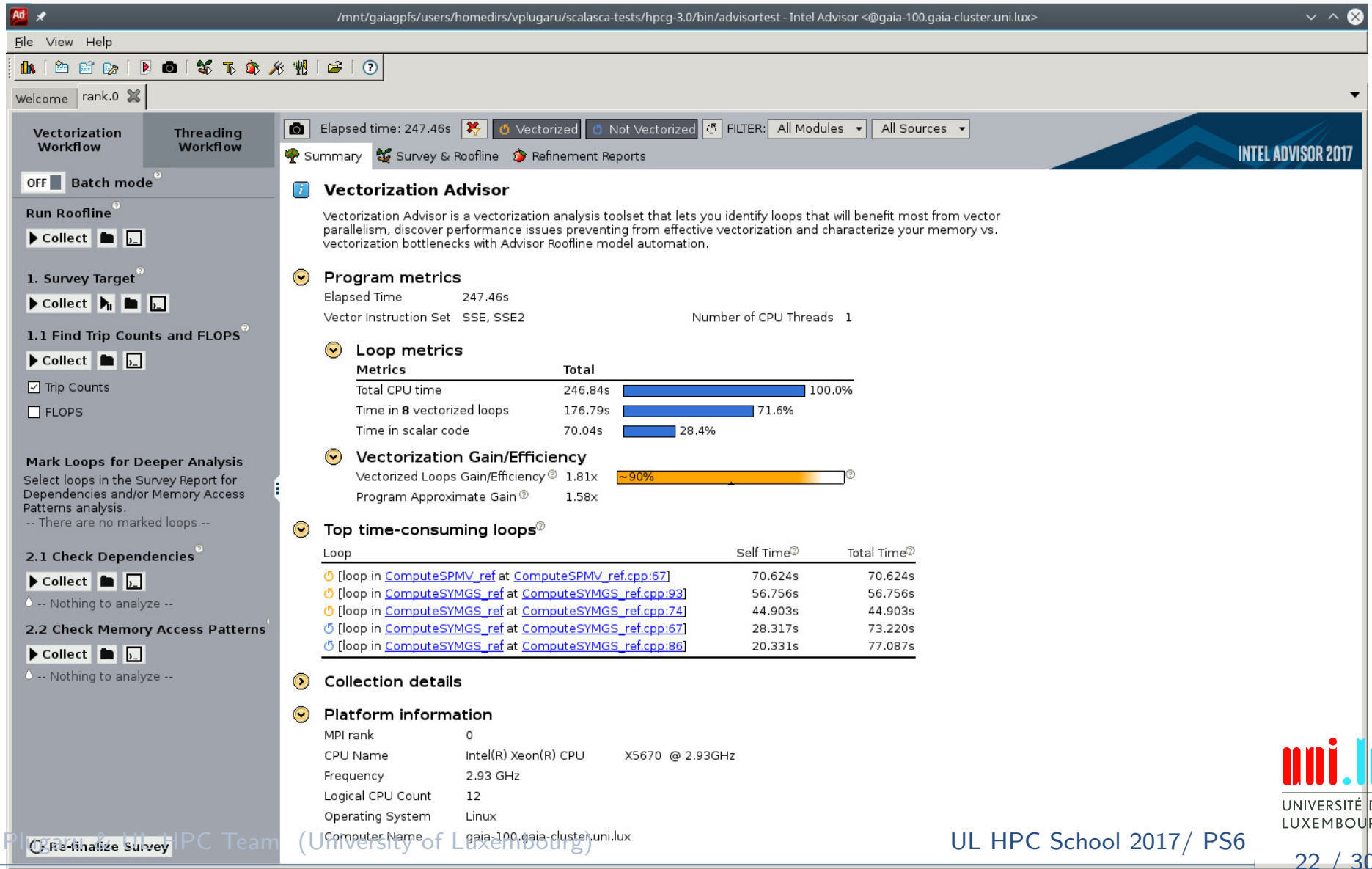
Using Intel Advisor

1. Load toolchain: `module load toolchain/intel`
2. Compile your code, e.g. `mpiicc $code.c -o $app`
3. Collect data e.g. on gaia:

```
mpirun -n 1 -gtool "advixe-cl -collect survey \
-project-dir ./advisortest:0" ./$app
```

4. Visualise results with `advixe-gui $HOME/advisortest`

Intel Advisor - interface



Scalasca & friends - highlights

Scalasca features

- Scalable performance analysis toolset
  - ↪ for large scale // applications on 100.000s of cores
- Support for C/C++/Fortran code with MPI, OpenMP, hybrid
- 3 stage workflow: instrument, measure, analyze
  - ↪ at compile time, run time and resp. postmortem
- Score-P for instrumentation + measurement, Cube for vis.
  - ↪ Score-P can also be used with Periscope, Vampir and Tau
- Facilities for measurement optimization to min. overhead
  - ↪ by selective recording, runtime filtering

Full details at `http://www.scalasca.org/about/about.html`

Scalasca - on ULHPC

Modules

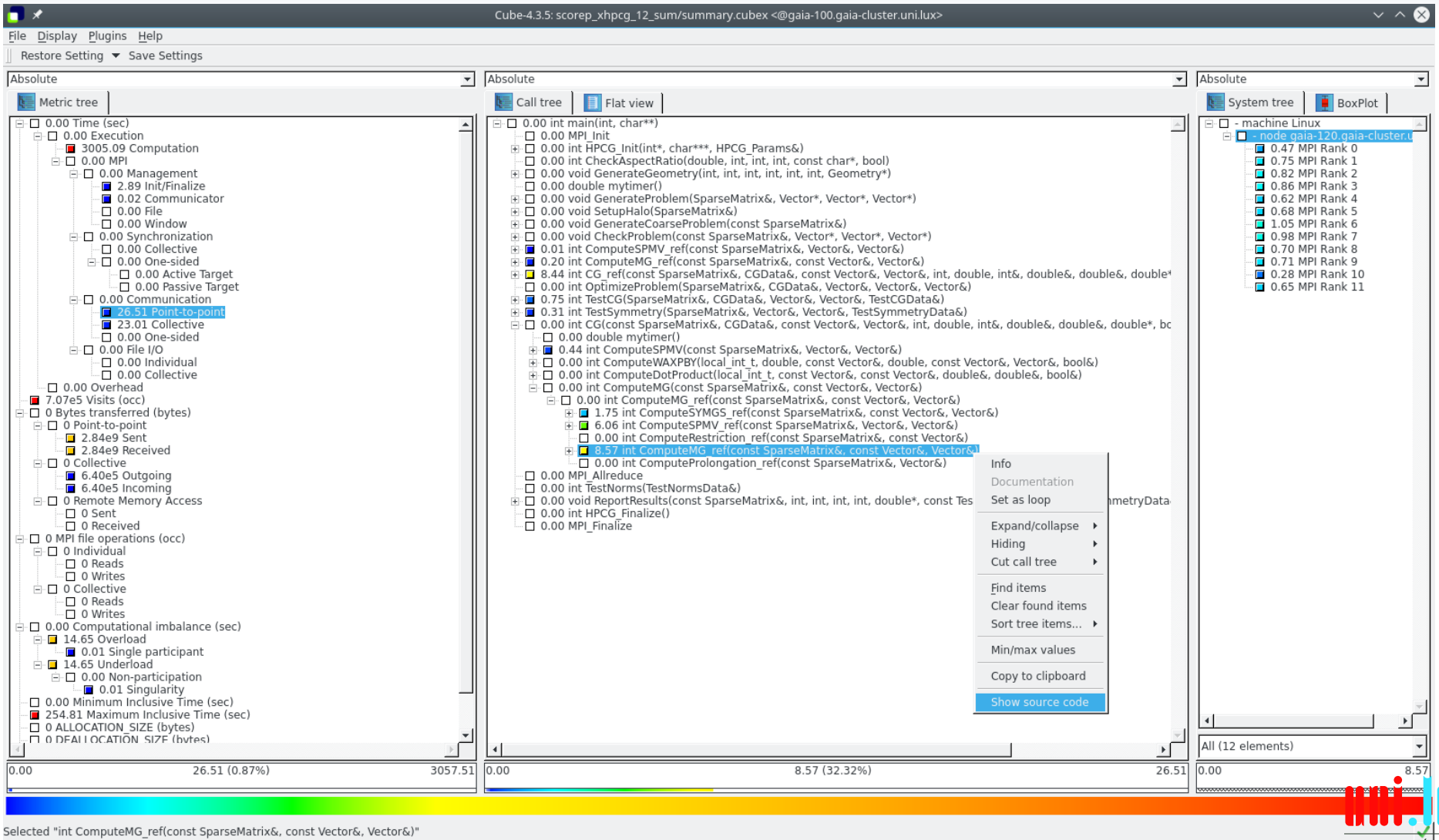
- On iris/gaia/chaos:

```
module load perf/Scalasca perf/Score-P
```

Using Scalasca

1. Load toolchain: `module load toolchain/foss`
2. Compile your code, e.g. `scorep mpicc $code.c -o $app`
3. Collect data e.g. on gaia: `scan -s mpirun -n 12 ./$app`
4. Visualise results with `square scorep_$app_12_sum`
   - ↪ or generate text report: `square -s scorep_$app_12_sum`
   - ↪ ... and print it: `cat scorep_$app_12_sum/scorep.score`

Scalasca visualisation with Cube-P



Summary

Now it's up to you

Easy right?

Now it's up to you

Easy right?

Well not exactly.

Now it's up to you

Easy right?

Well not exactly.
Debugging always takes effort and real applications are never trivial.

Now it's up to you

Easy right?

Well not exactly.
Debugging always takes effort and real applications are never trivial.

But we do guarantee it'll be /easier/ with these tools.

Conclusion and Practical Session start

We've discussed

- A couple of small utilities that can be of big help
- HPC oriented tools available for you on UL HPC

And now..

Short DEMO time!

Conclusion and Practical Session start

We've discussed

- A couple of small utilities that can be of big help
- HPC oriented tools available for you on UL HPC

And now..

Short DEMO time!

Your Turn!

Hands-on start

- We will first start with running HPCG (unmodified) as per:

http://ulhpc-tutorials.rtfd.io/en/latest/advanced/HPCG/

- ... your tasks:
  1. perform a timed first run using unmodified HPCG v3.0 (MPI only)
     - ✓ use `/usr/bin/time -v` to get details
     - ✓ single node, use $\geq$ `80 80 80` for input params (hpcg.dat)
  2. run HPCG (timed) through Allinea Perf. Report
     - ✓ use `perf-report` (bonus points if using `iris` to get energy metrics)
  3. instrument and measure HPCG execution with Scalasca
- Remember: pre-existing reservations for the workshop:
  - ↪ 'hpschool': Iris cluster resv. (use `--reservationname=hpcschool`)
  - ↪ 4248619: Gaia cluster regular nodes (use `-t inner=4248619`)
  - ↪ 4248620: Gaia cluster GPU nodes
  - ↪ 1614176: Chaos cluster

Questions?

http://hpc.uni.lu

High Performance Computing @ UL
Prof. Pascal Bouvry
Dr. Sebastien Varrette & the UL HPC Team
(V. Plugaru, S. Peter, H. Cartiaux & C. Parisot)
University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu