Uni.lu HPC School 2018

PS4b: Performance engineering - HPC debugging and profiling

Uni.lu High Performance Computing (HPC) Team

V. Plugaru

University of Luxembourg (UL), Luxembourg

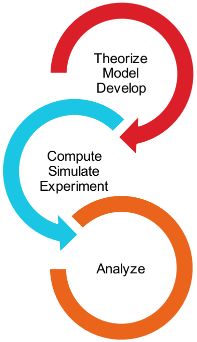
http://hpc.uni.lu

Latest versions available on Github:

UL HPC tutorials:	https://github.com/ULHPC/tutorials
UL HPC School:	http://hpc.uni.lu/hpc-school/
PS4b tutorial sources:	ulhpc-tutorials.rtfd.io/en/latest/debugging/advanced/

Summary

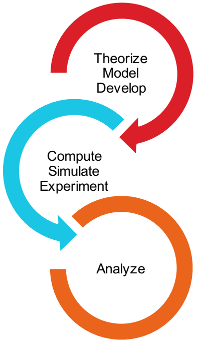
Main Objectives of this Session



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



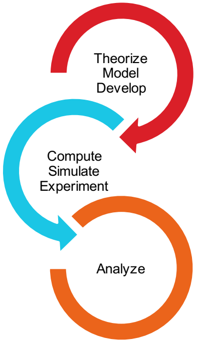
This session is meant to show you some of the various tools you have at your disposal on the UL HPC platform to:

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 - Arm (prev. Allinea)

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

Tools at your disposal (II)

HPC specific tools - Arm (prev. Allinea)

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

Arm tools are licensed

- license check integrated in SLURM: `scontrol show license`
- ask for licenses at job submission with e.g. `srun -L forge:16`

2017 software set lists the Arm tools under the previous *Allinea* name, the 2018 set will have them under *Arm*.

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
  - $\hookrightarrow$ Vectorization + threading advisor: check blockers and opport.
- Intel Inspector
  - $\hookrightarrow$ Memory and thread debugger: check leaks/corrupt., data races
- Intel Trace Analyzer and Collector
  - $\hookrightarrow$ MPI communications profiler and analyzer: evaluate patterns
- Intel VTune Amplifier
  - $\hookrightarrow$ 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`

Arm 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 Arm HPC Tools: Forge-DDT

Arm DDT - on ULHPC

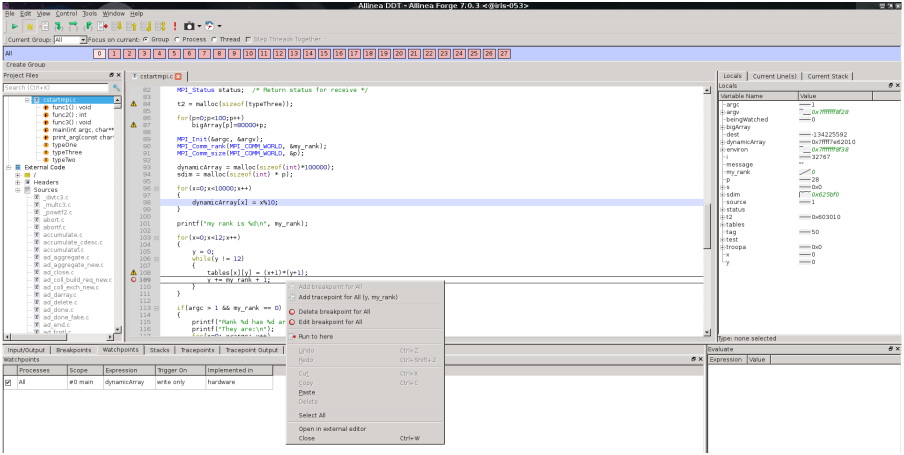
Modules

- 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`

Arm DDT - interface


Allinea DDT - Allinea Forge 7.0.3 <@iris-053>
File Edit View Control Tools Window Help
Current Group: All Focus on current: Group Process Thread Step Threads Together
All 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Create Group
Project Files
cstartmpi.c
Add breakpoint for All
Add tracepoint for All (y, my_rank)
Delete breakpoint for All
Edit breakpoint for All
Run to here
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste
Delete
Select All
Open in external editor
Close Ctrl+W
Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output
Locals Current Line(s) Current Stack
Evaluate
Expression Value
Type: none selected

```
    MPI_Status status;  /* Return status for receive */

    t2 = malloc(sizeof(typeThree));

    for(p=0;p<100;p++)
        bigArray[p]=80000+p;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    dynamicArray = malloc(sizeof(int)*100000);
    sdim = malloc(sizeof(int) * p);

    for(x=0;x<10000;x++)
    {
        dynamicArray[x] = x%10;
    }

    printf("my rank is %d\n", my_rank);

    for(x=0;x<12;x++)
    {
        y = 0;
        while(y != 12)
        {
            tables[x][y] = (x+1)*(y+1);
            y += my_rank + 1;
        }
    }

    if(argc > 1 && my_rank == 0)
    {
        printf("Rank %d has %d ar
        printf("They are:\n");
```

Locals

Variable Name	Value
argc	1
argv	0x7fffffff8f28
beingWatched	0
bigArray	
dest	-134225592
dynamicArray	0x7ffff7e62010
environ	0x7fffffff8f38
i	32767
message	""
my_rank	0
p	28
s	0x0
sdim	0x625bf0
source	1
status	
t2	0x603010
tables	
tag	50
test	
troopa	0x0
x	0
y	0

Watchpoints

	Processes	Scope	Expression	Trigger On	Implemented in
✓	All	#0 main	dynamicArray	write only	hardware

Arm 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 Arm HPC Tools: Forge-MAP

Arm MAP - on ULHPC

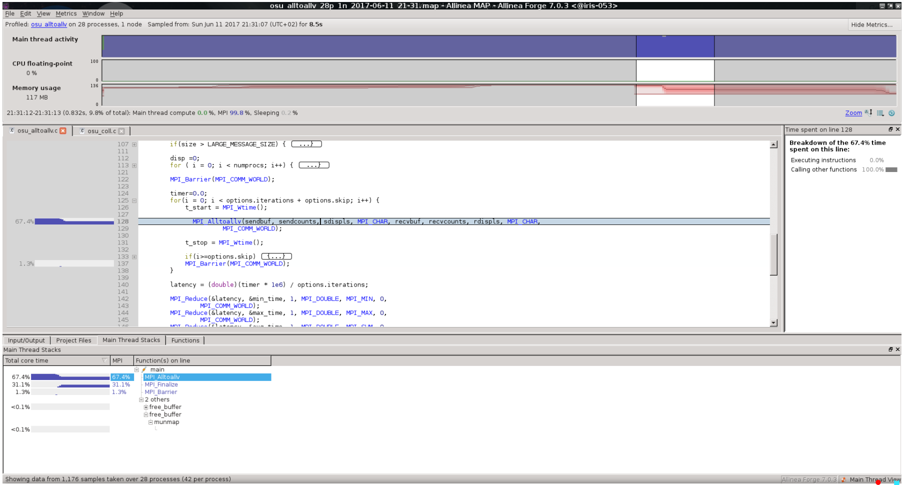
Modules

- 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`

Arm MAP - interface



Profiled: osu_alltoallv on 28 processes, 1 node Sampled from: Sun Jun 11 2017 21:31:07 (UTC+02) for 8.5s

21:31:12-21:31:13 (0.832s, 9.8% of total): Main thread compute 0.0 %, MPI 99.8 %, Sleeping 0.2 %

```
107         if(size > LARGE_MESSAGE_SIZE) { ...}
111
112         disp =0;
113         for ( i = 0; i < numprocs; i++) { ...}
121
122         MPI_Barrier(MPI_COMM_WORLD);
123
124         timer=0.0;
125         for(i = 0; i < options.iterations + options.skip; i++) {
126             t_start = MPI_Wtime();
127
128               MPI_Alltoallv(sendbuf, sendcounts, sdispls, MPI_CHAR, recvbuf, recvcounts, rdispls, MPI_CHAR,
129                       MPI_COMM_WORLD);
130
131             t_stop = MPI_Wtime();
132
133             if(i>=options.skip) {...}
137             MPI_Barrier(MPI_COMM_WORLD);
138         }
139
140         latency = (double)(timer * 1e6) / options.iterations;
141
142         MPI_Reduce(&latency, &min_time, 1, MPI_DOUBLE, MPI_MIN, 0,
143                 MPI_COMM_WORLD);
144         MPI_Reduce(&latency, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0,
145                 MPI_COMM_WORLD);
```

67.4% (line 128)
1.3% (line 137)

Time spent on line 128

Breakdown of the 67.4% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		main
67.4%	67.4%	MPI_Alltoallv
31.1%	31.1%	MPI_Finalize
1.3%	1.3%	MPI_Barrier
		2 others
<0.1%		free_buffer
		free_buffer
		munmap
<0.1%		

Showing data from 1,176 samples taken over 28 processes (42 per process)

Arm 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 Arm HPC Tools: Perf. Reports

Arm Perf. Reports - on ULHPC

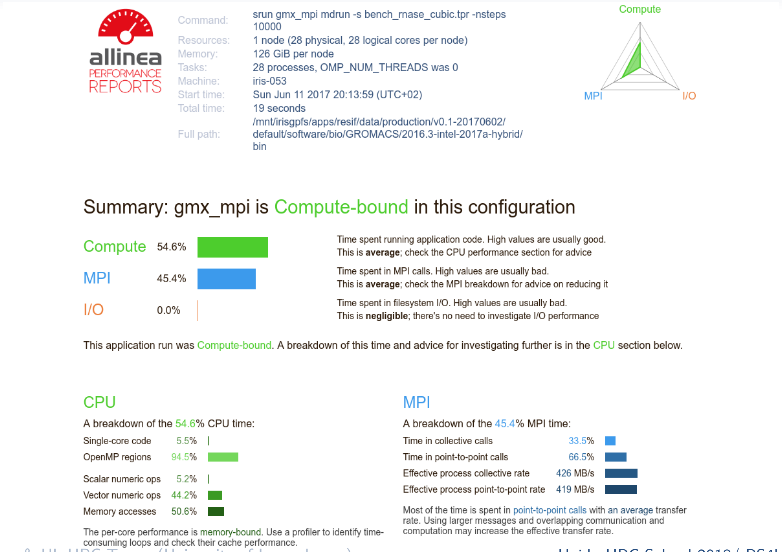
Modules

- 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.

Arm Perf. Reports - output (I)



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.

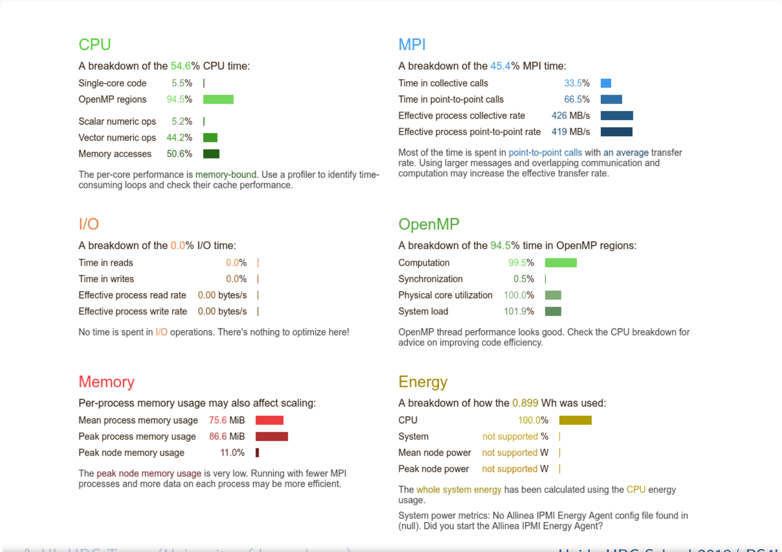
MPI

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.

Arm 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`

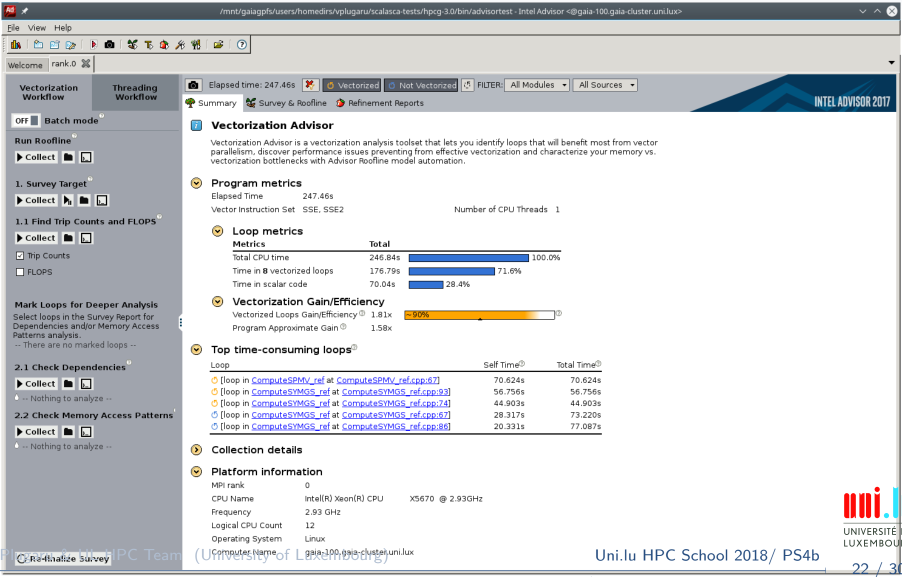
Using Intel Advisor

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



Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vector parallelism, discover performance issues preventing from effective vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automation.

Program metrics

Elapsed Time	247.46s		
Vector Instruction Set	SSE, SSE2	Number of CPU Threads	1

Loop metrics

Metrics	Total	
Total CPU time	246.84s	100.0%
Time in **8** vectorized loops	176.79s	71.6%
Time in scalar code	70.04s	28.4%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	1.81x	~90%
Program Approximate Gain	1.58x	

Top time-consuming loops

Loop	Self Time	Total Time
[loop in ComputeSPMV_ref at ComputeSPMV_ref.cpp:67]	70.624s	70.624s
[loop in ComputeSYMGS_ref at ComputeSYMGS_ref.cpp:93]	56.756s	56.756s
[loop in ComputeSYMGS_ref at ComputeSYMGS_ref.cpp:74]	44.903s	44.903s
[loop in ComputeSYMGS_ref at ComputeSYMGS_ref.cpp:67]	28.317s	73.220s
[loop in ComputeSYMGS_ref at ComputeSYMGS_ref.cpp:86]	20.331s	77.087s

Collection details

Platform information

MPI rank	0
CPU Name	Intel(R) Xeon(R) CPU X5670 @ 2.93GHz
Frequency	2.93 GHz
Logical CPU Count	12
Operating System	Linux
Computer Name	gaia-100.gaia-cluster.uni.lux

Scalasca & friends - highlights

Scalasca features

- Scalable performance analysis toolset
  - $\hookrightarrow$ 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
  - $\hookrightarrow$ at compile time, run time and resp. postmortem
- Score-P for instrumentation + measurement, Cube for vis.
  - $\hookrightarrow$ Score-P can also be used with Periscope, Vampir and Tau
- Facilities for measurement optimization to min. overhead
  - $\hookrightarrow$ by selective recording, runtime filtering

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

Scalasca - on ULHPC

Modules

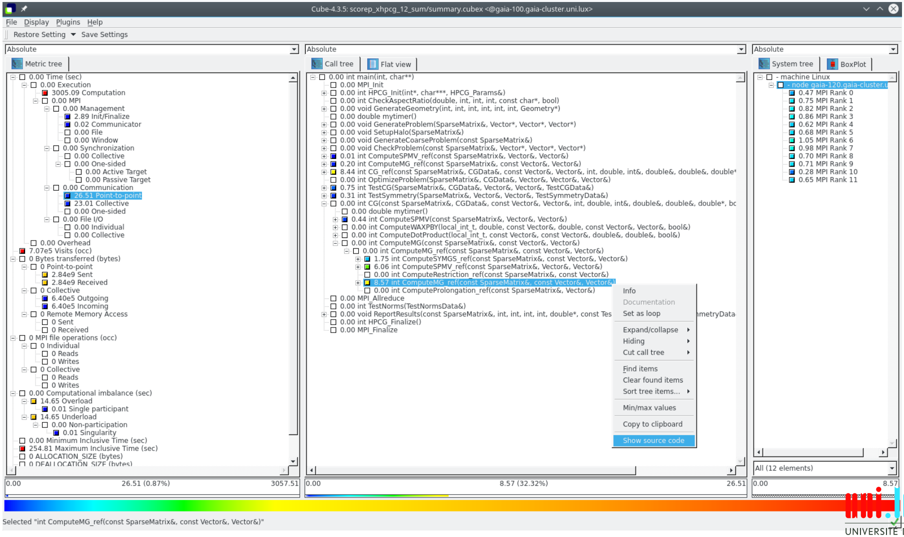
- On iris/gaia/chaos:

```
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


Cube-4.3.5: scorep_xhpcg_12_sum/summary.cubex <@gaia-100.gaia-cluster.uni.lux>

```
0.00 Time (sec)
  0.00 Execution
    3005.09 Computation
    0.00 MPI
      0.00 Management
        2.89 Init/Finalize
        0.02 Communicator
        0.00 File
        0.00 Window
      0.00 Synchronization
        0.00 Collective
        0.00 One-sided
          0.00 Active Target
          0.00 Passive Target
      0.00 Communication
        26.51 Point-to-point
        23.01 Collective
        0.00 One-sided
      0.00 File I/O
        0.00 Individual
        0.00 Collective
  0.00 Overhead
7.07e5 Visits (occ)
0 Bytes transferred (bytes)
  0 Point-to-point
    2.84e9 Sent
    2.84e9 Received
  0 Collective
    6.40e5 Outgoing
    6.40e5 Incoming
  0 Remote Memory Access
    0 Sent
    0 Received
0 MPI file operations (occ)
  0 Individual
    0 Reads
    0 Writes
  0 Collective
    0 Reads
    0 Writes
0.00 Computational imbalance (sec)
  14.65 Overload
    0.01 Single participant
  14.65 Underload
    0.00 Non-participation
      0.01 Singularity
0.00 Minimum Inclusive Time (sec)
254.81 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
```

```
0.00 int main(int, char**)
  0.00 MPI_Init
  0.00 int HPCG_Init(int*, char***, HPCG_Params&)
  0.00 int CheckAspectRatio(double, int, int, int, const char*, bool)
  0.00 void GenerateGeometry(int, int, int, int, int, int, Geometry*)
  0.00 double mytimer()
  0.00 void GenerateProblem(SparseMatrix&, Vector*, Vector*, Vector*)
  0.00 void SetupHalo(SparseMatrix&)
  0.00 void GenerateCoarseProblem(const SparseMatrix&)
  0.00 void CheckProblem(const SparseMatrix&, Vector*, Vector*, Vector*)
  0.01 int ComputeSPMV_ref(const SparseMatrix&, Vector&, Vector&)
  0.20 int ComputeMG_ref(const SparseMatrix&, const Vector&, Vector&)
  8.44 int CG_ref(const SparseMatrix&, CGData&, const Vector&, Vector&, int, double, int&, double&, double&, double*
  0.00 int OptimizeProblem(SparseMatrix&, CGData&, Vector&, Vector&, Vector&)
  0.75 int TestCG(SparseMatrix&, CGData&, Vector&, Vector&, TestCGData&)
  0.31 int TestSymmetry(SparseMatrix&, Vector&, Vector&, TestSymmetryData&)
  0.00 int CG(const SparseMatrix&, CGData&, const Vector&, Vector&, int, double, int&, double&, double&, double*, bc
    0.00 double mytimer()
    0.44 int ComputeSPMV(const SparseMatrix&, Vector&, Vector&)
    0.00 int ComputeWAXPBY(local_int_t, double, const Vector&, double, const Vector&, Vector&, bool&)
    0.00 int ComputeDotProduct(local_int_t, const Vector&, const Vector&, double&, double&, bool&)
    0.00 int ComputeMG(const SparseMatrix&, const Vector&, Vector&)
      0.00 int ComputeMG_ref(const SparseMatrix&, const Vector&, Vector&)
        1.75 int ComputeSYMGS_ref(const SparseMatrix&, const Vector&, Vector&)
        6.06 int ComputeSPMV_ref(const SparseMatrix&, Vector&, Vector&)
        0.00 int ComputeRestriction_ref(const SparseMatrix&, const Vector&)
        8.57 int ComputeMG_ref(const SparseMatrix&, const Vector&, Vector&)
        0.00 int ComputeProlongation_ref(const SparseMatrix&, Vector&)
  0.00 MPI_Allreduce
  0.00 int TestNorms(TestNormsData&)
  0.00 void ReportResults(const SparseMatrix&, int, int, int, int, double*, const Tes       metryData
  0.00 int HPCG_Finalize()
  0.00 MPI_Finalize
```

```
- machine Linux
  - node gaia-120.gaia-cluster.u
    0.47 MPI Rank 0
    0.75 MPI Rank 1
    0.82 MPI Rank 2
    0.86 MPI Rank 3
    0.62 MPI Rank 4
    0.68 MPI Rank 5
    1.05 MPI Rank 6
    0.98 MPI Rank 7
    0.70 MPI Rank 8
    0.71 MPI Rank 9
    0.28 MPI Rank 10
    0.65 MPI Rank 11
```

All (12 elements)

```
0.00          26.51 (0.87%)          3057.51
0.00          8.57 (32.32%)          26.51
0.00                                  8.57
```

Selected "int ComputeMG_ref(const SparseMatrix&, const Vector&, Vector&)"

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=hpcschoolday1`)
  - ↪ 4354151: Gaia cluster regular nodes (use `-t inner=4354151`)

Questions?

http://hpc.uni.lu

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement 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