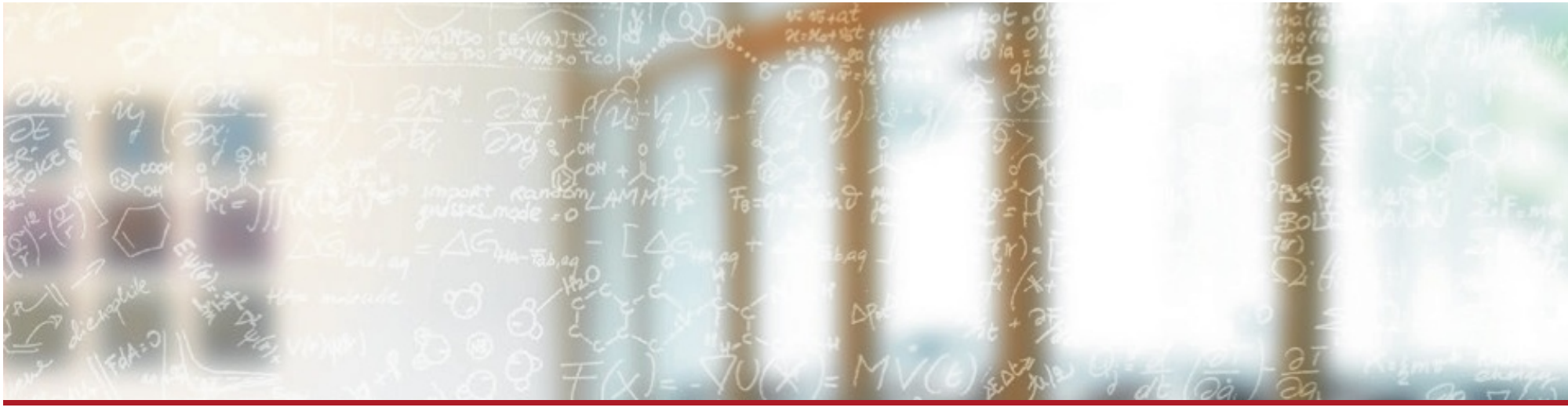
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Debugging and Performance Tools

CSCS User Lab Day

Matthias Kraushaar, JG. Piccinali

Tuesday, Sept. 11th 2018

Outline

- Introduction
- Debugging tools at CSCS
- Performance Analysis Tools at CSCS
- Concluding remarks

This presentation can be found here:
https://github.com/eth-cscs/UserLabDay/tree/master/slides/debugging_and_perf

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction

Why do this at all?

Performance data is a crucial part of your production proposal!
Performance analysis:

So that you are aware of...

- ...the limitations of your code on Piz Daint. (compute/communication/memory bound)
- ...how to best go about to improve your code's performance

Debugging: nobody wants to run a code that gives wrong results

Software used for this talk

- LAMMPS (https://lammps.sandia.gov):
  * A classical molecular dynamics code
  * Focus on materials modeling

  Used for the benchmark simulations.

- EasyBuild (https://github.com/eth-cscs/production):
  * A software build and installation framework

  Used to build LAMMPS.

- ReFrame (https://github.com/eth-cscs/reframe):
  * A framework for writing regression tests for HPC systems
  * In-house development at CSCS

  Used to run the LAMMPS simulation.

Software used for this talk: LAMMPS

http://lammps.sandia.gov

Wikipedia says:

- LAMMPS
  (**L**arge-scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator)
- A molecular dynamics program under the terms of the GNU General Public License
- Originally developed under a Cooperative Research and Development Agreement (CRADA) (DOE, private labs)
- As of 2016, it is maintained and distributed by researchers at the Sandia National Laboratories and Temple University.

For this presentation the Smooth Particle Hydrodynamics (SPH) user package of LAMMPS has been used.

Disclaimer

- What this talk is **not**:
  * A lecture on LAMMPS, SPH, EasyBuild, ReFrame
  * A lecture on how to do a performance analysis
  * How to write a production proposal

- What we are **not**:
  * LAMMPS developpers/experts
  * Performance/debugging tools developpers/experts

- Presentation contains profiling/debugging examples for
  * compiled languages (.c, .F), not interpreted ones (.py)
  * CPU performance only

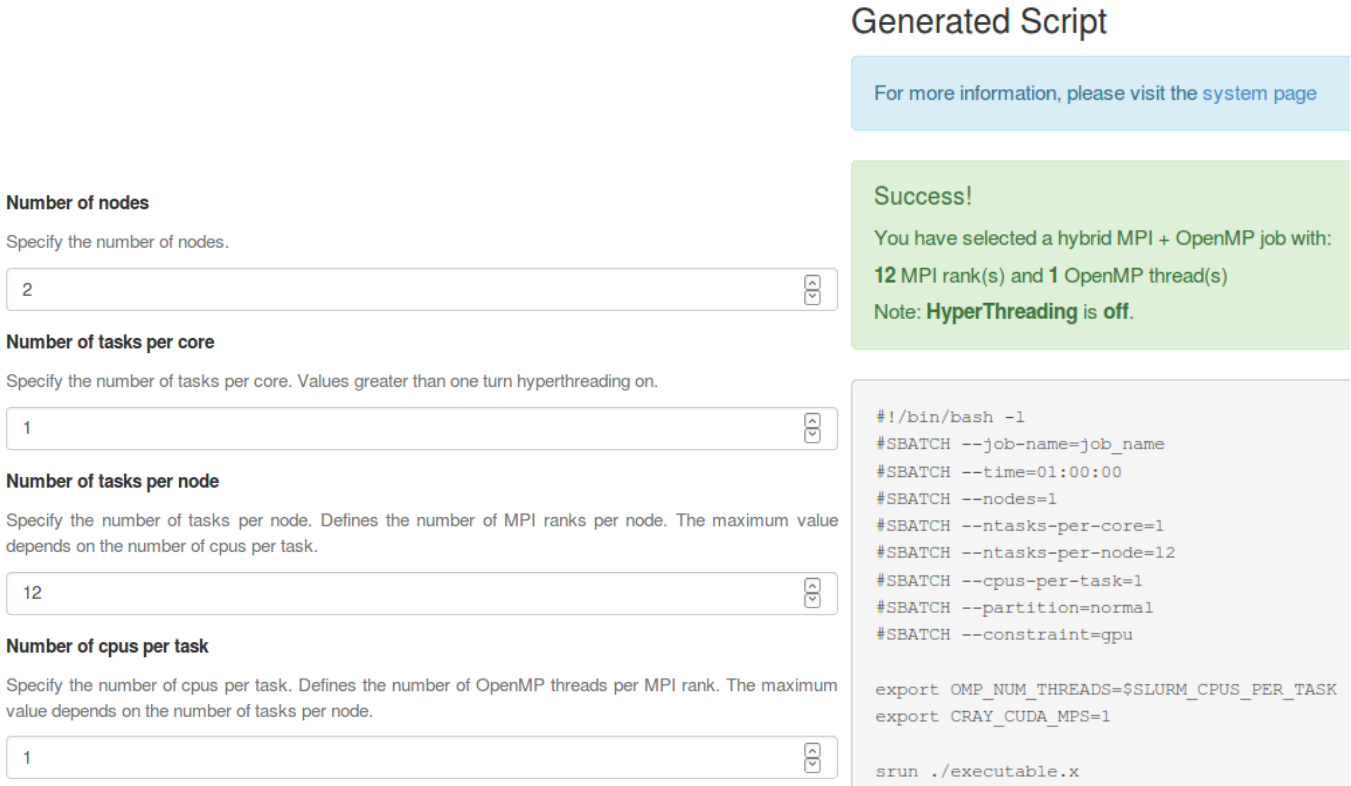
  Regarding the analysis and optimization of GPU codes:
  GPU Hackathon 2018

Piz Daint

`https://www.cscs.ch/computers/piz-daint/`

- Hybrid/Multicore Cray XC/40 and XC/50:
  - daint-mc: each XC/40 compute node hosts 2 Intel Broadwell CPUs
  - daint-gpu: each XC/50 compute node hosts 1 Intel Haswell CPU and 1 NVIDIA P100 GPU
  - Aries interconnect (dragonfly topology), Slurm

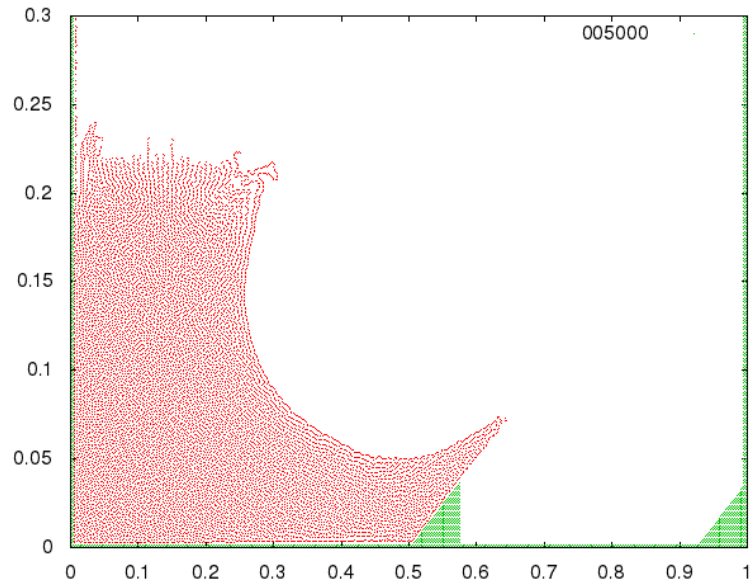
Piz Daint: slurm



- https://user.cscs.ch/access/running/jobscript_generator/

The test case

- Rectangular computational domain with obstacles.
- Boundary conditions are realised as SPH particles, which remain stationary.
- Water is modelled using Tait's EOS with
  - c = 10 m/s
  - ρ = 1000 kg/m^3
  - a = 10 (artificial viscosity)
- Variable time stepping, such that the fastest particle does not move more than $5/300 \cdot h$ (h - smoothing length)
- CFL is chosen, such that: $dt < 0.1 \frac{h}{c}$



Zoom into the solution of the water column simulation. (initial column is 4m x 0.25m)

- Red dots: water column
- Green dots: solid structures

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Tools at CSCS

Overview of the tools available on Piz Daint

Name	Dev/Maint organisation	License	Features
DDT	Arm Ltd.	Commercial	C, C++, Fortran
CrayPAT	Cray Inc.	Commercial	MPI, OpenMP, CUDA, OpenACC, ...
Score-P	SILC Partners	BSD	Serial, OpenMP, MPI, MPI+OpenMP
Scalasca	Research Center Jülich (FZJ)	Open source	MPI, OpenMP
Vampir	TU Dresden	Commercial	MPI, OpenMP, pthreads, CUDA, Java Threads
Extrae	Barcelona Supercomputing Center	LGPL	MPI, OpenMP, CUDA, pthreads, OmpSs
Intel Advisor	Intel	Commercial	Serial
Intel Vtune	Intel	Commercial	OpenMP, Intel TBB, native threads
likwid	RRZ Erlangen	Commercial	MPI, OpenMP
Extra-P	TU Darmstadt, LLNL, FZJ	New BSD	MPI, OpenMP

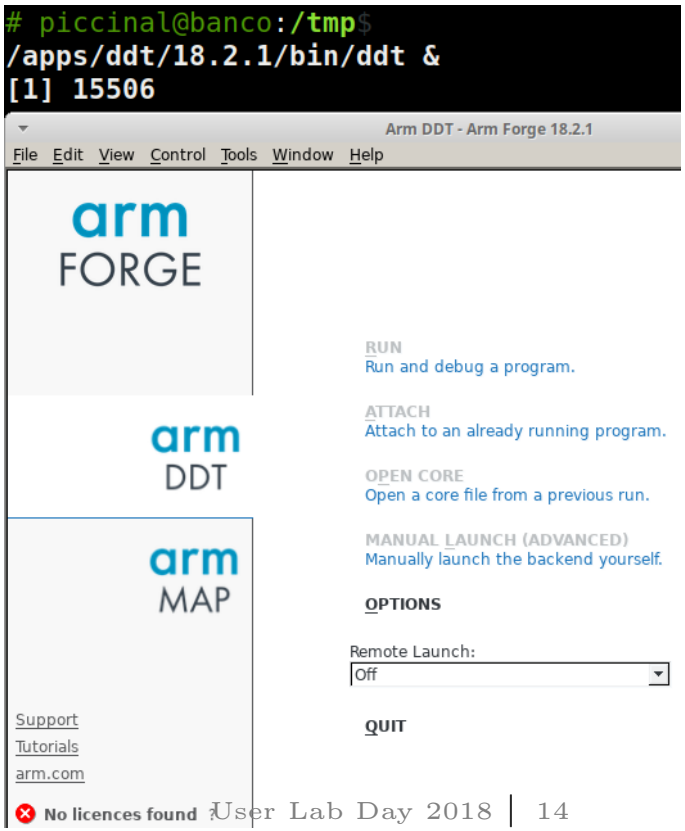
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

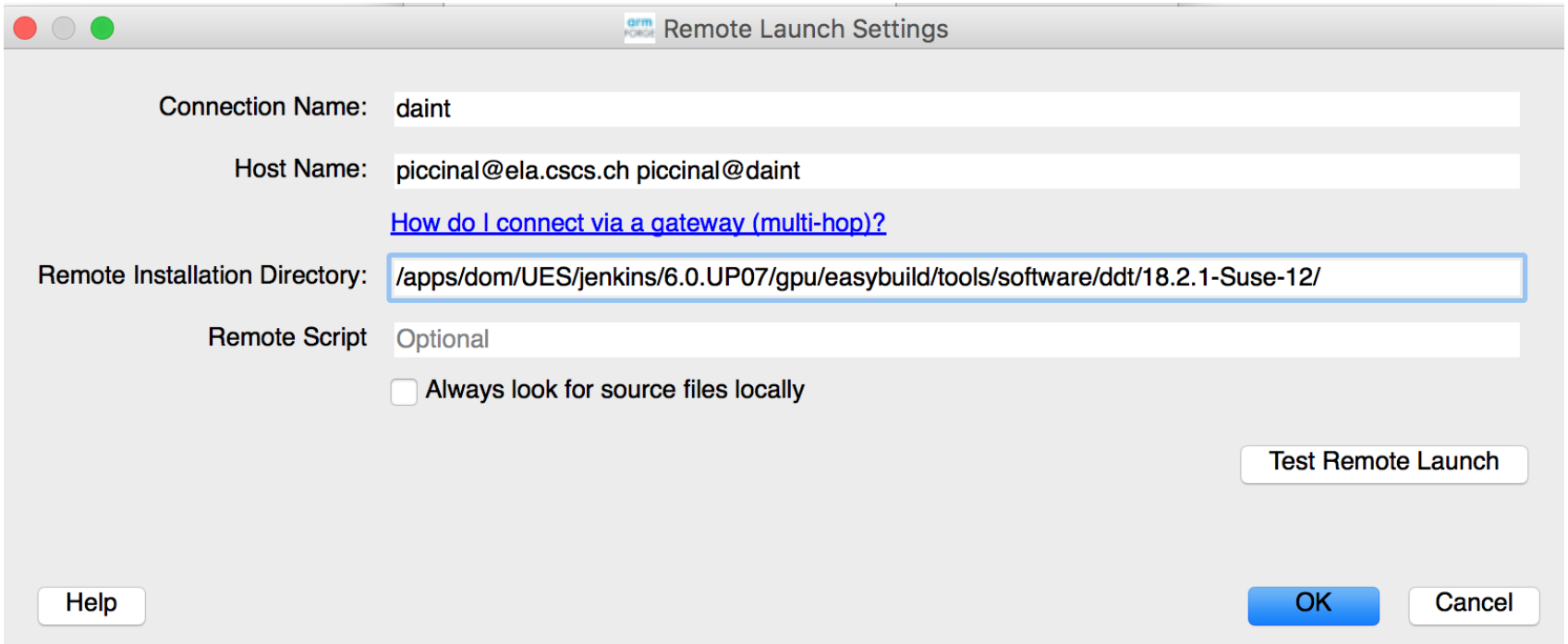
Debugging with DDT

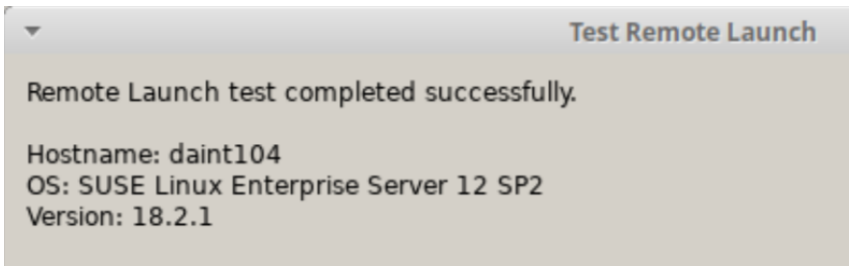
DDT: install the remote client

- `https://user.cscs.ch/computing/analysis/ddt/`
- Download the remote client from:
  - `https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge`
- and install it on your laptop.

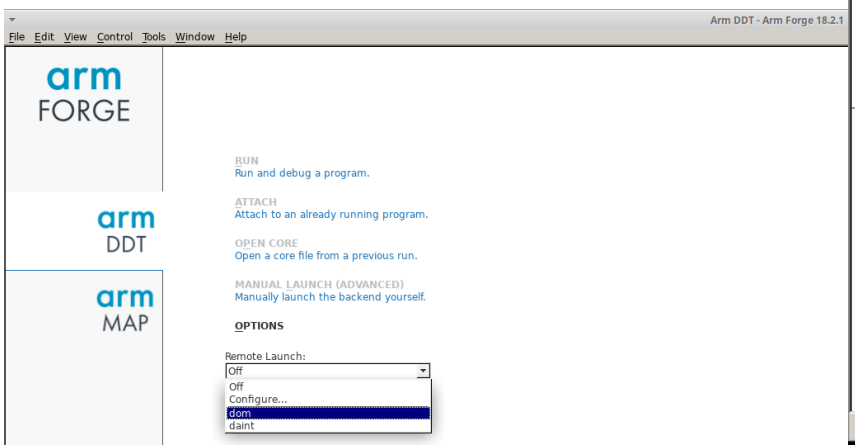


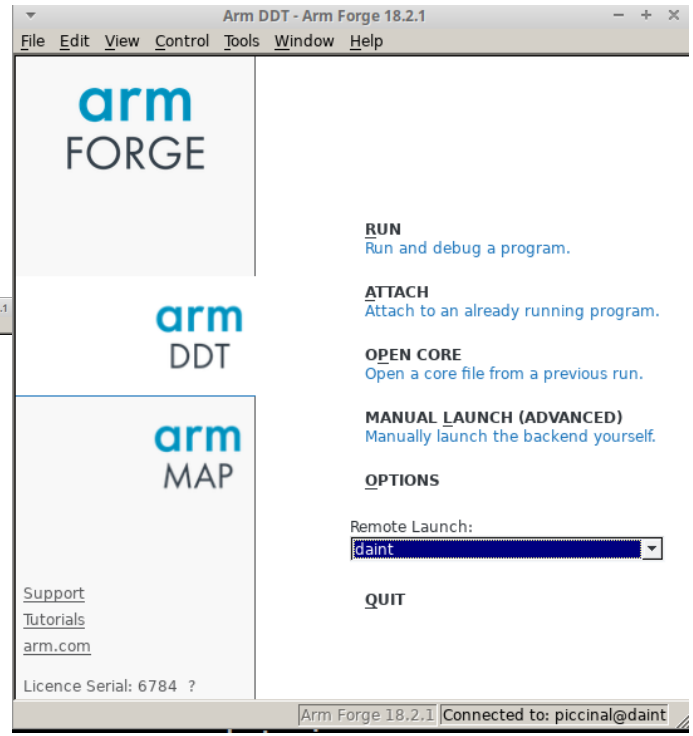
DDT: configure the remote client





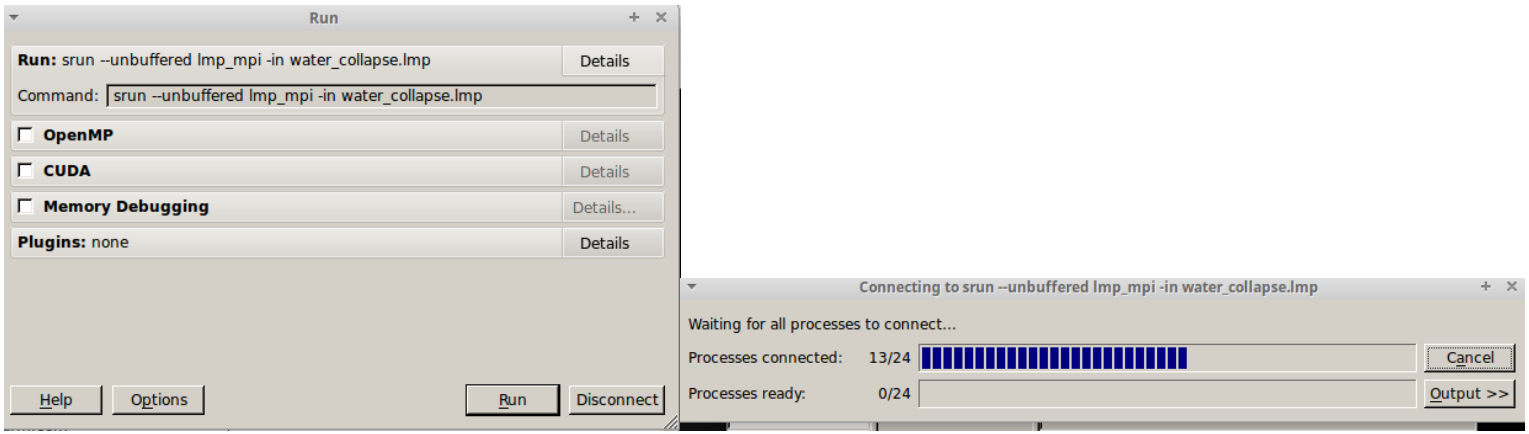
DDT: connect the remote client to Piz Daint





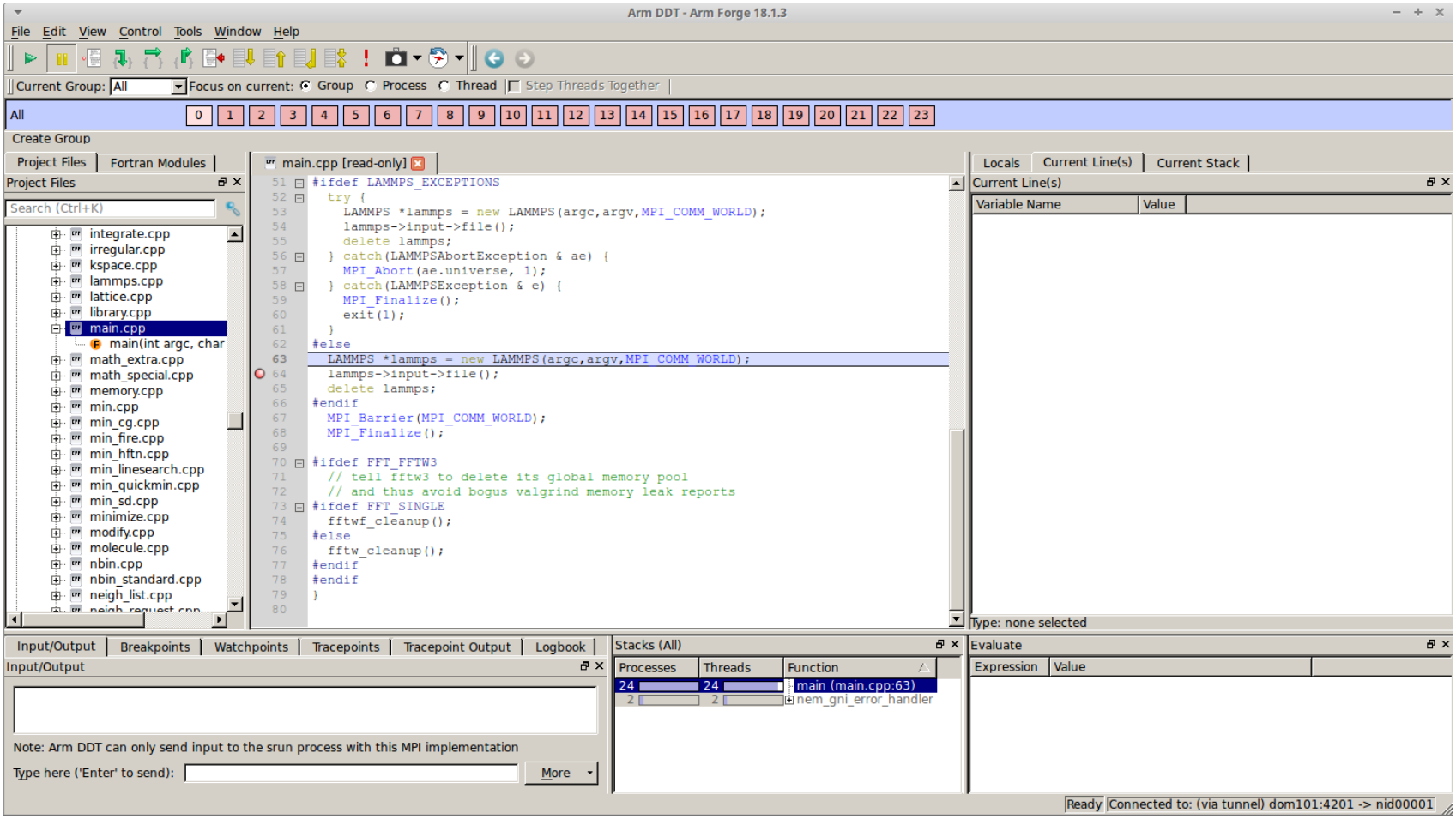
DDT: launch a job (on Piz Daint)

- easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-ddt.eb
- Submit the slurm job on daint and wait for the remote client to connect:
  * ssh daint
  * reframe: lammps_sph_strong_scaling_010000wp+ddt–connect.py
    i.e **ddt --connect** srun ...

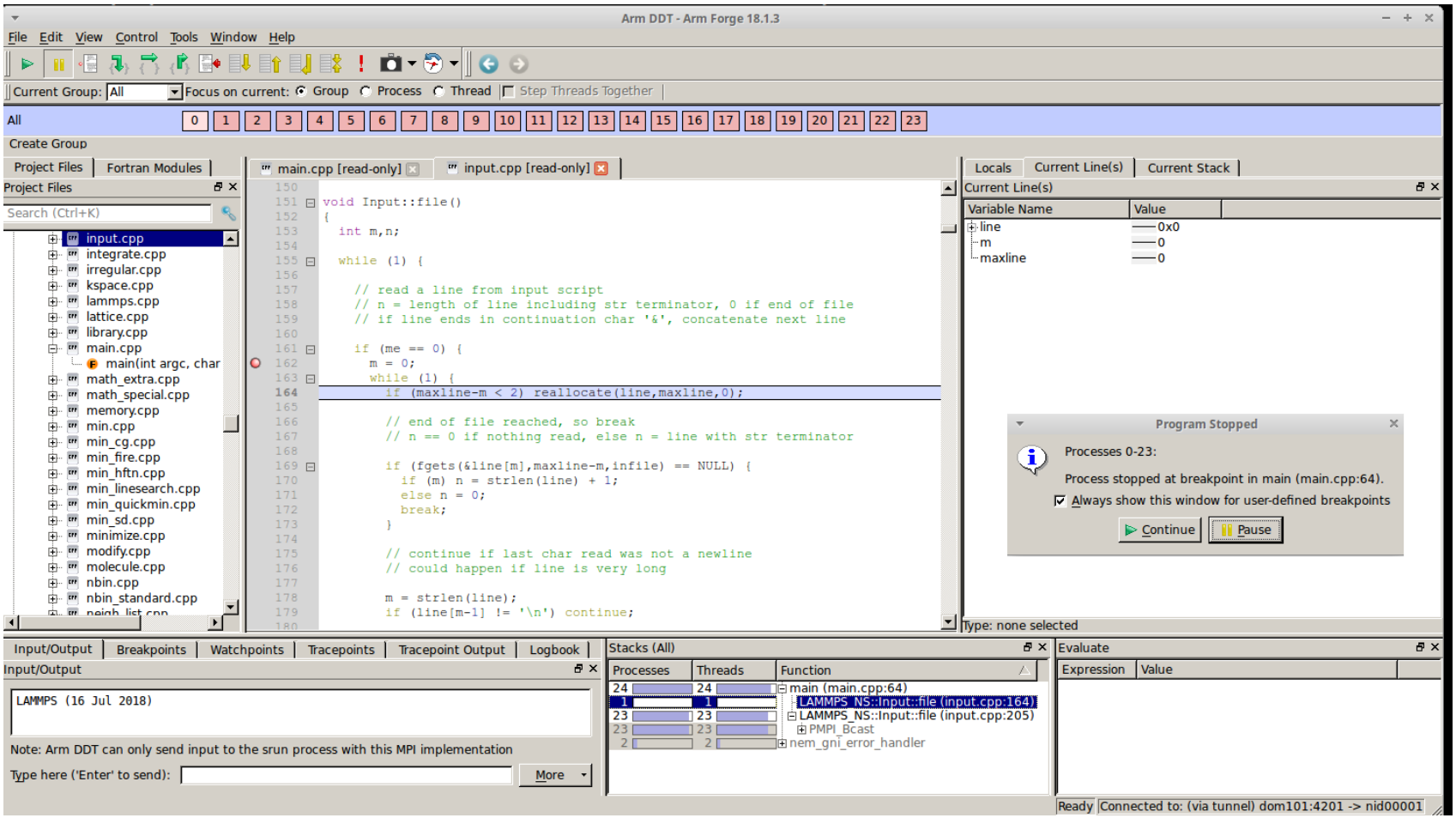


DDT: control the execution of the job

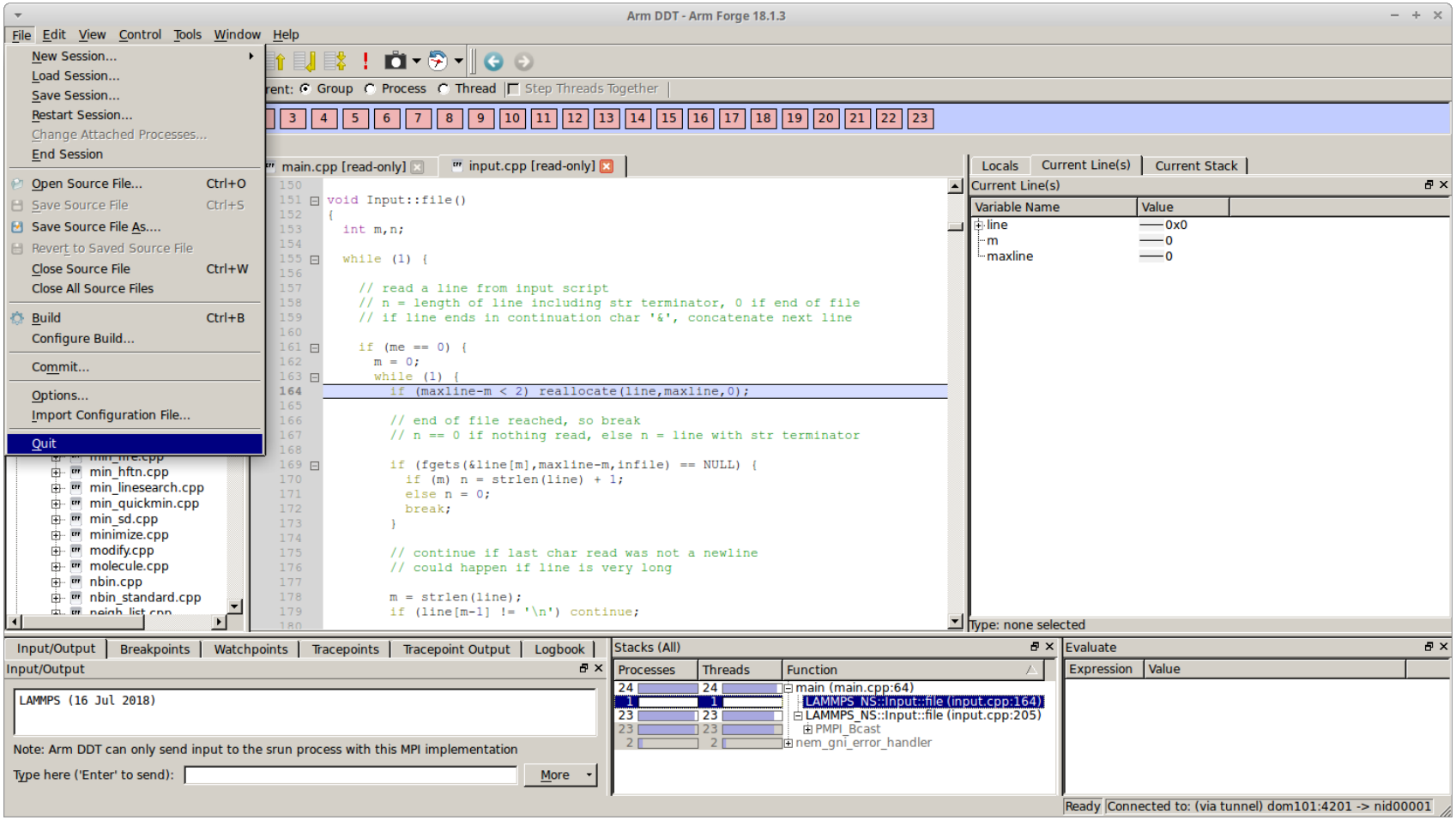
- ddt will stop and wait for your instructions:
- You can then set breakpoints, inspect data, navigate between tasks/threads, and much more...



DDT: breakpoint example

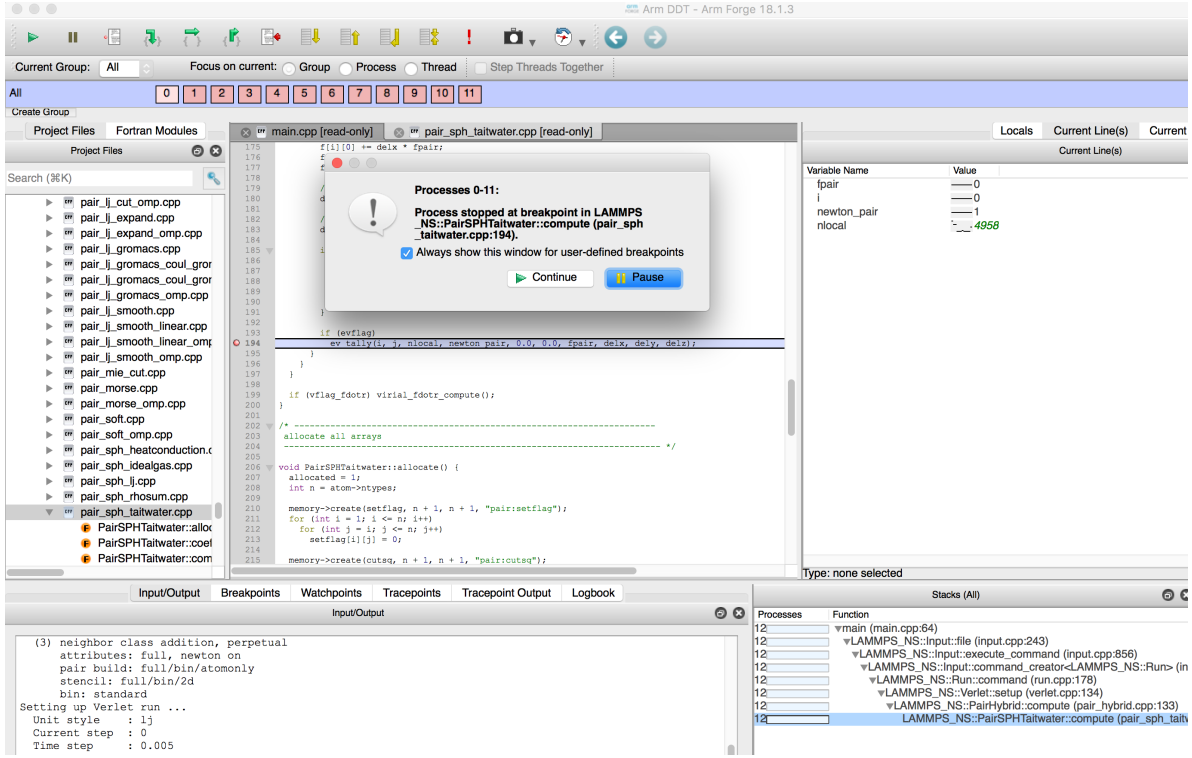


DDT: quit ddt when you are done



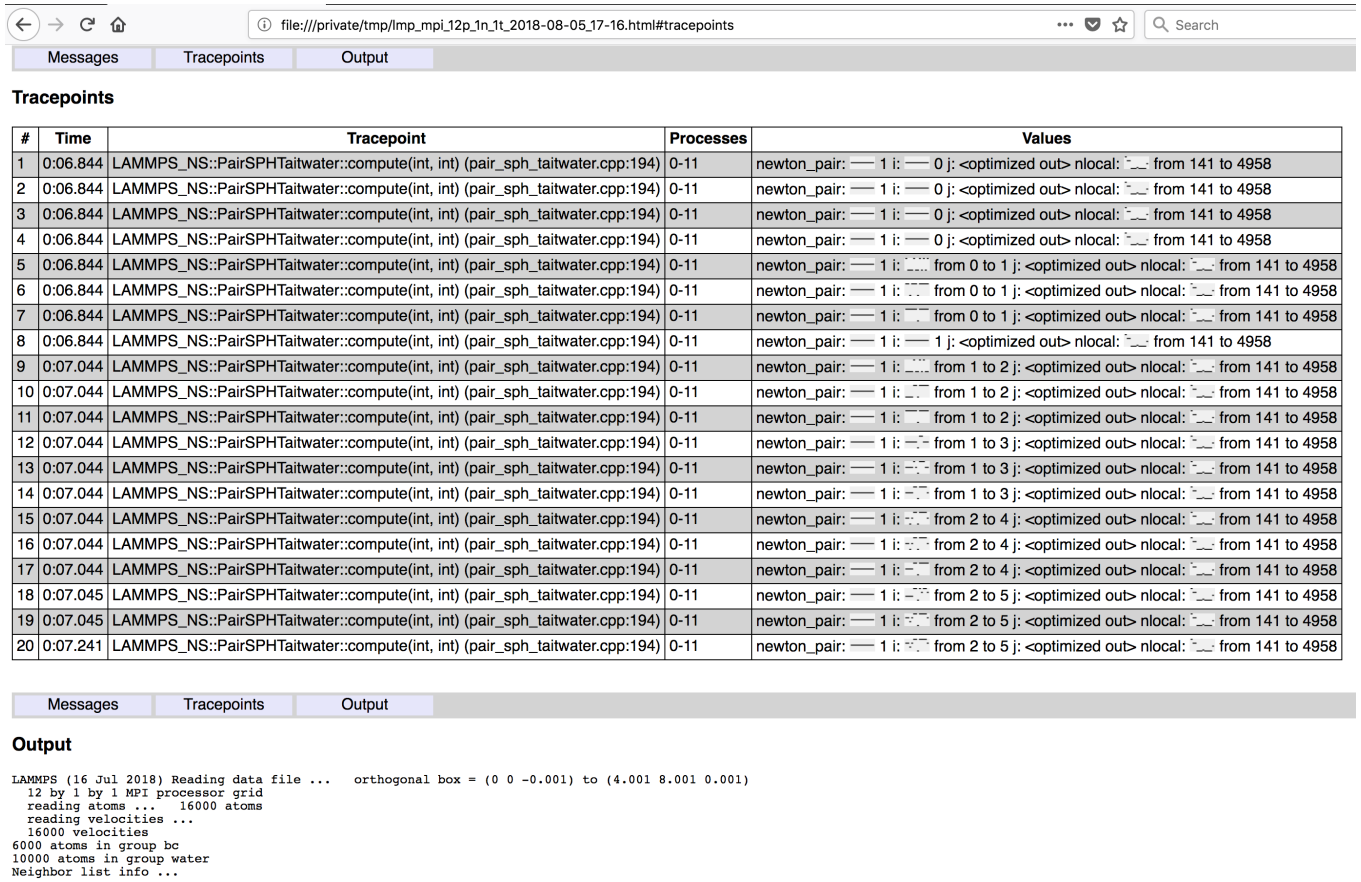
DDT: offline mode

- It is also possible to use ddt in offline mode:
  * ssh daint
  * reframe: lammps_sph_strong_scaling_010000wp+ddt-offline.py
    i.e **ddt --offline --trace-at** ... srun ...



DDT: offline mode report

- A debugging report will look like:



file:///private/tmp/lmp_mpi_12p_1n_1t_2018-08-05_17-16.html#tracepoints

Search

Messages | Tracepoints | Output

Tracepoints

#	Time	Tracepoint	Processes	Values
1	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — 0 j: <optimized out> nlocal: — from 141 to 4958
2	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — 0 j: <optimized out> nlocal: — from 141 to 4958
3	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — 0 j: <optimized out> nlocal: — from 141 to 4958
4	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — 0 j: <optimized out> nlocal: — from 141 to 4958
5	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 0 to 1 j: <optimized out> nlocal: — from 141 to 4958
6	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 0 to 1 j: <optimized out> nlocal: — from 141 to 4958
7	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 0 to 1 j: <optimized out> nlocal: — from 141 to 4958
8	0:06.844	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — 1 j: <optimized out> nlocal: — from 141 to 4958
9	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 2 j: <optimized out> nlocal: — from 141 to 4958
10	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 2 j: <optimized out> nlocal: — from 141 to 4958
11	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 2 j: <optimized out> nlocal: — from 141 to 4958
12	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 3 j: <optimized out> nlocal: — from 141 to 4958
13	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 3 j: <optimized out> nlocal: — from 141 to 4958
14	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 1 to 3 j: <optimized out> nlocal: — from 141 to 4958
15	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 4 j: <optimized out> nlocal: — from 141 to 4958
16	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 4 j: <optimized out> nlocal: — from 141 to 4958
17	0:07.044	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 4 j: <optimized out> nlocal: — from 141 to 4958
18	0:07.045	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 5 j: <optimized out> nlocal: — from 141 to 4958
19	0:07.045	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 5 j: <optimized out> nlocal: — from 141 to 4958
20	0:07.241	LAMMPS_NS::PairSPHTaitwater::compute(int, int) (pair_sph_taitwater.cpp:194)	0-11	newton_pair: — 1 i: — from 2 to 5 j: <optimized out> nlocal: — from 141 to 4958

Messages | Tracepoints | Output

Output

```
LAMMPS (16 Jul 2018) Reading data file ...   orthogonal box = (0 0 -0.001) to (4.001 8.001 0.001)
  12 by 1 by 1 MPI processor grid
  reading atoms ...   16000 atoms
  reading velocities ...
  16000 velocities
6000 atoms in group bc
10000 atoms in group water
Neighbor list info ...
```

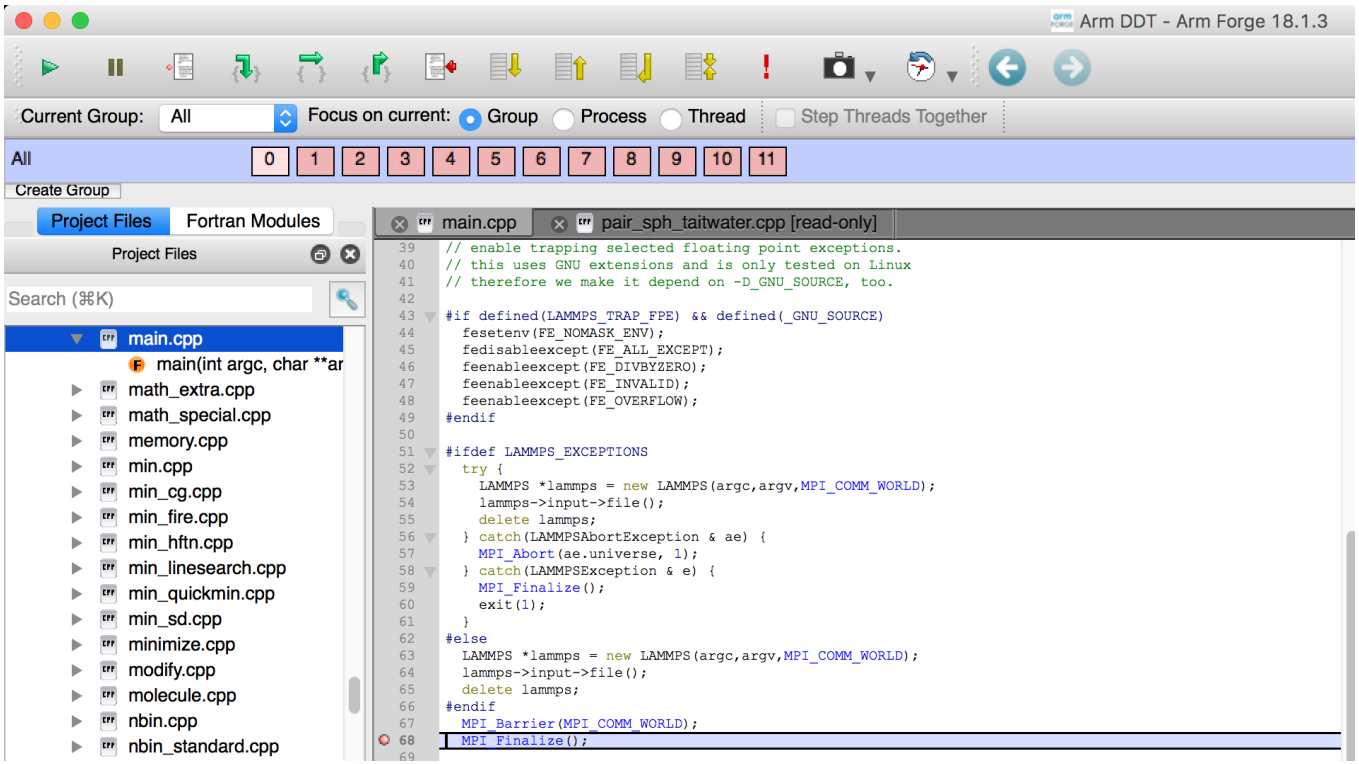
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

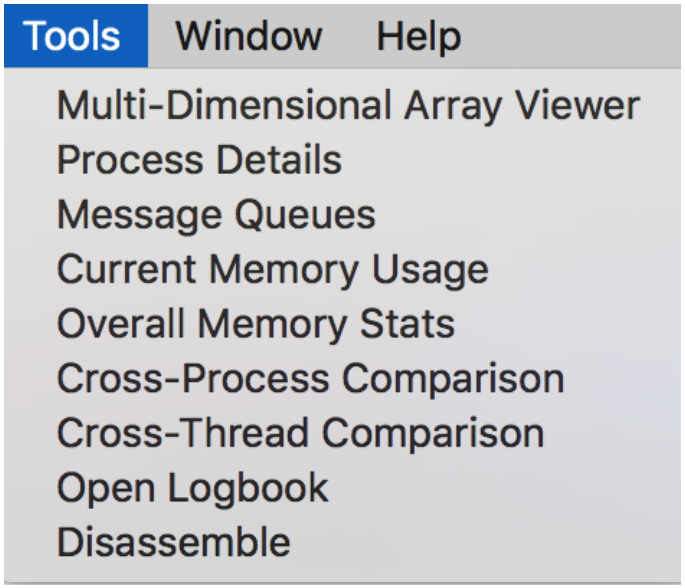
Memory debugging with DDT

DDT: memory debugging

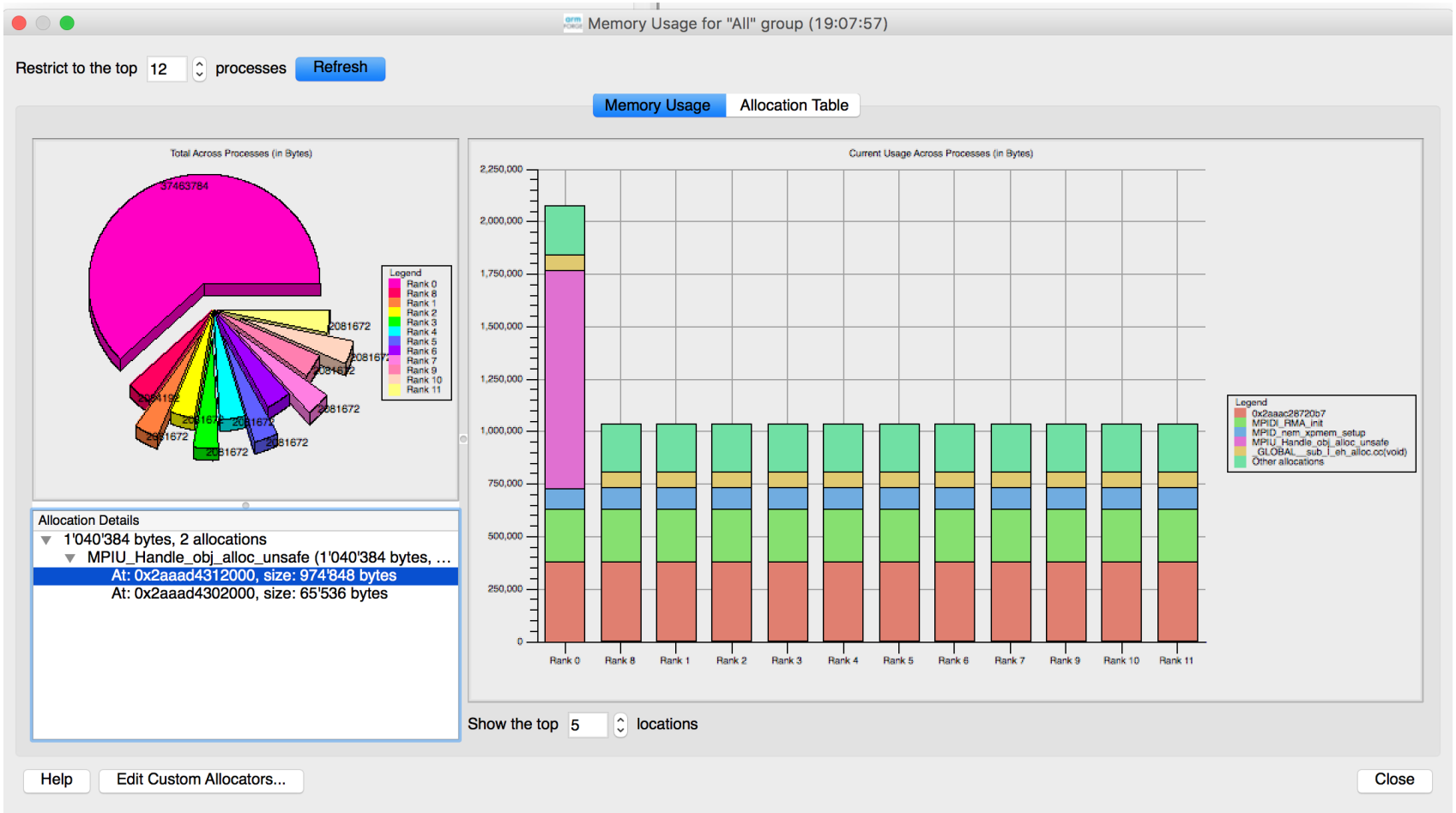
- easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-ddt.eb
- reframe: lammps_sph_strong_scaling_010000wp+ddt-connect+memory.py
  i.e **ddt --connect** srun ...



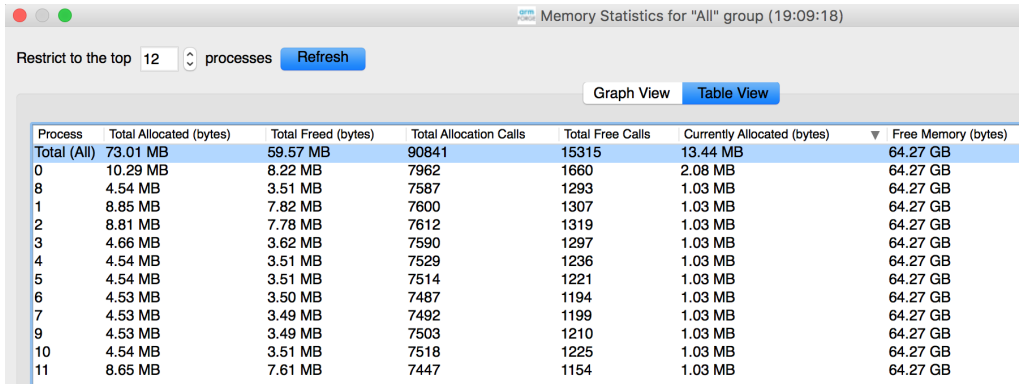
DDT: memory debugging options



DDT: memory usage

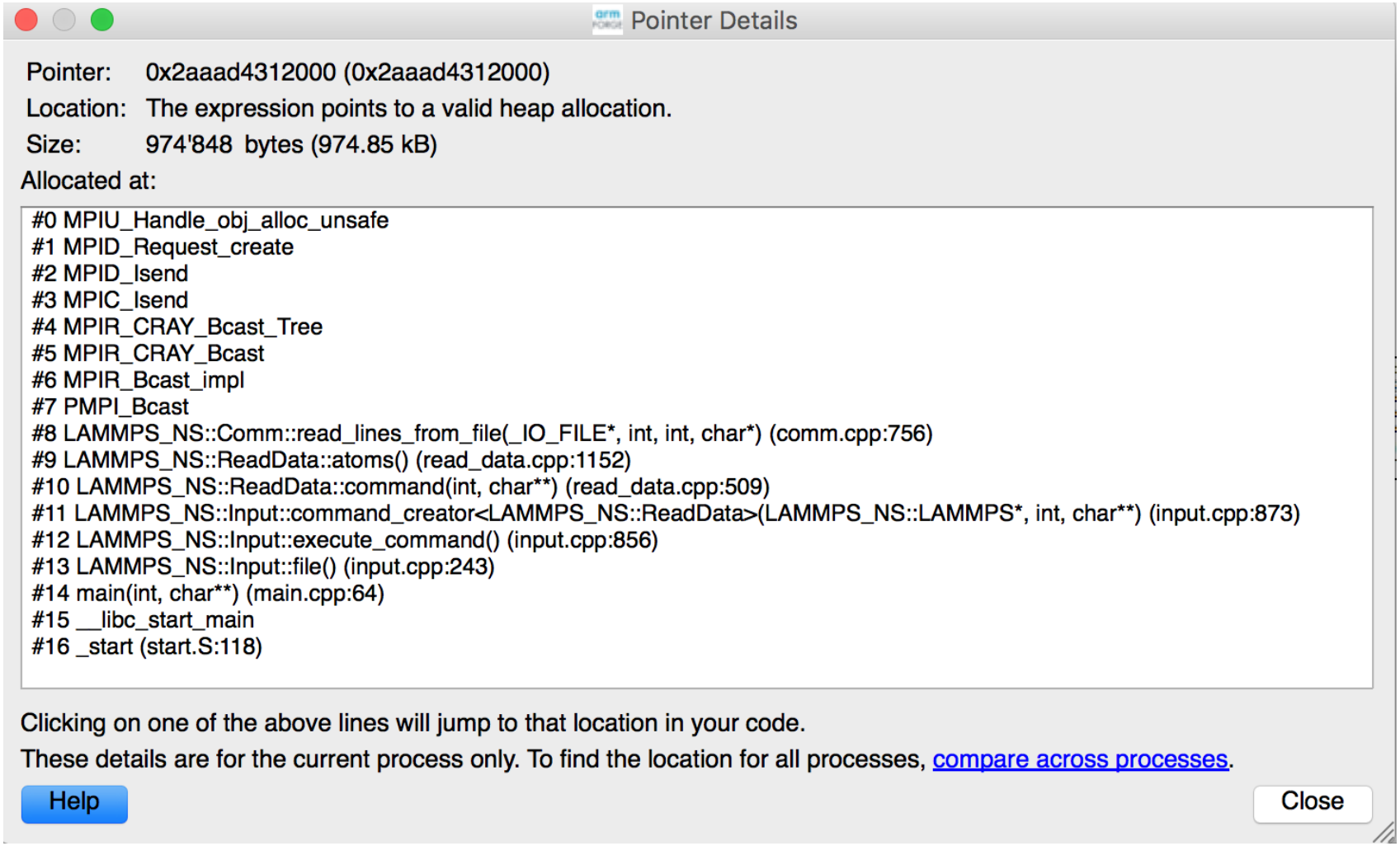


DDT: memory statistics



Process	Total Allocated (bytes)	Total Freed (bytes)	Total Allocation Calls	Total Free Calls	Currently Allocated (bytes)	Free Memory (bytes)
Total (All)	73.01 MB	59.57 MB	90841	15315	13.44 MB	64.27 GB
0	10.29 MB	8.22 MB	7962	1660	2.08 MB	64.27 GB
8	4.54 MB	3.51 MB	7587	1293	1.03 MB	64.27 GB
1	8.85 MB	7.82 MB	7600	1307	1.03 MB	64.27 GB
2	8.81 MB	7.78 MB	7612	1319	1.03 MB	64.27 GB
3	4.66 MB	3.62 MB	7590	1297	1.03 MB	64.27 GB
4	4.54 MB	3.51 MB	7529	1236	1.03 MB	64.27 GB
5	4.54 MB	3.51 MB	7514	1221	1.03 MB	64.27 GB
6	4.53 MB	3.50 MB	7487	1194	1.03 MB	64.27 GB
7	4.53 MB	3.49 MB	7492	1199	1.03 MB	64.27 GB
9	4.53 MB	3.49 MB	7503	1210	1.03 MB	64.27 GB
10	4.54 MB	3.51 MB	7518	1225	1.03 MB	64.27 GB
11	8.65 MB	7.61 MB	7447	1154	1.03 MB	64.27 GB

DDT: memory callstack



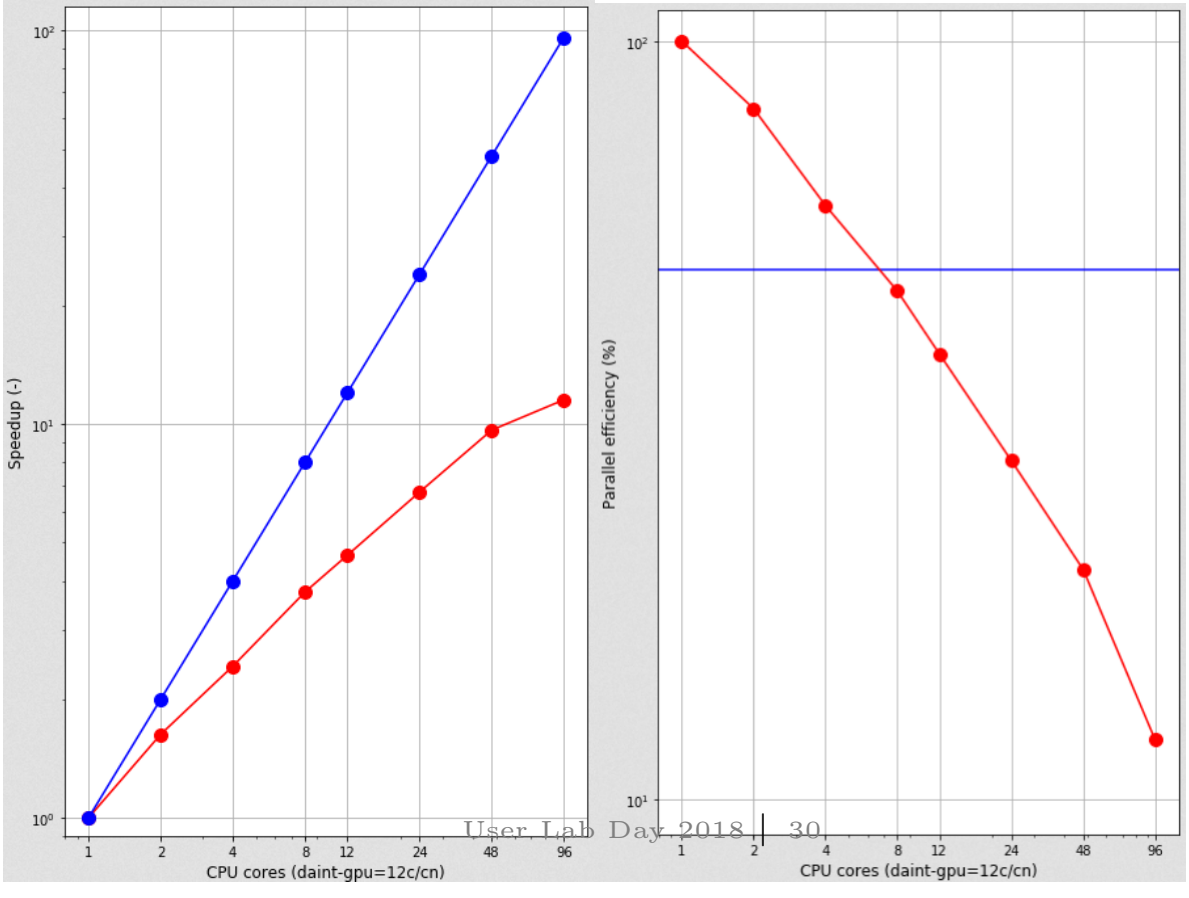
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Performance analysis with LAMMPS built-in timers

Strong scaling results

np	mpi tasks	seconds	measured speedup	ideal speedup	// efficiency
15702	01	317.51	1.0	001	100%
15702	02	194.92	1.6	002	081%
15702	04	130.78	2.4	004	061%
15702	08	84.75	3.7	008	047%
15702	12	68.39	4.6	012	039%
15702	24	47.26	6.7	024	028%
15702	48	32.88	9.7	048	020%
15702	96	27.54	11.5	096	012%



Timings reported by LAMMPS

- easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1.eb
- reframe: PrgEnv-gnu/LAMMPS_*_010000wp_strong.out
  i.e **srun** lmp_mpi ...

```
MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 0.021976   | 8.1453     | 28.253     | 409.7 | 23.36
Neigh   | 0.0037389  | 1.1927     | 5.0269     | 178.5 |  3.42
Comm    | 0.025769   | 2.5804     | 16.377     | 303.2 |  7.40
Output  | 0.00095081 | 0.00098038 | 0.0010929  |   0.0 |  0.00
Modify  | 0.59374    | 22.934     | 34.808     | 321.4 | 65.78
Other   |            | 0.01382    |            |       |  0.04
lammps.log                                                 56,26
```

```
181     timer->init();
182     timer->barrier_start();
183     update->integrate->run(nsteps);
184     timer->barrier_stop();
run.cpp
```

```
107 void Timer::init()
108 {
109   for (int i = 0; i < NUM_TIMER; i++) {
110     cpu_array[i] = 0.0;
111     wall_array[i] = 0.0;
112   }
113 }
timer.cpp
```

```
302   timer->stamp();
303
304   if (n_pre_force) {
305     modify->pre_force(vflag);
306     timer->stamp(Timer::MODIFY);
307   }
308
309   if (pair_compute_flag) {
310     force->pair->compute(eflag,vflag
311     timer->stamp(Timer::PAIR);
312   }
verlet.cpp                                                 3
```

```
117 void Timer::_stamp(enum ttype which)
118 {
119   double current_cpu=0.0, current_wall=0.0;
120
121   if (_level > NORMAL) current_cpu = CPU_Time();
122   current_wall = MPI_Wtime();
123
124   if ((which > TOTAL) && (which < NUM_TIMER)) {
125     const double delta_cpu = current_cpu - previous_cpu;
126     const double delta_wall = current_wall - previous_wall;
127
128     cpu_array[which]  += delta_cpu;
129     wall_array[which] += delta_wall;
130     cpu_array[ALL]    += delta_cpu;
131     wall_array[ALL]   += delta_wall;
132   }
timer.cpp
```

```
182   timer->barrier_start();
183   update->integrate->run(nsteps);
184   timer->barrier_stop();
185
186   update->integrate->cleanup();
187
188   Finish finish(lmp);
189   finish.end(postflag);
run.cpp
```

```
927
928   if (me == 0) {
929     tmp = time/time_loop*100.0;
930     if (t->has_full()) {
931       const char fmt[] = "%-8s|%- 12.5g|%- 12.5g|%- 12.5g|%6.1f |%6.1f |%6.
932       if (scr)
```

Timings reported by LAMMPS

- Timing breakdown (% of total time, between 2 and 12 mpi tasks, final step):

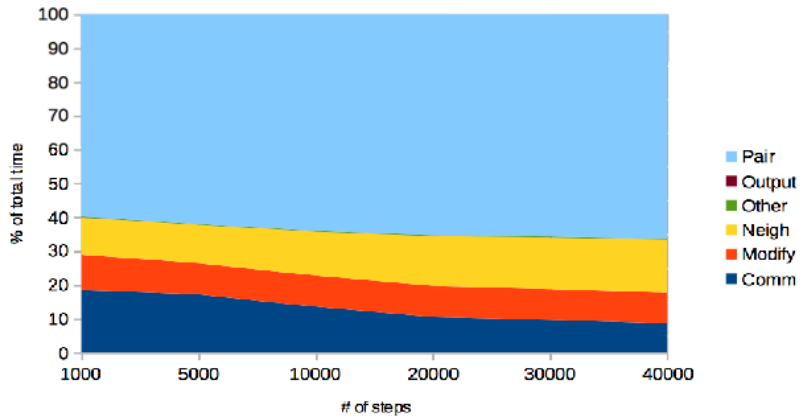
Pair:	66%	↘	35%
Neigh:	15%	↘	8%
Comm:	8%	↗	12%
Modify:	9%	↗	43%

  - `https://lammps.sandia.gov/doc/timer.html`

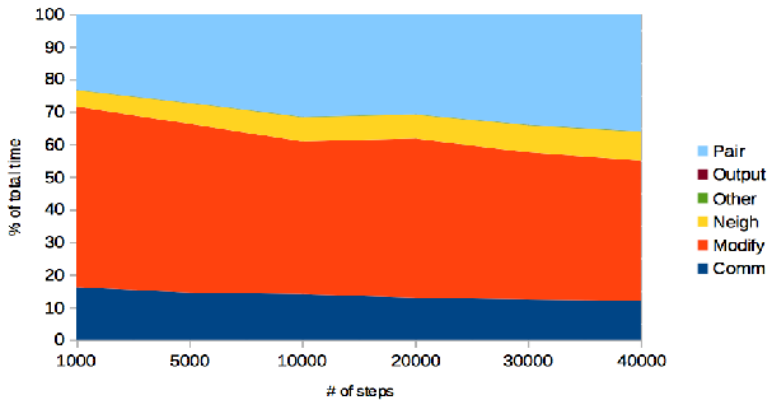
Timings reported by LAMMPS

- Duration of the job has to be considered too:

steps (2cores)	1000	5000	10000	20000	30000	40000
Comm	18.4	17.09	13.47	10.45	9.59	8.49
Modify	10.53	9.24	9.22	9.22	9.15	9.09
Neigh	10.92	11.41	12.96	14.7	15.21	15.74
Other	0.24	0.21	0.23	0.26	0.27	0.28
Output	0.02	0.01	0.01	0.01	0.01	0.01
Pair	59.9	62.04	64.11	65.36	65.76	66.38



steps (12cores)	1000	5000	10000	20000	30000	40000
Comm	16.02	14.4	13.95	12.87	12.34	11.95
Modify	55.5	51.85	46.84	48.82	45.14	42.95
Neigh	4.97	6.32	7.51	7.39	8.29	8.86
Other	0.12	0.12	0.14	0.13	0.15	0.16
Output	0.02	0.02	0.02	0.02	0.02	0.02
Pair	23.38	27.29	31.53	30.76	34.06	36.07



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Performance analysis with Cray tools

General approach to performance analysis

- **C** - compile your application with the tool
  read the easybuild scripts or check the tools' guide
- **P** - profile your application
  get an understanding of your code's performance
- **F** - filter before tracing your application
  reduce the amount of data recorded by the tool
- **T** - trace your application
  extract detailed information from the filtered part

Cray perftools: pat_run

- https://user.cscs.ch/computing/analysis/craypat/
  * easybuild: dynamically linked binaries (no recompilation needed)
  * reframe: lammps_sph_strong_scaling_100000wp+patrun.py
    i.e srun **pat_run** lmp_mpi ...
  * **pat_report** ./lmp_mpi+29655-1s/index.ap2

```
Table 1:  Profile by Function
  Samp% |     Samp |    Imb. |  Imb. | Group
        |          |    Samp | Samp% |  Function
        |          |         |       |   PE=HIDE
 100.0% | 13,169.3 |      -- |    -- | Total
|-------------------------------------------------------------------------
|  75.3% |  9,921.1 |      -- |    -- | MPI
||------------------------------------------------------------------------
||  68.4% |  9,013.9 | 4,112.1 | 32.0% | MPI_Allreduce
||   3.7% |    487.6 | 2,414.4 | 85.0% | MPI_Wait
||   3.0% |    389.9 | 1,370.1 | 79.5% | MPI_Send
||========================================================================
|  24.6% |  3,242.8 |      -- |    -- | USER
||------------------------------------------------------------------------
||  12.4% |  1,633.8 | 4,732.2 | 75.9% | LAMMPS_NS::PairSPHTaitwater::compute
||   7.2% |    949.5 | 2,822.5 | 76.4% | LAMMPS_NS::PairSPHRhoSum::compute
||   3.0% |    399.5 | 1,112.5 | 75.1% | LAMMPS_NS::NPairFullBinAtomonly::build
|========================================================================
```

Cray perftools: perftools-lite

- https://user.cscs.ch/computing/analysis/craypat/
  * man perftools-lite
  * easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-ptl-7.0.2.eb
  * reframe: lammps_sph_strong_scaling_100000wp+ptl.py

```
Table 1:  Profile by Function
   Samp% |    Samp |    Imb. |  Imb. | Group
         |         |    Samp | Samp% |  Function=[MAX10]
         |         |         |       |   PE=HIDE
  100.0% | 7,309.7 |      -- |    -- | Total
 |-------------------------------------------------------------------------
 |  76.4% | 5,587.5 |      -- |    -- | MPI
 ||------------------------------------------------------------------------
 ||  71.8% | 5,251.7 | 2,009.3 | 28.0% | MPI_Allreduce
 ||   2.2% |   157.7 | 2,550.3 | 95.2% | MPI_Wait
 ||   1.9% |   135.9 |   419.1 | 76.3% | MPI_Send
 ||========================================================================
 |  23.4% | 1,714.1 |      -- |    -- | USER
 ||------------------------------------------------------------------------
 ||  12.1% |   887.9 | 2,585.1 | 75.2% | LAMMPS_NS::PairSPHTaitwater::compute
 ||   6.7% |   492.7 | 1,474.3 | 75.7% | LAMMPS_NS::PairSPHRhoSum::compute
 ||   2.7% |   199.3 |   584.7 | 75.4% | LAMMPS_NS::NPairFullBinAtomonly::build
 |========================================================================
```

Cray perftools: how to choose ?

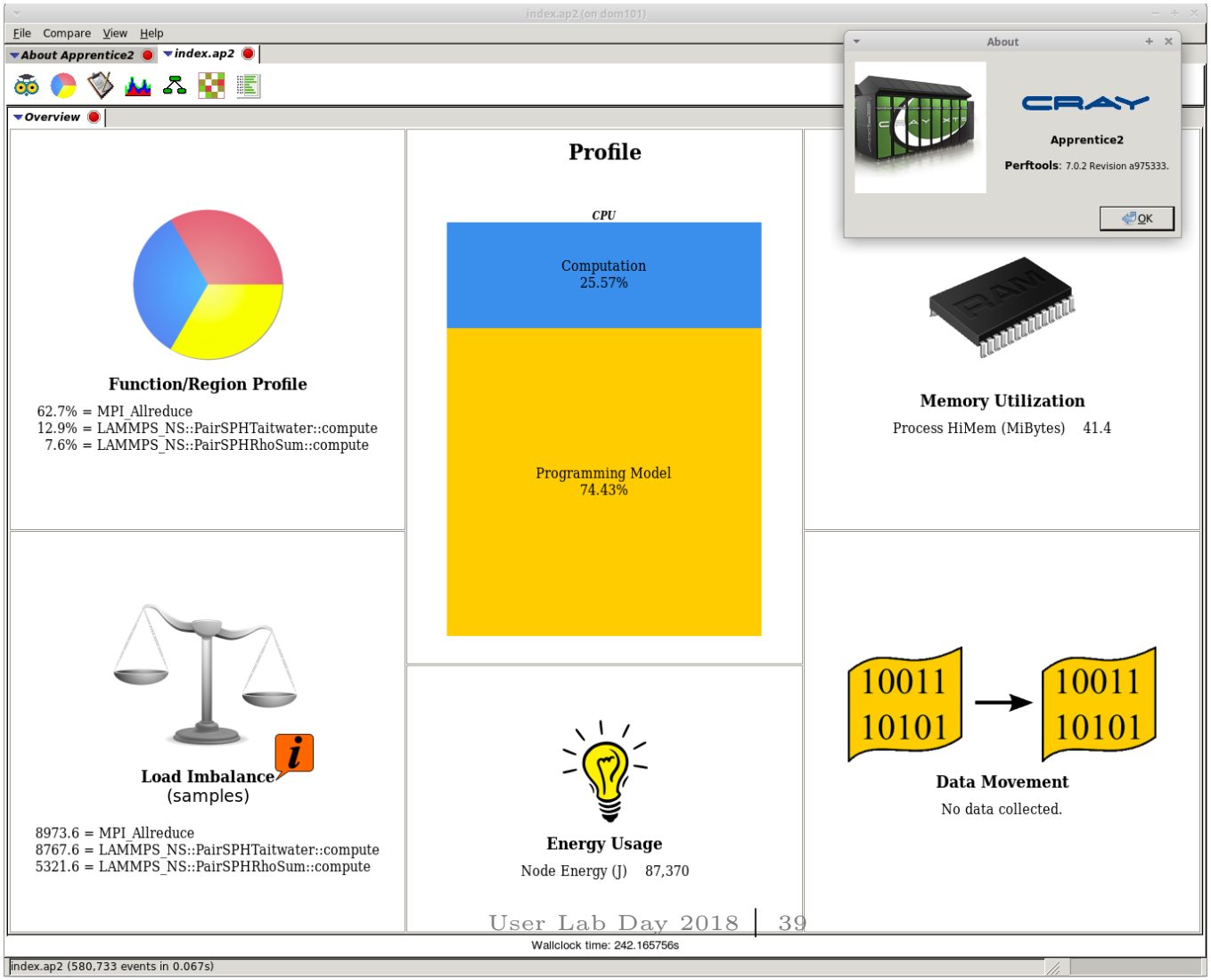
- The table below summarizes the major differences between pat_run and pat_build:

man pat_run	man pat_build; man perftools[-lite]
- supports only dynamically-linked programs	- supports both statically-linked and dynamically-linked programs
- use the appropriate compiler option to instrument functions in user-owned source files	- functions defined in user-owned files can be individually selected for tracing
- all functions in a trace group are traced (selecting an individual one selects all)	- functions belonging to a trace group can be individually selected for tracing
- functions in user-owned source files are only traced in lite modes if instrumented using compiler options	- some lite modes trace functions in user-owned source files
- does not support 64-bit MPI	- supports 64-bit MPI

Cray perftools: Apprentice2

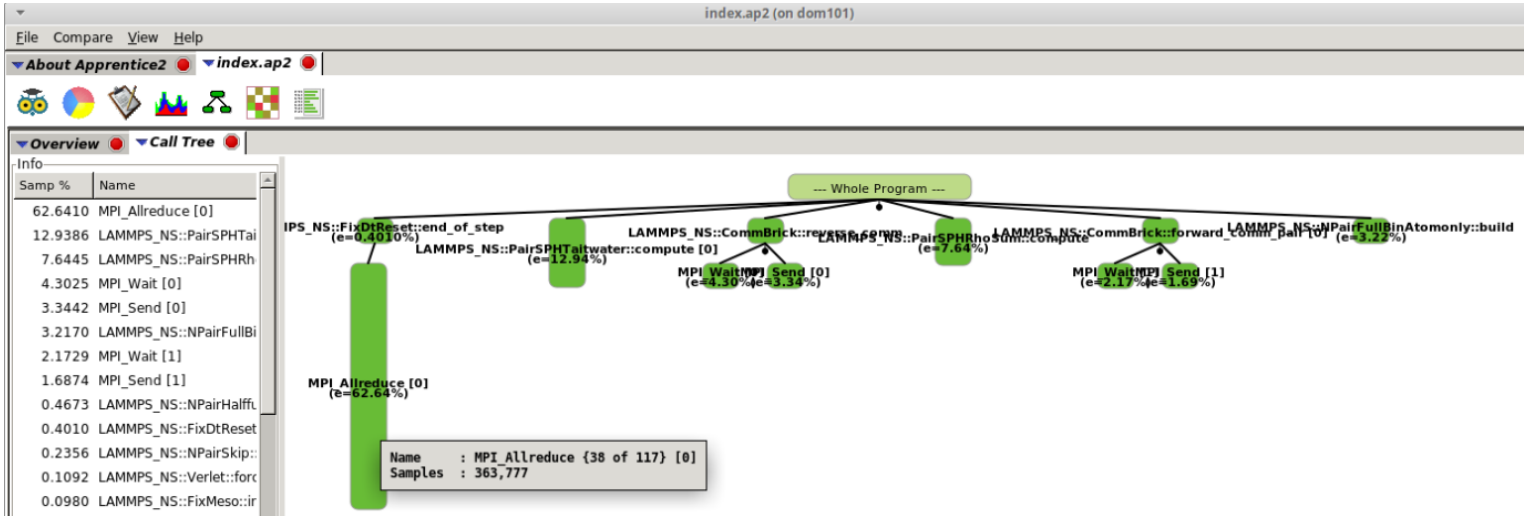
- Visualize the performance report with:

```
# app2 ./lmp_mpi+29655-1s/index.ap2
```

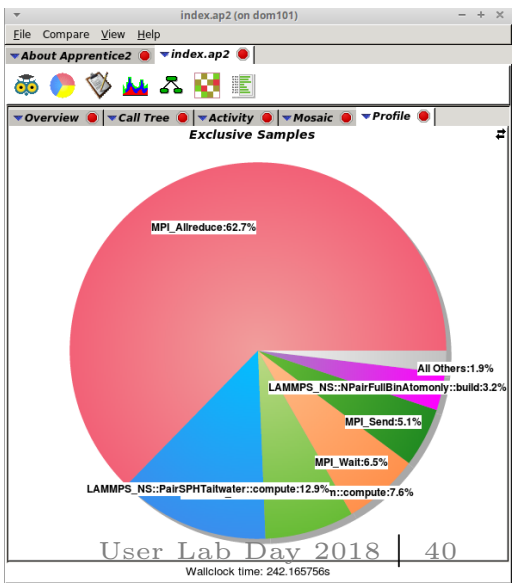


Cray perftools: Apprentice2

- Call tree:

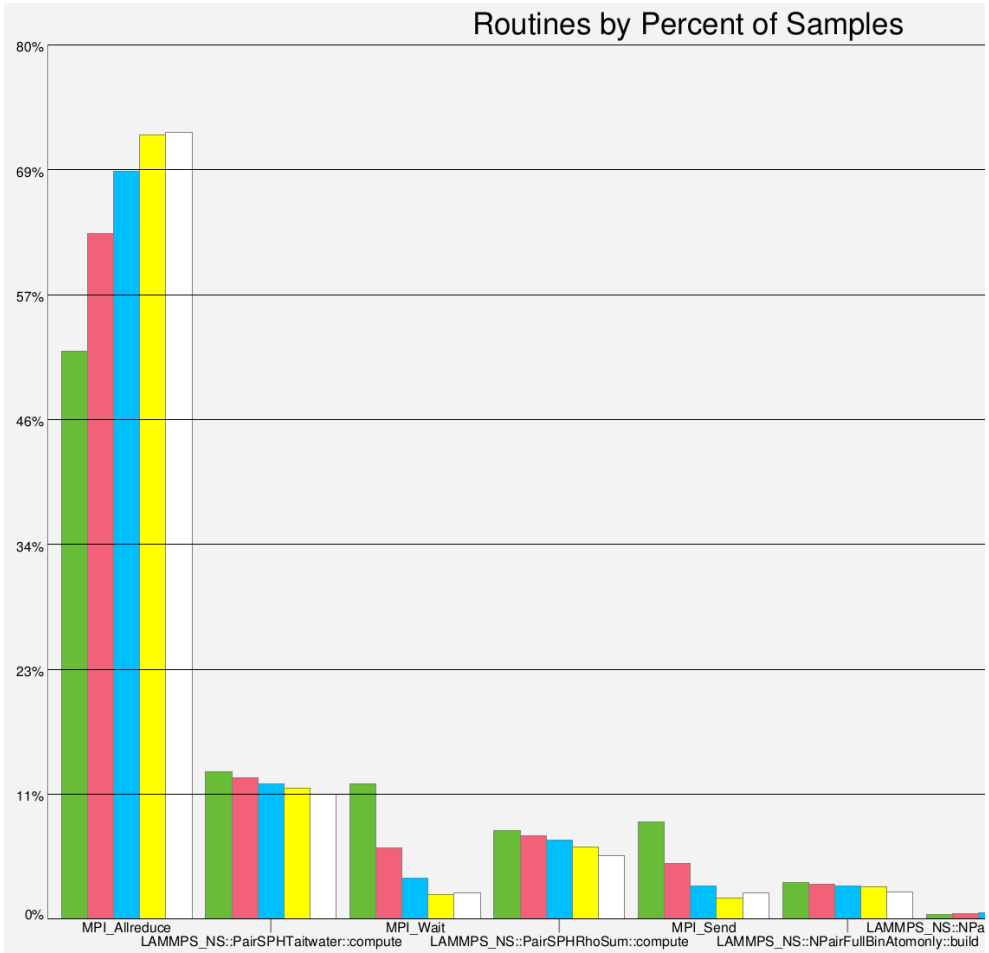


- Timing breakdown:



Cray perftools: pat_view

- Create a scaling graph (from 12 to 192 mpi tasks, 10^5 particles) with:
  # pat_view --bottom --relative *s/index.ap2

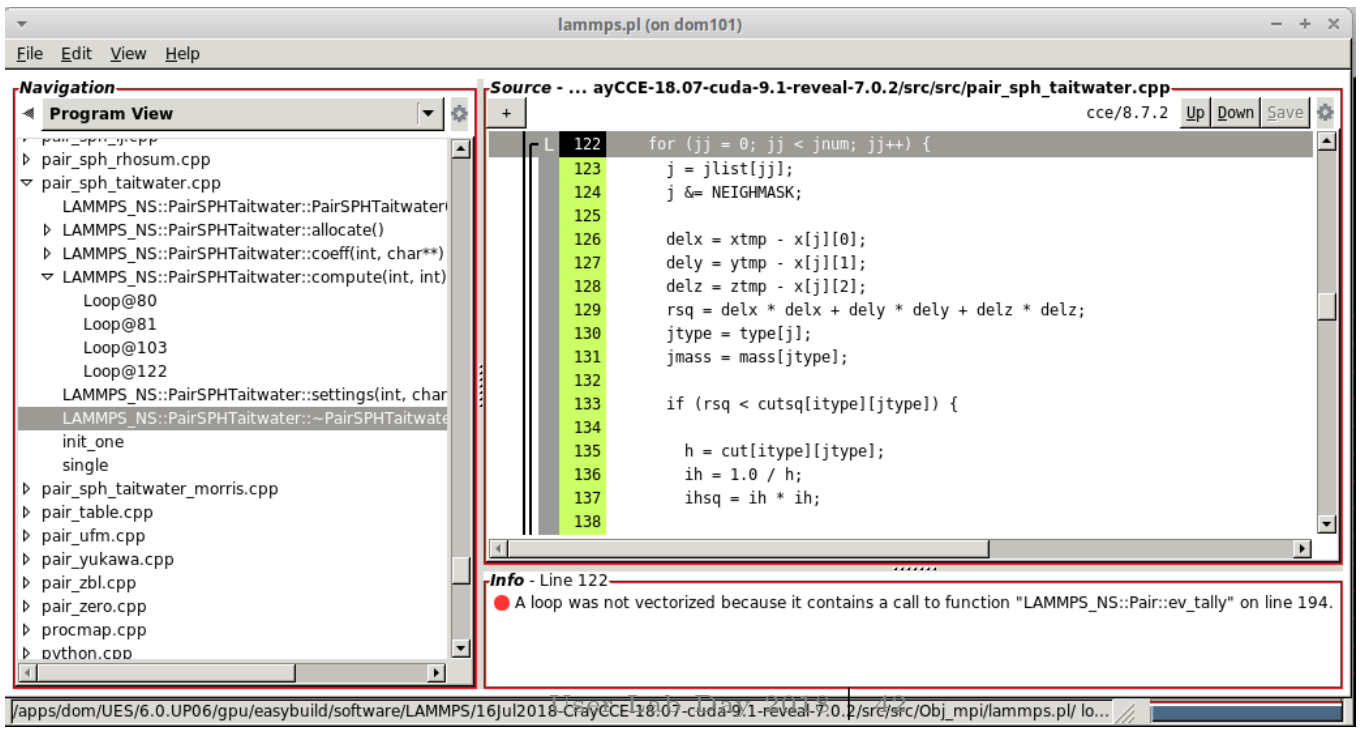



```
LAMMPS_12_100000wp_strong+patrun/PrgEnv-gnu/lmp_mpi+30259-0s/index.ap2: loaded. [PEs=12 Threads=1]
LAMMPS_24_100000wp_strong+patrun/PrgEnv-gnu/lmp_mpi+29655-1s/index.ap2: loaded. [PEs=24 Threads=1]
LAMMPS_48_100000wp_strong+patrun/PrgEnv-gnu/lmp_mpi+908-3s/index.ap2:   loaded. [PEs=48 Threads=1]
LAMMPS_96_100000wp_strong+patrun/PrgEnv-gnu/lmp_mpi+2792-7s/index.ap2:  loaded. [PEs=96 Threads=1]
LAMMPS_192_100000wp_strong+patrun/PrgEnv-gnu/lmp_mpi+551-0s/index.ap2:  loaded. [PEs=192 Threads=1]
```

Cray perftools: reveal

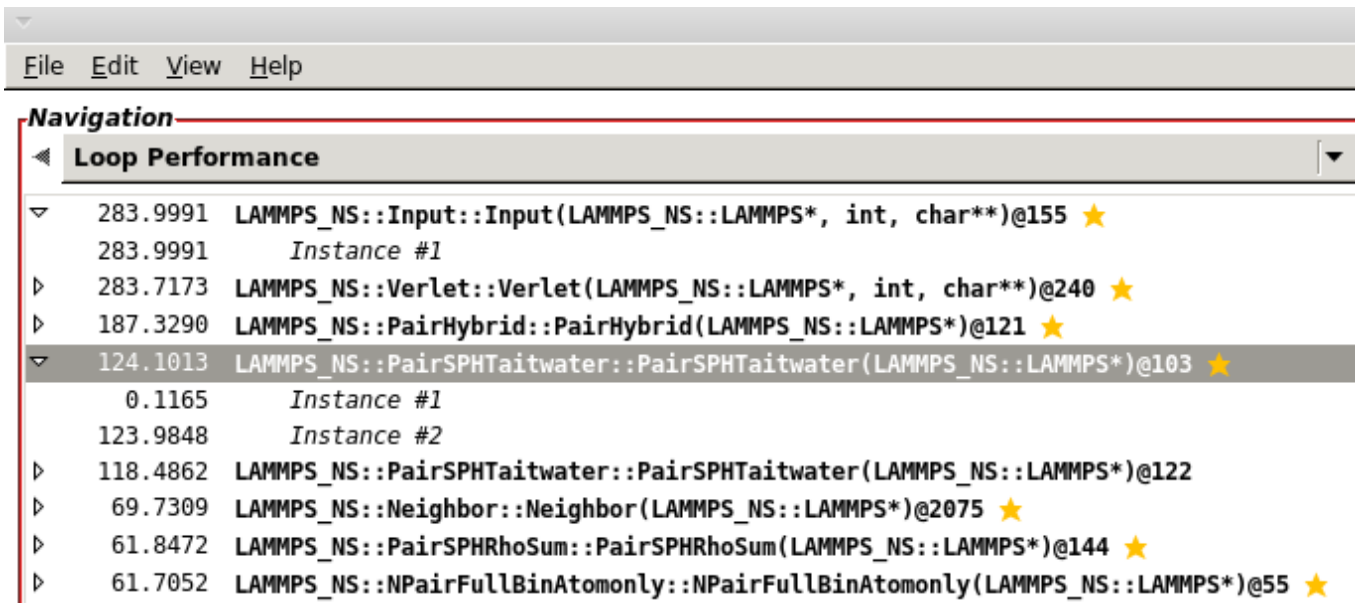
- Reveal (**man reveal**) helps to understand which high-level serial loops could benefit from improved parallelism by combining:
  a) whole-program analysis data for codes compiled with **CCE**,
  b) and runtime performance statistics from **cray-perftools**.

- Generate the program_library (a) and get compiler feedback with:
  * easybuild: LAMMPS-16Jul2018-CrayCCE-18.07-cuda-9.1-reveal-7.0.2.eb
  * module load LAMMPS/16Jul2018-CrayCCE-18.07-cuda-9.1-reveal-7.0.2
  * **reveal** $EBROOTLAMMPS/src/src/Obj_mpi/**lammps.pl**/ &



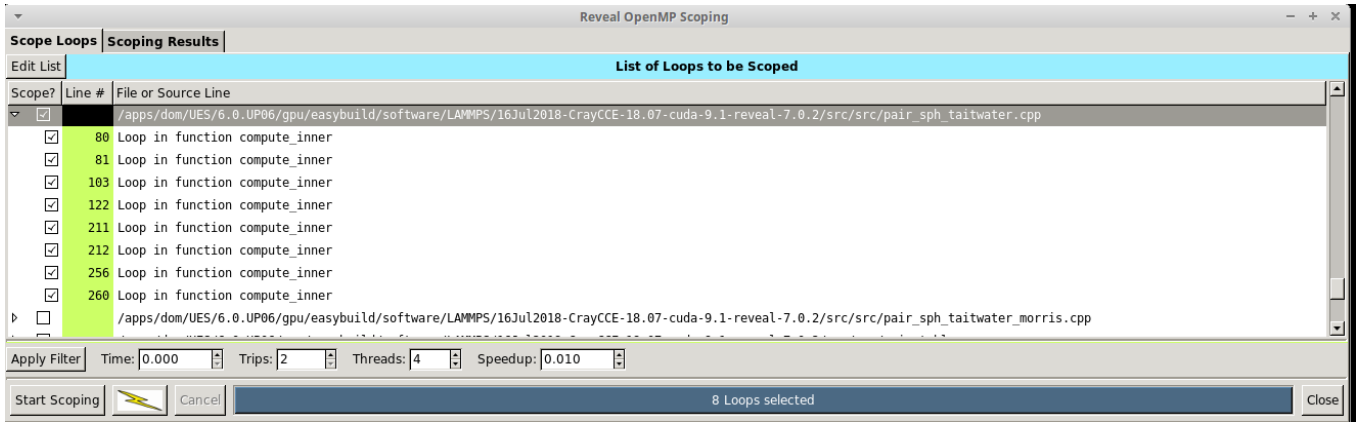
Cray perftools: reveal

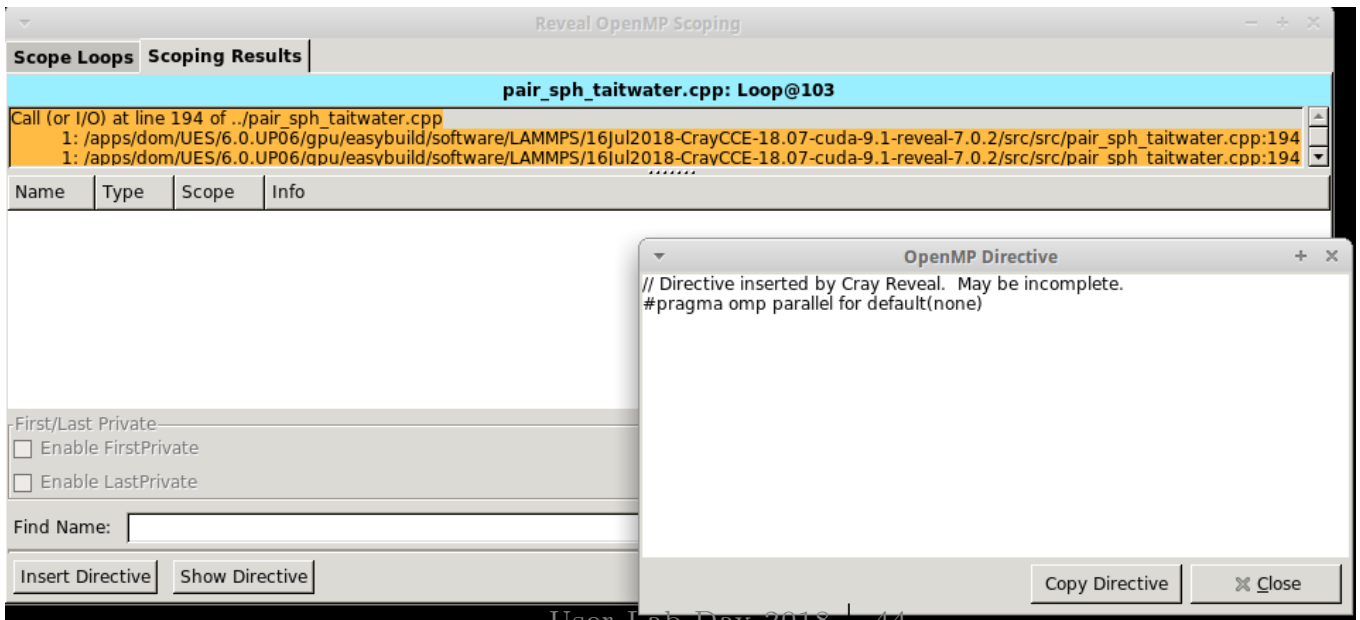
- Generate the performance report (b) with:
  * easybuild: LAMMPS-16Jul2018-CrayCCE-18.07-cuda-9.1-ptl-loops.eb
  * reframe: lammps_sph_strong_scaling_010000wp+ptl-loops.py
  * **reveal** \
    $EBROOTLAMMPS/src/src/Obj_mpi/**lammps.pl**/ \
    LAMMPS_1_010000wp_strong+ptl-loops/PrgEnv-cray/**lmp_mpi+32084-0t**/ &



Cray perftools: reveal

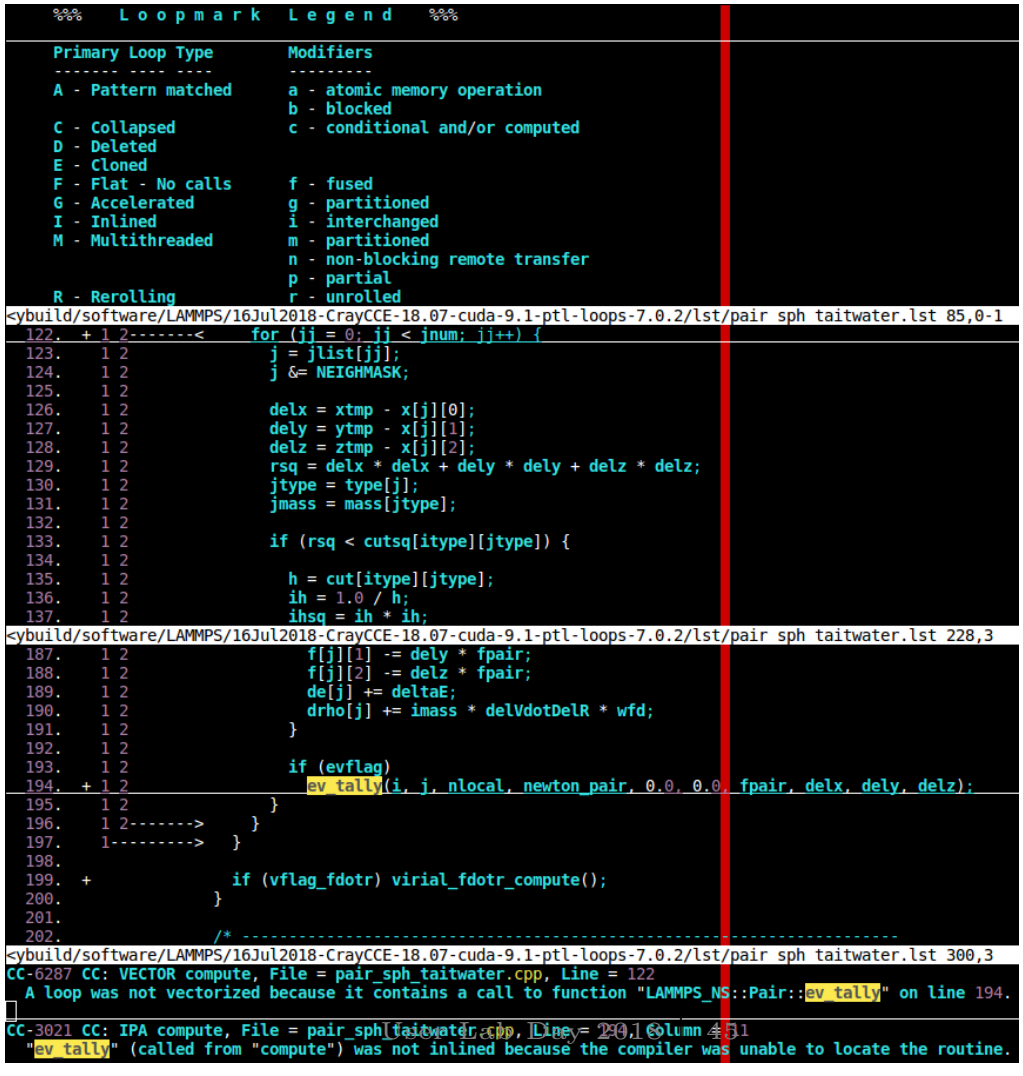
- Use reveal to explore opportunities for code/loops optimization:





Cray perftools: reveal

- The compiler feedback (a) can also be read with:
  - easybuild: LAMMPS-16Jul2018-CrayCCE-18.07-cuda-9.1-ptl-loops.eb
  - module load LAMMPS/16Jul2018-CrayCCE-18.07-cuda-9.1-ptl-loops-7.0.2
  - **vim** $EBROOTLAMMPS/lst/pair_sph_taitwater.lst



Cray perftools: reveal

- The performance report (b) can also be read with:
  * easybuild: LAMMPS-16Jul2018-CrayCCE-18.07-cuda-9.1-ptl-loops.eb
  * reframe: lammps_sph_strong_scaling_010000wp+ptl-loops.py
  * **vim** lmp_mpi+17326-38t/rpt-files/**RUNTIME.rpt**

Table 1: Inclusive and Exclusive Time in Loops (from -hprofile_generate)

Loop Incl Time%	Loop Incl Time	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
100.0%	39.823566	1	67.0	67	67	LAMMPS_NS::Input::file().LOOP.1.li.155
95.4%	38.004556	1	500.0	500	500	LAMMPS_NS::Verlet::run(int).LOOP.1.li.240
62.3%	24.816514	500	1.0	1	1	LAMMPS_NS::Modify::end_of_step().LOOP.1.li.477
23.2%	9.251915	501	2.0	2	2	LAMMPS_NS::PairHybrid::compute(int, int).LOOP.2.li.121
11.3%	4.517321	501	4,262.9	69	16,872	LAMMPS_NS::PairSPHTaitwater::compute(int, int).LOOP.3.li.103
11.3%	4.491951	2,135,723	93.5	1	230	LAMMPS_NS::PairSPHTaitwater::compute(int, int).LOOP.4.li.122
7.5%	2.983443	501	4,012.9	360	16,800	LAMMPS_NS::PairSPHRhoSum::compute(int, int).LOOP.4.li.144
7.4%	2.966345	2,010,473	196.1	4	251	LAMMPS_NS::PairSPHRhoSum::compute(int, int).LOOP.5.li.153
7.0%	2.803558	501	2.0	2	2	LAMMPS_NS::CommBrick::reverse_comm().LOOP.1.li.539
4.4%	1.738144	501	2.0	2	2	LAMMPS_NS::CommBrick::forward_comm_pair(LAMMPS_NS::Pair*).LOOP.1.li.892
3.6%	1.437810	101	3.0	3	3	LAMMPS_NS::Neighbor::build(int).LOOP.5.li.2075
3.0%	1.201679	101	4,262.4	69	16,872	LAMMPS_NS::NPairFullBinAtomonly::build(LAMMPS_NS::NeighList*).LOOP.1.li.55
3.0%	1.195668	430,500	25.0	25	25	LAMMPS_NS::NPairFullBinAtomonly::build(LAMMPS_NS::NeighList*).LOOP.3.li.69
2.8%	1.096637	10,183,526	15.6	1	30	LAMMPS_NS::NPairFullBinAtomonly::build(LAMMPS_NS::NeighList*).LOOP.4.li.70
2.4%	0.969656	1	4.0	4	4	LAMMPS_NS::Modify::setup(int).LOOP.2.li.298

Cray perftools: performance counters

- man intro_craypat
- Collect hardware performance counter events with **$PAT_RT_PERFCTR**
- Get the list of predefined performance groups of counters with:
  * daint-gpu (E5-2690 v3 Haswell cpu): **pat_help** counters intel_fam6mod63 groups
  * daint-mc (E5-2695 v4 Broadwell cpu): **pat_help** counters intel_fam6mod79 groups

```
 0: D1 with instruction counts
 1: Summary -- cache and TLB metrics
 2: D1, D2, L3 Metrics
 6: Micro-op queue stalls
 7: Back end stalls
 8: Instructions and branches
 9: Instruction cache
10: Cache Hierarchy
11: Floating point operations (unsupported)
12: AVX floating point operations (unsupported)
13: SSE and AVX floating point operations SP (unsupported)
14: SSE and AVX floating point operations DP (unsupported)
19: Prefetchs
23: Cache metrics (same as 1)
```

```
br_mispredict_latency: Branch mispredict latency
data_sharing_latency: read and write data sharing latencies
default: group 1
default_samp: default plus PM_ENERGY (for apa)
dram_l3_latency: local and remote dram and l3 access latencies
dtlb_l3_misses: Load misses in DTLB or L3 plus ICP
dtlb_miss_latency: dtlb miss latencies, first- and second-level
efficiency: CPI and slots retired
fe_latency: itlb, icache, and lcp latencies
hbm: load hits in LFB and resource stalls
hbm_loads: retired loads and resource stalls
hbm_stores: retired stores and resource stalls
load_block_latency: load block latencies
machine_clears: Count of machine clears
microcode_assist_details: Causes of assists from microcode sequencer
microcode_assists: Assists from microcode sequencer
```

- Performance counters are based on PAPI

  PAPI (Performance Application Programming Interface) provides a consistent interface for use of the performance counter hardware found in modern CPUs/GPUs. It enables to see the relation between software performance and processor events.

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

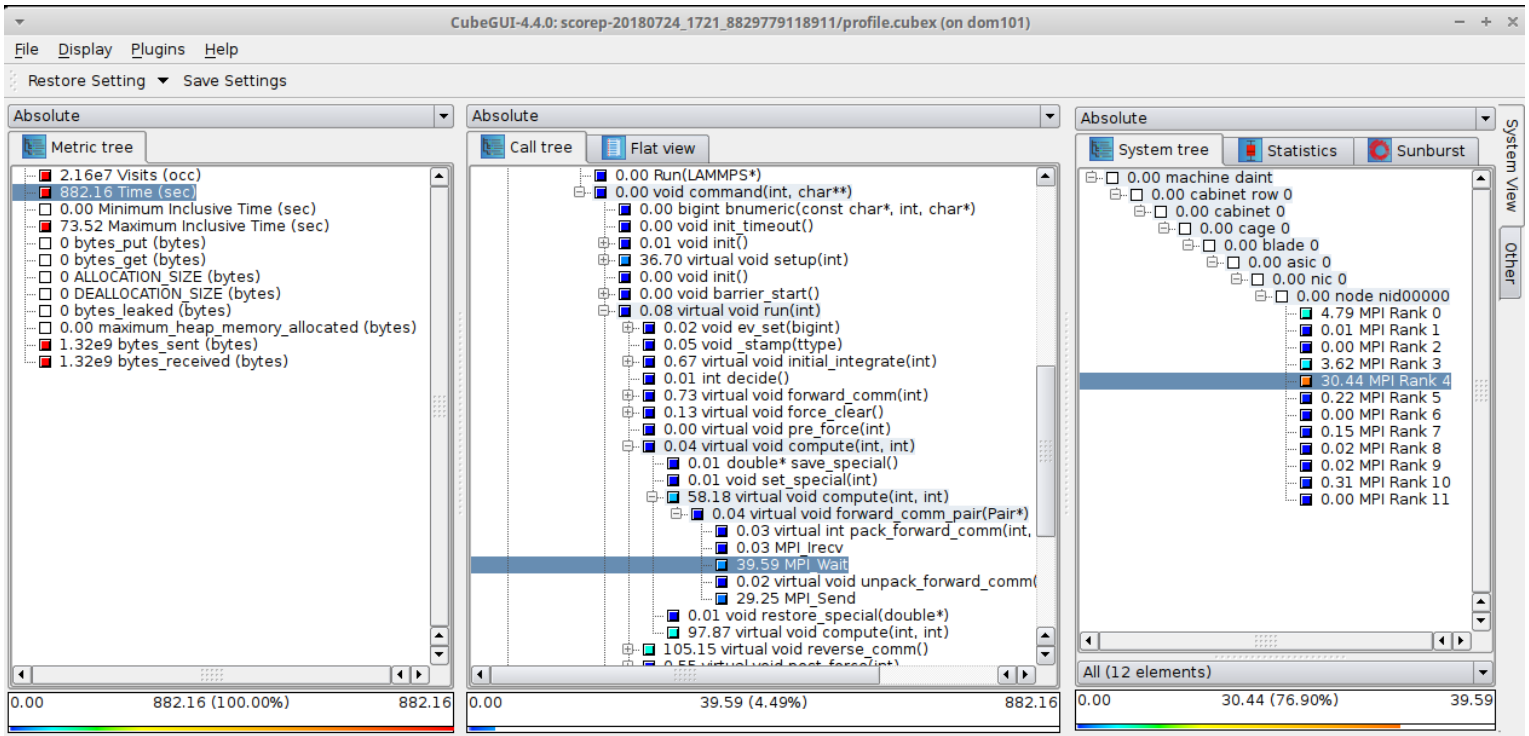
Performance analysis with Score-P tools

General approach to performance analysis

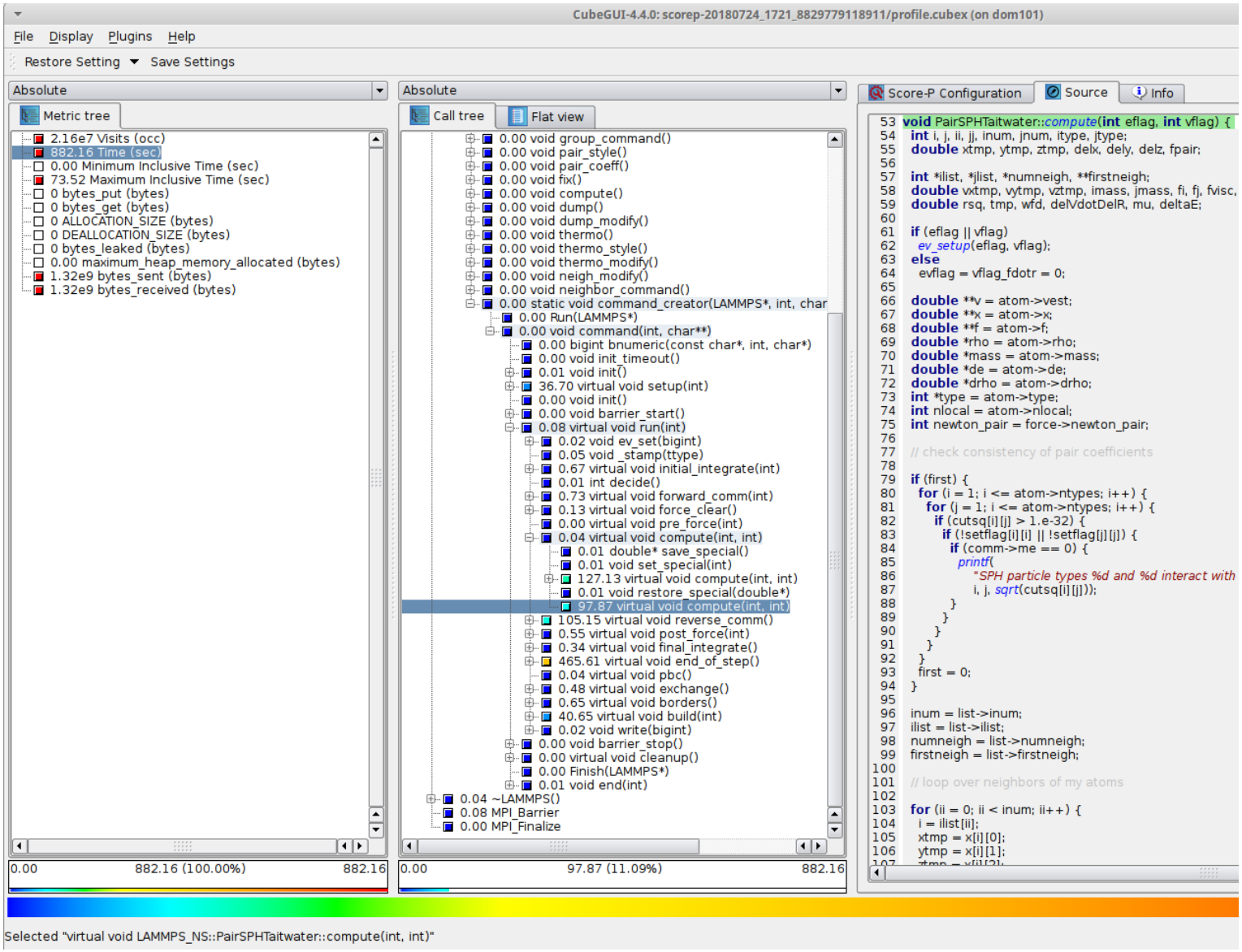
- **C** - compile your application with the tool
  read the easybuild scripts or check the tools' guide
- **P** - profile your application
  get an understanding of your code's performance
- **F** - filter before tracing your application
  reduce the amount of data recorded by the tool
- **T** - trace your application
  extract detailed information from the filtered part

Score-P: Profiling

- `https://user.cscs.ch/computing/analysis/vihps/`
  * easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-scorep-4.0.eb
    i.e **scorep --mpp=mpi** CC ...
  * reframe: lammps_sph_strong_scaling_100000wp+scorep_profiling.py
    i.e **SCOREP_ENABLE_PROFILING**=true; srun lmp_mpi ...
  * **cube** ./scorep-20180724_1721_8829779118911/profile.cubex &



Score-P: Profiling



Score-P: Filtering

- Without filtering: scorep-score -r ./scorep-*/profile.cubex

```
Estimated aggregate size of event trace:                   863MB <--------
Estimated requirements for largest trace buffer (max_buf): 59MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)
.
flt     type max_buf[B]     visits time[s] time[%] time/visit[us]  region
         ALL 61,645,990 34,060,317 1032.22   100.0          30.31  ALL
         USR 60,857,763 33,337,772  176.85    17.1           5.30  USR
         MPI    685,218    532,369  792.40    76.8        1488.44  MPI
         COM    103,009    190,176   62.97     6.1         331.14  COM
```

- With filtering: scorep-score **-f ./myfilter.txt** ./scorep-*/profile.cubex

```
Estimated aggregate size of event trace:                   69MB <-------
Estimated requirements for largest trace buffer (max_buf): 1478kB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)
.
flt     type max_buf[B]     visits time[s] time[%] time/visit[us]  region
 -       ALL 61,645,990 34,060,317 1032.22   100.0          30.31  ALL
 -       USR 60,857,763 33,337,772  176.85    17.1           5.30  USR
 -       MPI    685,218    532,369  792.40    76.8        1488.44  MPI
 -       COM    103,009    190,176   62.97     6.1         331.14  COM
```

Score-P: Filtering

- **scorep-score** will help to create a filter file, but because LAMMPS is a C++ code, scorep-score will list function names as **mangled** symbols:

```
flt     type  max_buf[B]     visits time[s] time[%] time/visit[us]  region
         MPI          26         12    0.78     0.1       65182.74  MPI_Init
         ...
-------> USR 102,546,860 12,369,160    6.93     0.8           0.56  _ZN9LAMMPS_NS4NBin9coord2binEPd <-----
         MPI     195,688     35,501   62.18     7.0        1751.60  MPI_Send
         USR      13,026      6,012  102.08    11.6       16979.01  _ZN9LAMMPS_NS16PairSPHTaitwater7computeEii
         MPI      83,408     35,501  119.12    13.5        3355.32  MPI_Wait
         MPI      39,848      7,032  484.57    54.9       68909.91  MPI_Allreduce
         ALL 171,575,290 21,557,652  882.16   100.0          40.92  ALL
```

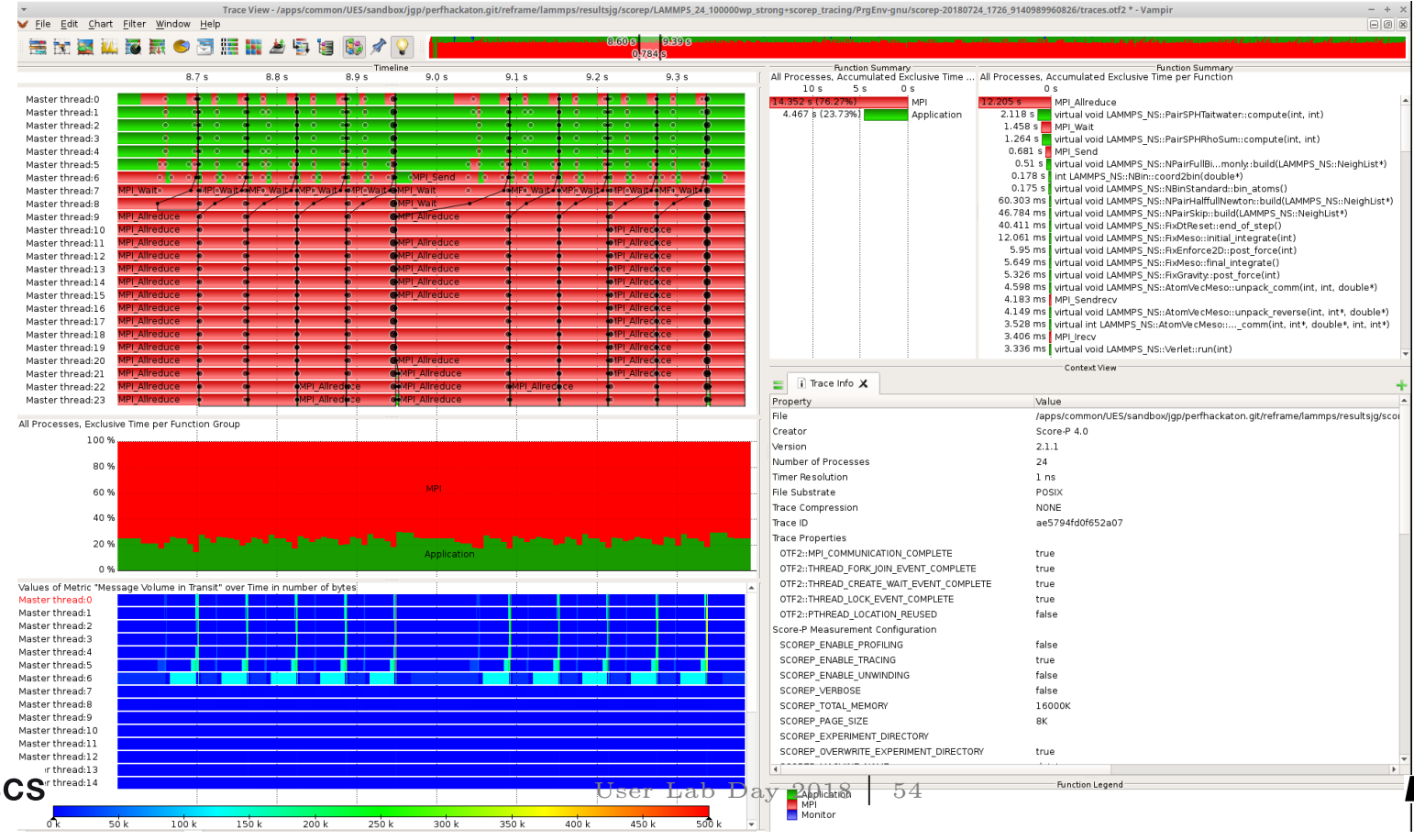
- To demangle function names: **c++filt** _ZN9LAMMPS_NS4NBin9coord2binEPd
  # LAMMPS_NS::NBin::coord2bin(double*)
- Example filtering file: cat ./myfilter.txt

```
SCOREP_REGION_NAMES_BEGIN
 EXCLUDE
 MANGLED
_ZN9LAMMPS_NS11AtomVecMeso13pack_exchangeEiPd
_ZN9LAMMPS_NS11AtomVecMeso15unpack_exchangeEPd
_ZN9LAMMPS_NS11AtomVecMeso4copyEiii
_ZN9LAMMPS_NS11AtomVecMeso9data_atomEPdiPPc
_ZN9LAMMPS_NS4NBin9coord2binEPd     <----------
SCOREP_REGION_NAMES_END
```

Score-P: Tracing

- Rerun with the same executable used for profiling:
  * easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-scorep-4.0.eb
  * reframe: lammps_sph_strong_scaling_100000wp+scorep_tracing.py

    SCOREP_ENABLE_TRACING=true
    SCOREP_FILTERING_FILE=myfilter.txt
    srun lmp_mpi ...
  * **vampir** scorep-*/traces.otf2 &



Score-P: Memory tracing

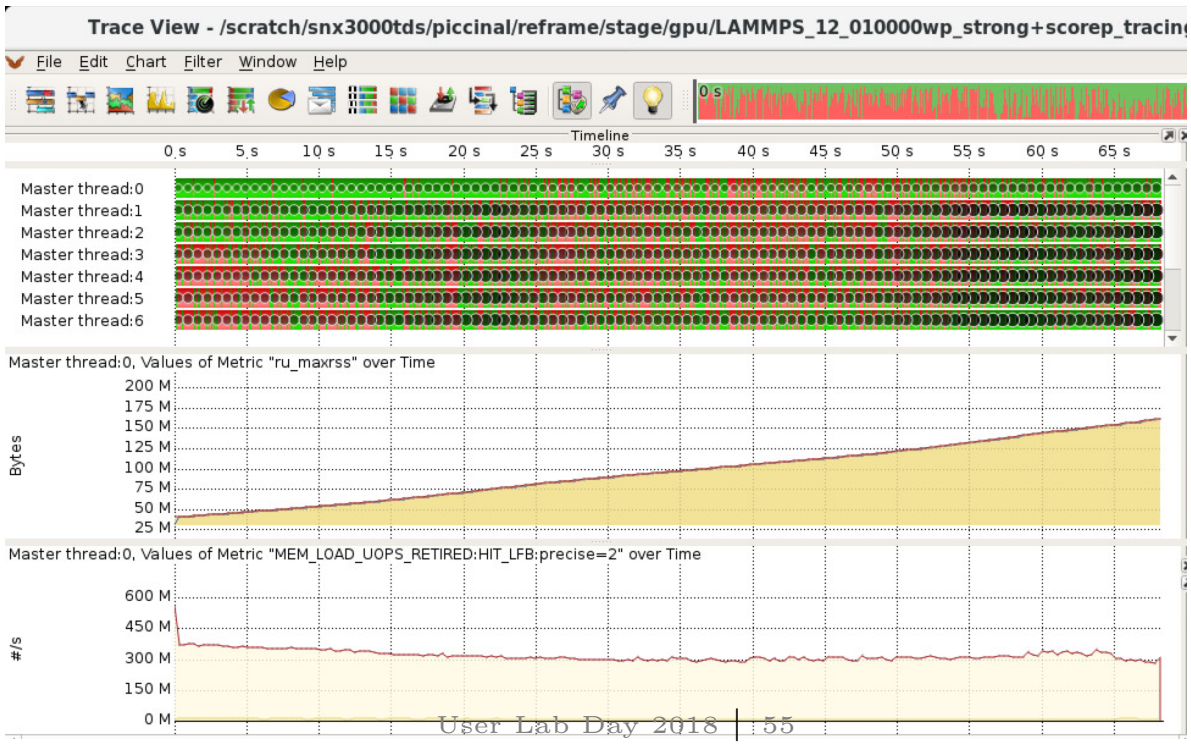
- Rerun the same executable with:

The Unix system call `getrusage` provides information about consumed resources

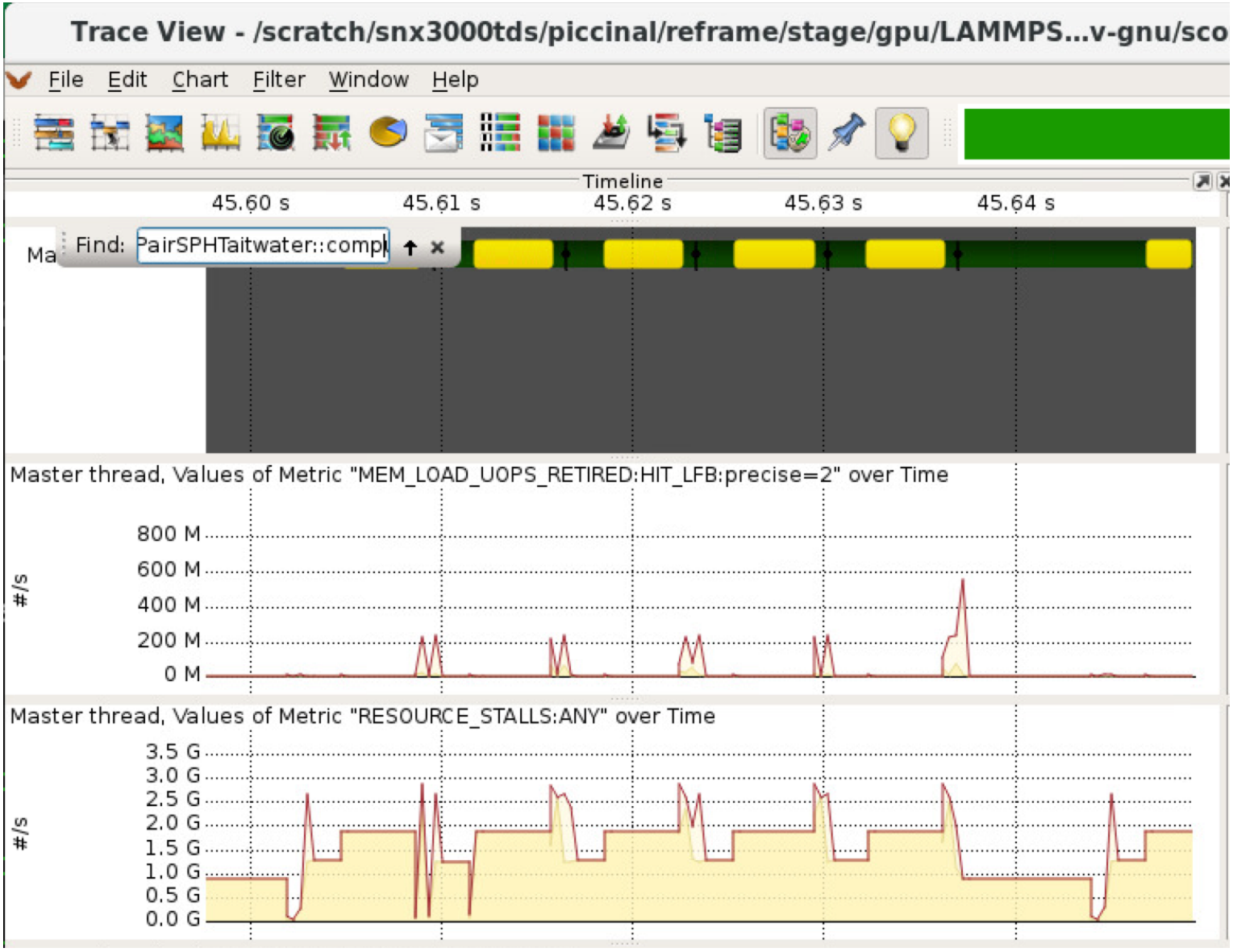
```
% export SCOREP_METRIC_RUSAGE=ru_stime:ru_majflt
```

Name	Unit	Linux	Description
ru_utime	ms	x	Total amount of user time used.
ru_stime	ms	x	Total amount of system time used.
ru_maxrss	kB		Maximum resident set size.
ru_ixrss	kB/s		Integral shared memory size (text segment).
ru_idrss	kB/s		Integral data segment memory used over runtime.
ru_isrss	kB/s		Integral stack memory used over the runtime.
ru_minflt	#	x	Number of soft page faults.
ru_majflt	#	x	Number of hard page faults.

* ru_ixrss,ru_idrss abd ru_isrss are unsupported on Linux (man getrusage)



Score-P: Memory tracing



Score-P: performance counters

- Rerun the same executable with:
  - **SCOREP_METRIC_PAPI**=PAPI_TOT_INS,PAPI_TOT_CYC
  - **vampir** scorep-*/traces.otf2 &

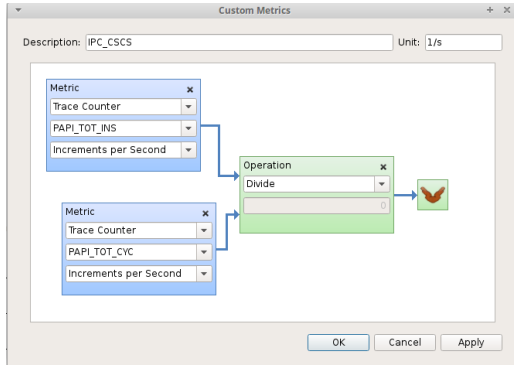
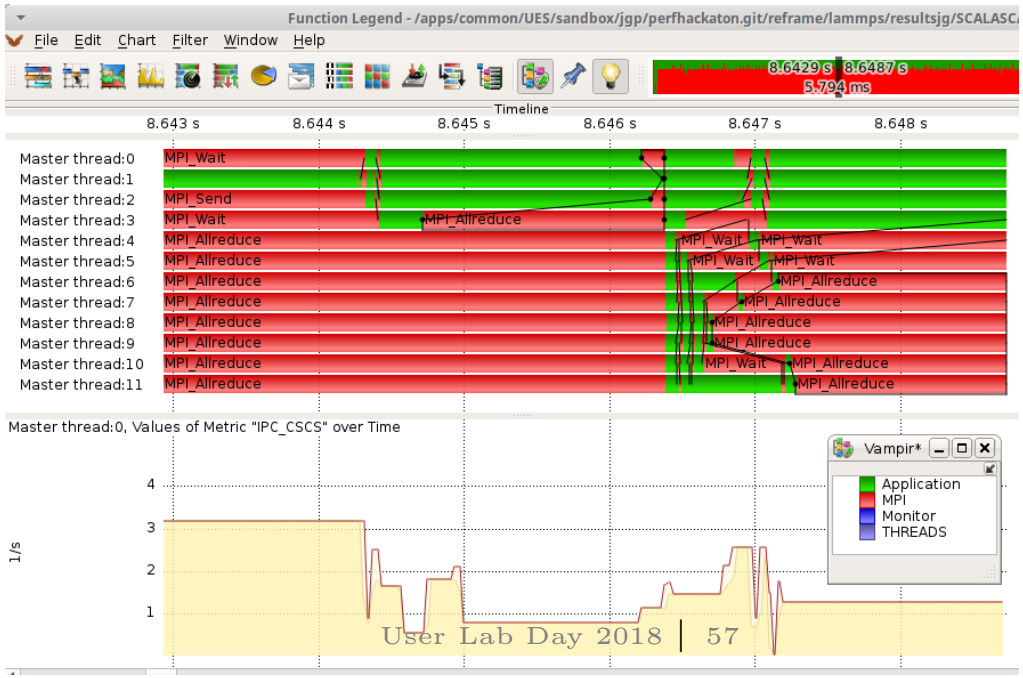


Chart menu/counter data timeline/right click/custom metric



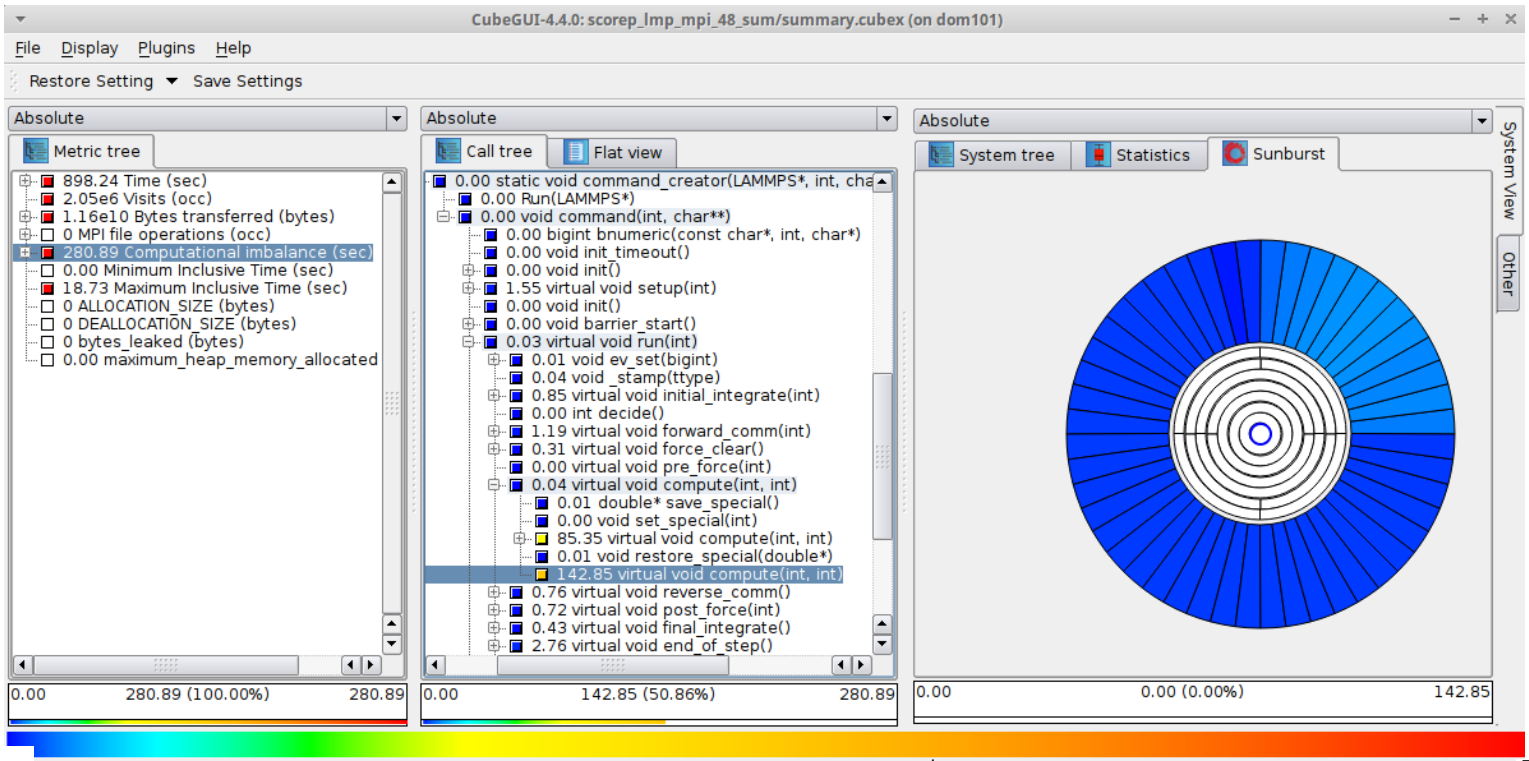
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Performance analysis with Scalasca tools

Scalasca: Profiling

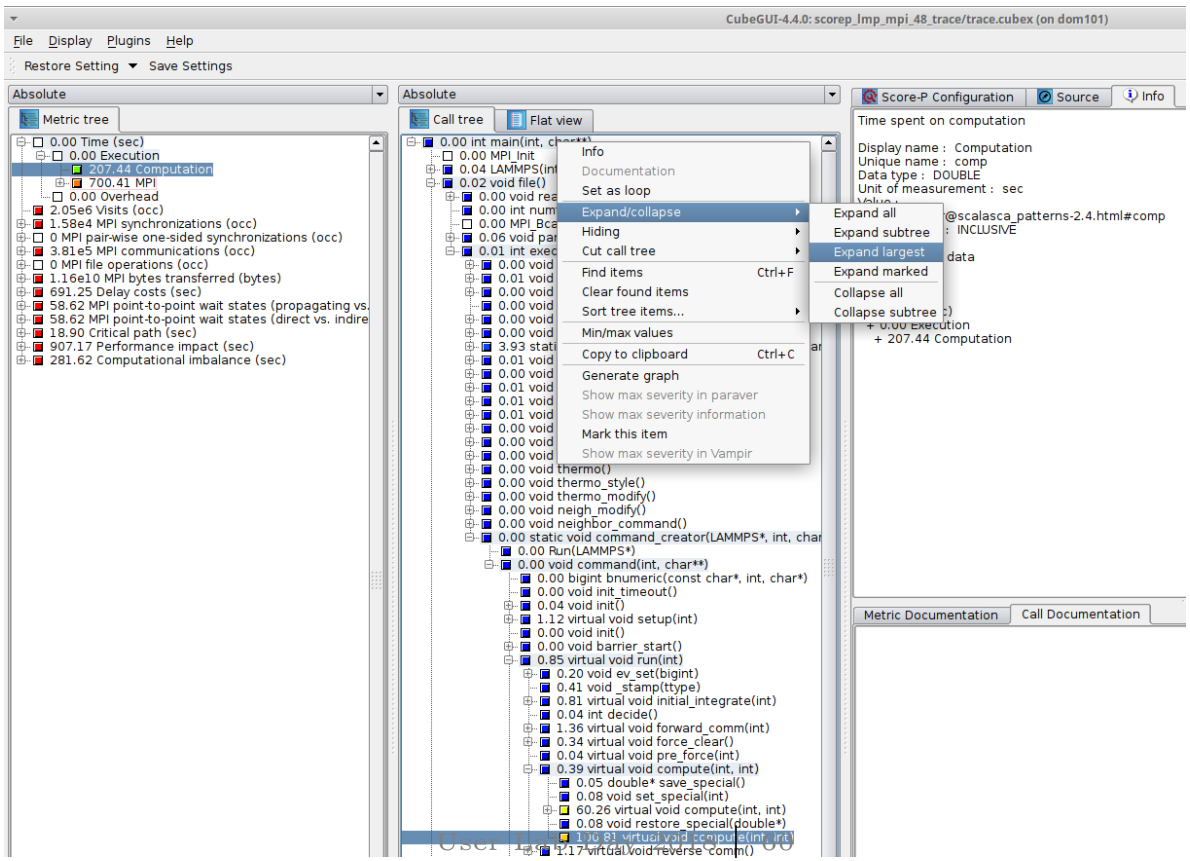
- `https://user.cscs.ch/computing/analysis/vihps/`
  * easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-scorep-4.0.eb
    i.e **scorep --mpp=mpi CC ...**
  * reframe: lammps_sph_strong_scaling_100000wp+scalasca_profiling.py
    i.e **scalasca -analyze** srun lmp_mpi ...
  * **cube** ./scorep_lmp_mpi_12_sum/summary.cubex &



Scalasca: Filtering and Tracing

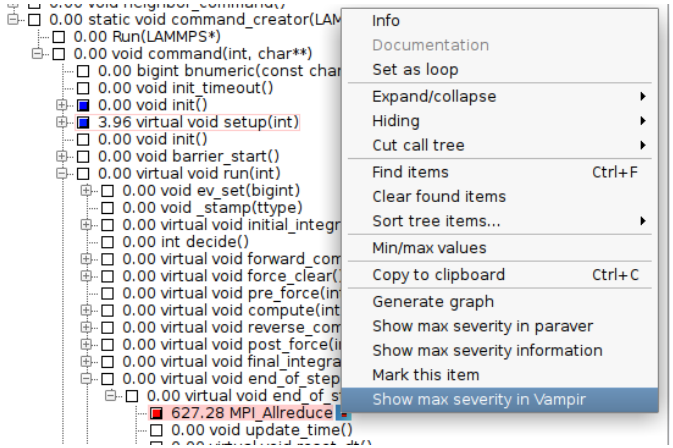
- Rerun with the same executable used for profiling:
  * easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-scorep-4.0.eb
  * reframe: lammps_sph_strong_scaling_100000wp+scalasca_tracing.py

    SCOREP_FILTERING_FILE=myfilter.txt
    scalasca -analyze -t srun lmp_mpi ...
  * **square** ./scorep_lmp_mpi_12_trace/**trace.cubex** &

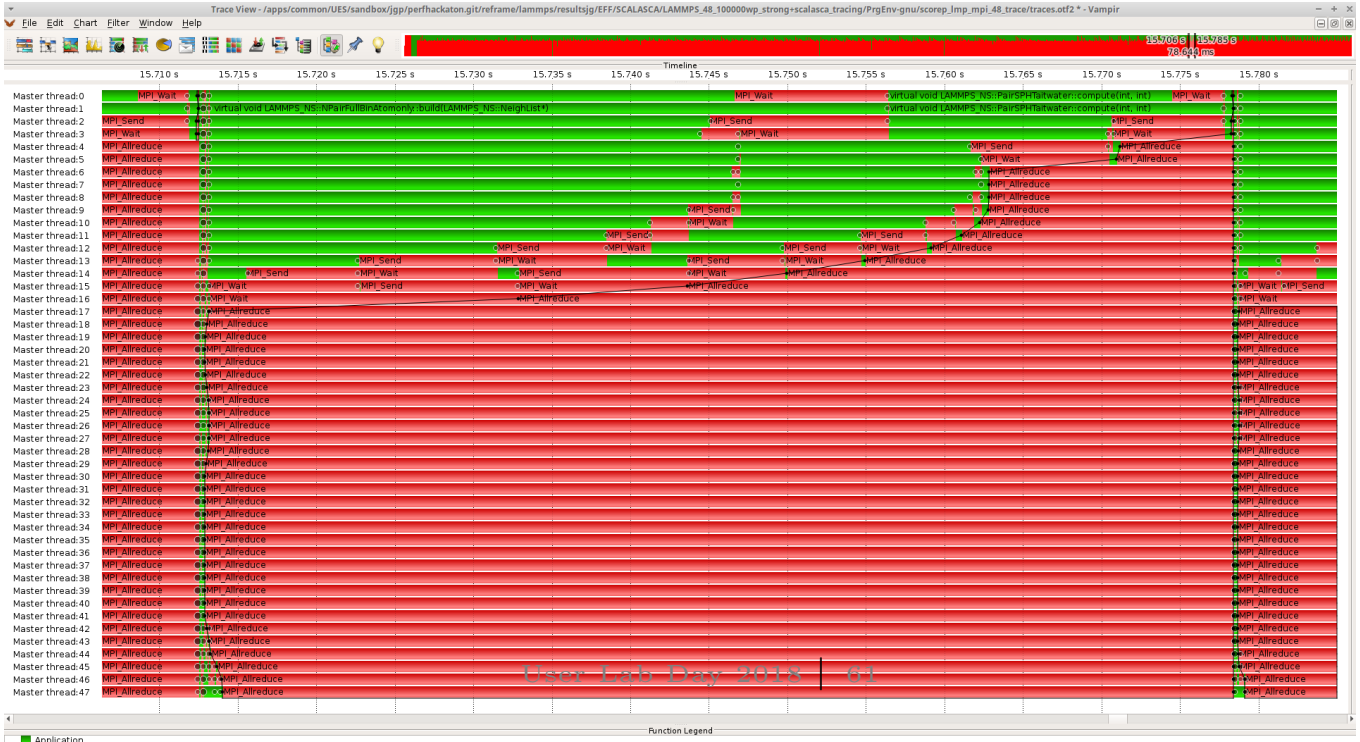


Scalasca: connect to Vampir

- **Find the largest compute time...**

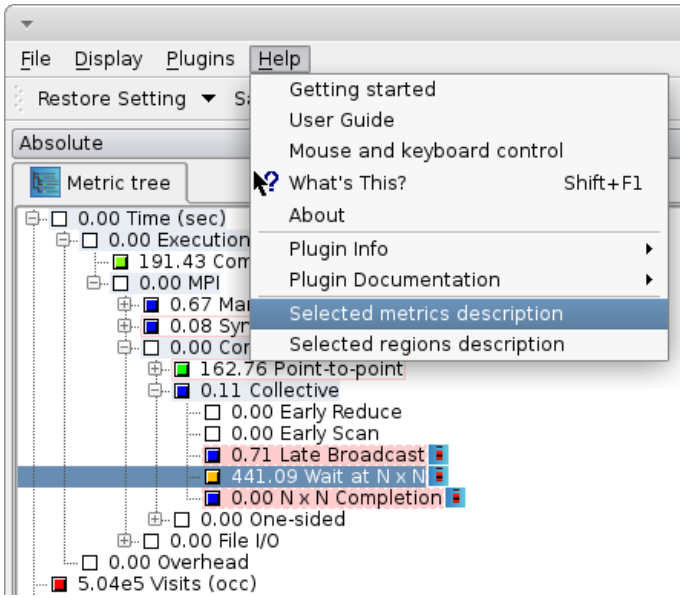


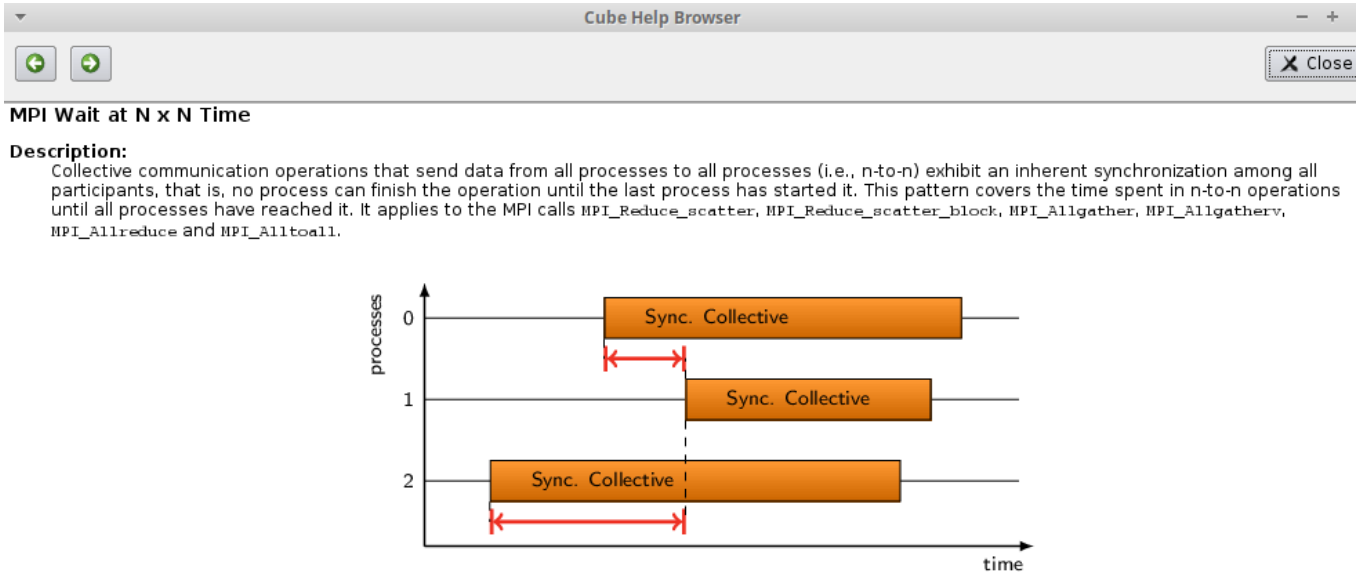
- **... and connect it to Vampir**



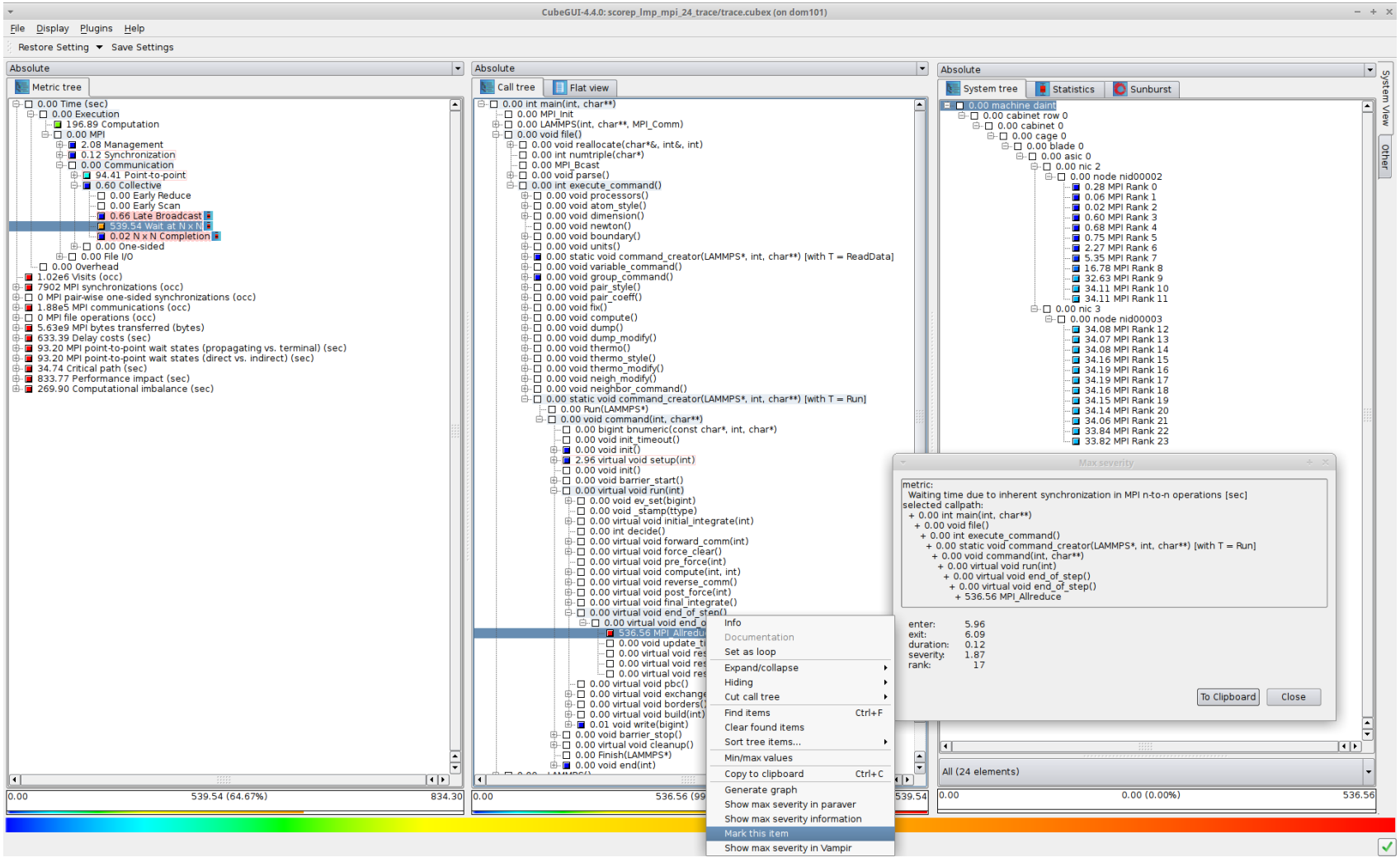
Scalasca: Load imbalance detection

- **Tracing with Scalasca detects communication inefficiencies**



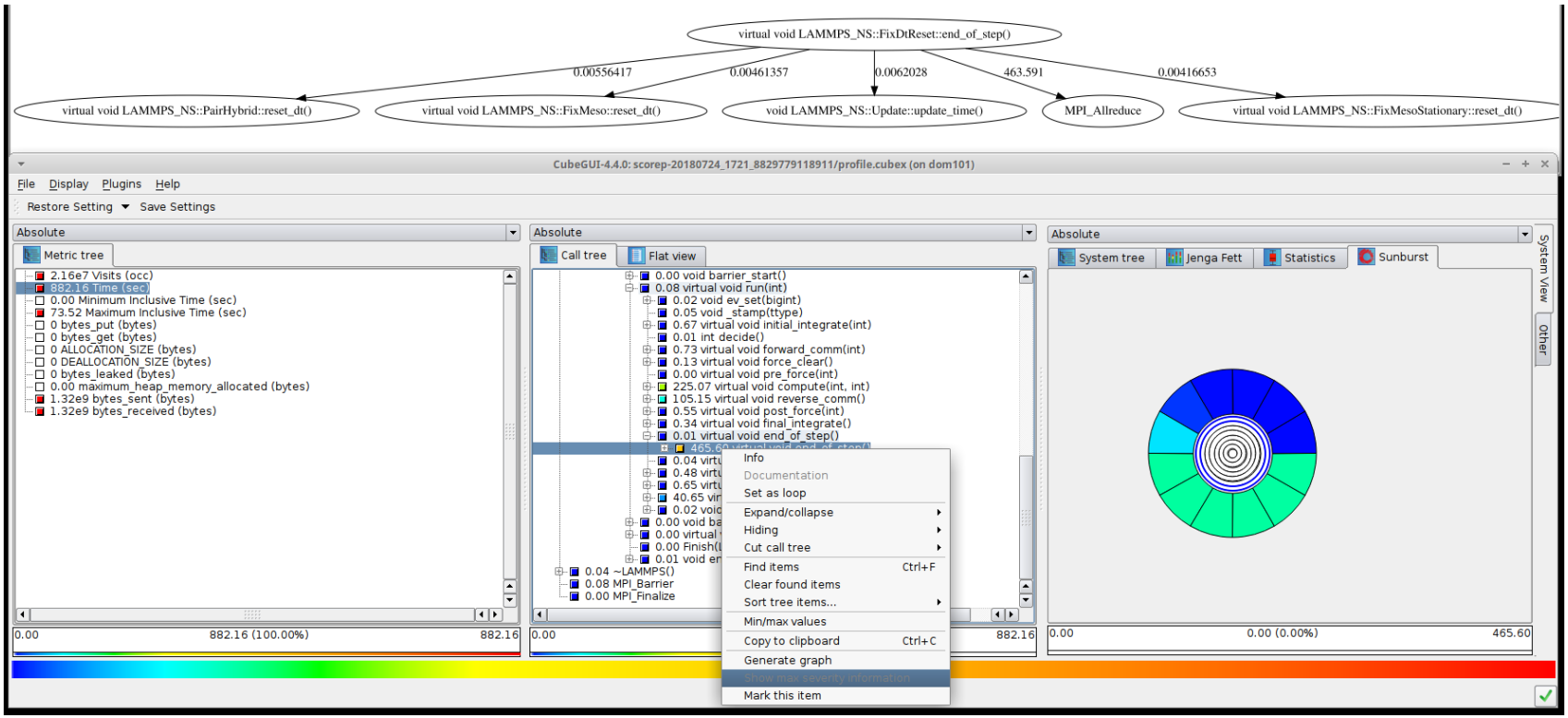


Scalasca: Load imbalance detection



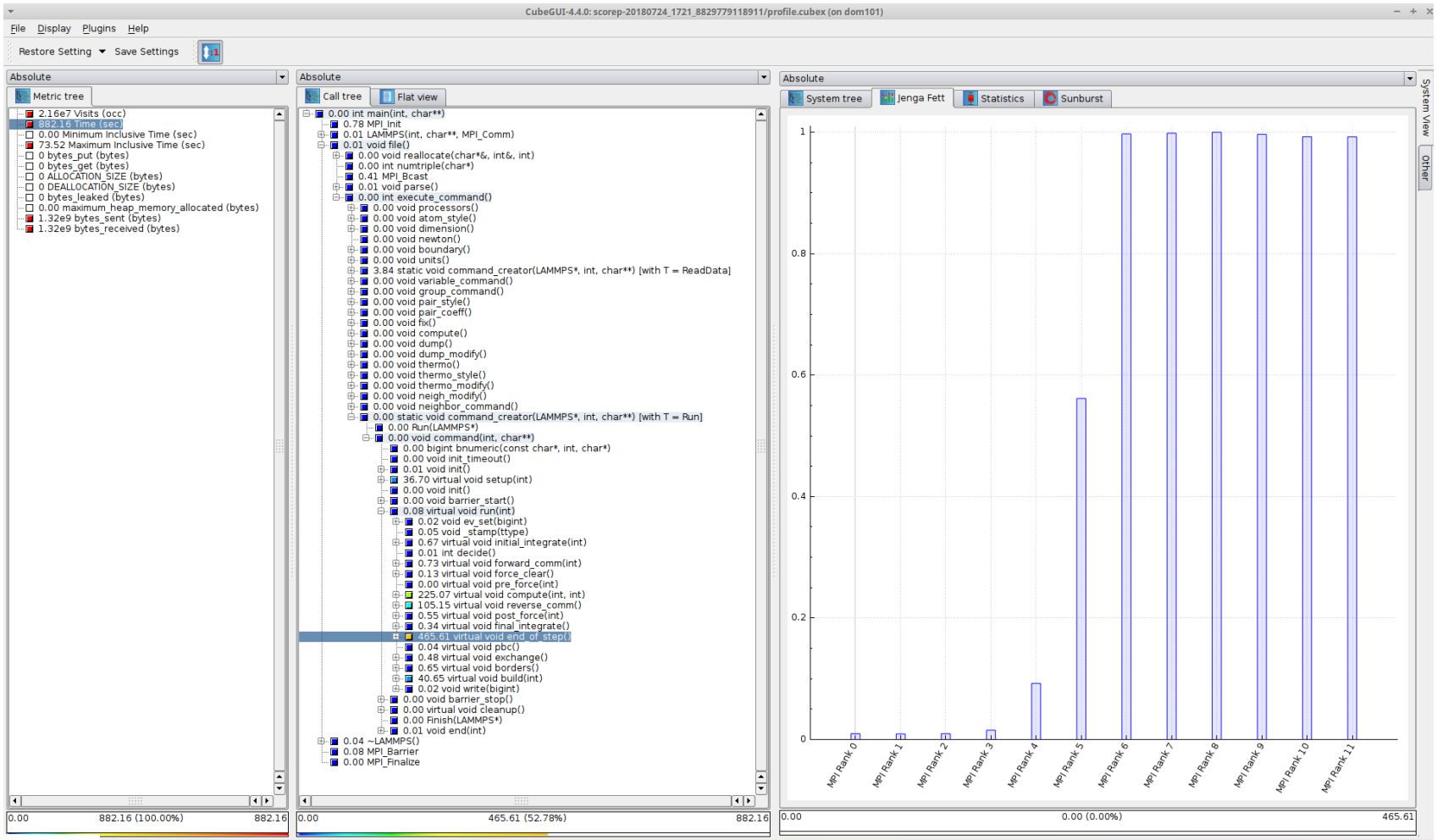
Scalasca: callgraph plugin

- Load the plugin with: **module load callgraphplugin/0.1**
- This plugin builds a call graph based on call tree and assigns the metric values to the edges. It allows to detect critical calls in the program execution.



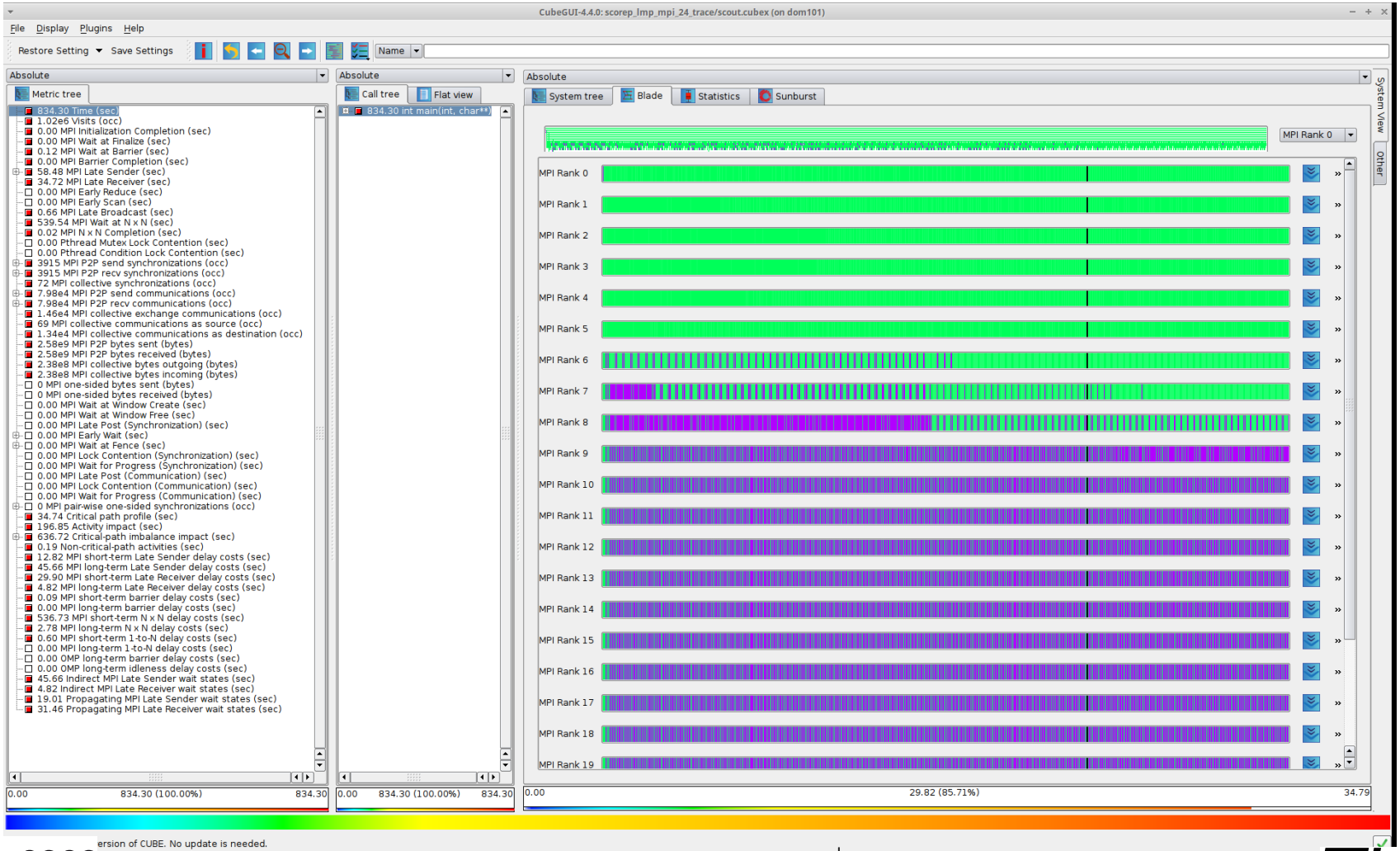
Scalasca: jengafett plugin

- Load the plugin with: **module load jengafettplugin/0.1**
- This plugin allows to display correlation between metrics as a parallel bar charts distribution across the system tree.



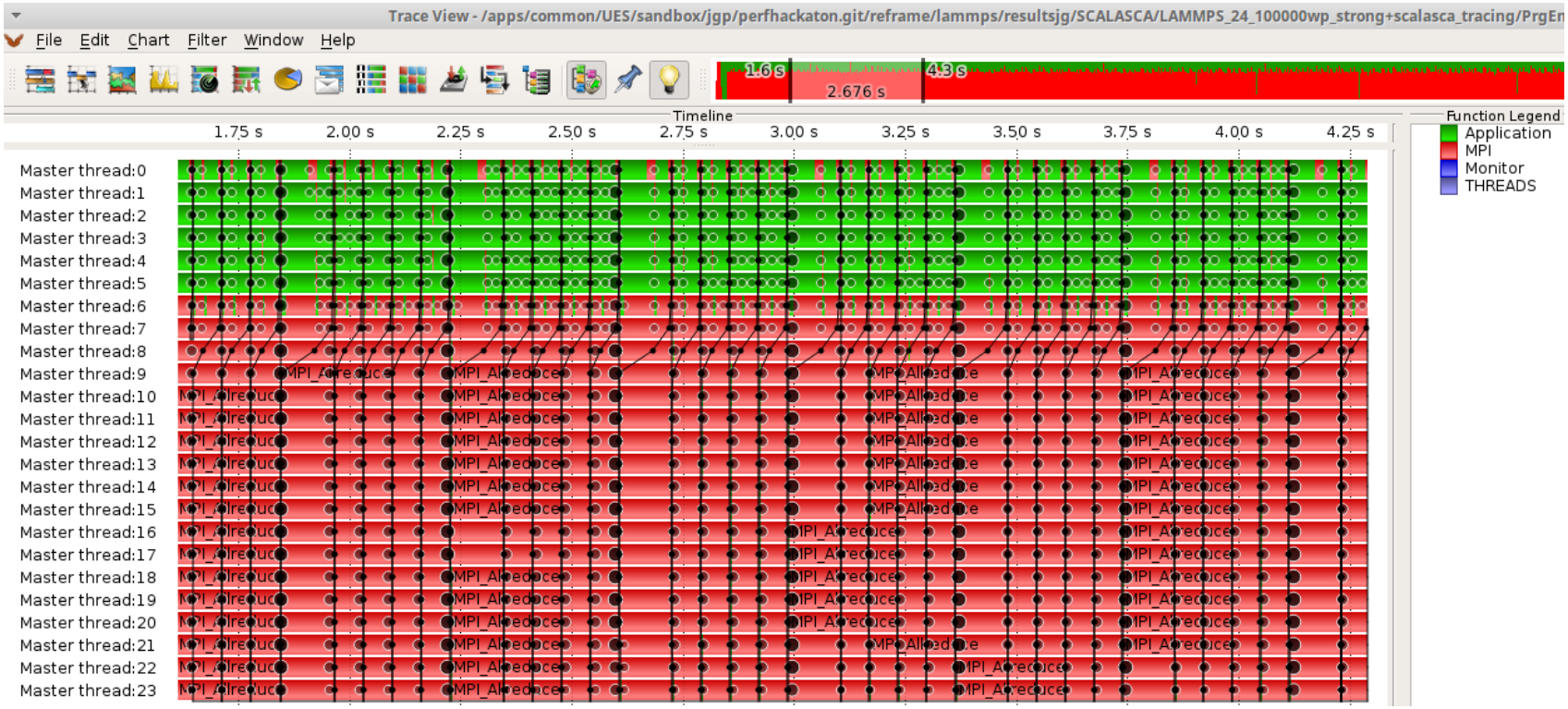
Scalasca: blade plugin

- Load the plugin with: **module load bladeplugin/0.1**
- This plugin is a simple OTF2 (vampir like) trace explorer.



Scalasca: blade plugin

- Same tracefile visualised with Vampir:



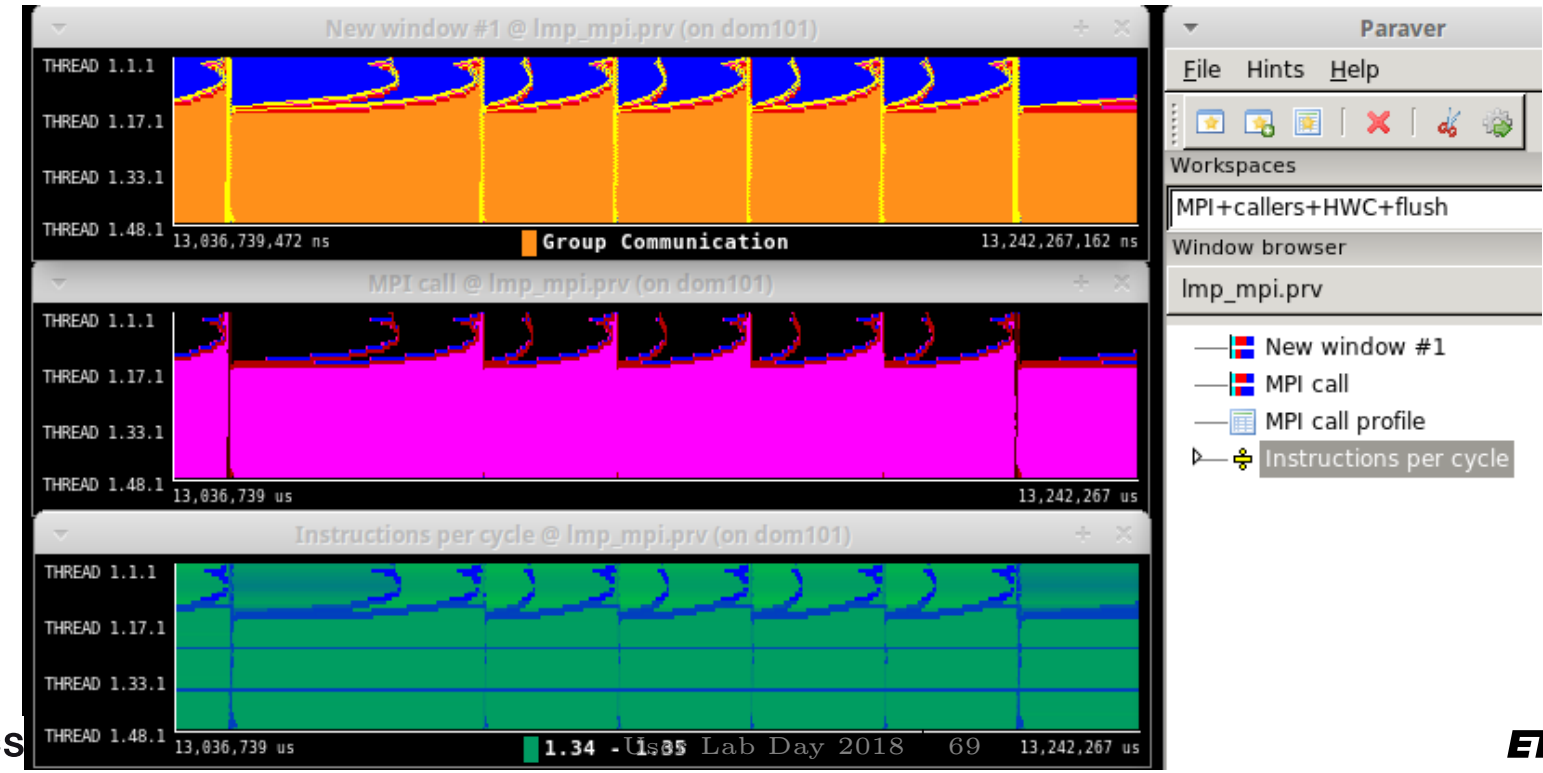
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Performance analysis with Extrae tools

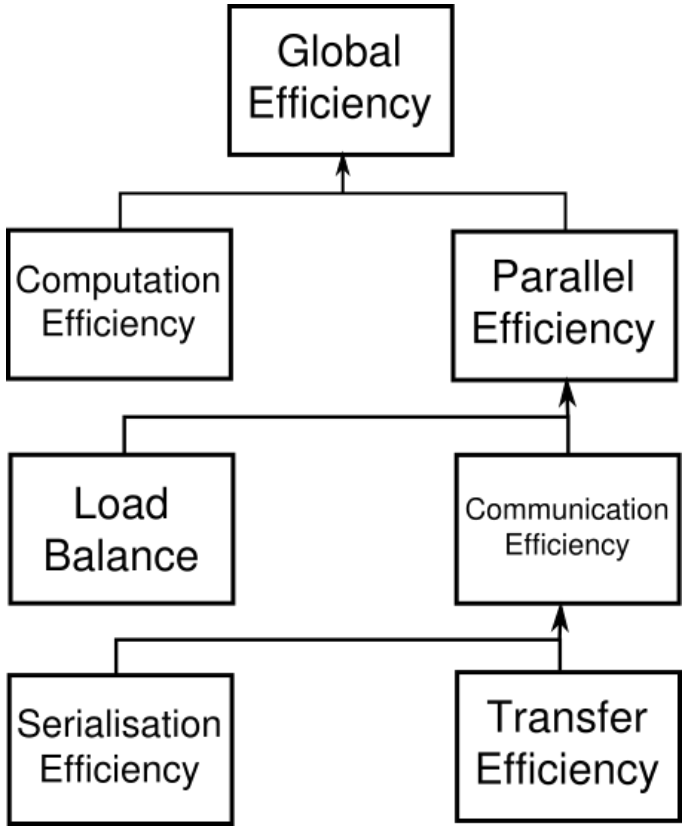
Extrae

- `http://www.bsc.es/computer-sciences/performance-tools`
  - Extrae only requires a dynamically linked executable:
    easybuild: LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1-extrae-3.5.4.eb
    i.e CC **-dynamic ...**
  - reframe: lammps_sph_strong_scaling_100000wp+extrae.py
    i.e srun **./extrae/extrae_daint.sh** ...
  - **wxparaver** ./lmp_mpi.prv &



Extrae: POP performance metrics

- Extrae can derive predefined POP (Performance Optimisation and Productivity) metrics from .prv files. These metrics can be used to calculate performance efficiencies (higher numbers being better) in the code.



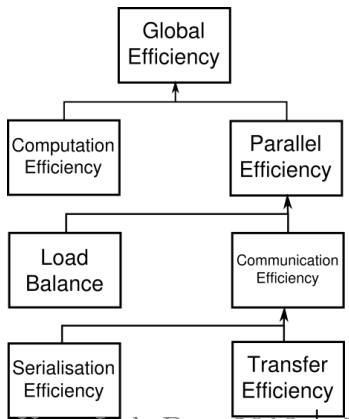
Extrae: POP performance metrics

- *Global efficiency* indicates the overall parallelization quality. Global efficiency is the product of two sub-metrics: Parallel Efficiency and Computation Efficiency.

- *Parallel efficiency* reveals the inefficiency in decomposing the computation over processes and the associated data communicated between processes.

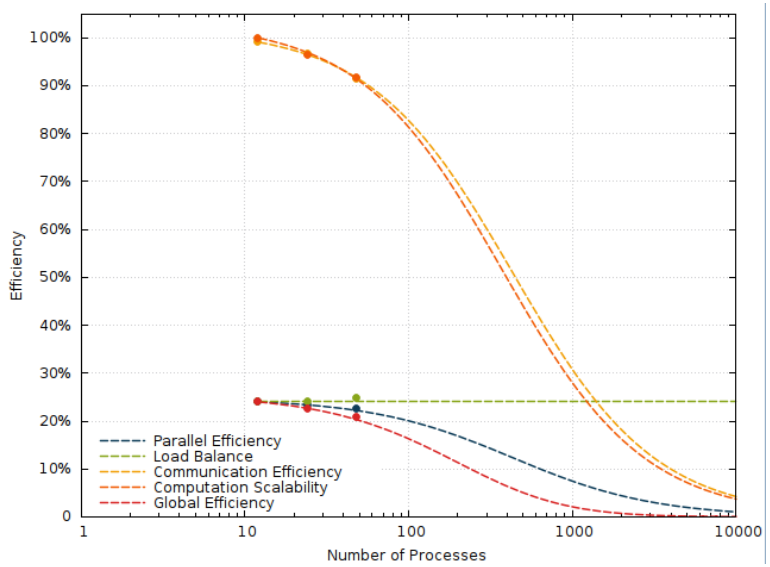
- *Load Balance* is computed as the ratio between average useful computation time (across all processes) and maximum useful computation time (also across all processes). It reflects the quality of the distribution of work to processes.

- *Computation Efficiency* is the ratio of total time in useful computation for a reference job size to the total time as the number of processes (or nodes) increases.



Extrae: POP performance metrics

- **gnuplot** ./modelfactors.gp



- **modelfactors.py** */PrgEnv-gnu/lmp_mpi.prv

```
Overview of the computed model factors:
                              |         12 |         24 |         48
=====================================================================
Parallel efficiency           |     24.00% |     23.34% |     22.71%
  Load balance                |     24.21% |     24.11% |     24.80%
  Communication efficiency    |     99.11% |     96.82% |     91.58%
    Serialization efficiency  |     99.50% |     98.05% |     93.19%
    Transfer efficiency       |     99.61% |     98.75% |     98.27%
Computation scalability       |    100.00% |     96.45% |     91.63%
Global efficiency             |     24.00% |     22.51% |     20.81% <-----
                              |            |            |
IPC scalability               |    100.00% |     97.96% |     96.86%
Instruction scalability       |    100.00% |     99.07% |     96.34%
Frequency scalability         |    100.00% |     99.37% |     98.18%
Average IPC                   |       1.39 |       1.37 |       1.35
Average frequency (GHz)       |       3.07 |       3.05 |       3.02
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

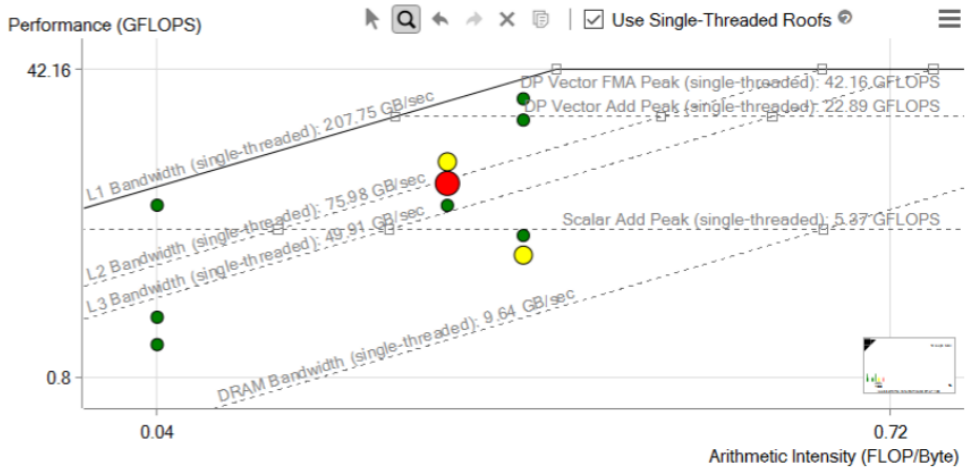
Performance analysis with Intel tools

Intel Advisor: Roofline Model

- The Roofline Model is a visual representation of application performance in relation to hardware limitations, including memory bandwidth and computational peaks.
  * https://software.intel.com
  * http://crd.lbl.gov

A Roofline Chart plots application performance against hardware limitations.

- Where are the bottlenecks?
- How much performance is being left on the table?
- Which bottlenecks can be addressed, and which *should* be addressed?
- What's the most likely cause?
- What are the next steps?



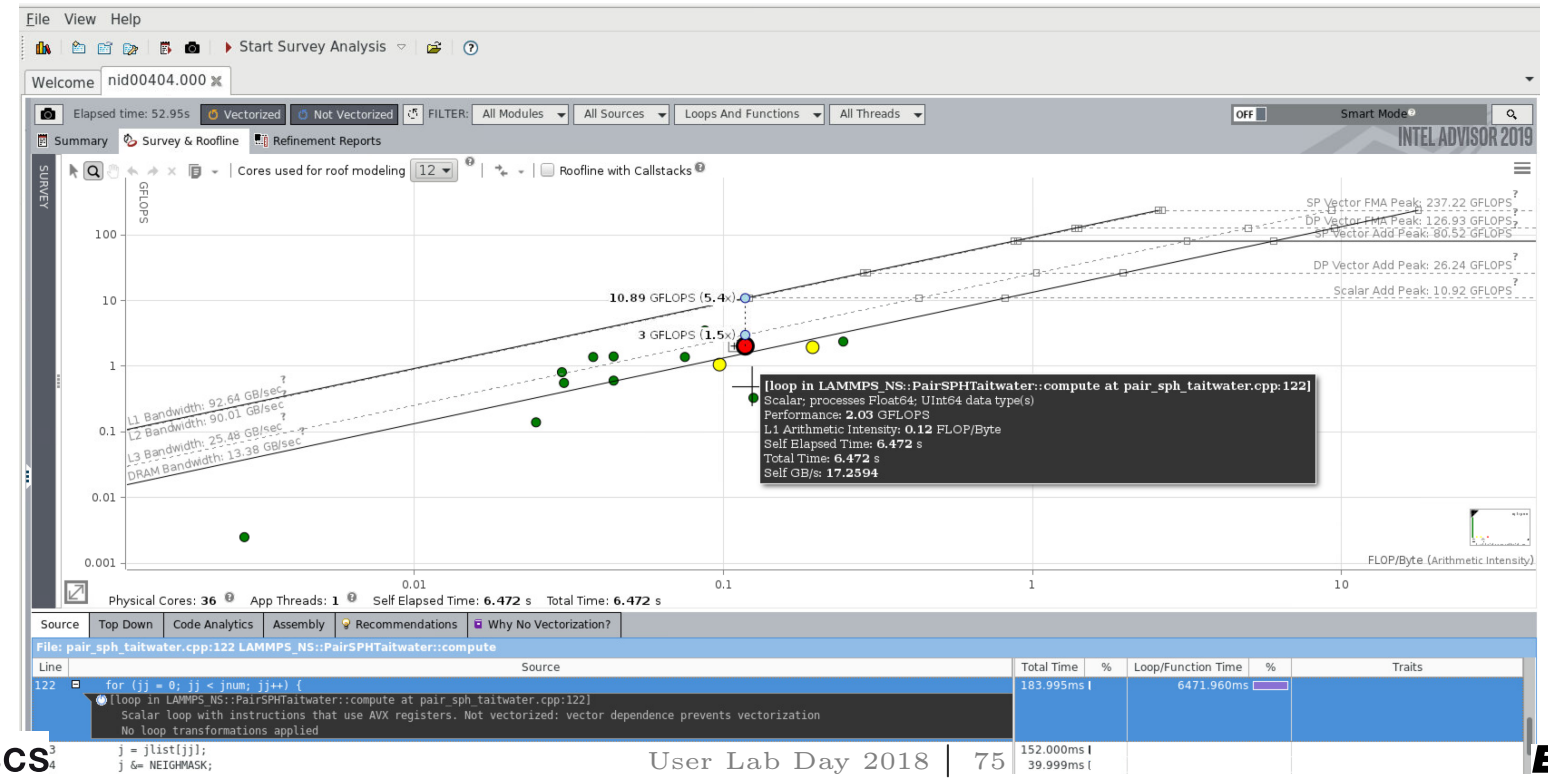
Roofline first proposed by University of California at Berkeley:
Roofline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache-aware variant proposed by University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013

Intel Advisor: Roofline Analysis (LAMMPS)

* easybuild: LAMMPS-16Jul2018-CrayIntel-18.07.eb

* 2 steps are needed to get a roofline model:
  reframe: lammps_sph_strong_scaling_010000wp+roofline_step1.py
  i.e srun **advixe-cl -collect survey -trace-mpi** lmp_mpi ...
  reframe: lammps_sph_strong_scaling_010000wp+roofline_step2.py
  i.e srun **advixe-cl -collect tripcounts -flop -trace-mpi** lmp_mpi ...

* **advixe-gui** ./lammps_roofline_12.advixeproj &



Intel Advisor: Roofline Analysis example

- Step by step optimisation example:

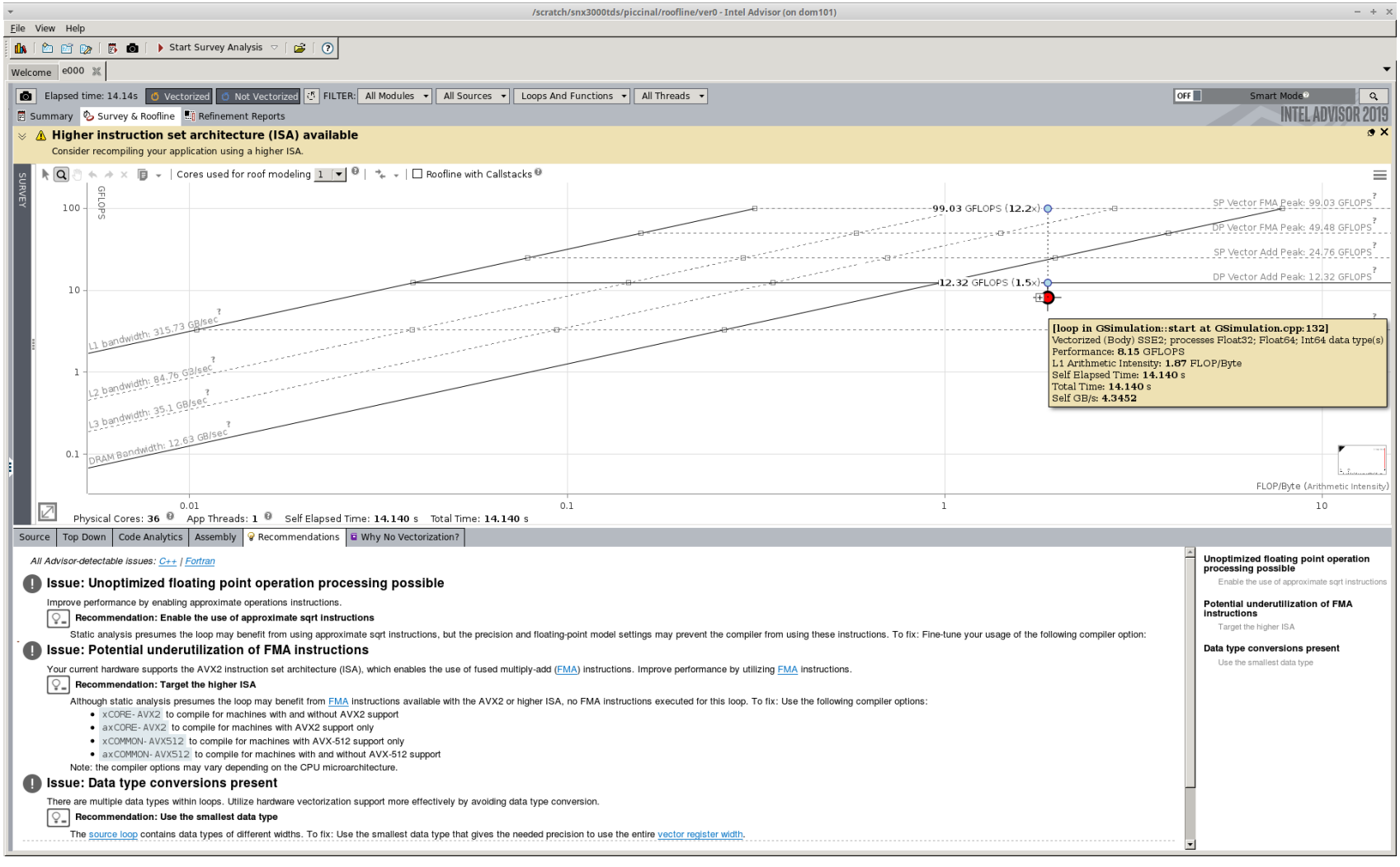
  `https://github.com/fbaru-dev/code-optimization/tree/master/nbody`

- ver0: a simple "C++" n-body simulation code with room for improvements

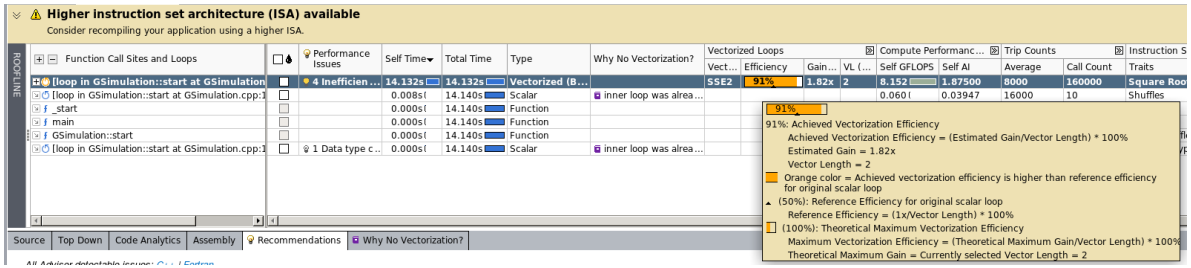
```
127 for (int s=1; s<=get_nsteps(); ++s)
128   {
129     ts0 += time.start();
130     for (i = 0; i < n; i++)// update acceleration
131     {
132       for (j = 0; j < n; j++)
133       {
134       real_type dx, dy, dz;
135       real_type distanceSqr = 0.0;
136       real_type distanceInv = 0.0;
137
138       dx = particles[j].pos[0] - particles[i].pos[0];   //1flop
139       dy = particles[j].pos[1] - particles[i].pos[1];   //1flop
140       dz = particles[j].pos[2] - particles[i].pos[2];   //1flop
141
142       distanceSqr = dx*dx + dy*dy + dz*dz + softeningSquared;   //6flops
143       distanceInv = 1.0 / sqrt(distanceSqr);            //1div+1sqrt
144
145       particles[i].acc[0] += dx * G * particles[j].mass * distanceInv * distanceInv * distanceInv;
146       particles[i].acc[1] += dy * G * particles[j].mass * distanceInv * distanceInv * distanceInv;
147       particles[i].acc[2] += dz * G * particles[j].mass * distanceInv * distanceInv * distanceInv;
148
149       }
150     }
GSimulation.cpp
```

- ver1: compile with the AVX2 ISA instructions set
- ver2: fix for data type conversions issue reported by advisor in ver1
- ver3: fix for data layout issue (SoA instead of AoS) reported by advisor in ver2 (strides)

Intel Advisor: version0 roofline



Intel Advisor: version0 survey and summary



Elapsed time: 14.14s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Summary | Survey & Roofline | Refinement Reports

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	14.14s		
Vector Instruction Set	SSE2, SSE	Number of CPU Threads	1
Total GFLOP Count	115.20	Total GFLOPS	8.14
Total Arithmetic Intensity	1.87464		

Loop metrics

Metrics	Total	
Total CPU time	14.14s	100.0%
Time in **1** vectorized loop	14.14s	100.0%
Time in scalar code	0s	
Total GFLOP Count	115.20	100.0%
Total GFLOPS	8.14	

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	1.82x	91%
Program Approximate Gain	1.82x	

Per program recommendations

Higher instruction set architecture (ISA) available

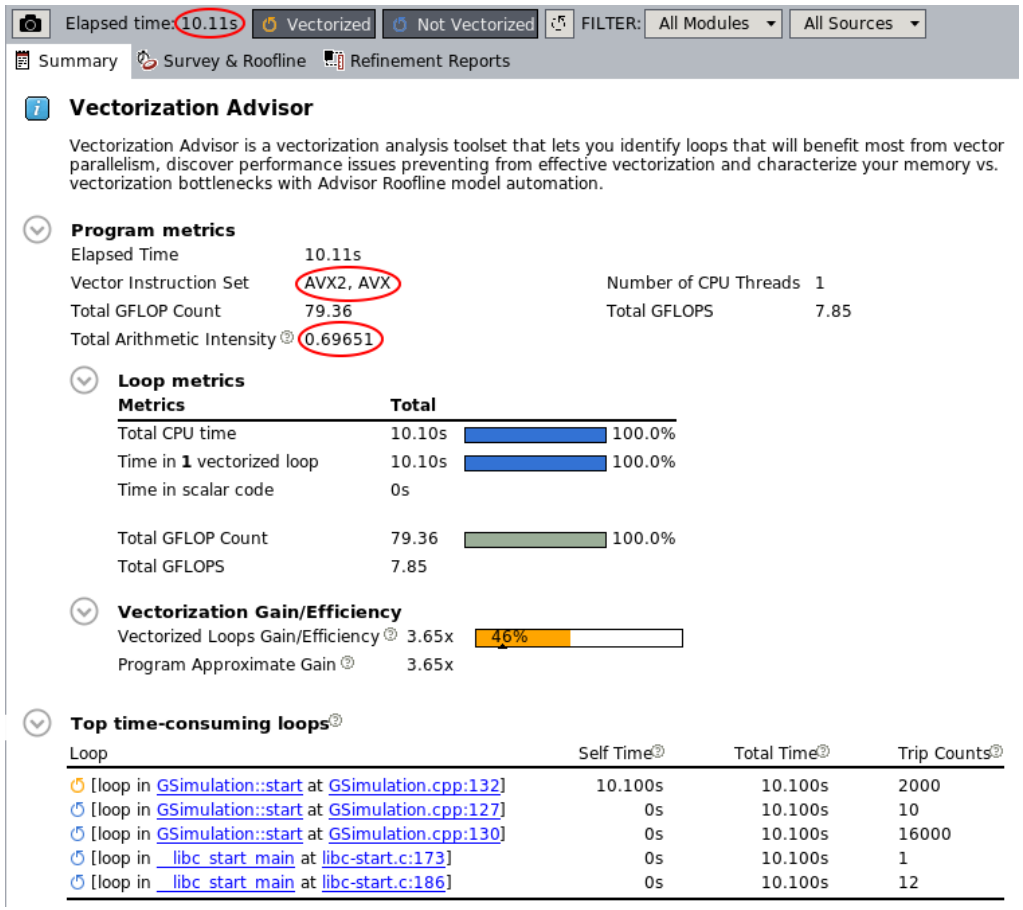
Your application was compiled using the SSE2 instruction set, which is lower than AVX2 - the highest ISA available on the target machine. To compile the application using the highest ISA available on the target machine: Use the **-xCORE-AVX2** option. Show less

Top time-consuming loops

Loop	Self Time	Total Time	Trip Counts
[loop in GSimulation::start at GSimulation.cpp:132]	14.140s	14.140s	8000
[loop in GSimulation::start at GSimulation.cpp:127]	0s	14.140s	10
[loop in GSimulation::start at GSimulation.cpp:130]	0s	14.140s	16000
[loop in __libc_start_main at libc-start.c:173]	0s	14.140s	1
[loop in __libc_start_main at libc-start.c:186]	0s	14.140s	12

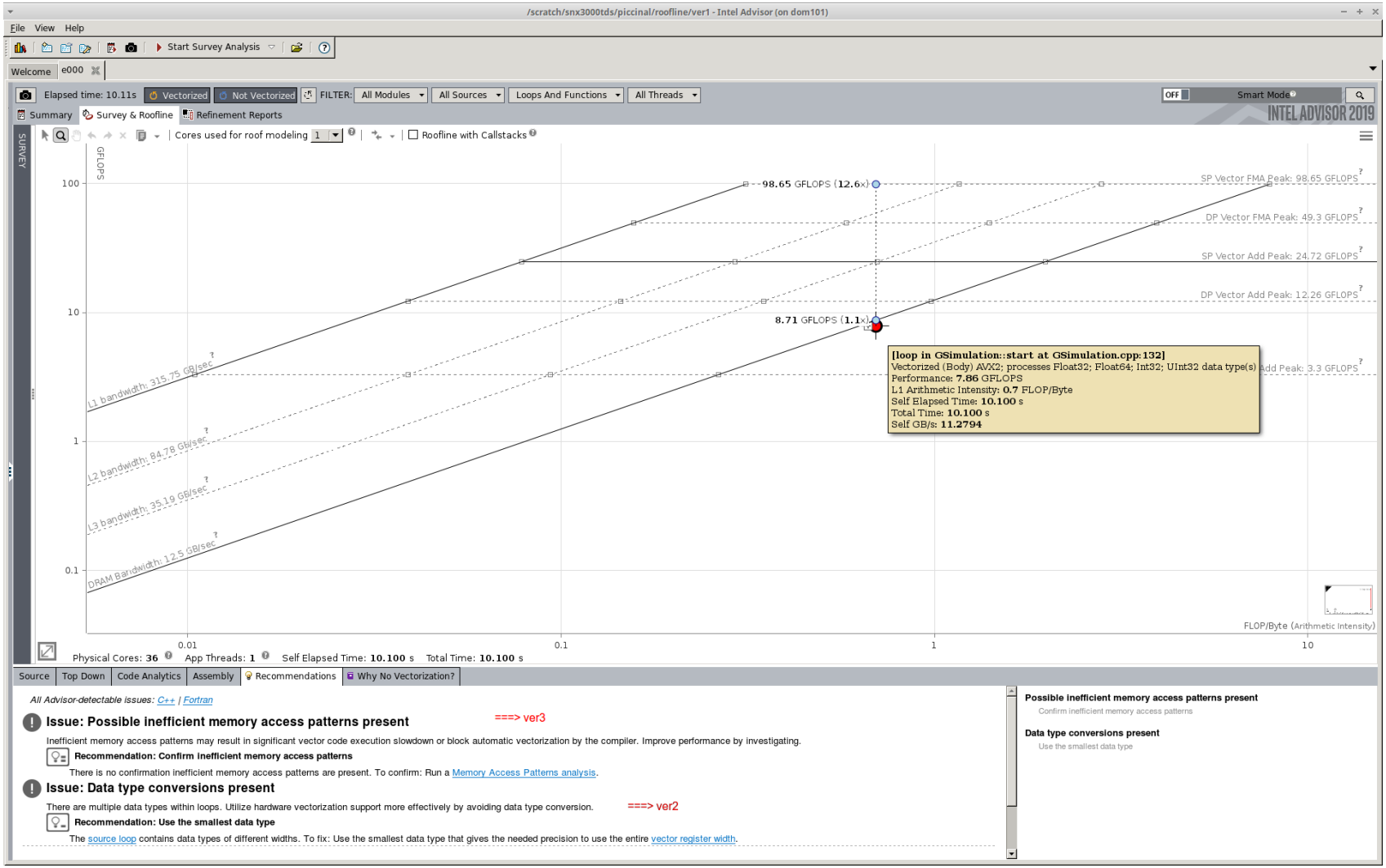
Intel Advisor: Roofline Analysis (ver1)

- ver0: a simple n-body simulation code with lot of room for improvements
- ver1: compiling with the AVX2 ISA instructions speeds up the code:

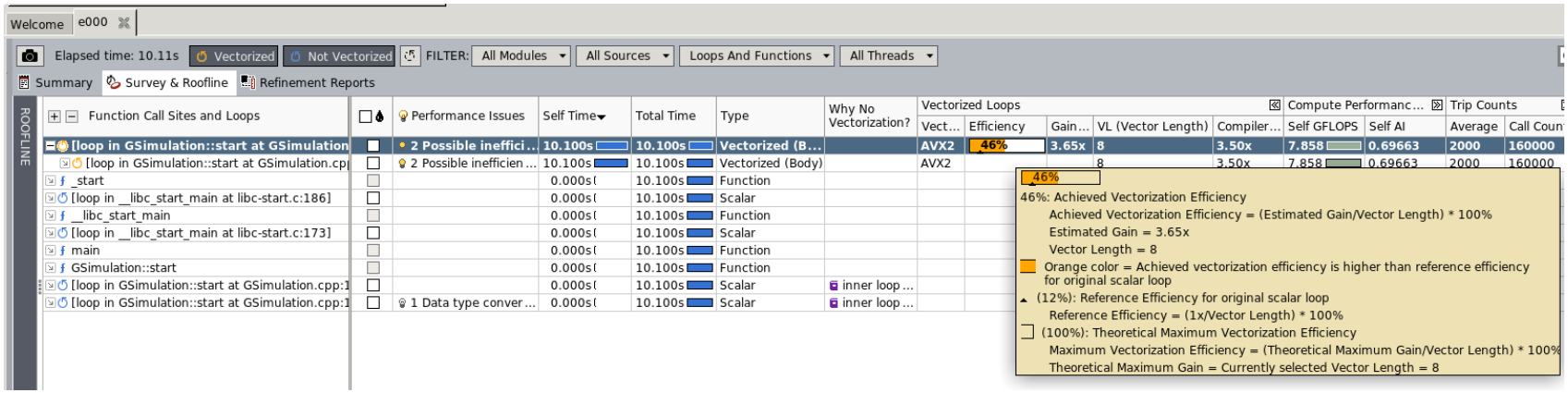


- ver2: fix for data type conversions issue reported by advisor in ver1
- ver3: fix for data layout issue (SoA instead of AoS) reported by advisor in ver2 (strides)

Intel Advisor: version1 roofline



Intel Advisor: version1 survey



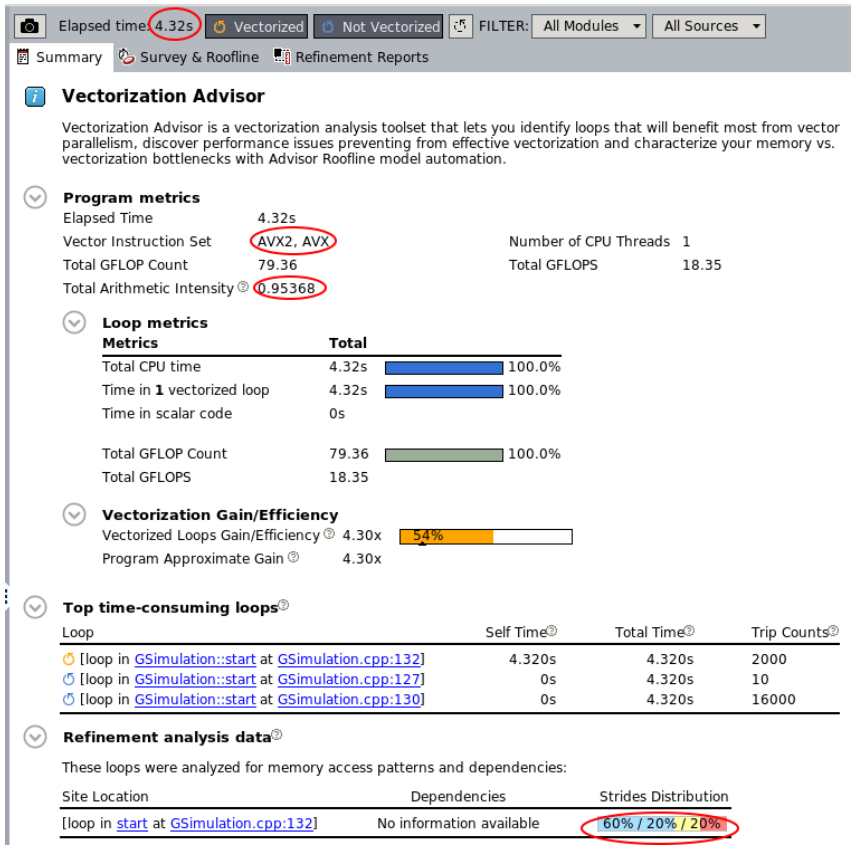

```
sdiff -s 1/GSimulation.cpp                    2/GSimulation.cpp

const double softeningSquared = 1e-3;       |   const float softeningSquared = 1e-3f;
const double G = 6.67259e-11;               |   const float G = 6.67259e-11f;
real_type distanceSqr = 0.0;                |   real_type distanceSqr = 0.0f;
real_type distanceInv = 0.0;                |   real_type distanceInv = 0.0f;
distanceInv = 1.0 / sqrt(distanceSqr);      |   distanceInv = 1.0f / sqrtf(distanceSqr);
```

a double has 2x the size of a float

Intel Advisor: Roofline Analysis (ver2)

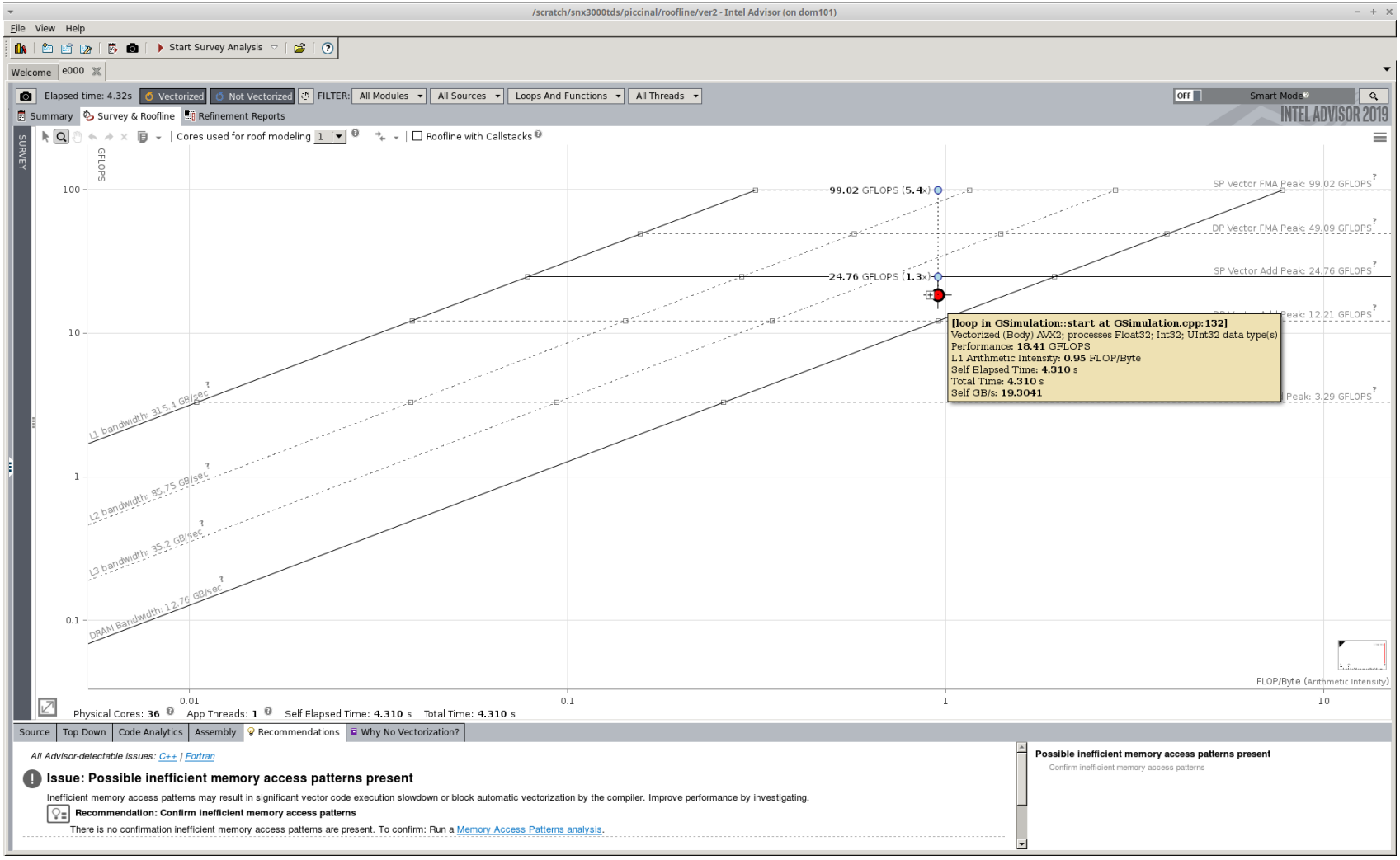
- ver0: a simple n-body simulation code with lot of room for improvements
- ver1: compiling with the AVX2 ISA instructions speeds up the code
- ver2: fixing the data type conversions issue reported in ver1 speeds up the code



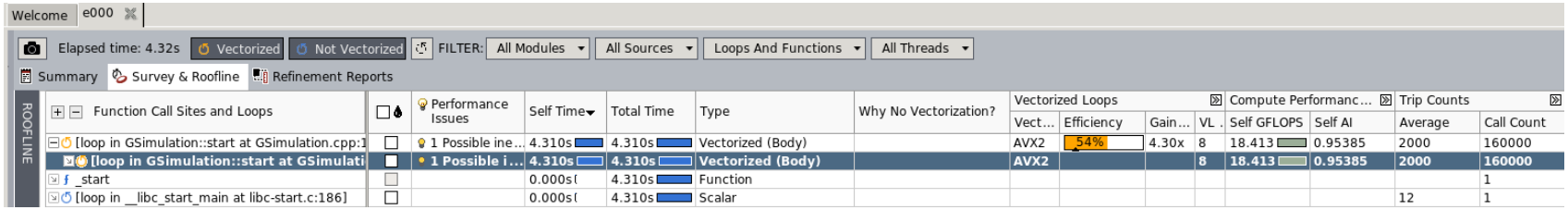
strides distribution: Unit/Constant/Variable stride ratio

- ver3: fix for data layout issue (SoA instead of AoS) reported by advisor in ver2 (strides)

Intel Advisor: version2 roofline



Intel Advisor: version2 survey




```
25
26 struct Particle
27 {
28   public:
29     Particle() { init();}
30     void init()
31     {
32       pos[0] = 0.; pos[1] = 0.; pos[2] = 0.;
33       vel[0] = 0.; vel[1] = 0.; vel[2] = 0.;
34       acc[0] = 0.; acc[1] = 0.; acc[2] = 0.;
35       mass   = 0.;
36     }
37     real_type pos[3];
38     real_type vel[3];
39     real_type acc[3];
40     real_type mass;
41 };
42
```

2/Particle.hpp 25,0-1 90%

```
21 typedef float real_type;
../types.hpp 11,0-1
42
43 struct ParticleSoA
44 {
45   public:
46     ParticleSoA() { init();}
47     void init()
48     {
49       pos_x = NULL; pos_y = NULL; pos_z = NULL;
50       vel_x = NULL; vel_y = NULL; vel_z = NULL;
51       acc_x = NULL; acc_y = NULL; acc_z = NULL;
52       mass  = NULL;
53     }
54     real_type *pos_x, *pos_y, *pos_z;
55     real_type *vel_x, *vel_y, *vel_z;
56     real_type *acc_x, *acc_y, *acc_z;
57     real_type *mass;
```

3/Particle.hpp 57,1

```
152   for (i = 0; i < n; ++i)// update position and velocity
153   {
154     particles[i].vel[0] += particles[i].acc[0] * dt;  //2flops
155     particles[i].vel[1] += particles[i].acc[1] * dt;  //2flops
156     particles[i].vel[2] += particles[i].acc[2] * dt;  //2flops
157
158     particles[i].pos[0] += particles[i].vel[0] * dt;  //2flops
159     particles[i].pos[1] += particles[i].vel[1] * dt;  //2flops
160     particles[i].pos[2] += particles[i].vel[2] * dt;  //2flops
161
162     particles[i].acc[0] = 0.;
163     particles[i].acc[1] = 0.;
164     particles[i].acc[2] = 0.;
165
166     energy += particles[i].mass * (
167       particles[i].vel[0]*particles[i].vel[0] +
168             particles[i].vel[1]*particles[i].vel[1] +
169             particles[i].vel[2]*particles[i].vel[2]); //7flops
```

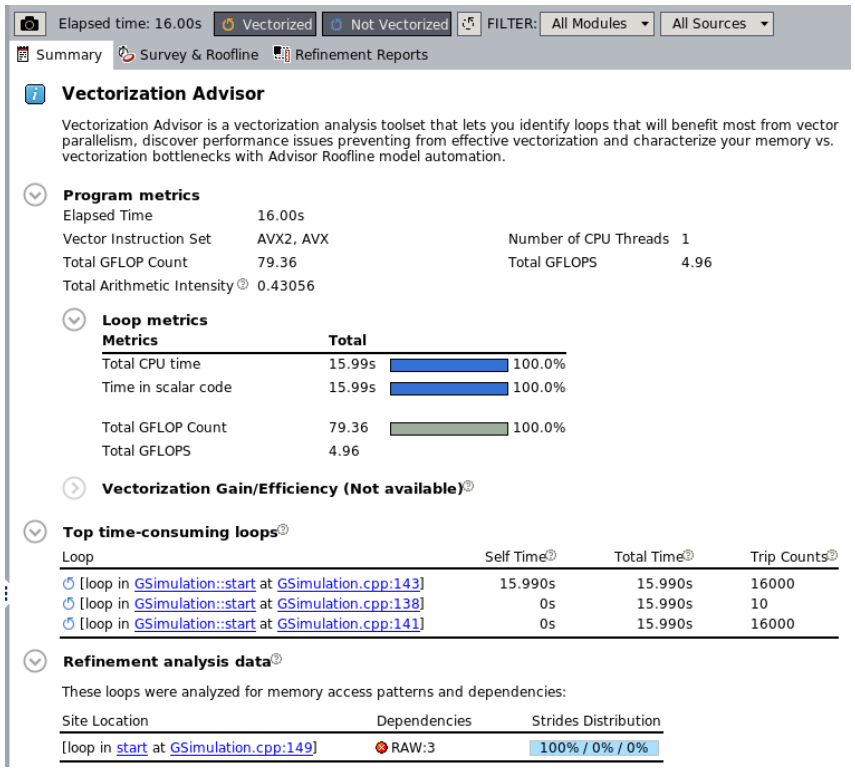
ons/2/GSimulation.cpp [+] 115,41 60%

```
163   for (i = 0; i < n; ++i)// update position and velocity
164   {
165     particles->vel_x[i] += particles->acc_x[i] * dt; //2flops
166     particles->vel_y[i] += particles->acc_y[i] * dt; //2flops
167     particles->vel_z[i] += particles->acc_z[i] * dt; //2flops
168
169     particles->pos_x[i] += particles->vel_x[i] * dt; //2flops
170     particles->pos_y[i] += particles->vel_y[i] * dt; //2flops
171     particles->pos_z[i] += particles->vel_z[i] * dt; //2flops
172
173     particles->acc_x[i] = 0.;
174     particles->acc_y[i] = 0.;
175     particles->acc_z[i] = 0.;
176
177     energy += particles->mass[i] * (
178           particles->vel_x[i]*particles->vel_x[i] +
179                 particles->vel_y[i]*particles->vel_y[i] +
180                 particles->vel_z[i]*particles->vel_z[i]);
181   }
```

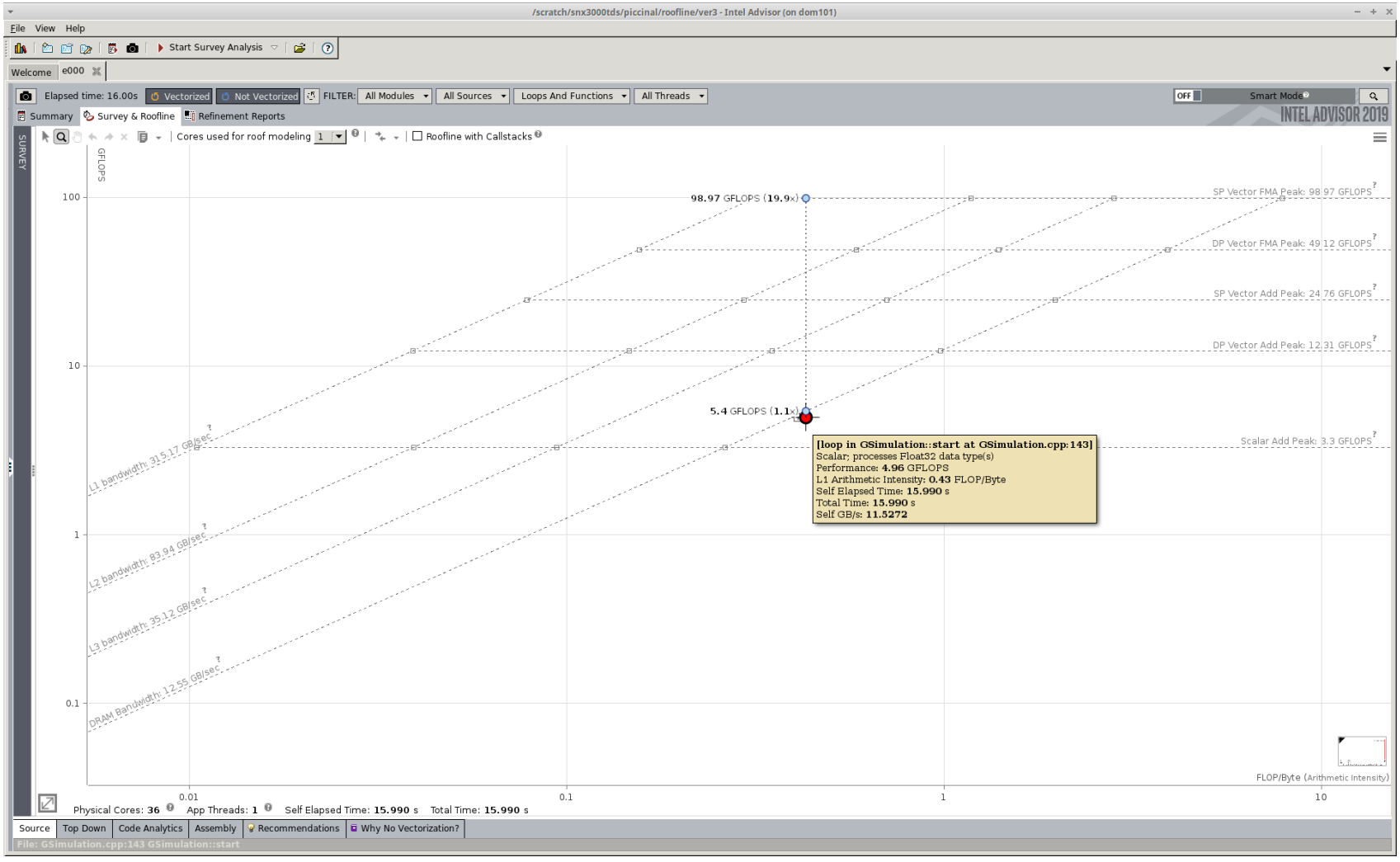
src/C++/versions/3/GSimulation.cpp

Intel Advisor: Roofline Analysis (ver3)

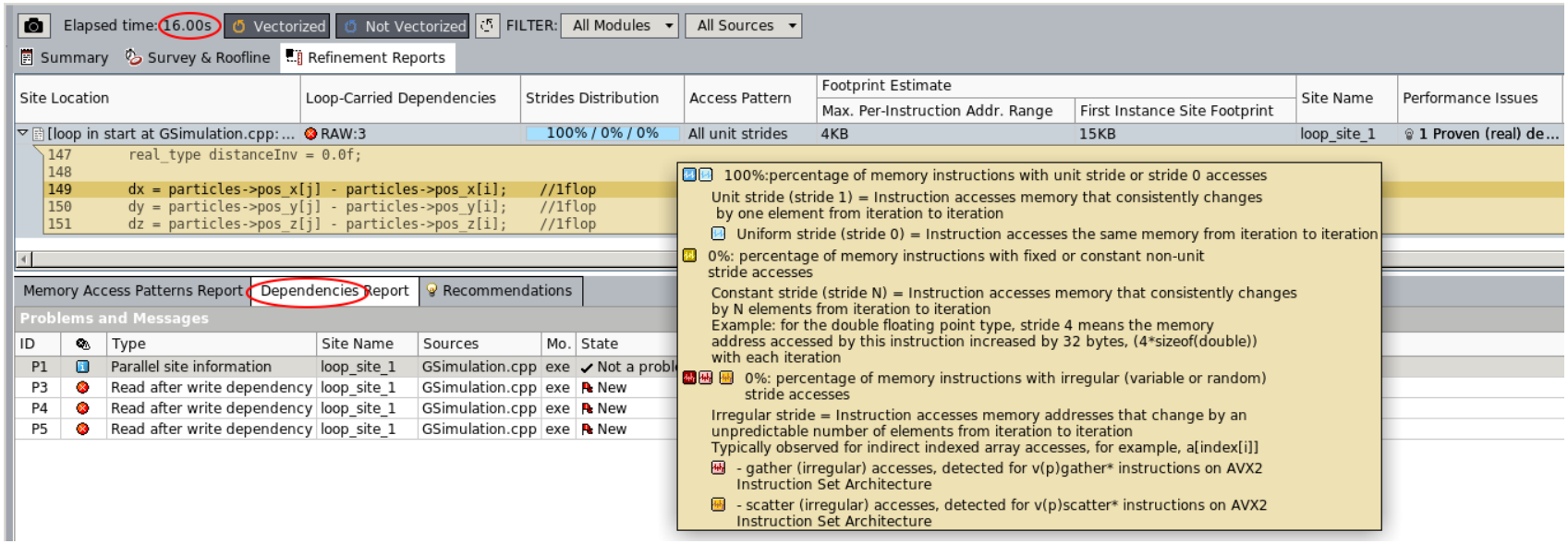
- ver0: a simple n-body simulation code with lot of room for improvements
- ver1: compiling with the AVX2 ISA instructions speeds up the code
- ver2: fixing the data type conversions issue reported in ver1 speeds up the code
- ver3: changing the data layout issue (SoA instead of AoS) fixes the strides issue reported in ver2 but slowdowns the code

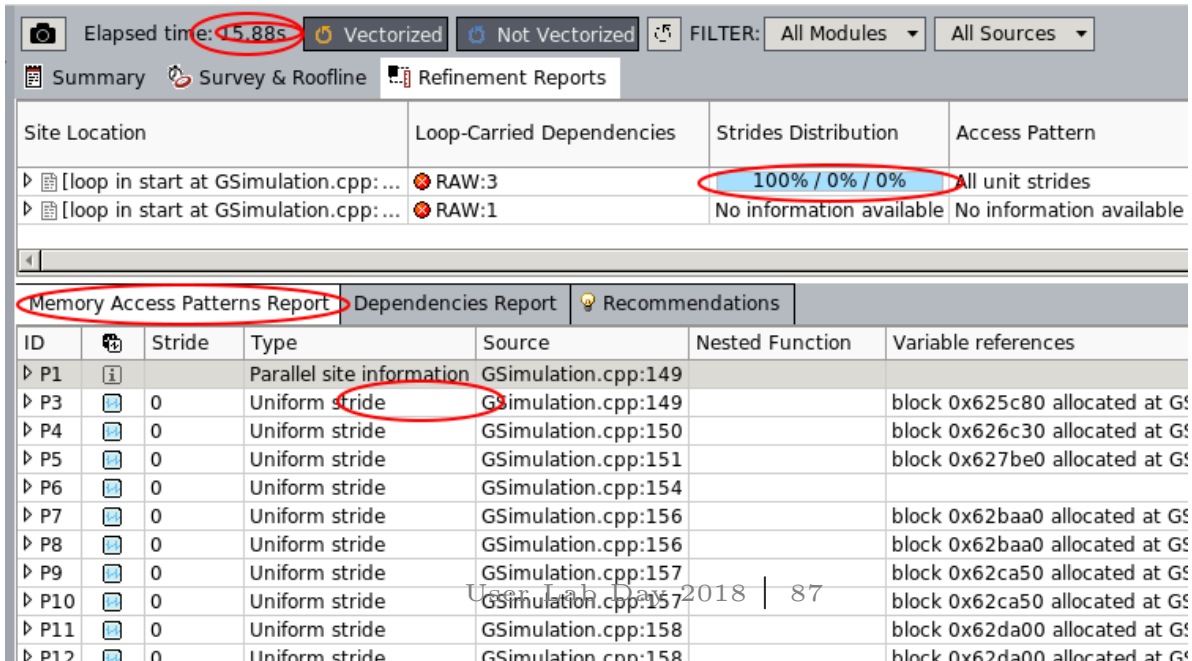


Intel Advisor: version3 roofline

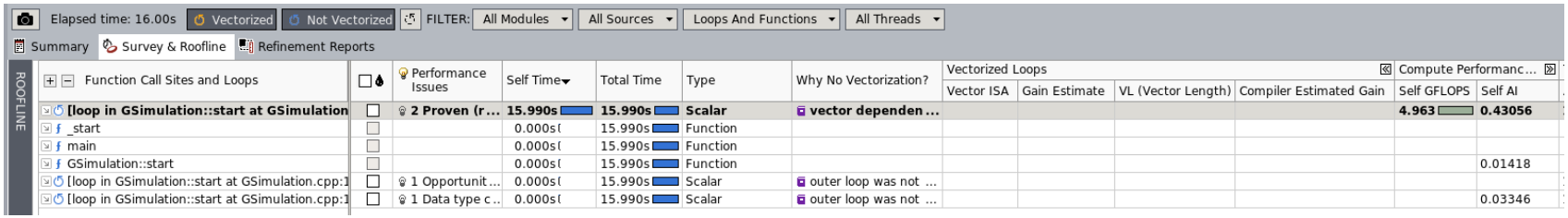


Intel Advisor: version3 memory accesses



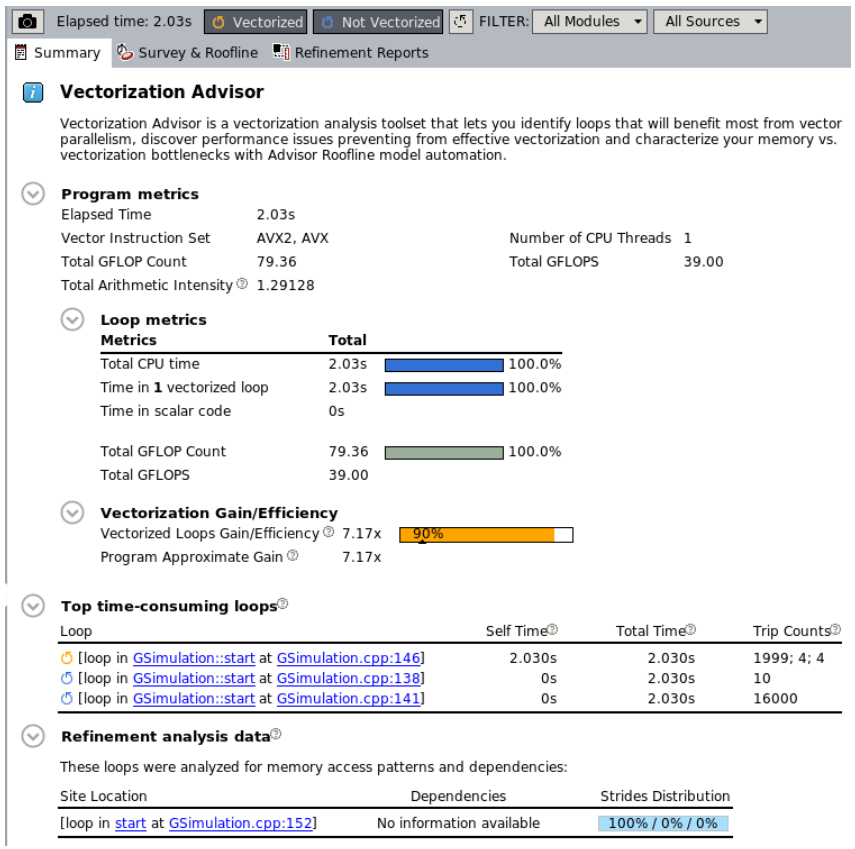


Intel Advisor: version3 survey

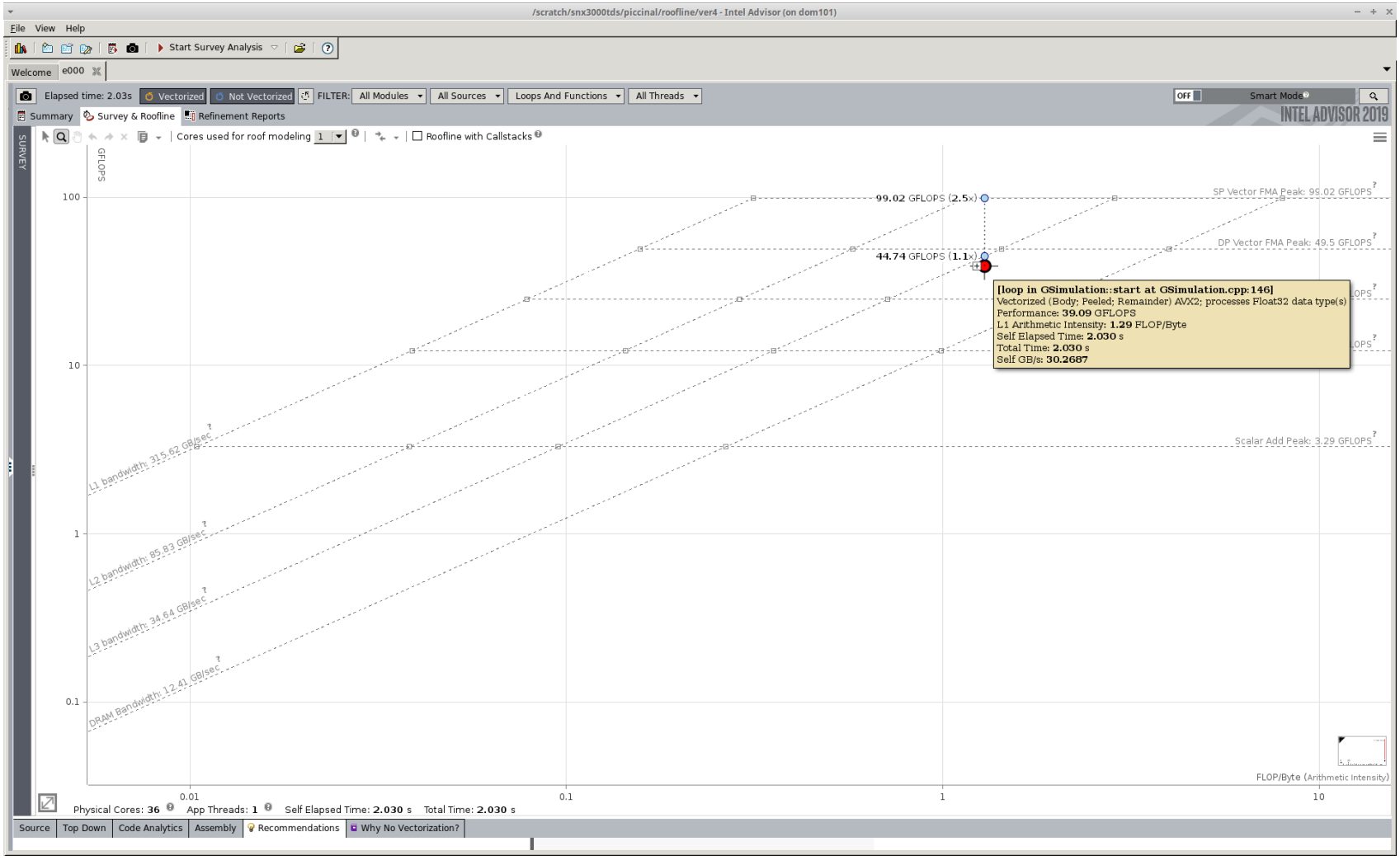


Intel Advisor: Roofline Analysis (ver4)

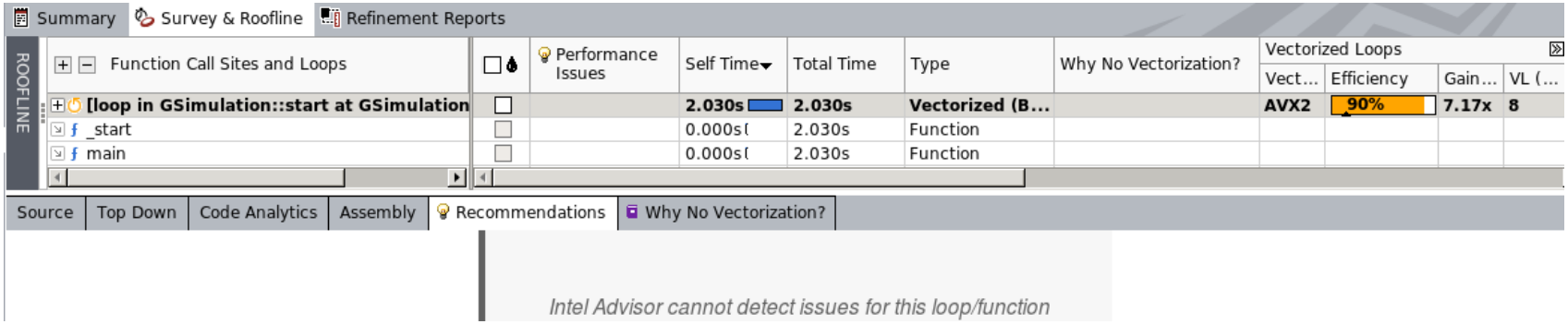
- ver0: a simple n-body simulation code with lot of room for improvements
- ver1: compiling with the AVX2 ISA instructions speeds up the code
- ver2: fixing the data type conversions issue reported in ver1 speeds up the code
- ver3: changing the data layout issue (SoA instead of AoS) fixes the strides issue reported in ver2 but slowdowns the code
- ver4: fixing the issue reported in ver3 speeds up the code



Intel Advisor: version4 roofline

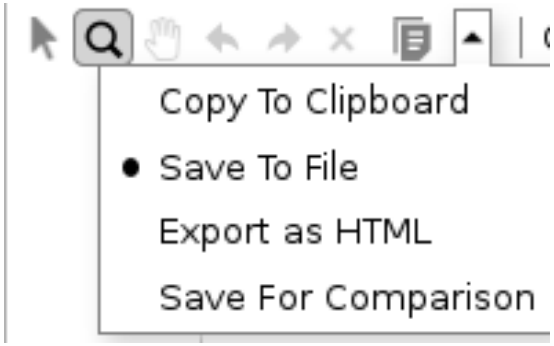


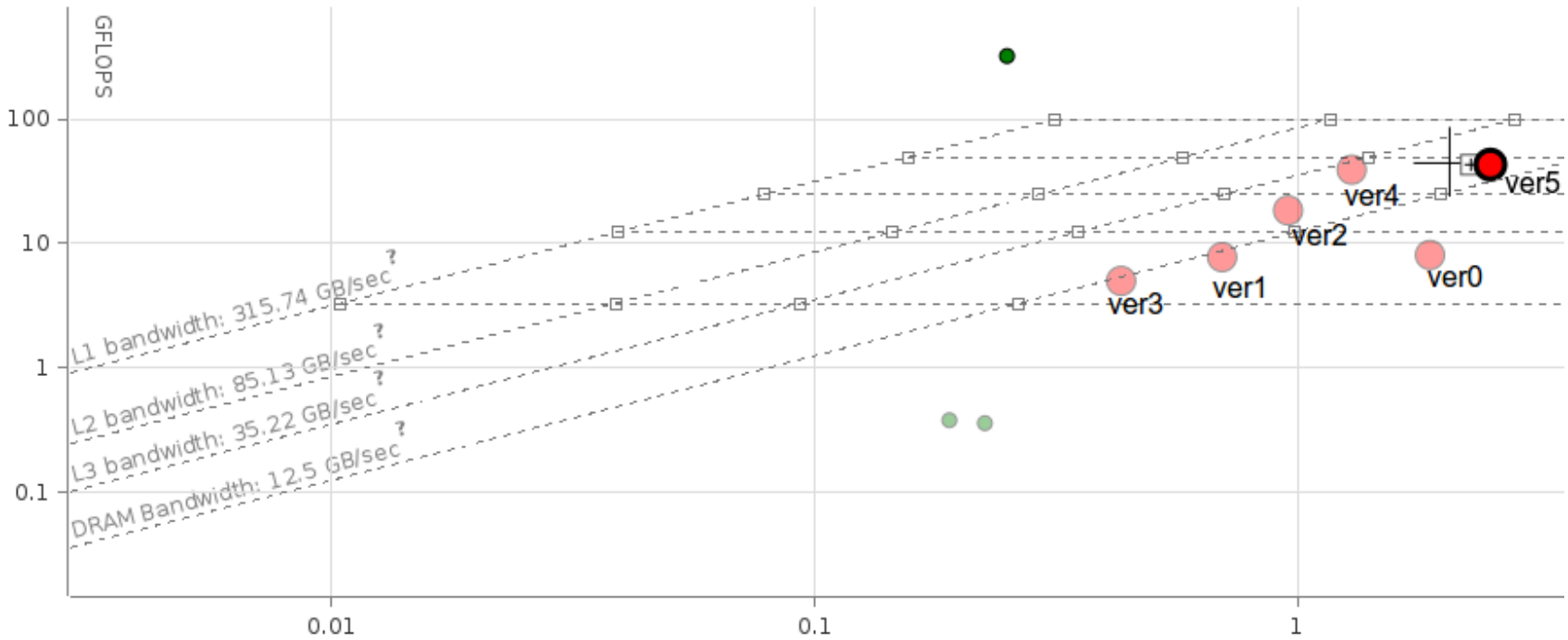
Intel Advisor: version4 survey



Intel Advisor: conclusion

- ver0: 14 seconds - 8.15 GFLOPS
- ver1: 10 seconds - 7.86 GFLOPS
- ver2: 4 seconds - 18.41 GFLOPS
- ver3: 16 seconds - 4.96 GFLOPS
- ver4: 2 seconds - 39.09 GFLOPS





Concluding Remarks

Summary

For the example of the collapsing water column simulation using the LAMMPS SPH solver, this presentation provided

- an overview of the debugging & performance tools on Piz Daint
- an idea of the area of application & the capabilities of each tool
- a basic introduction to performance analysis & why it is needed

Editorial comment:

Regarding the analysis and optimization of GPU codes:
GPU Hackathon 2018

Appendix

Piz Daint: daint-gpu

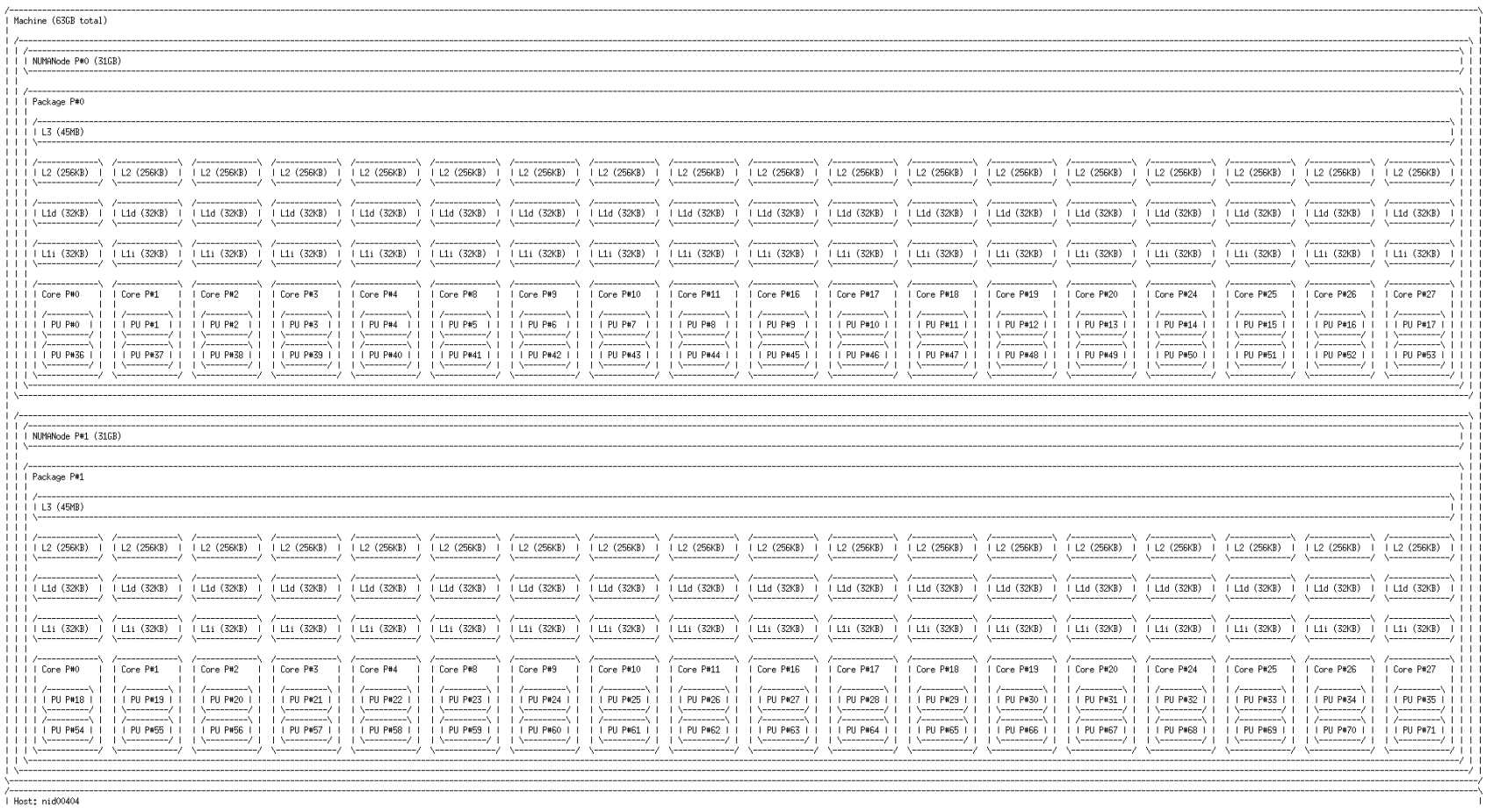
```
srun -Cgpu -t1 -n1 lstopo --output-format txt --no-io --no-bridges

Machine (63GB)
  Package P#0
    L3 (30MB)
      L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)  L2 (256KB)
      L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)  L1d (32KB)
      L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)  L1i (32KB)
      Core P#0    Core P#1    Core P#2    Core P#3    Core P#4    Core P#5    Core P#8    Core P#9    Core P#10   Core P#11   Core P#12   Core P#13
        PU P#0      PU P#1      PU P#2      PU P#3      PU P#4      PU P#5      PU P#6      PU P#7      PU P#8      PU P#9      PU P#10     PU P#11
        PU P#12     PU P#13     PU P#14     PU P#15     PU P#16     PU P#17     PU P#18     PU P#19     PU P#20     PU P#21     PU P#22     PU P#23
Host: nid00000
```

Back to the Piz Daint slide.

Piz Daint: daint-mc

```
srun -Cmc -t1 -n1 lstopo --output-format txt --no-io --no-bridges
```



Back to the Piz Daint slide.

Building LAMMPS with Easybuild

- eb LAMMPS-16Jul2018-CrayGNU-18.07.eb

```
== temporary log file in case of crash /tmp/eb-ZP9vJ0/easybuild-m0\_Pb6.log
== processing EasyBuild easyconfig LAMMPS-16Jul2018-CrayGNU-18.07-cuda-9.1.eb
== building and installing LAMMPS/16Jul2018-CrayGNU-18.07-cuda-9.1...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== permissions...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file(s)
   /software/LAMMPS/16Jul2018-CrayGNU-18.07-cuda-9.1/easybuild/
    easybuild-LAMMPS-16Jul2018-20180906.164506.log
== Build succeeded for 1 out of 1
== Temporary log file(s) /dev/shm/piccinal/easybuild/stage/tmp/
            eb-ZP9vJ0/easybuild-m0\_Pb6.log* have been removed.
== Temporary directory /dev/shm/piccinal/easybuild/stage/tmp/
            eb-ZP9vJ0 has been removed.
```