HPCTools

Release 0.1

SCS/CSCS

May 11, 2021

TABLE OF CONTENTS:

ReFrame is a framework for writing regression tests for HPC systems. This repository showcases how to use ReFrame together with HPC tools.

HPC tools	Focus		Prog. Model			Language			
	node level	parallel	MPI/ Openmp	CUDA	OpenACC	c++/fortran	python	License	Contact
Score-P/ Vampir	Y	Y	Y	Y	Y	Y	x	Open src	EU
Scalasca	Y	Y	Y	x	x	Y	x	Open src	JSC
Likwid	Y	x	Y	Y	Y	Y	Y	Open src	Erlangen
Extrae	Y	Y	Y	Y	x	Y	Y	Open src	BSC
mpiP	x	Y	Y	x	x	Y		Open src	LLNL
Intel Vtune	Y	Y	Y	x	x	Y	Y	Commercial	Intel
Intel Advisor	Y	Y	Y	x	x	Y		Commercial	Intel
CrayPAT	Y	Y	Y	Y	x	Y	x	Commercial	Cray
NVIDIA® Nsight™	Y	Y	Y	Y	Y	Y		Commercial	Nvidia
Debugging									
Arm Forge DDT	Y	Y	Y	Y	Y	Y		Commercial	ARM
Cray ATP		Y	Y			Y		Commercial	Cray
GNU gdb	Y	Y	Y	x	x	Y		Open src	GNU
Nvidia gdb	Y	Y	Y	Y	Y	Y		Open src	Nvidia

Fig. 1: HPC tools overview

CHAPTER

ONE

GETTING STARTED

Usage of performance tools to identify and remove bottlenecks is part of most modern performance engineering workflows. ReFrame is a regression testing framework for HPC systems that allows to write portable regression tests, focusing only on functionality and performance. It has been used in production at CSCS since 2016 and is being actively developed. All the tests used in this guide are freely available here.

This page will guide you through writing ReFrame tests to analyze the performance of your code. You should be familiar with ReFrame, this link to the ReFrame tutorial can serve as a starting point. As a reference test code, we will use the SPH-EXA mini-app. This code is based on the smoothed particle hydrodynamics (SPH) method, which is a particle-based, meshfree, Lagrangian method for simulating multidimensional fluids with arbitrary geometries, commonly employed in astrophysics, cosmology, and computational fluid-dynamics. The mini-app is a C++14, lightweight, flexible, and header-only code with no external software dependencies. Parallelism is expressed via multiple programming models such as OpenMP and HPX for node level parallelism, MPI for internode communication and Cuda, OpenACC and OpenMP targets for accelerators. Our reference HPC system is Piz Daint.

The most simple case of a ReFrame test is a test that compiles, runs and checks the output of the job. Looking into the `Class` shows how to setup and run the code with the tool. In this example, we set the 3 parameters as follows: 24 mpi tasks, a cube size of 100^3 particles in the 3D square patch test and only 1 step of simulation.

1.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/notool/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
--keep-stage-files \
-c ./internal_timers_mpi.py
```

where:

- `-C` points to the site config-file,
- `--system` selects the targeted system,
- `--prefix` sets output directory,
- `-r` runs the selected check,
- `-c` points to the check.

A typical output from ReFrame will look like this:

```
Reframe version: 2.22
Launched on host: daint101
Reframe paths
=============
    Check prefix      :
    Check search path : 'internal_timers_mpi.py'
    Stage dir prefix     : /scratch/snx3000tds/piccinal/stage/
    Output dir prefix    : /scratch/snx3000tds/piccinal/output/
    Perf. logging prefix : /scratch/snx3000tds/piccinal/perflogs
[==========] Running 1 check(s)

[----------] started processing sphexa_timers_sqpatch_024mpi_001omp_100n_0steps (Strong␣
↪scaling study)
[ RUN      ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪cray
[       OK ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪cray
[ RUN      ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪cray_classic
[       OK ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪cray_classic
[ RUN      ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
[       OK ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
[ RUN      ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪intel
[       OK ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪intel
[ RUN      ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪pgi
[       OK ] sphexa_timers_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪pgi
[----------] finished processing sphexa_timers_sqpatch_024mpi_001omp_100n_0steps (Strong␣
↪scaling study)

[  PASSED  ] Ran 5 test case(s) from 1 check(s) (0 failure(s))
```

By default, the test is run with every programming environment set inside the check. It is possible to select only one programming environment with the *-p* flag (`-p PrgEnv-gnu` for instance).

1.2 Sanity checking

All of our tests passed. Sanity checking checks if a test passed or not. In this simple example, we check that the job output reached the end of the first step, this is coded in the `self.sanity_patterns` part of the `Class`.

1.3 Performance reporting

The mini-app calls the `std::chrono` library to measure and report the elapsed time for each timestep from different parts of the code in the job output. ReFrame supports the extraction and manipulation of performance data from the program output, as well as a comprehensive way of setting performance thresholds per system and per system partitions. In addition to performance checking, it is possible to print a performance report with the `--performance-report` flag. A typical report for the mini-app with PrgEnv-gnu will look like this:

```
PERFORMANCE REPORT
----------------------------------------------
sphexa_timers_sqpatch_024mpi_001omp_100n_0steps
- daint:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 3.6201 s
      * _Elapsed: 5 s
      * domain_build: 0.0956 s
      * mpi_synchronizeHalos: 0.4567 s
      * BuildTree: 0 s
      * FindNeighbors: 0.3547 s
      * Density: 0.296 s
      * EquationOfState: 0.0024 s
      * IAD: 0.6284 s
      * MomentumEnergyIAD: 1.0914 s
      * Timestep: 0.6009 s
      * UpdateQuantities: 0.0051 s
      * EnergyConservation: 0.0012 s
      * SmoothingLength: 0.0033 s
      * %MomentumEnergyIAD: 30.15 %
      * %Timestep: 16.6 %
      * %mpi_synchronizeHalos: 12.62 %
      * %FindNeighbors: 9.8 %
      * %IAD: 17.36 %
```

This report is generated from the data collected from the job output and processed in the `self.perf_patterns` part of the check. For example, the time spent in the `MomentumEnergyIAD` is extracted with the `seconds_energ` method. Similarly, the percentage of walltime spent in the `MomentumEnergyIAD` is calculated with the *`pctg_MomentumEnergyIAD`* method.

1.4 Summary

We have covered the basic aspects of a ReFrame test. The next section will expand this test with the integration of a list of commonly used performance tools.

CHAPTER

TWO

CONTAINERS

CSCS currently supports the following container runtimes for running HPC workloads on Piz Daint:

- Sarus
- Singularity

2.1 Getting started

2.1.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/notool/

~/reframe.git/bin/reframe \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-r -c ./internal_timers_mpi_containers.py
```

A successful ReFrame output will look like the following:

```
[ReFrame Setup]
 version: 3.1-dev0 (rev: 6adcc347)

[=====] Running 10 check(s)

[-----] started processing compute_singularity_24mpi_2steps (Tool validation)
[ RUN ] compute_singularity_24mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_singularity_24mpi_2steps (Tool validation)

[-----] started processing compute_singularity_48mpi_2steps (Tool validation)
[ RUN ] compute_singularity_48mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_singularity_48mpi_2steps (Tool validation)

[-----] started processing compute_singularity_96mpi_2steps (Tool validation)
```

(continues on next page)

(continued from previous page)

```
[ RUN ] compute_singularity_96mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_singularity_96mpi_2steps (Tool validation)

[-----] started processing compute_singularity_192mpi_2steps (Tool validation)
[ RUN ] compute_singularity_192mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_singularity_192mpi_2steps (Tool validation)

[-----] started processing compute_sarus_24mpi_2steps (Tool validation)
[ RUN ] compute_sarus_24mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_sarus_24mpi_2steps (Tool validation)

[-----] started processing compute_sarus_48mpi_2steps (Tool validation)
[ RUN ] compute_sarus_48mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_sarus_48mpi_2steps (Tool validation)

[-----] started processing compute_sarus_96mpi_2steps (Tool validation)
[ RUN ] compute_sarus_96mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_sarus_96mpi_2steps (Tool validation)

[-----] started processing compute_sarus_192mpi_2steps (Tool validation)
[ RUN ] compute_sarus_192mpi_2steps on daint:gpu using PrgEnv-gnu
[-----] finished processing compute_sarus_192mpi_2steps (Tool validation)

[-----] started processing postproc_containers (MPI_Collect_Logs_Test)
[ RUN ] postproc_containers on daint:gpu using PrgEnv-gnu
[  DEP ] postproc_containers on daint:gpu using PrgEnv-gnu
[------] finished processing postproc_containers (MPI_Collect_Logs_Test)

[------] started processing performance_containers (MPI_Plot_Test)
[ RUN ] performance_containers on daint:gpu using PrgEnv-gnu
[  DEP ] performance_containers on daint:gpu using PrgEnv-gnu
[---] finished processing performance_containers (MPI_Plot_Test)

[----] waiting for spawned checks to finish
[ OK ] ( 1/10) compute_singularity_96mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 2/10) compute_singularity_24mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 3/10) compute_singularity_48mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 4/10) compute_singularity_192mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 5/10) compute_sarus_24mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 6/10) compute_sarus_48mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 7/10) compute_sarus_96mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 8/10) compute_sarus_192mpi_2steps on daint:gpu using PrgEnv-gnu
[ OK ] ( 9/10) postproc_containers on daint:gpu using PrgEnv-gnu
[ OK ] (10/10) performance_containers on daint:gpu using PrgEnv-gnu
[----] all spawned checks have finished

[  PASSED  ] Ran 10 test case(s) from 10 check(s) (0 failure(s))
```

2.1.2 Performance reporting

This test has 3 parts: first, run the code with Singularity, Sarus and Native (i.e no container), then postprocess the jobs output and plot the timings as a result. The report is generated automatically after all the compute jobs have been terminated. The performance data for a couple of steps weak scaling job from 24 to 192 mpi tasks will typically look like this:

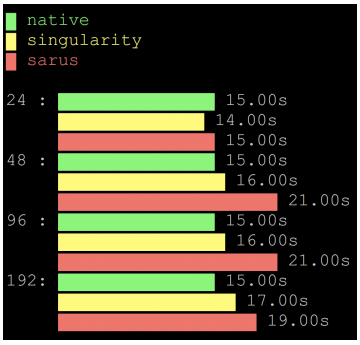

Fig. 1: Weak scaling (launched with cat performance_containers/termgraph.rpt)

Looking into the `classes` shows how to setup and run the code with the containers. Notice that the postprocessing and plotting classes depend on the compute classes.

CHAPTER

THREE

INTEL® TOOLS

3.1 Intel® Inspector

Intel® Inspector is a dynamic memory and threading error checking tool. This page will guide you through writing ReFrame tests to analyze our code with this tool. Looking into the *Class* shows how to setup and run the code with the tool.

3.1.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/intel/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./intel_inspector.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps (Tool␣
↪validation)
[ RUN      ] sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using␣
↪PrgEnv-gnu
[       OK ] sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using␣
↪PrgEnv-gnu
[----------] finished processing sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps␣
↪(Tool validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Several analyses are available.

The `mi1` (memory leak) analysis is triggered by setting the `executable_opts`.

Use `self.post_run` to generate the report with the tool.

3.1.2 Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
-----------------------------------------------------
sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 8.899 s
      ...
      * Memory not deallocated: 1
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the check. The number of (`memory not deallocated`) problems detected by the tool is extracted with the *`inspector_not_deallocated`* method. Looking at the report with the tool shows that the problem comes from a system library (`libpmi.so`) hence we can assume there is no problem with the code.

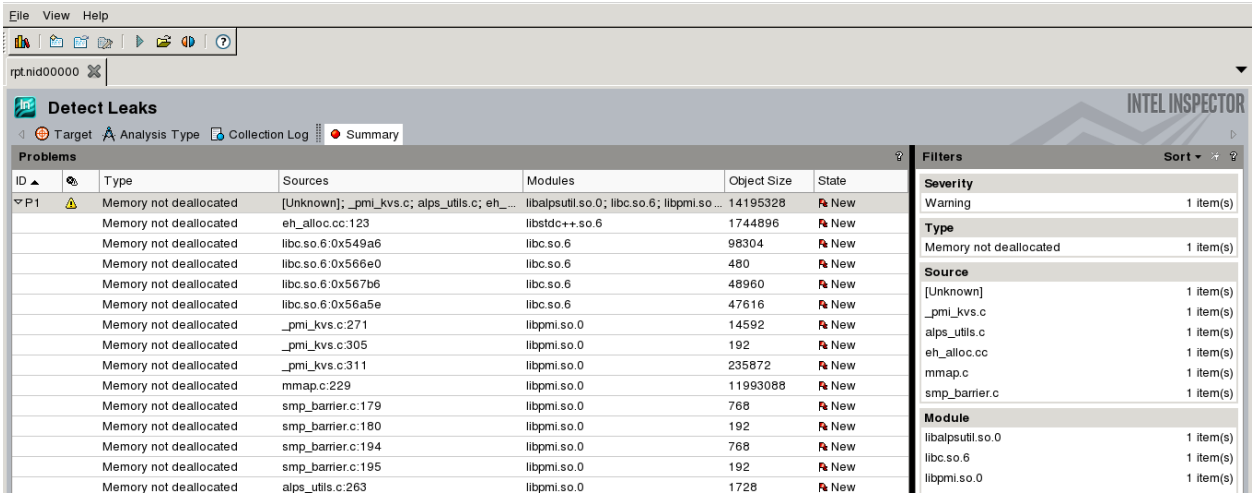

Fig. 1: Intel Inspector (launched with: inspxe-gui rpt.nid00000/)

3.2 Intel® Vtune

Intel® Vtune is Intel's performance profiler for C, C++, Fortran, Assembly and Python.

3.2.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/intel/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-intel \
--performance-report \
--keep-stage-files \
-c ./intel_vtune.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev2 (rev: 6d543136)
Launched on host: daint101

[----------] waiting for spawned checks to finish
[       OK ] sphexa_vtune_sqpatch_024mpi_001omp_100n_1steps on daint:gpu using PrgEnv-
↪intel
[       OK ] sphexa_vtune_sqpatch_048mpi_001omp_125n_1steps on daint:gpu using PrgEnv-
↪intel
[       OK ] sphexa_vtune_sqpatch_096mpi_001omp_157n_1steps on daint:gpu using PrgEnv-
↪intel
[----------] all spawned checks have finished

[  PASSED  ] Ran 3 test case(s) from 3 check(s) (0 failure(s))
```

Several analyses are available:

```
Available analysis types are: ``vtune -h collect``

    .. code-block:: none

      hotspots            <--
      memory-consumption
      uarch-exploration
      memory-access
      threading
      hpc-performance
      system-overview
      graphics-rendering
      io
      fpga-interaction
      gpu-offload
      gpu-hotspots
      throttling
      platform-profiler
```

Looking into the *Class* shows how to setup and run the code with the tool. Notice that this class is a derived class hence `super().__init__()` is required. The `hotspots` analysis is triggered by setting the `executable_opts`.

Use `self.post_run` to generate the report with the tool. Notice that the tool collects performance data per compute node.

3.2.2 Performance reporting

An overview of the performance data for a 4 compute nodes job will typically look like this:

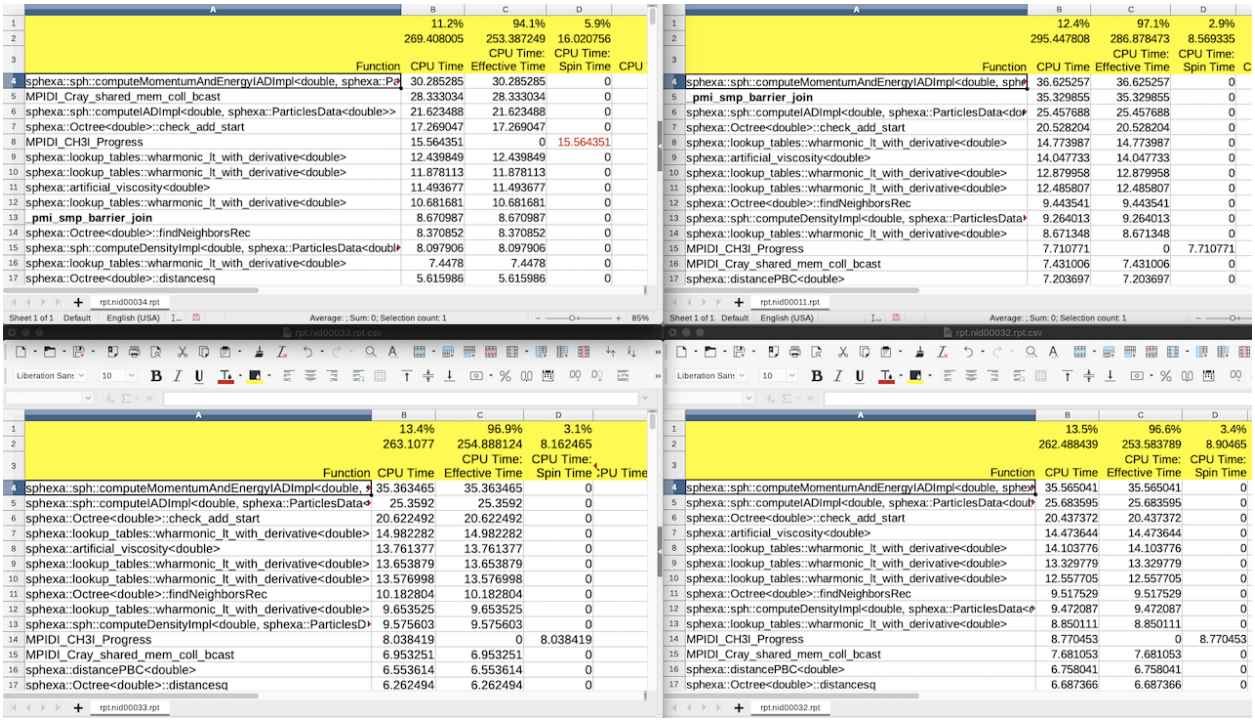

Fig. 2: Intel Vtune (overview)

As a result, a typical output from the `--performance-report` flag will look like this:

```
sphexa_vtune_sqpatch_096mpi_001omp_157n_1steps
   - PrgEnv-intel
      * num_tasks: 96
      * Elapsed: 5.0388 s
      * _Elapsed: 37 s
      * domain_distribute: 0.207 s
      * mpi_synchronizeHalos: 0.5091 s
      * BuildTree: 0 s
      * FindNeighbors: 0.8136 s
      * Density: 0.5672 s
      * EquationOfState: 0.001 s
      * IAD: 0.9999 s
      * MomentumEnergyIAD: 1.5887 s
      * Timestep: 0.1516 s
      * UpdateQuantities: 0.0087 s
      * EnergyConservation: 0.0028 s
      * SmoothingLength: 0.002 s
      * %MomentumEnergyIAD: 31.53 %
```

(continues on next page)

(continued from previous page)

```
      * %Timestep: 3.01 %
      * %mpi_synchronizeHalos: 10.1 %
      * %FindNeighbors: 16.15 %
      * %IAD: 19.84 %
      * vtune_elapsed_min: 7.408 s
      * vtune_elapsed_max: 9.841 s
      * vtune_elapsed_cput: 8.1791 s
      * vtune_elapsed_cput_efft: 7.985 s
      * vtune_elapsed_cput_spint: 0.3246 s
      * vtune_elapsed_cput_spint_mpit: 0.3191 s
      * %vtune_effective_physical_core_utilization: 87.8 %
      * %vtune_effective_logical_core_utilization: 86.6 %
      * vtune_cput_cn0: 196.3 s
      * %vtune_cput_cn0_efft: 96.0 %
      * %vtune_cput_cn0_spint: 4.0 %
      * vtune_cput_cn1: 195.71 s
      * %vtune_cput_cn1_efft: 97.9 %
      * %vtune_cput_cn1_spint: 2.1 %
      * vtune_cput_cn2: 189.77 s
      * %vtune_cput_cn2_efft: 98.1 %
      * %vtune_cput_cn2_spint: 1.9 %
      * vtune_cput_cn3: 148.46 s
      * %vtune_cput_cn3_efft: 97.6 %
      * %vtune_cput_cn3_spint: 2.4 %
```

This report is generated from the data collected from the tool and processed in the `set_vtune_perf_patterns_rpt` method of the *check*.

Looking at the report with the tool gives more insight into the performance of the code:

3.3 Intel® Advisor

Intel® Advisor is a Vectorization Optimization and Thread Prototyping tool:

- Vectorize & thread code for maximum performance
- Easy workflow + data + tips = faster code faster
- Prioritize, Prototype & Predict performance gain.

3.3.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/intel/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
```

(continues on next page)

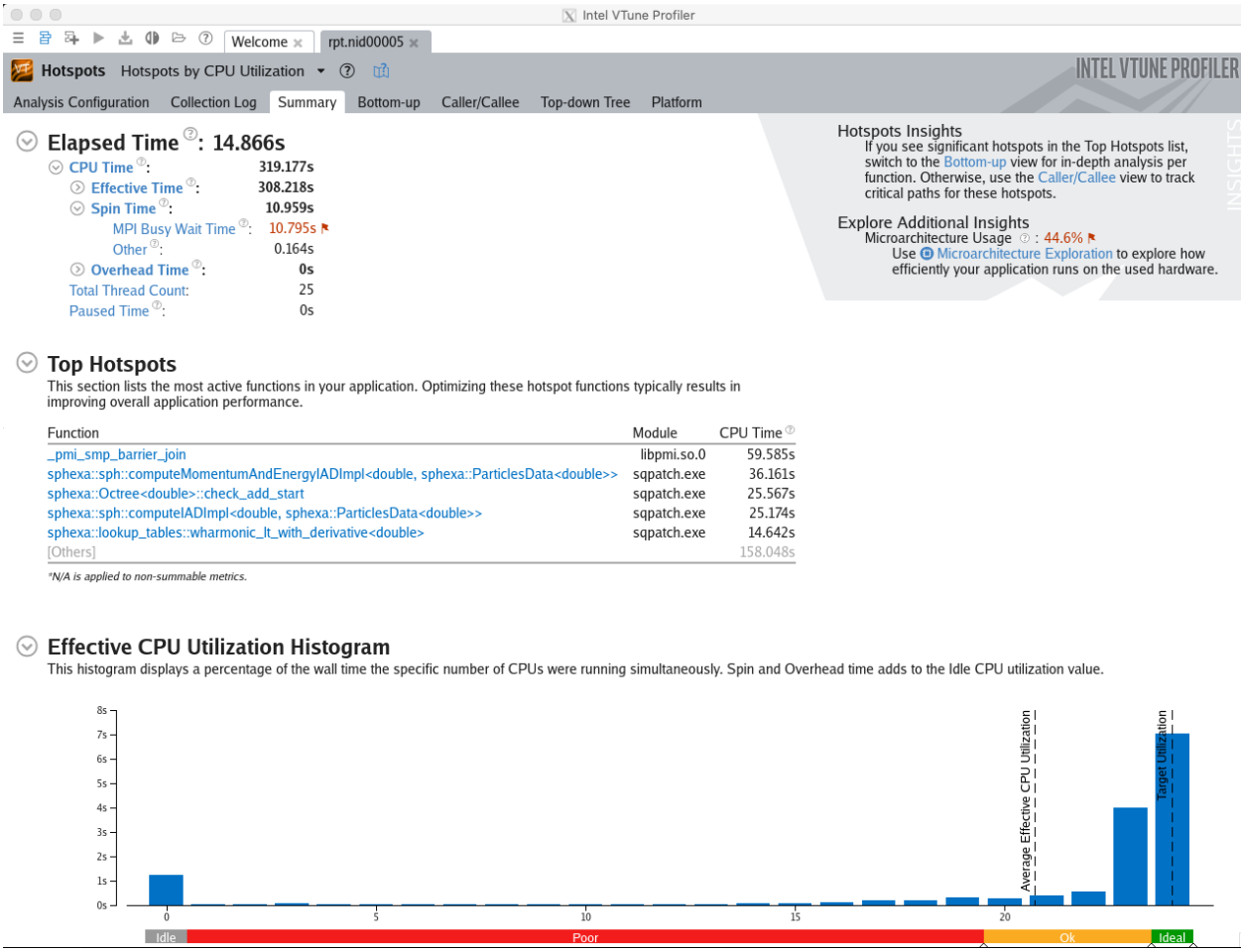

Fig. 3: Intel Vtune Summary view (launched with: vtune-gui rpt.nid00034/rpt.nid00034.vtune)

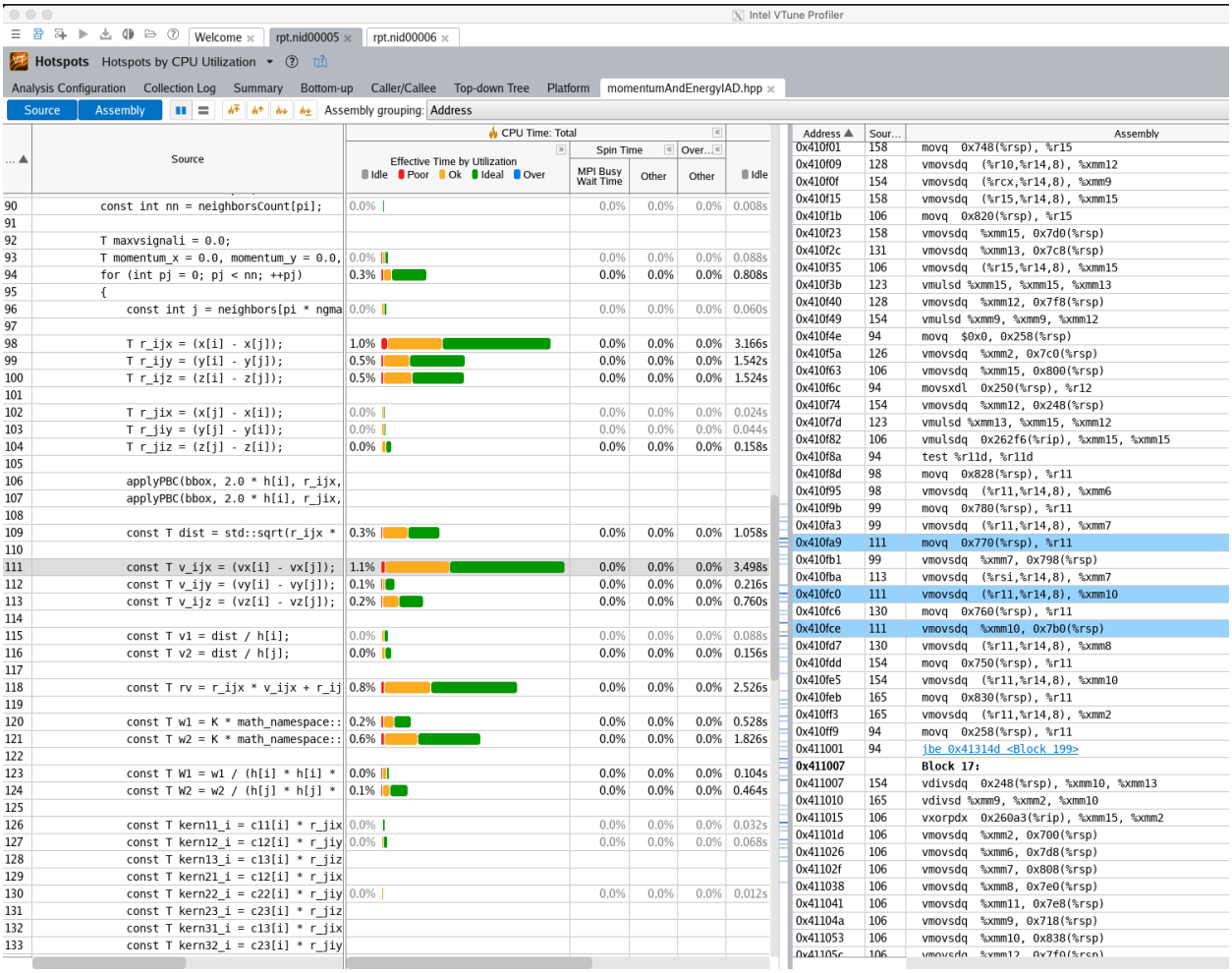

Fig. 4: Intel Vtune Hotspots (Src/Assembly view)

(continued from previous page)

```
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./intel_advisor.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_advisor_sqpatch_024mpi_001omp_100n_0steps (Tool␣
↪validation)
[ RUN      ] sphexa_advisor_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
[       OK ] sphexa_advisor_sqpatch_024mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
[----------] finished processing sphexa_advisor_sqpatch_024mpi_001omp_100n_0steps (Tool␣
↪validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Several analyses are available:

Looking into the *Class* shows how to setup and run the code with the tool. The `survey` analysis is triggered by setting the `executable_opts`.

Use `self.post_run` to generate the report with the tool.

3.3.2 Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
------------------------------------------------------
sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 3.6147 s
      ...
      * advisor_elapsed: 2.13 s
      * advisor_loop1_line: 94 (momentumAndEnergyIAD.hpp)
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the *check*. This information (elapsed walltime, source filename and line number) is extracted for mpi rank 0 with the *advisor_elapsed*, *advisor_loop1_filename*, and *advisor_loop1_line* methods. Looking at the report with the tool gives more insight into the performance of the code:

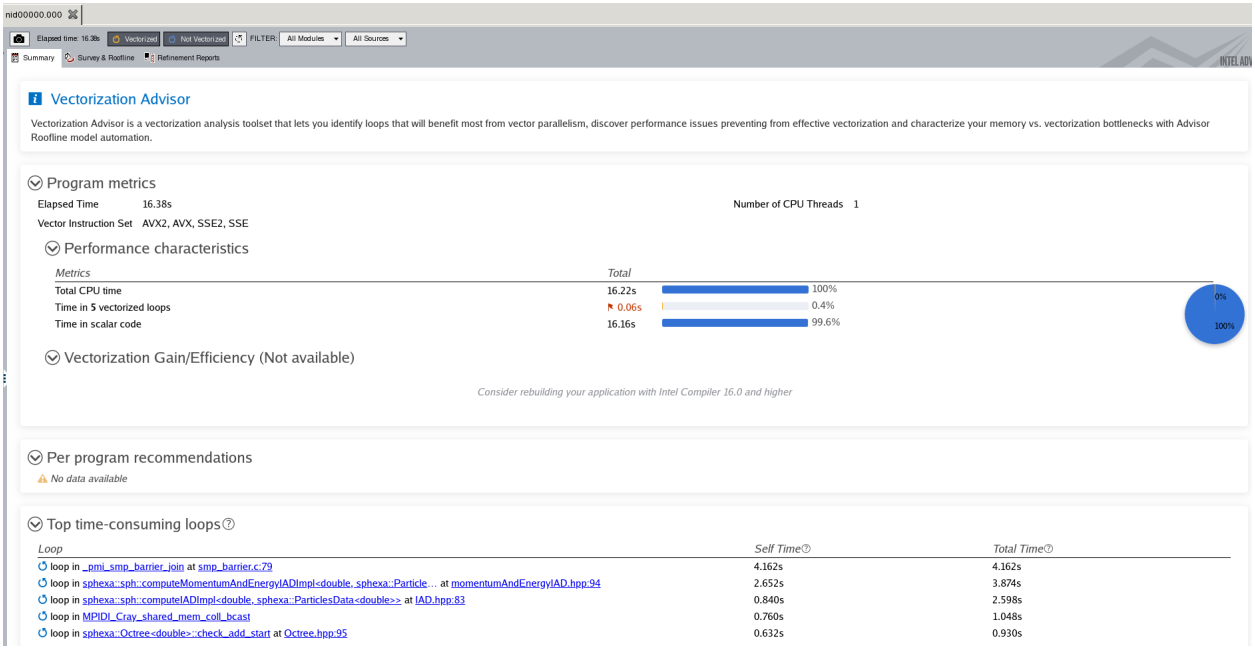

Fig. 5: Intel Advisor (launched with: advixe-gui rpt/rpt.advixeproj)

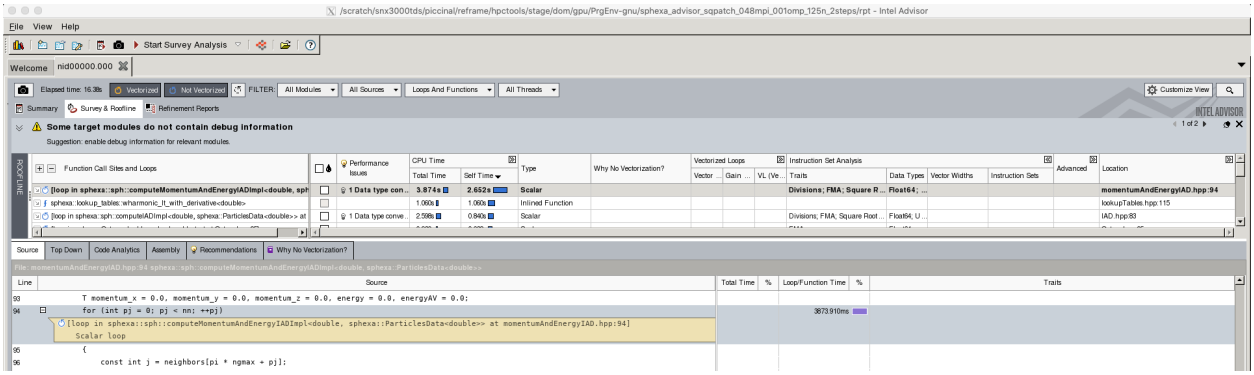

Fig. 6: Intel Advisor (survey analysis)

CHAPTER

FOUR

VI-HPS TOOLS

The VI-HPS Institute (Virtual Institute for High Productivity Supercomputing) provides tools that can assist developers of simulation codes to address their needs in performance analysis:

- Score-P is a highly scalable instrumentation and measurement infrastructure for profiling, event tracing, and online analysis. It supports a wide range of HPC platforms and programming models. Score-P provides core measurement services for a range of specialized analysis tools, such as Vampir, Scalasca and others.
- Scalasca supports the performance optimization of parallel programs with a collection of scalable trace-based tools for in-depth analyses of concurrent behavior. The analysis identifies potential performance bottlenecks - in particular those concerning communication and synchronization - and offers guidance in exploring their causes.
- Vampir is a performance visualizer that allows to quickly study the program runtime behavior at a fine level of details. This includes the display of detailed performance event recordings over time in timelines and aggregated profiles. Interactive navigation and zooming are the key features of the tool, which help to quickly identify inefficient or faulty parts of a program.

For further information, please check:

- Training material.
- VI-HPS Tools Guide.

4.1 Score-P

4.1.1 Profiling

Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/scorep/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./scorep_sampling_profiling.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_scorepS+P_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)
[ RUN      ] sphexa_scorepS+P_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[       OK ] sphexa_scorepS+P_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[----------] finished processing sphexa_scorepS+P_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool.

Set `self.build_system.cxx` to instrument the code and set the SCOREP runtime variables with `self.variables` to trigger the (sampling based) *profiling* analysis. Use `self.post_run` to generate the tool's report.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------------
sphexa_scorepS+P_sqpatch_024mpi_001omp_100n_0steps_1000000cycles
- daint:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 3.8245 s
      * _Elapsed: 6 s
      * domain_distribute: 0.1039 s
      * mpi_synchronizeHalos: 0.4705 s
      * BuildTree: 0 s
      * FindNeighbors: 0.384 s
      * Density: 0.3126 s
      * EquationOfState: 0.0047 s
      * IAD: 0.6437 s
      * MomentumEnergyIAD: 1.1131 s
      * Timestep: 0.6459 s
      * UpdateQuantities: 0.0078 s
      * EnergyConservation: 0.0024 s
      * SmoothingLength: 0.0052 s
      * %MomentumEnergyIAD: 29.1 %
      * %Timestep: 16.89 %
      * %mpi_synchronizeHalos: 12.3 %
      * %FindNeighbors: 10.04 %
      * %IAD: 16.83 %
      * scorep_elapsed: 4.8408 s
      * %scorep_USR: 66.1 %
      * %scorep_MPI: 21.9 %
```

(continues on next page)

(continued from previous page)

```
        * scorep_top1: 24.2 % (void sphexa::sph::computeMomentumAndEnergyIADImpl)
        * %scorep_Energy_exclusive: 24.216 %
        * %scorep_Energy_inclusive: 24.216 %
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the `Class`. For example, the information about the top1 function is extracted with the `scorep_top1_pct` method. Looking at the report with the tool gives more insight into the performance of the code:

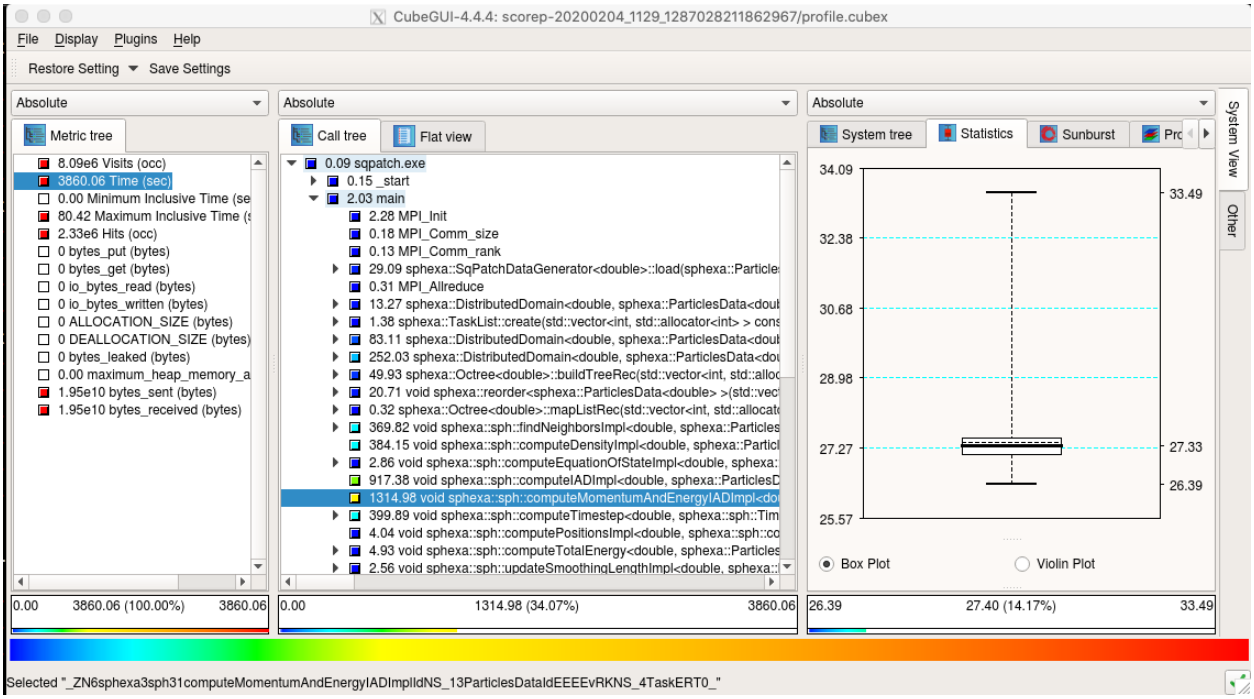

Fig. 1: Score-P Cube (launched with: cube scorep-/profile.cubex)

4.1.2 Tracing

Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/scorep/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./scorep_sampling_tracing.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_scorepS+T_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)
[ RUN      ] sphexa_scorepS+T_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[       OK ] sphexa_scorepS+T_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[----------] finished processing sphexa_scorepS+T_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool.

Set `self.build_system.cxx` to instrument the code and set the SCOREP runtime variables with `self.variables` to trigger the (sampling based) *tracing* analysis.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------------
sphexa_scorepS+T_sqpatch_024mpi_001omp_100n_4steps_5000000cycles
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 20.5236 s
      * _Elapsed: 27 s
      * domain_distribute: 0.4484 s
      * mpi_synchronizeHalos: 2.4355 s
      * BuildTree: 0 s
      * FindNeighbors: 1.875 s
      * Density: 1.8138 s
      * EquationOfState: 0.019 s
      * IAD: 3.7302 s
      * MomentumEnergyIAD: 6.1363 s
      * Timestep: 3.5853 s
      * UpdateQuantities: 0.0272 s
      * EnergyConservation: 0.0087 s
      * SmoothingLength: 0.0232 s
      * %MomentumEnergyIAD: 29.9 %
      * %Timestep: 17.47 %
      * %mpi_synchronizeHalos: 11.87 %
      * %FindNeighbors: 9.14 %
      * %IAD: 18.18 %
      * max_ipc_rk0: 1.297827 ins/cyc
      * max_rumaxrss_rk0: 127448 kilobytes
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the `Class`. For example, the information about the `ipc` for rank 0 is extracted with the *`ipc_rk0`* method. Looking at the

report with the tool gives more insight into the performance of the code:

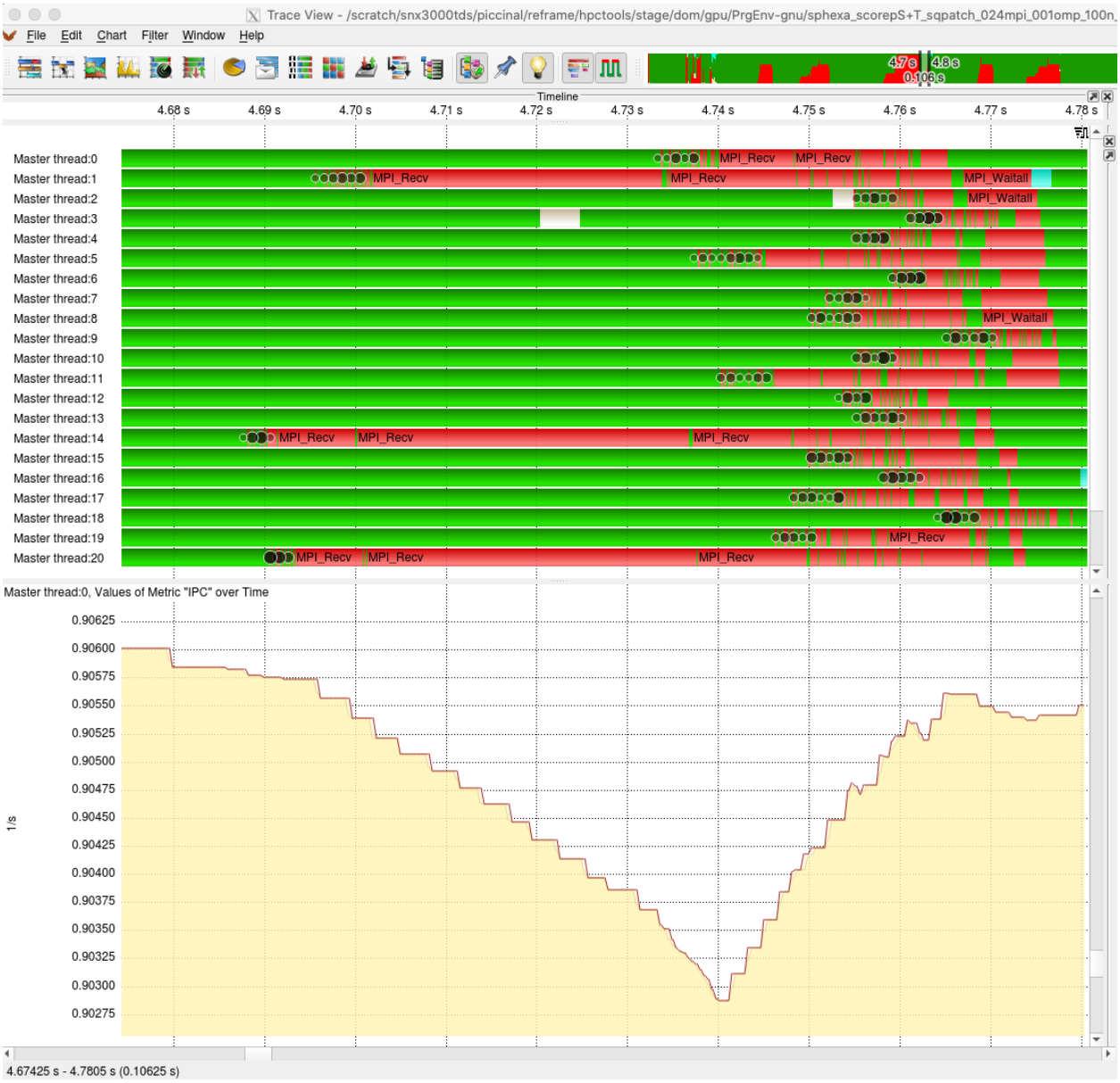

Fig. 2: Score-P Vampir (launched with: vampir scorep-/traces.otf2)

4.2 Scalasca

4.2.1 Profiling

Running the test

The test can be run from the command-line:

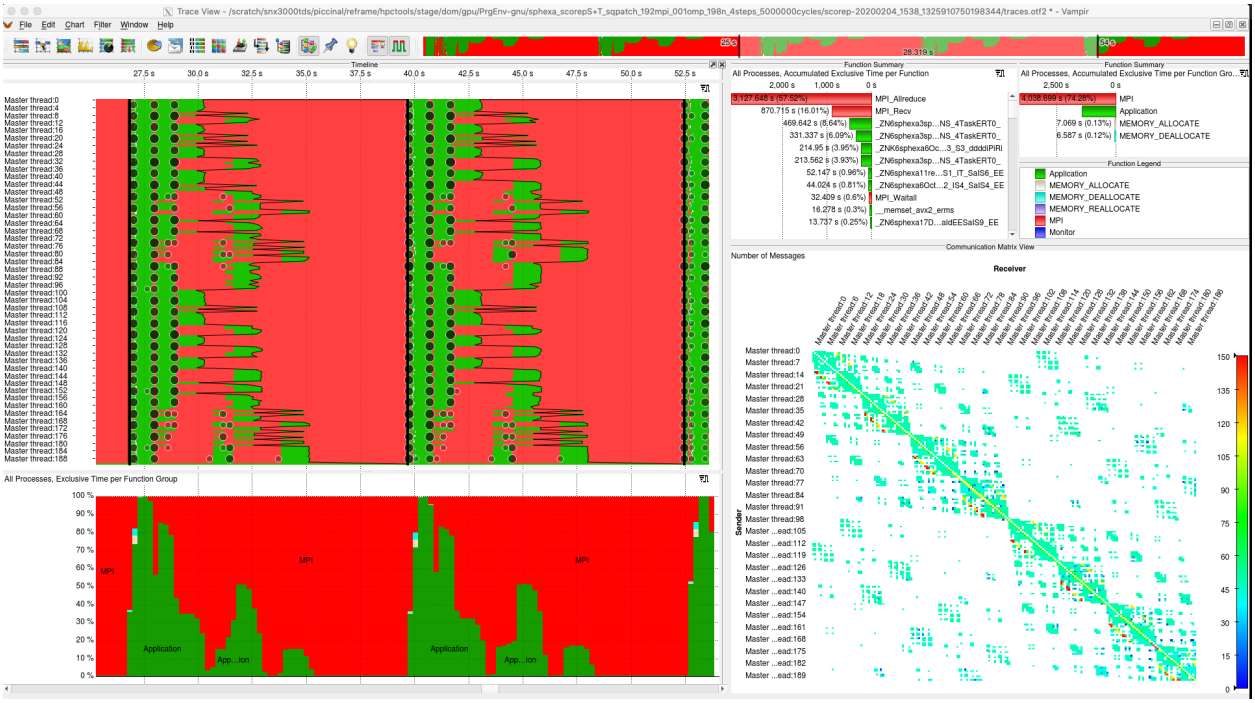

Fig. 3: Score-P Vampir (communication matrix)

```
module load reframe
cd hpctools.git/reframechecks/scalasca

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./scalasca_sampling_profiling.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)
[ RUN      ] sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[       OK ] sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[----------] finished processing sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the *Class* shows how to setup and run the code with the tool.

Set `self.build_system.cxx` to instrument the code and set the SCOREP runtime variables with `self.variables` to trigger the (sampling based) *profiling* analysis. Use `self.post_run` to generate the tool's report.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
--------------------------------------------------------------------------------
sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_4steps_5000000cycles
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 20.4549 s
      * _Elapsed: 38 s
      * domain_distribute: 0.4089 s
      * mpi_synchronizeHalos: 2.4644 s
      * BuildTree: 0 s
      * FindNeighbors: 1.8787 s
      * Density: 1.8009 s
      * EquationOfState: 0.0174 s
      * IAD: 3.726 s
      * MomentumEnergyIAD: 6.1141 s
      * Timestep: 3.5887 s
      * UpdateQuantities: 0.0424 s
      * EnergyConservation: 0.0177 s
      * SmoothingLength: 0.017 s
      * %MomentumEnergyIAD: 29.89 %
      * %Timestep: 17.54 %
      * %mpi_synchronizeHalos: 12.05 %
      * %FindNeighbors: 9.18 %
      * %IAD: 18.22 %
      * scorep_elapsed: 21.4262 s
      * %scorep_USR: 71.0 %
      * %scorep_MPI: 23.3 %
      * scorep_top1: 30.1 % (void sphexa::sph::computeMomentumAndEnergyIADImpl)
      * %scorep_Energy_exclusive: 30.112 %
      * %scorep_Energy_inclusive: 30.112 %
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the *Class*. For example, the information about the top1 function is extracted with the `scorep_top1_pct` method. Notice that the same sanity functions used with Score-P can be used with Scalasca too. Looking at the report with the tool gives more insight into the performance of the code:

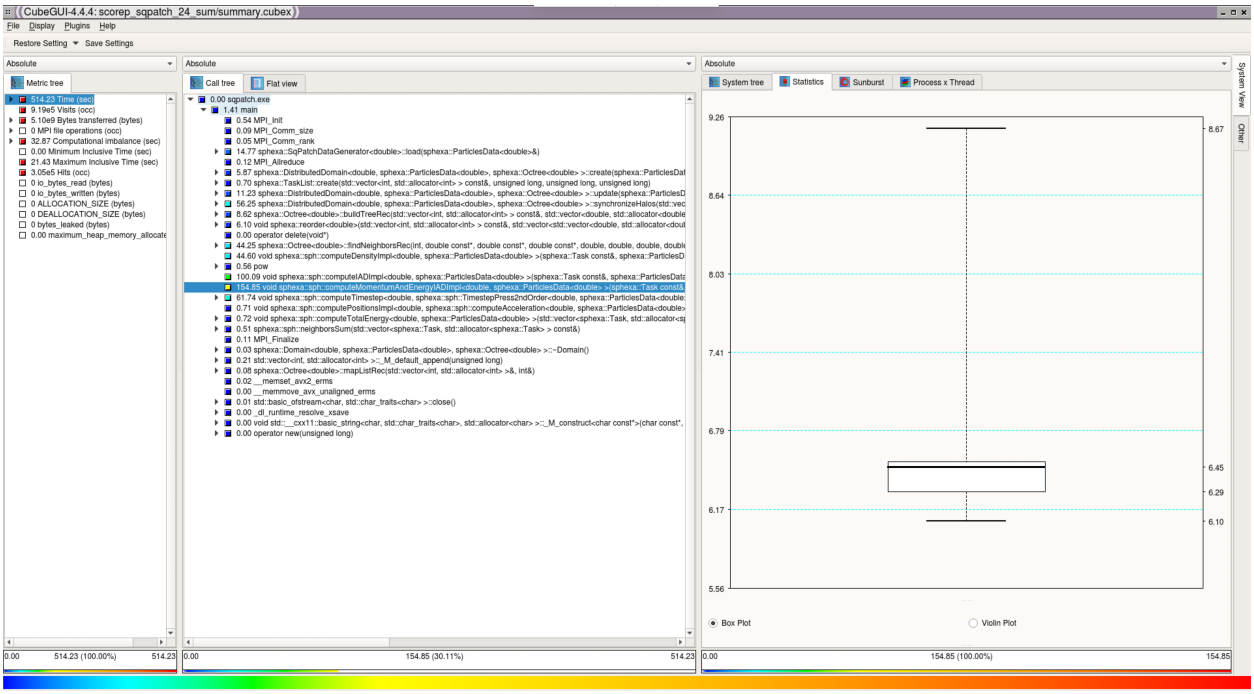

Fig. 4: Scalasca Cube (launched with: cube scorep_sqpatch_24_sum/profile.cubex)

4.2.2 Tracing

Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/scalasca

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./scalasca_sampling_tracing.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)
[ RUN      ] sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
[       OK ] sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_10steps_1000000cycles on␣
↪daint:gpu using PrgEnv-gnu
```

(continues on next page)

(continued from previous page)

```
[----------] finished processing sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_10steps_
↪1000000cycles (Tool validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the *Class* shows how to setup and run the code with the tool.

Set `self.build_system.cxx` to instrument the code and set the SCOREP runtime variables with `self.variables` to trigger the (sampling based) *tracing* analysis. Use `self.post_run` to generate the tool's report.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------
sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_4steps_5000000cycles
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 20.5242 s
      * _Elapsed: 28 s
      * domain_distribute: 0.4712 s
      * mpi_synchronizeHalos: 2.4623 s
      * BuildTree: 0 s
      * FindNeighbors: 1.8752 s
      * Density: 1.8066 s
      * EquationOfState: 0.0174 s
      * IAD: 3.7259 s
      * MomentumEnergyIAD: 6.1355 s
      * Timestep: 3.572 s
      * UpdateQuantities: 0.0273 s
      * EnergyConservation: 0.0079 s
      * SmoothingLength: 0.017 s
      * %MomentumEnergyIAD: 29.89 %
      * %Timestep: 17.4 %
      * %mpi_synchronizeHalos: 12.0 %
      * %FindNeighbors: 9.14 %
      * %IAD: 18.15 %
      * mpi_latesender: 2090 count
      * mpi_latesender_wo: 19 count
      * mpi_latereceiver: 336 count
      * mpi_wait_nxn: 1977 count
      * max_ipc_rk0: 1.294516 ins/cyc
      * max_rumaxrss_rk0: 127932 kilobytes
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the *Class*. For example, the information about the numer of MPI Late Sender is extracted with the `rpt_trace_stats_d` method (latesender). Cube help describes this metric as the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv or MPI_Wait) that is posted earlier than the corresponding send operation. Looking at the report with the tools (Cube and Vampir) gives more insight into the performance of the code:

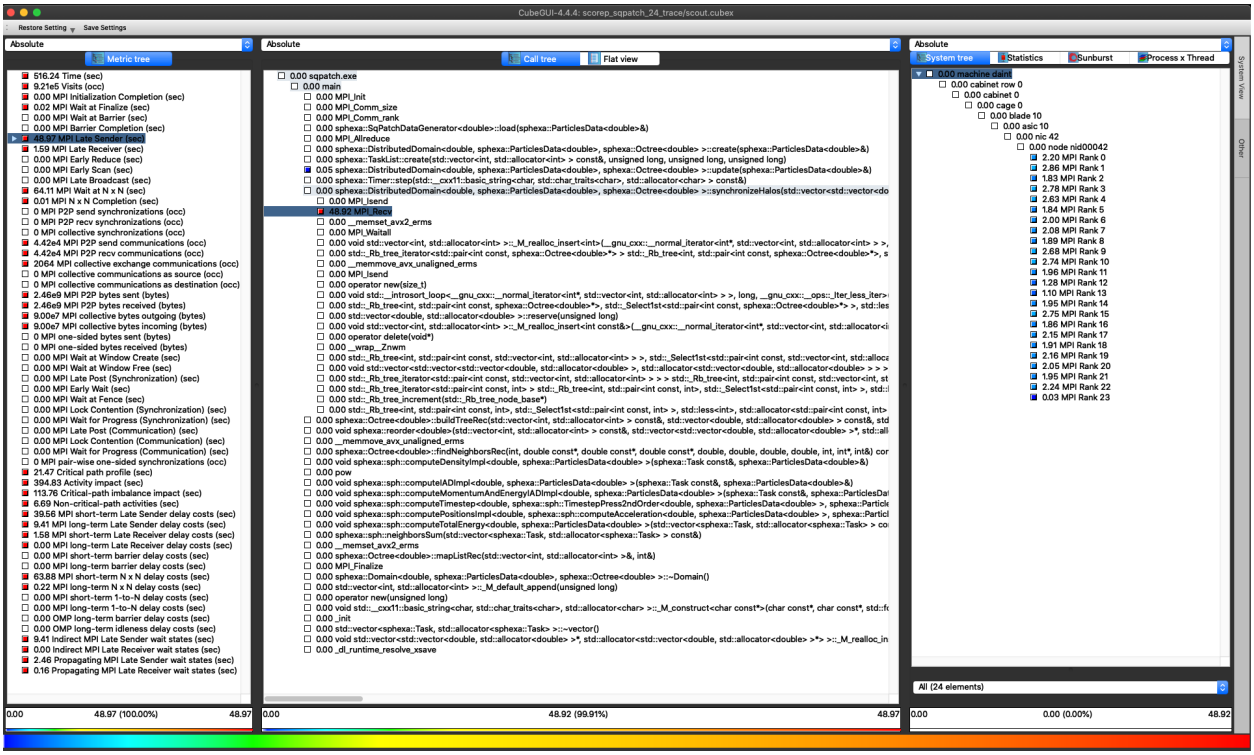

Fig. 5: Scalasca Cube (launched with: cube scorep_sqpatch_24_trace/scout.cubex)

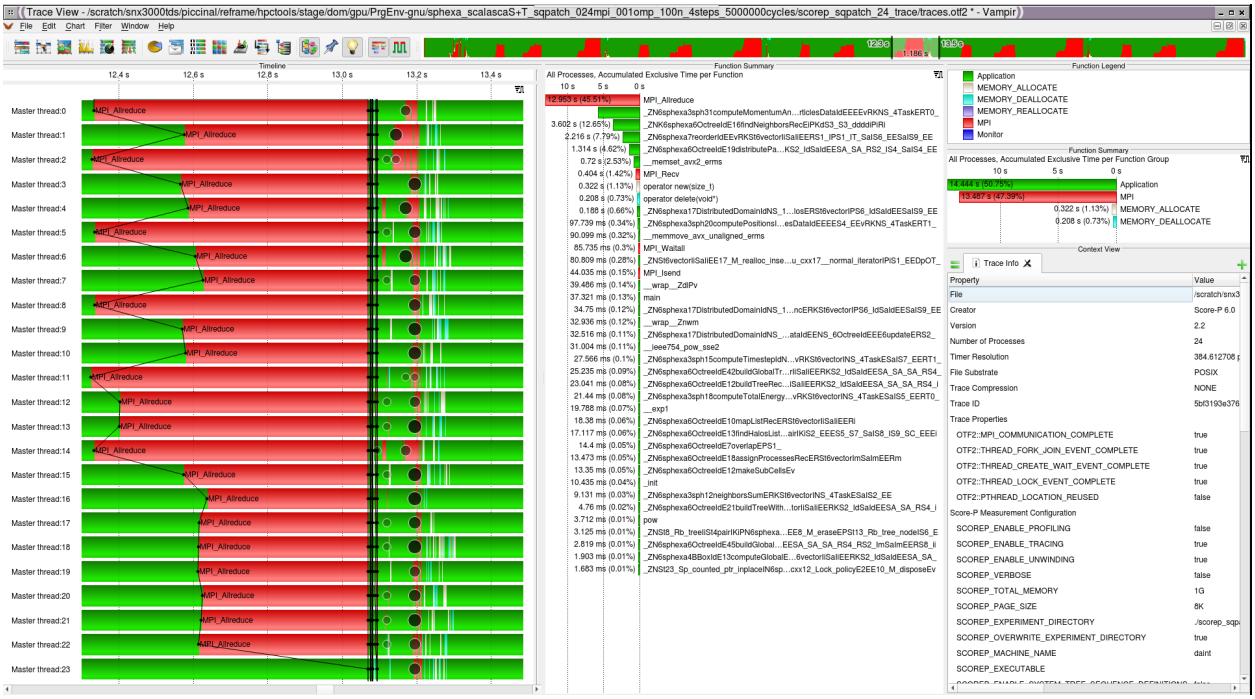

Fig. 6: Scalasca Vampir (launched with: vampir scorep_sqpatch_24_trace/traces.otf2)

CHAPTER

FIVE

EXTRAE

Extrae is the core instrumentation package developed by the Performance Tools group at BSC. Extrae is capable of instrumenting applications based on MPI, OpenMP, pthreads, CUDA1, OpenCL1, and StarSs1 using different instrumentation approaches. The information gathered by Extrae typically includes timestamped events of runtime calls, performance counters and source code references. Besides, Extrae provides its own API to allow the user to manually instrument his or her application.

5.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/extrae/

~/reframe.git/bin/reframe \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--report-file $HOME/rpt.json \
--keep-stage-files \
-c ./extrae.py
```

A successful ReFrame output will look like the following:

```
Launched on host: daint101

[----------] started processing sphexa_extrae_sqpatch_024mpi_001omp_100n_1steps (Tool␣
↪validation)
[ RUN      ] sphexa_extrae_sqpatch_024mpi_001omp_100n_1steps on daint:gpu using PrgEnv-
↪gnu
[       OK ] sphexa_extrae_sqpatch_024mpi_001omp_100n_1steps on daint:gpu using PrgEnv-
↪gnu
[----------] finished processing sphexa_extrae_sqpatch_024mpi_001omp_100n_1steps (Tool␣
↪validation)

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool.

Because the tool relies on `LD_PRELOAD` to instrument the executable, it is required to set a list of shell commands to execute before launching the job: `self.pre_run` will create a wrapper runscript that will launch the code together with a custom configuration xml file.

5.2 Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
PERFORMANCE REPORT
-------------------------------------------------------------------------------
sphexa_extrae_sqpatch_024mpi_001omp_100n_1steps
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 7.743 s
      * num_comms_0-10B: 2704
      * num_comms_10B-100B: 0
      * num_comms_100B-1KB: 0
      * num_comms_1KB-10KB: 434
      * num_comms_10KB-100KB: 15206
      * num_comms_100KB-1MB: 2848
      * num_comms_1MB-10MB: 0
      * num_comms_10MB: 0
      * %_of_bytes_sent_0-10B: 0.0 %
      * %_of_bytes_sent_10B-100B: 0.0 %
      * %_of_bytes_sent_100B-1KB: 0.0 %
      * %_of_bytes_sent_1KB-10KB: 0.24 %
      * %_of_bytes_sent_10KB-100KB: 57.83 %
      * %_of_bytes_sent_100KB-1MB: 41.93 %
      * %_of_bytes_sent_1MB-10MB: 0.0 %
      * %_of_bytes_sent_10MB: 0.0 %
```

This report is generated from the data collected from the tool with `self.post_run` and processed in the `self.perf_patterns` part of the `Class`. For example, the information about the MPI communications for which the size of the messages is between 10KB and 100KB (`%_of_bytes_sent_10KB-100KB`) is extracted with the *rpt_mpistats* method (hundKB_b). Looking at the report with the tool gives more insight into the performance of the code:

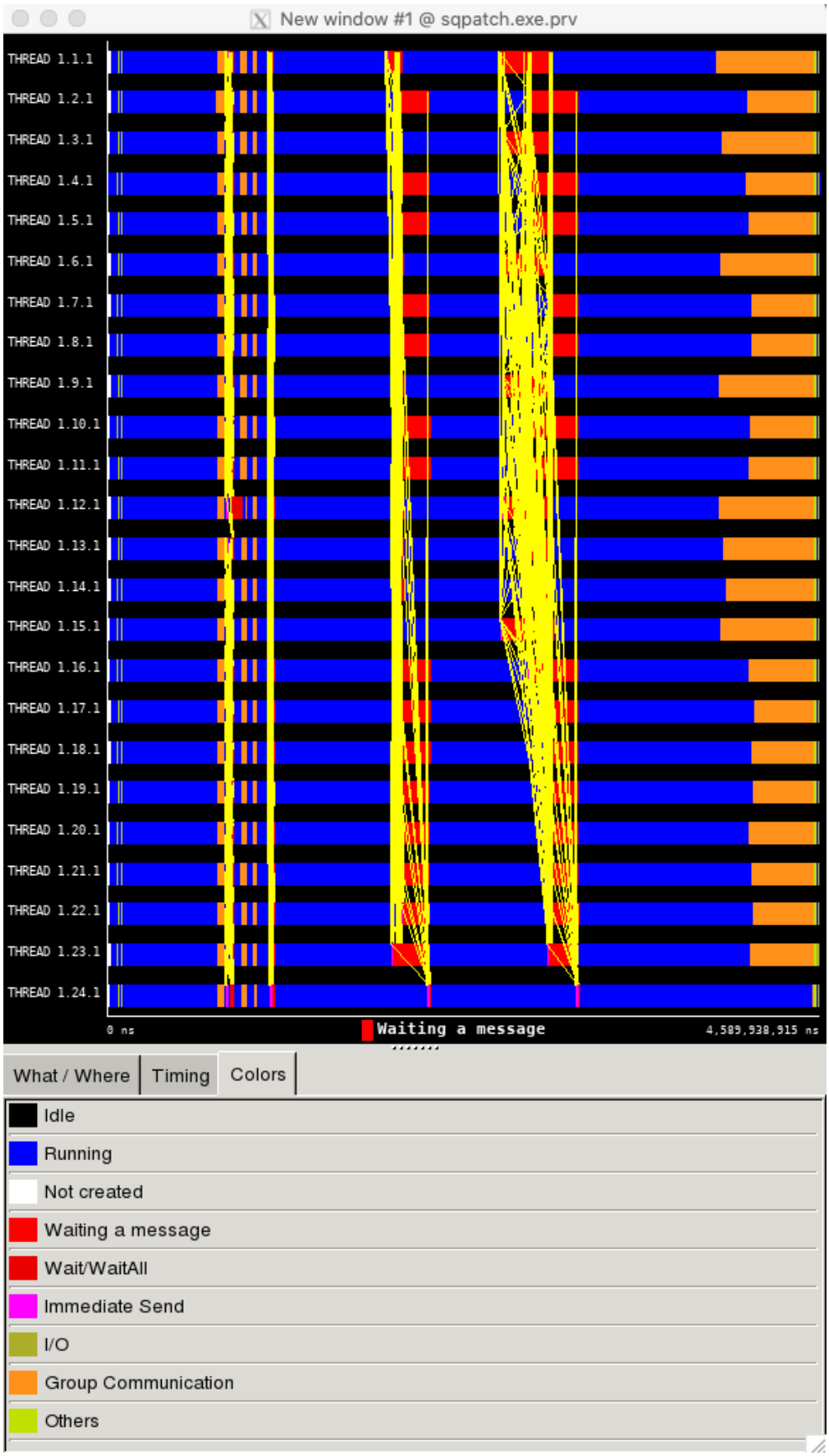

Fig. 1: Extrae Paraver (launched with: wxparaver sqpatch.exe.prv)

CHAPTER

SIX

MPIP

mpiP is LLNL's light-weight MPI profiler.

6.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/mpip/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./mpip.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev2 (rev: 6d543136)
Launched on host: daint101

[----------] waiting for spawned checks to finish
[       OK ] sphexa_mpiP_sqpatch_024mpi_001omp_100n_3steps on daint:gpu using PrgEnv-gnu
[       OK ] sphexa_mpiP_sqpatch_048mpi_001omp_125n_3steps on daint:gpu using PrgEnv-gnu
[       OK ] sphexa_mpiP_sqpatch_096mpi_001omp_157n_3steps on daint:gpu using PrgEnv-gnu
[----------] all spawned checks have finished

[  PASSED  ] Ran 3 test case(s) from 3 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool. Notice that this class is a derived class hence `super().__init__()` is required.

The performance report is generated automatically at the end of the job.

6.2 Performance reporting

An overview of the performance data for a job with 3 mpi ranks will typically look like this:

mpiP							
MPI Rank	App (sec)	MPI (sec)	MPI% _of rank		Non MPI (sec)	MPI% _of wallt	Non MPI% _of wallt
0	8.08	0.414	5.13		7.67	1.7%	31.7%
1	8.06	0.206	2.55		7.85	0.9%	32.4%
2	8.07	5.65	70		2.42	23.3%	10.0%
*	**24.2**	**6.3**	**25.89**		**17.94**	25.9%	74.1%
	100%	25.9%			74.1%		

Fig. 1: mpiP (overview)

As a result, a typical output from the `--performance-report` flag will look like this:

```
sphexa_mpiP_sqpatch_096mpi_001omp_157n_3steps
   - PrgEnv-gnu
      * num_tasks: 96
      * Elapsed: 16.0431 s
      * _Elapsed: 19 s
      * domain_distribute: 0.3262 s
      * mpi_synchronizeHalos: 0.8793 s
      * BuildTree: 0 s
      * FindNeighbors: 1.557 s
      * Density: 1.5117 s
      * EquationOfState: 0.0132 s
      * IAD: 3.6159 s
      * MomentumEnergyIAD: 5.2786 s
      * Timestep: 2.5658 s
      * UpdateQuantities: 0.0202 s
      * EnergyConservation: 0.0092 s
      * SmoothingLength: 0.0131 s
      * %MomentumEnergyIAD: 32.9 %
      * %Timestep: 15.99 %
      * %mpi_synchronizeHalos: 5.48 %
      * %FindNeighbors: 9.71 %
      * %IAD: 22.54 %
      * mpip_avg_app_time: 16.88 s
      * mpip_avg_mpi_time: 3.59 s
      * %mpip_avg_mpi_time: 21.27 %
      * %mpip_avg_mpi_time_max: 95.86 %
      * %mpip_avg_mpi_time_min: 15.75 %
      * %mpip_avg_non_mpi_time: 78.73 %
```

This report is generated from the data collected from the tool and processed in the *set_mpip_perf_patterns* method of the *MpipBaseTest* class.

This tool outputs text performance report files only.

CHAPTER

SEVEN

CRAYPAT

CrayPAT (Cray Performance Measurement and Analysis toolset) is Cray's performance analysis tool.

CrayPAT provides detailed information about application performance. It can be used for profiling, tracing and hardware performance counter based analysis. It also provides access to a wide variety of performance experiments that measure how an executable program consumes resources while it is running, as well as several different user interfaces that provide access to the experiment and reporting functions.

CrayPAT consists of the following main components

- pat_run - a simplified, easy-to-use version of CrayPAT for dynamically-linked executables,
- perftools-lite - an easy-to-use version of CrayPAT,
- perftools - CrayPAT, for advanced users:
  - pat_build - used to instrument the program to be analyzed,
  - pat_report - a standalone text report generator that can be use to further explore the data generated by instrumented program execution,
- Apprentice2 - a graphical analysis tool that can be used, in addition to pat_report to further explore and visualize the data generated by instrumented program execution.
- Reveal - Reveal helps to identify top time consuming loops, with compiler feedback on dependency and vectorization.
- pat_view - a graphical analysis tool that can be used to view CrayPat data. pat_view takes as input several ap2 data sets and creates either a graph or the raw data of the scaling information for the input data.

7.1 pat_run

7.1.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/perftools/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
```

(continues on next page)

(continued from previous page)

```
--performance-report \
--keep-stage-files \
-c ./patrun.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev2 (rev: 6d543136)
Launched on host: daint101

[----------] waiting for spawned checks to finish
[       OK ] sphexa_patrun_sqpatch_024mpi_001omp_100n_4steps on daint:gpu using PrgEnv-
↪gnu
[       OK ] sphexa_patrun_sqpatch_048mpi_001omp_125n_4steps on daint:gpu using PrgEnv-
↪gnu
[       OK ] sphexa_patrun_sqpatch_096mpi_001omp_157n_4steps on daint:gpu using PrgEnv-
↪gnu
[----------] all spawned checks have finished

[  PASSED  ] Ran 3 test case(s) from 3 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool. Notice that this class is a derived class hence `super().__init__()` is required.

The performance report is generated automatically at the end of the job.

7.1.2 Performance reporting

An overview of the performance data for a job with 96 mpi ranks will typically look like this:

```
Table 10:  Wall Clock Time, Memory High Water Mark

   Process |   Process | PE=[mmm]
      Time |     HiMem |
           | (MiBytes) |

 18.744450 |      76.8 | Total
|-------------------------------
| 18.755150 |      60.1 | pe.12
| 18.744323 |      99.6 | pe.20
| 18.721262 |      53.8 | pe.93
|===============================
Table 1:  Profile by Function

  Samp% |    Samp |    Imb. |  Imb. | Group
        |         |    Samp | Samp% |  Function
        |         |         |       |   PE=HIDE

 100.0% | 1,858.4 |      -- |    -- | Total
|-------------------------------------------------------------------------------
|  84.5% | 1,570.2 |      -- |    -- | USER
||------------------------------------------------------------------------------
||  34.7% |   645.1 |    40.9 |  6.0% | sphexa::sph::computeMomentumAndEnergyIADImpl<>
```

(continues on next page)

(continued from previous page)

```
||  23.8% |   442.9 |    26.1 |  5.6% | sphexa::sph::computeIADImpl<>
||  10.7% |   198.1 |    14.9 |  7.1% | sphexa::Octree<>::findNeighborsRec
||  10.3% |   191.0 |    12.0 |  6.0% | sphexa::sph::computeDensityImpl<>
||   1.5% |    28.2 |    16.8 | 37.8% | sphexa::reorder<>
||   1.1% |    20.2 |     8.8 | 30.6% | sphexa::Octree<>::buildTreeRec
||   1.1% |    19.9 |    11.1 | 36.2% | main
||=========================================================================
|  11.1% |   205.9 |      -- |    -- | MPI
||-------------------------------------------------------------------------
||   6.6% |   122.2 | 1,664.8 | 94.1% | MPI_Allreduce
||   3.5% |    64.7 |   676.3 | 92.2% | MPI_Recv
||=========================================================================
|   4.2% |    78.3 |      -- |    -- | ETC
||-------------------------------------------------------------------------
||   2.2% |    40.7 |     9.3 | 18.8% | __sin_avx
||   1.1% |    19.9 |    14.1 | 42.0% | __memset_avx2_erms

Table 8:  Program energy and power usage (from Cray PM)

   Node |     Node |   Process | Node Id
 Energy |    Power |      Time |  PE=HIDE
    (J) |      (W) |           |

 13,096 |  698.660 | 18.744450 | Total
```

As a result, a typical output from the `--performance-report` flag will look like this:

```
      * patrun_wallt_max: 18.7552 s
      * patrun_wallt_avg: 18.7445 s
      * patrun_wallt_min: 18.7213 s
      * patrun_mem_max: 60.1 MiBytes
      * patrun_mem_min: 53.8 MiBytes
      * patrun_memory_traffic_global: 53.95 GB
      * patrun_memory_traffic_local: 53.95 GB
      * %patrun_memory_traffic_peak: 4.2 %
      * patrun_memory_traffic: 2.15 GB
      * patrun_ipc: 0.64
      * %patrun_stallcycles: 58.0 %
      * %patrun_user: 84.7 % (slow: 1677.0 smp [pe14] / mean:1570.2 median:1630.0 /␣
↪fast:26.0 [pe95])
      * %patrun_mpi: 11.1 % (slow: 1793.0 smp [pe94] / mean:205.9 median:146.0 / fast:91.
↪0 [pe56])
      * %patrun_etc: 4.2 % (slow: 97.0 smp [pe63] / mean:78.3 median:78.5 / fast:38.0␣
↪[pe93])
      * %patrun_total: 100.0 % (slow: 1862.0 smp [pe92] / mean:1854.4 median:1854.0 /␣
↪fast:1835.0 [pe5])
      * %patrun_user_slowest: 90.5 % (pe.14)
      * %patrun_mpi_slowest: 5.6 % (pe.14)
      * %patrun_etc_slowest: 3.9 % (pe.14)
      * %patrun_user_fastest: 1.4 % (pe.95)
      * %patrun_mpi_fastest: 96.3 % (pe.95)
      * %patrun_etc_fastest: 2.3 % (pe.95)
```

(continues on next page)

(continued from previous page)

```
      * %patrun_avg_usr_reported: 84.5 %
      * %patrun_avg_mpi_reported: 11.1 %
      * %patrun_avg_etc_reported: 4.4 %
      * %patrun_hotspot1: 34.7 % (sphexa::sph::computeMomentumAndEnergyIADImpl<>)
      * %patrun_mpi_h1: 6.6 % (MPI_Allreduce)
      * %patrun_mpi_h1_imb: 94.1 % (MPI_Allreduce)
      * patrun_avg_energy: 3274.0 J
      * patrun_avg_power: 174.665 W
```

This report is generated from the performance data collected from the tool and processed in the *patrun_walltime_and_memory* and *set_tool_perf_patterns* methods of the *SphExaPatRunCheck* class.

Looking at the report with the tool gives more insight into the performance of the code:

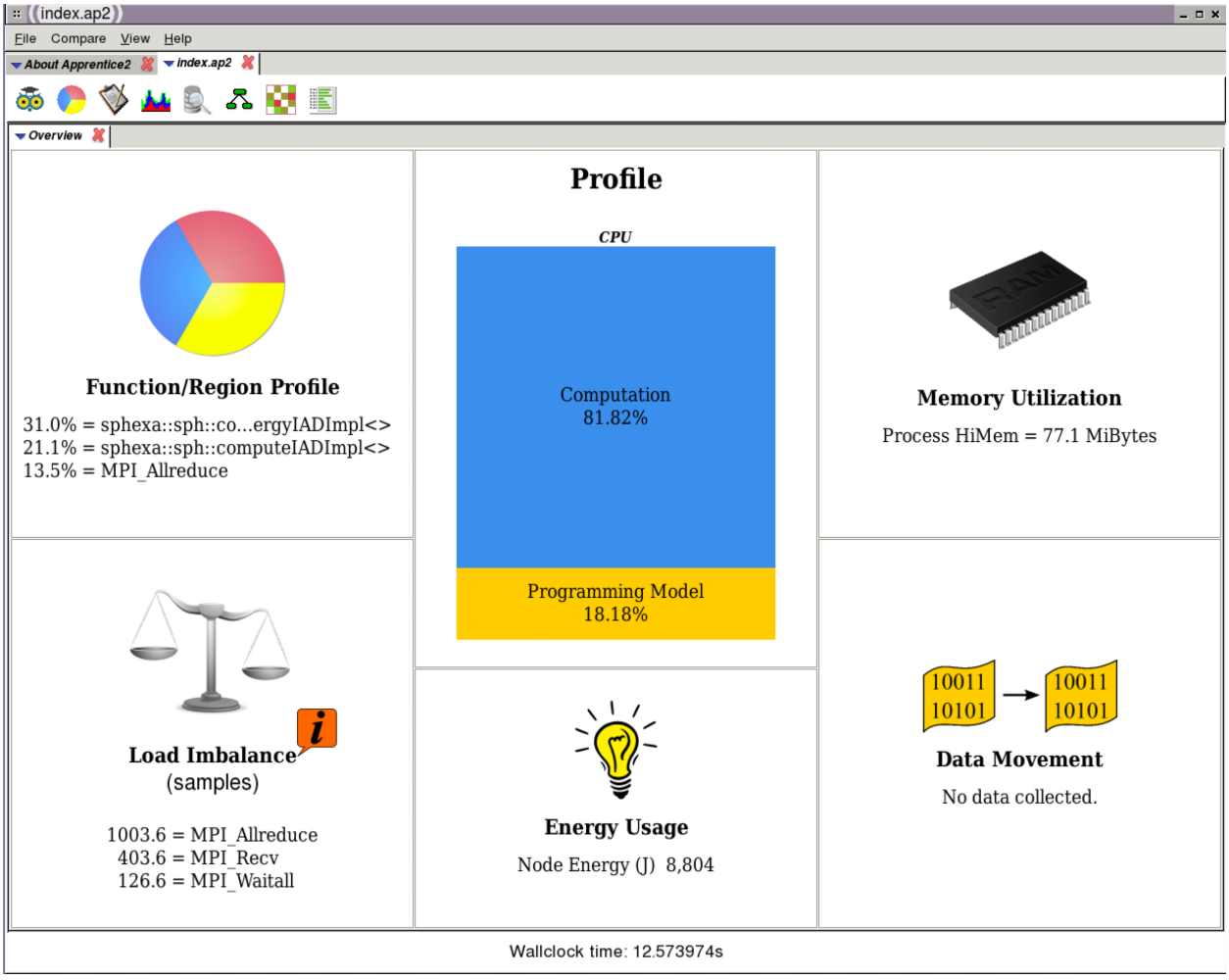

Fig. 1: Apprentice2 (launched with app2 sqpatch.exe+15597-5s/index.ap2)

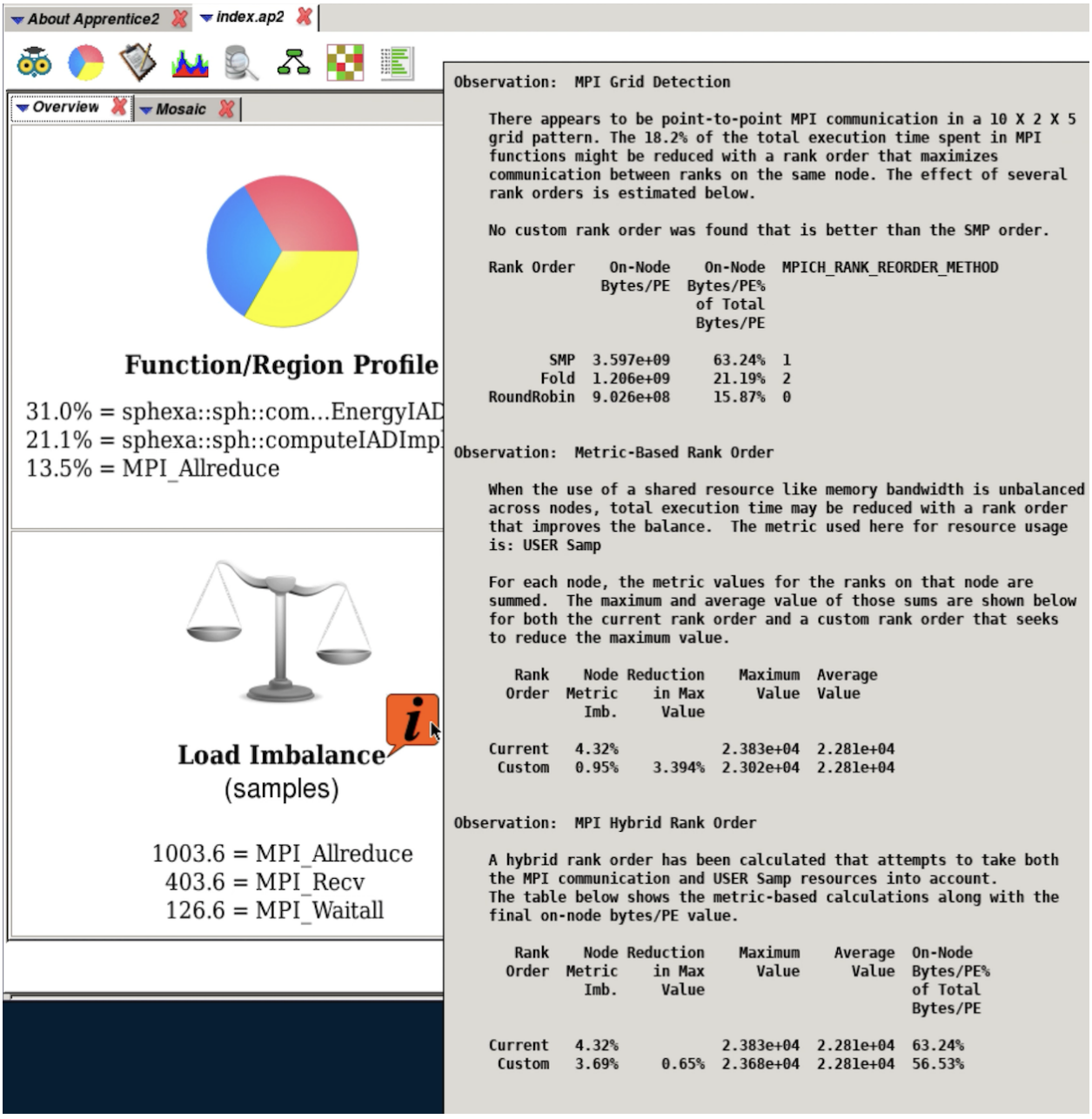

Fig. 2: Apprentice2 (overview: load imbalance)

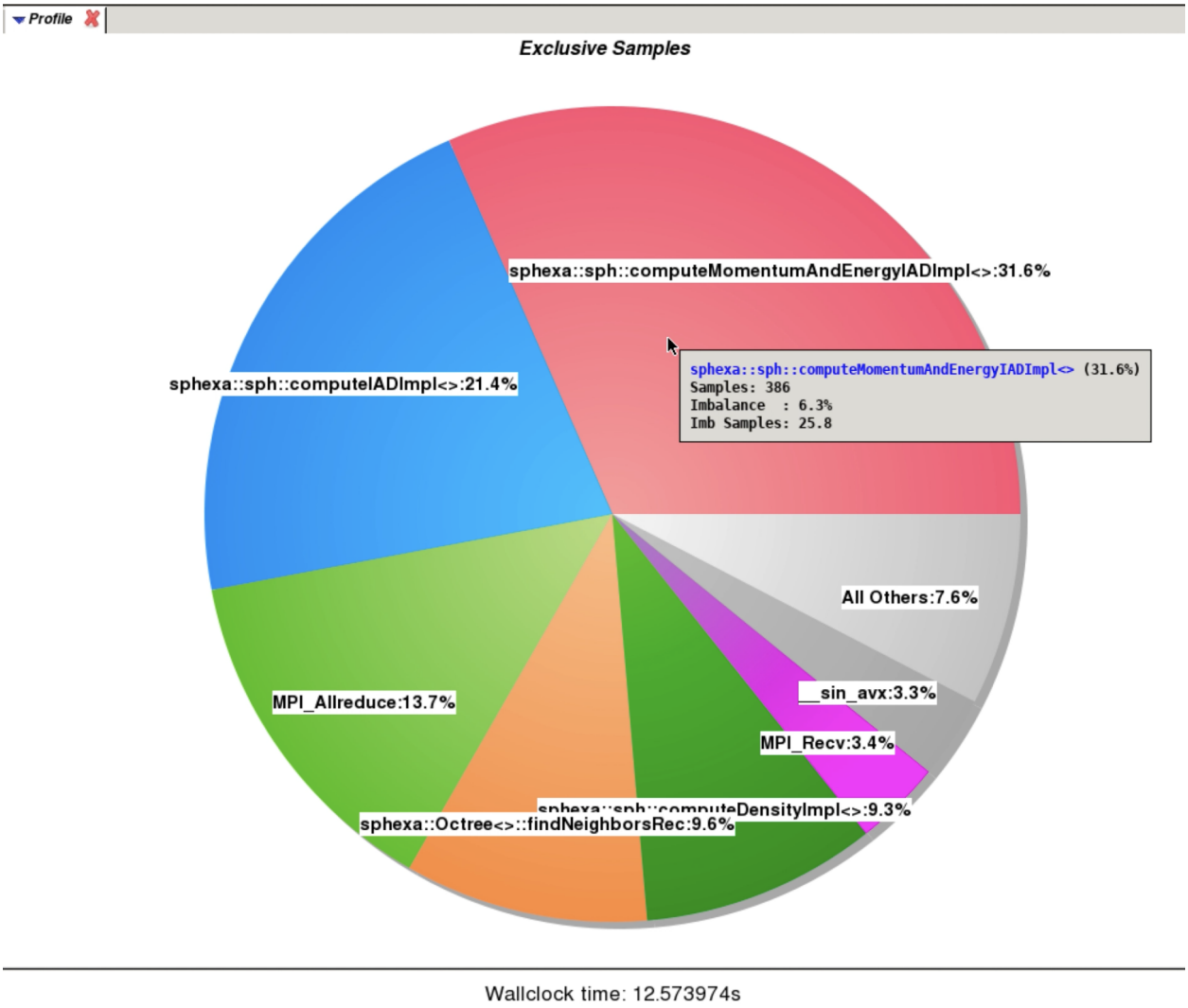

Fig. 3: Apprentice2 (overview: exclusive samples)

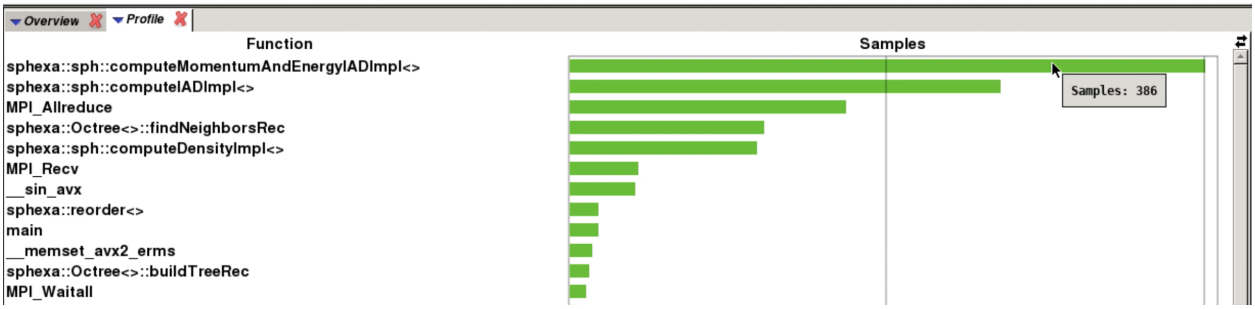

Fig. 4: Apprentice2 (profile)

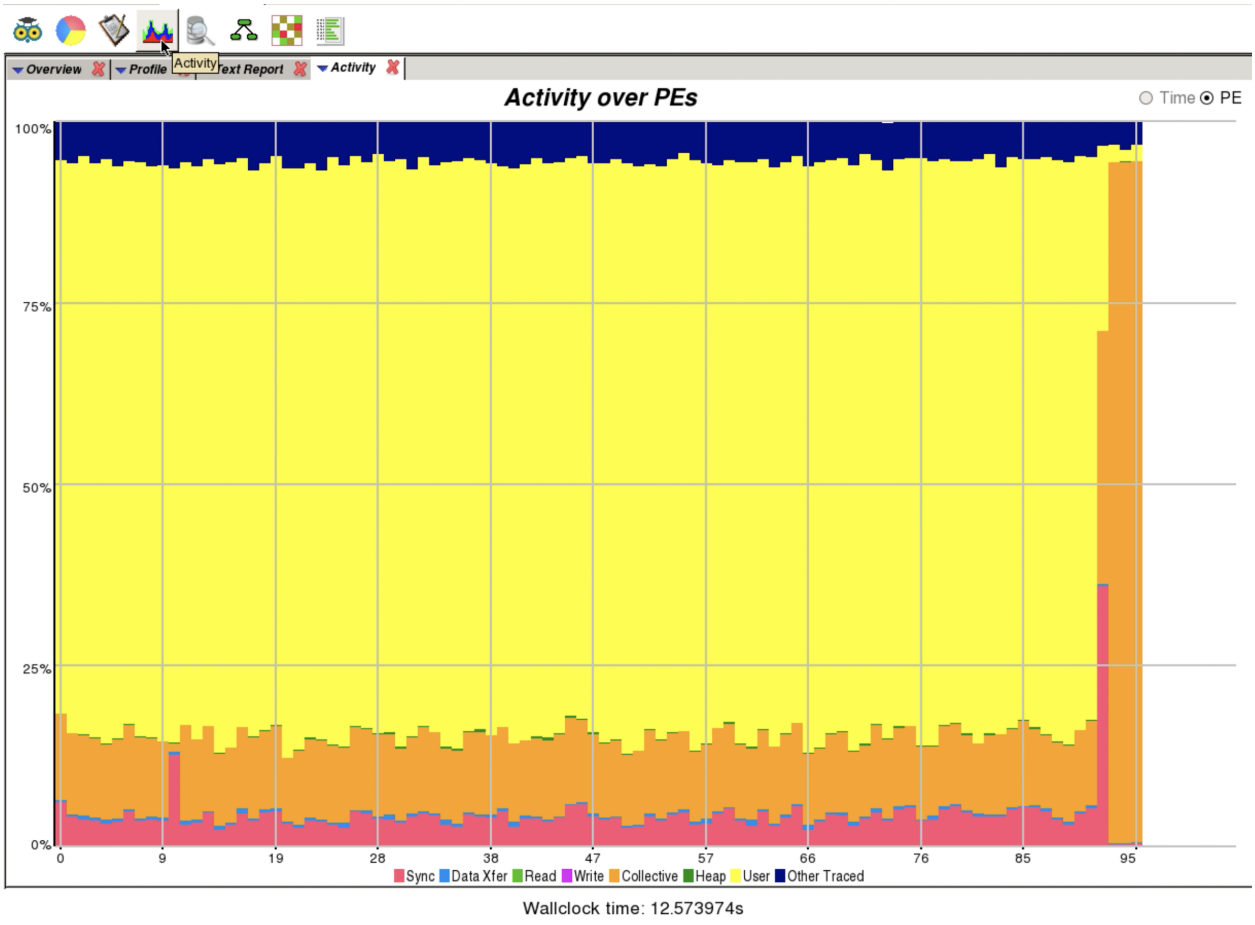

Fig. 5: Apprentice2 (timeline)

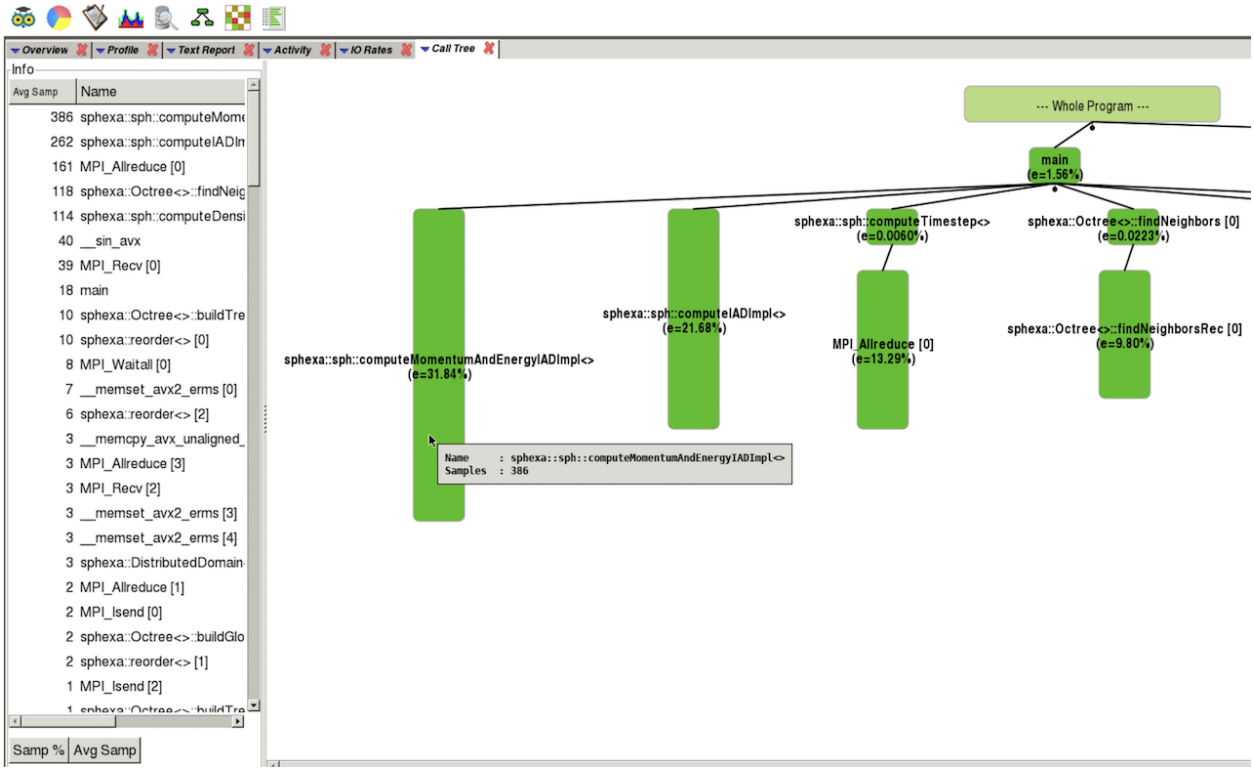

Fig. 6: Apprentice2 (calltree)

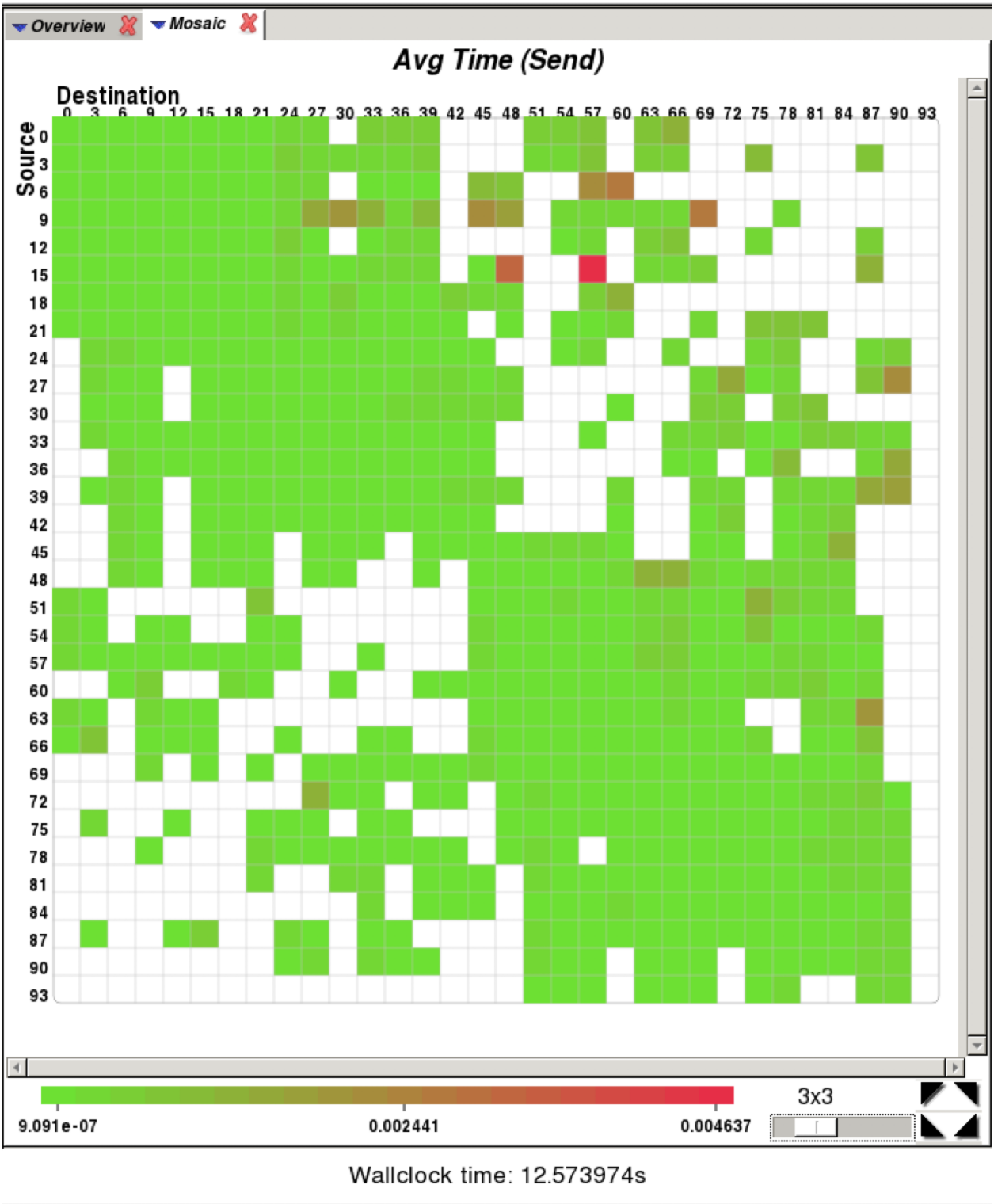

Fig. 7: Apprentice2 (communication matrix)

CHAPTER

EIGHT

REFERENCE GUIDE

8.1 Regression tests

8.1.1 internal_timers_mpi.py

Sanity checks

reframechecks.common.sphexa.sanity.**elapsed_time_from_date**(*self*)
Reports elapsed time in seconds using the linux date command:

```
starttime=1579725956
stoptime=1579725961
reports: _Elapsed: 5 s
```

reframechecks.common.sphexa.sanity.**pctg_FindNeighbors**(*obj*)
reports: * %FindNeighbors: 9.8 %

reframechecks.common.sphexa.sanity.**pctg_IAD**(*obj*)
reports: * %IAD: 17.36 %

reframechecks.common.sphexa.sanity.**pctg_MomentumEnergyIAD**(*obj*)
reports: * %MomentumEnergyIAD: 30.15 %

reframechecks.common.sphexa.sanity.**pctg_Timestep**(*obj*)
reports: * %Timestep: 16.6 %

reframechecks.common.sphexa.sanity.**pctg_mpi_synchronizeHalos**(*obj*)
reports: * %mpi_synchronizeHalos: 12.62 %

reframechecks.common.sphexa.sanity.**seconds_elaps**(*self*)
Reports elapsed time in seconds using the internal timer from the code

```
=== Total time for iteration(0) 3.61153s
reports: * Elapsed: 3.6115 s
```

reframechecks.common.sphexa.sanity.**seconds_timers**(*self*, *region*)
Reports timings (in seconds) using the internal timer from the code

```
# domain::sync: 0.118225s
# updateTasks: 0.00561256s
# FindNeighbors: 0.266282s
# Density: 0.120372s
# EquationOfState: 0.00255166s
```

(continues on next page)

(continued from previous page)

```
# mpi::synchronizeHalos: 0.116917s
# IAD: 0.185804s
# mpi::synchronizeHalos: 0.0850162s
# MomentumEnergyIAD: 0.423282s
# Timestep: 0.0405346s
# UpdateQuantities: 0.0140938s
# EnergyConservation: 0.0224118s
# UpdateSmoothingLength: 0.00413466s
```

8.1.2 internal_timers_mpi_containers.py

class reframechecks.notool.internal_timers_mpi_containers.**SphExa_Container_Base_Check**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

2 parameters can be set for simulation:

Parameters

- **mpi_task** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpi_task, but cubesize could also be on the list of parameters,
- **step** – number of simulation steps.

Dependencies are:

- compute: inputs (mpi_task, step) —srun—> *job.out
- postprocess logs: inputs (*job.out) —x—> termgraph.in
- plot data: inputs (termgraph.in) —termgraph.py—> termgraph.rpt

class reframechecks.notool.internal_timers_mpi_containers.**MPI_Collect_Logs_Test**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

collect_logs()

extract_data()

class reframechecks.notool.internal_timers_mpi_containers.**MPI_Compute_Sarus_Test**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class run the executable with Sarus

class reframechecks.notool.internal_timers_mpi_containers.**MPI_Compute_Singularity_Test**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class run the executable with Singularity (and natively too for comparison)

class reframechecks.notool.internal_timers_mpi_containers.**MPI_Plot_Test**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

class reframechecks.notool.internal_timers_mpi_containers.**SphExa_Container_Base_Check**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

2 parameters can be set for simulation:

Parameters

- **mpi_task** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpi_task, but cubesize could also be on the list of parameters,
- **step** – number of simulation steps.

Dependencies are:

- compute: inputs (mpi_task, step) —srun—> *job.out
- postprocess logs: inputs (*job.out) —x—> termgraph.in
- plot data: inputs (termgraph.in) —termgraph.py—> termgraph.rpt

8.1.3 Intel

intel_inspector.py

class reframechecks.intel.intel_inspector.**SphExaIntelInspectorCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with Intel Inspector (mpi only): https://software.intel.com/en-us/inspector

Available analysis types are: `inspxe-cl -h collect`

```
mi1   Detect Leaks
mi2   Detect Memory Problems
mi3   Locate Memory Problems
ti1   Detect Deadlocks
ti2   Detect Deadlocks and Data Races
ti3   Locate Deadlocks and Data Races
```

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

Typical performance reporting:

```
PERFORMANCE REPORT
-------------------------------------------------
sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 8.899 s
      ...
      * Memory not deallocated: 1
```

intel_vtune.py

class reframechecks.intel.intel_vtune.**SphExaVtuneCheck**(**args: Any*, ***kwargs: Any*)
Bases: sphexa.sanity_vtune.

This class runs the test code with Intel(R) VTune(TM) (mpi only): https://software.intel.com/en-us/vtune

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

class reframechecks.intel.intel_vtune.**SphExaVtuneCheck**(**args: Any*, ***kwargs: Any*)
Bases: sphexa.sanity_vtune.

This class runs the test code with Intel(R) VTune(TM) (mpi only): https://software.intel.com/en-us/vtune

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

Sanity checks

class reframechecks.common.sphexa.sanity_vtune.**VtuneBaseTest**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

set_basic_perf_patterns()
A set of basic perf_patterns shared between the tests

set_vtune_perf_patterns_rpt()
More perf_patterns for the tool

Typical performance reporting:

```
      * vtune_elapsed_min: 6.695 s
      * vtune_elapsed_max: 6.695 s
      * vtune_elapsed_cput: 5.0858 s
      * vtune_elapsed_cput_efft: 4.8549 s
      * vtune_elapsed_cput_spint: 0.2309 s
```

(continues on next page)

(continued from previous page)

```
      * vtune_elapsed_cput_spint_mpit: 0.2187 s
      * %vtune_effective_physical_core_utilization: 85.3 %
      * %vtune_effective_logical_core_utilization: 84.6 %
      * vtune_cput_cn0: 122.06 s
      * %vtune_cput_cn0_efft: 95.5 %
      * %vtune_cput_cn0_spint: 4.5 %
```

intel_advisor.py

class reframechecks.intel.intel_advisor.**SphExaIntelAdvisorCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with Intel Advisor (mpi only): https://software.intel.com/en-us/advisor

Available analysis types are: `advixe-cl -h collect`

```
survey       - Discover efficient vectorization and/or threading
dependencies - Identify and explore loop-carried dependencies for loops
map          - Identify and explore complex memory accesses
roofline     - Run the Survey analysis + Trip Counts & FLOP analysis
suitability  - Check predicted parallel performance
tripcounts   - Identify the number of loop iterations.
```

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

Typical performance reporting:

```
PERFORMANCE REPORT
--------------------------------------------------
sphexa_inspector_sqpatch_024mpi_001omp_100n_0steps
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 3.6147 s
      ...
      * advisor_elapsed: 2.13 s
      * advisor_loop1_line: 94 (momentumAndEnergyIAD.hpp)
```

Sanity checks

reframechecks.common.sphexa.sanity_intel.**advisor_elapsed**(*obj*)
Reports the elapsed time (sum of `Self Time` in seconds) measured by the tool

```
> summary.rpt
ID / Function Call Sites and Loops / Total Time / Self Time /  Type
71 [loop in sphexa::sph::computeMomentumAndEnergyIADImpl<double,
  ... sphexa::ParticlesData<double>> at momentumAndEnergyIAD.hpp:94]
  ... 1.092s      0.736s              Scalar  momentumAndEnergyIAD.hpp:94
34 [loop in MPIDI_Cray_shared_mem_coll_bcast]
  ... 0.596s      0.472s              Scalar  libmpich_gnu_82.so.3
etc.
returns: * advisor_elapsed: 2.13 s
```

reframechecks.common.sphexa.sanity_intel.**advisor_loop1_filename**(*obj*)
Reports the name of the source file (filename) of the most time consuming loop

```
> summary.rpt
ID / Function Call Sites and Loops / Total Time / Self Time /  Type
71 [loop in sphexa::sph::computeMomentumAndEnergyIADImpl<double,
  ... sphexa::ParticlesData<double>> at momentumAndEnergyIAD.hpp:94]
  ... 1.092s      0.736s              Scalar  momentumAndEnergyIAD.hpp:94
34 [loop in MPIDI_Cray_shared_mem_coll_bcast]
  ... 0.596s      0.472s              Scalar  libmpich_gnu_82.so.3
etc.
returns: * advisor_loop1_line: 94 (momentumAndEnergyIAD.hpp)
```

reframechecks.common.sphexa.sanity_intel.**advisor_loop1_line**(*obj*)
Reports the line (fline) of the most time consuming loop

```
> summary.rpt
ID / Function Call Sites and Loops / Total Time / Self Time /  Type
71 [loop in sphexa::sph::computeMomentumAndEnergyIADImpl<double,
  ... sphexa::ParticlesData<double>> at momentumAndEnergyIAD.hpp:94]
  ... 1.092s      0.736s              Scalar  momentumAndEnergyIAD.hpp:94
34 [loop in MPIDI_Cray_shared_mem_coll_bcast]
  ... 0.596s      0.472s              Scalar  libmpich_gnu_82.so.3
etc.
returns: * advisor_loop1_line: 94 (momentumAndEnergyIAD.hpp)
```

reframechecks.common.sphexa.sanity_intel.**advisor_version**(*obj*)
Checks tool's version:

```
> advixe-cl --version
Intel(R) Advisor 2020 (build 604394) Command Line Tool
returns: True or False
```

reframechecks.common.sphexa.sanity_intel.**inspector_not_deallocated**(*obj*)
Reports number of Memory not deallocated problem(s)

```
> summary.rpt
2 new problem(s) found
1 Memory leak problem(s) detected
```

(continues on next page)

(continued from previous page)

```
1 Memory not deallocated problem(s) detected
returns: * Memory not deallocated: 1
```

reframechecks.common.sphexa.sanity_intel.**inspector_version**(*obj*)

Checks tool's version:

```
> inspxe-cl --version
Intel(R) Inspector 2020 (build 603904) Command Line tool
returns: True or False
```

reframechecks.common.sphexa.sanity_intel.**vtune_logical_core_utilization**(*self*)

Reports the minimum `Physical Core Utilization` (%) measured by the tool

```
Effective Logical Core Utilization: 96.0% (23.028 out of 24)
Effective Logical Core Utilization: 95.9% (23.007 out of 24)
Effective Logical Core Utilization: 95.5% (22.911 out of 24)
```

reframechecks.common.sphexa.sanity_intel.**vtune_momentumAndEnergyIAD**(*self*)

sphexa::sph::computeMomentumAndEnergyIADImpl<...>	sqpatch.exe	40.919s
sphexa::sph::computeMomentumAndEnergyIADImpl<...>	sqpatch.exe	38.994s
sphexa::sph::computeMomentumAndEnergyIADImpl<...>	sqpatch.exe	40.245s
sphexa::sph::computeMomentumAndEnergyIADImpl<...>	sqpatch.exe	39.487s

reframechecks.common.sphexa.sanity_intel.**vtune_perf_patterns**(*obj*)

Dictionary of default `perf_patterns` for the tool

reframechecks.common.sphexa.sanity_intel.**vtune_physical_core_utilization**(*self*)

Reports the minimum `Physical Core Utilization` (%) measured by the tool

```
Effective Physical Core Utilization: 96.3% (11.554 out of 12)
Effective Physical Core Utilization: 96.1% (11.534 out of 12)
Effective Physical Core Utilization: 95.9% (11.512 out of 12)
```

reframechecks.common.sphexa.sanity_intel.**vtune_time**(*self*)

Vtune creates 1 report per compute node. For example, a 48 mpi tasks job (= 2 compute nodes when running with 24 c/cn) will create 2 directories: * rpt.nid00001/rpt.nid00001.vtune * rpt.nid00002/rpt.nid00002.vtune

Typical output (for each compute node) is:

```
Elapsed Time:     14.866s
    CPU Time:     319.177s            /24 = 13.3
        Effective Time:   308.218s    /24 = 12.8
            Idle: 0s
            Poor: 19.725s
            Ok:   119.570s
            Ideal:        168.922s
            Over: 0s
        Spin Time:        10.959s             /24 =  0.4
            MPI Busy Wait Time:   10.795s
            Other:        0.164s
        Overhead Time:    0s
Total Thread Count:       25
Paused Time:      0s
```

reframechecks.common.sphexa.sanity_intel.**vtune_tool_reference**(*obj*)
Dictionary of default `reference` for the tool

reframechecks.common.sphexa.sanity_intel.**vtune_version**(*obj*)
Checks tool's version:

```
> vtune --version
Intel(R) VTune(TM) Profiler 2020 (build 605129) Command Line Tool
returns: True or False
```

8.1.4 Score-P

scorep_sampling_profiling.py

scorep_sampling_tracing.py

Sanity checks

reframechecks.common.sphexa.sanity_scorep.**ipc_rk0**(*obj*)
Reports the IPC (instructions per cycle) for rank 0

reframechecks.common.sphexa.sanity_scorep.**program_begin_count**(*obj*)
Reports the number of `PROGRAM_BEGIN` in the otf2 trace file

reframechecks.common.sphexa.sanity_scorep.**program_end_count**(*obj*)
Reports the number of `PROGRAM_END` in the otf2 trace file

reframechecks.common.sphexa.sanity_scorep.**ru_maxrss_rk0**(*obj*)
Reports the `maximum resident set size`

reframechecks.common.sphexa.sanity_scorep.**scorep_assert_version**(*obj*)
Checks tool's version:

```
> scorep --version
Score-P 6.0
returns: True or False
```

reframechecks.common.sphexa.sanity_scorep.**scorep_com_pct**(*obj*)
Reports COM % measured by the tool

```
type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
COM      4,680 1,019,424     891  303.17    12.0         297.39  COM
                                          ****
```

reframechecks.common.sphexa.sanity_scorep.**scorep_elapsed**(*obj*)
Typical performance report from the tool (profile.cubex)

```
   type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
   ALL  1,019,921 2,249,107 934,957  325.00   100.0   144.50  ALL
                                    ******
   USR    724,140 1,125,393 667,740  226.14    69.6   200.94  USR
   MPI    428,794    59,185 215,094   74.72    23.0  1262.56  MPI
   COM     43,920 1,061,276  48,832   21.96     6.8    20.69  COM
MEMORY      9,143     3,229   3,267    2.16     0.7   669.59  MEMORY
SCOREP         94        24      24    0.01     0.0   492.90  SCOREP
```

(continues on next page)

(continued from previous page)

```
   USR    317,100   283,366 283,366   94.43    29.1         333.24 ...
_ZN6sphexa3sph31computeMomentumAndEnergyIADImplIdNS_13ParticlesData ...
IdEEEEvRKNS_4TaskERT0_
```

reframechecks.common.sphexa.sanity_scorep.**scorep_exclusivepct_energy**(*obj*)

Reports % of elapsed time (exclusive) for MomentumAndEnergy function (small scale job)

```
> sqpatch_048mpi_001omp_125n_10steps_1000000cycles/rpt.exclusive
0.0193958 (0.0009252%) sqpatch.exe
1.39647 (0.06661%)       + main
...
714.135 (34.063%)   |   + ...
         *******
  _ZN6sphexa3sph31computeMomentumAndEnergyIADImplIdNS_13 ...
  ParticlesDataIdEEEEvRKNS_4TaskERT0_
0.205453 (0.0098%)  |   +
  _ZN6sphexa3sph15computeTimestepIdNS0_21TimestepPress2ndOrderIdNS_13 ...
  ParticlesDataIdEEEES4_EEvRKSt6vectorINS_4TaskESaIS7_EERT1_
201.685 (9.62%)     |   |   + MPI_Allreduce


 type max_buf[B]    visits    hits time[s] time[%] time/visit[us]  region
 OMP  1,925,120    81,920       0   63.84     2.5         779.29
  !$omp parallel @momentumAndEnergyIAD.hpp:87 ***
 OMP    920,500    81,920  48,000  125.41     5.0        1530.93
  !$omp for @momentumAndEnergyIAD.hpp:87      ***
 OMP    675,860    81,920       1   30.95     1.2         377.85
  !$omp implicit barrier @momentumAndEnergyIAD.hpp:93
                                             ***
```

reframechecks.common.sphexa.sanity_scorep.**scorep_inclusivepct_energy**(*obj*)

Reports % of elapsed time (inclusive) for MomentumAndEnergy function (small scale job)

```
> sqpatch_048mpi_001omp_125n_10steps_1000000cycles/rpt.exclusive
0.0193958 (0.0009252%) sqpatch.exe
1.39647 (0.06661%)       + main
...
714.135 (34.063%)   |   + ...
         *******
  _ZN6sphexa3sph31computeMomentumAndEnergyIADImplIdNS_13 ...
  ParticlesDataIdEEEEvRKNS_4TaskERT0_
0.205453 (0.0098%)  |   +
  _ZN6sphexa3sph15computeTimestepIdNS0_21TimestepPress2ndOrderIdNS_13 ...
  ParticlesDataIdEEEES4_EEvRKSt6vectorINS_4TaskESaIS7_EERT1_
201.685 (9.62%)     |   |   + MPI_Allreduce
```

reframechecks.common.sphexa.sanity_scorep.**scorep_info_cuda_support**(*obj*)

Checks tool's configuration (Cuda support)

```
> scorep-info config-summary
CUDA support:  yes
```

reframechecks.common.sphexa.sanity_scorep.**scorep_info_papi_support**(*obj*)
Checks tool's configuration (papi support)

```
> scorep-info config-summary
PAPI support: yes
```

reframechecks.common.sphexa.sanity_scorep.**scorep_info_perf_support**(*obj*)
Checks tool's configuration (perf support)

```
> scorep-info config-summary
metric perf support: yes
```

reframechecks.common.sphexa.sanity_scorep.**scorep_info_unwinding_support**(*obj*)
Checks tool's configuration (libunwind support)

```
> scorep-info config-summary
Unwinding support: yes
```

reframechecks.common.sphexa.sanity_scorep.**scorep_mpi_pct**(*obj*)
Reports MPI % measured by the tool

```
type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
MPI    428,794    59,185 215,094   74.72    23.0  1262.56  MPI
                                             ****
```

reframechecks.common.sphexa.sanity_scorep.**scorep_omp_pct**(*obj*)
Reports OMP % measured by the tool

```
type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
OMP 40,739,286 3,017,524 111,304 2203.92    85.4         730.37  OMP
                                             ****
```

reframechecks.common.sphexa.sanity_scorep.**scorep_top1_name**(*obj*)
Reports demangled name of top1 function name, for instance:

```
> c++filt ...
_ZN6sphexa3sph31computeMomentumAndEnergyIADImplIdNS_13 ...
ParticlesDataIdEEEEvRKNS_4TaskERT0_

void sphexa::sph::computeMomentumAndEnergyIADImpl  ...
      <double, sphexa::ParticlesData<double> > ...
      (sphexa::Task const&, sphexa::ParticlesData<double>&)
```

reframechecks.common.sphexa.sanity_scorep.**scorep_top1_tracebuffersize**(*obj*)
Reports max_buf[B] for top1 function

```
   type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
   ...
   USR    317,100   283,366 283,366   94.43    29.1         333.24 ...
_ZN6sphexa3sph31computeMomentumAndEnergyIADImplIdNS_13ParticlesData ...

   USR    430,500    81,902  81,902   38.00     1.5         463.99  ...
gomp_team_barrier_wait_end
```

reframechecks.common.sphexa.sanity_scorep.**scorep_top1_tracebuffersize_name**(*obj*)
Reports function name for top1 (max_buf[B]) function

reframechecks.common.sphexa.sanity_scorep.**scorep_usr_pct**(*obj*)

Reports USR % measured by the tool

```
type max_buf[B]   visits    hits time[s] time[%] time/visit[us] region
USR    724,140 1,125,393 667,740  226.14    69.6   200.94  USR
                                            ****
```

reframechecks.common.sphexa.sanity_scorep.**scorep_version**(*obj*)

Checks tool's version:

```
> scorep --version
Score-P 7.0
returns: version string
```

8.1.5 Scalasca

scalasca_sampling_profiling.py

class reframechecks.scalasca.scalasca_sampling_profiling.**SphExaScalascaProfilingCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with Scalasca (mpi only):

3 parameters can be set for simulation:

Parameters

- **mpi_task** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.
- **cycles** – sampling sources generate interrupts that trigger a sample. SCOREP_SAMPLING_EVENTS sets the sampling source: see $EBROOTSCOREM-INP/share/doc/scorep/html/sampling.html . Very large values will produce unreliable performance report, very small values will have a large runtime overhead.

Typical performance reporting:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------
sphexa_scalascaS+P_sqpatch_024mpi_001omp_100n_4steps_5000000cycles
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 20.4549 s
      * _Elapsed: 38 s
      * domain_distribute: 0.4089 s
      * mpi_synchronizeHalos: 2.4644 s
      * BuildTree: 0 s
      * FindNeighbors: 1.8787 s
      * Density: 1.8009 s
      * EquationOfState: 0.0174 s
```

(continues on next page)

(continued from previous page)

```
      * IAD: 3.726 s
      * MomentumEnergyIAD: 6.1141 s
      * Timestep: 3.5887 s
      * UpdateQuantities: 0.0424 s
      * EnergyConservation: 0.0177 s
      * SmoothingLength: 0.017 s
      * %MomentumEnergyIAD: 29.89 %
      * %Timestep: 17.54 %
      * %mpi_synchronizeHalos: 12.05 %
      * %FindNeighbors: 9.18 %
      * %IAD: 18.22 %
      * scorep_elapsed: 21.4262 s
      * %scorep_USR: 71.0 %
      * %scorep_MPI: 23.3 %
      * scorep_top1: 30.1 % (void sphexa::sph::computeMomentumAndEnergyIADImpl)
      * %scorep_Energy_exclusive: 30.112 %
      * %scorep_Energy_inclusive: 30.112 %
```

set_runflags()

scalasca_sampling_tracing.py

class reframechecks.scalasca.scalasca_sampling_tracing.**SphExaScalascaTracingCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with Scalasca (mpi only):

3 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.
- **cycles** – sampling sources generate interrupts that trigger a sample. SCOREP_SAMPLING_EVENTS sets the sampling source: see $EBROOTSCOREMINP/share/doc/scorep/html/sampling.html . Very large values will produce unreliable performance report, very small values will have a large runtime overhead.

Typical performance reporting:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------
sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_4steps_5000000cycles
- dom:gpu
   - PrgEnv-gnu
      * num_tasks: 24
      * Elapsed: 20.5242 s
      * _Elapsed: 28 s
      * domain_distribute: 0.4712 s
```

(continues on next page)

(continued from previous page)

```
      * mpi_synchronizeHalos: 2.4623 s
      * BuildTree: 0 s
      * FindNeighbors: 1.8752 s
      * Density: 1.8066 s
      * EquationOfState: 0.0174 s
      * IAD: 3.7259 s
      * MomentumEnergyIAD: 6.1355 s
      * Timestep: 3.572 s
      * UpdateQuantities: 0.0273 s
      * EnergyConservation: 0.0079 s
      * SmoothingLength: 0.017 s
      * %MomentumEnergyIAD: 29.89 %
      * %Timestep: 17.4 %
      * %mpi_synchronizeHalos: 12.0 %
      * %FindNeighbors: 9.14 %
      * %IAD: 18.15 %
      * mpi_latesender: 2090 count
      * mpi_latesender_wo: 19 count
      * mpi_latereceiver: 336 count
      * mpi_wait_nxn: 1977 count
      * max_ipc_rk0: 1.294516 ins/cyc
      * max_rumaxrss_rk0: 127932 kilobytes
```

set_runflags()

Sanity checks

reframechecks.common.sphexa.sanity_scalasca.**rpt_tracestats_mpi**(*obj*)

Reports MPI statistics (mpi_latesender, mpi_latesender_wo, mpi_latereceiver, mpi_wait_nxn, mpi_nxn_completion) by reading the stat_rpt (`trace.stat`) file reported by the tool. Columns are (for each PatternName): Count Mean Median Minimum Maximum Sum Variance Quartil25 and Quartil75. Count (=second column) is used here.

Typical performance reporting:

```
> sphexa_scalascaS+T_sqpatch_024mpi_001omp_100n_4steps_5000000cycles/scorep_sqpatch_
↪24_trace/trace.stat

PatternName                Count      Mean    Median      Minimum      Maximum     ␣
↪Sum     Variance    Quartil25    Quartil75
mpi_latesender                  2087 0.0231947 0.0024630 0.0000001623 0.2150408312␣
↪48.4074203231 0.0029201162 0.0007851545 0.0067652356
mpi_latesender_wo                 15 0.0073685 0.0057757 0.0011093750 0.0301200084␣
↪0.1105282126 0.0000531833 0.0025275522 0.0104651418
mpi_latereceiver                 327 0.0047614 0.0000339 0.0000362101 0.0139404002␣
↪1.5569782562 0.0000071413 0.0000338690 0.0000338690
mpi_wait_nxn                    1978 0.0324812 0.0002649 0.0000000015 0.7569314451␣
↪64.2478221177 0.0164967346 0.0001135482 0.0004163433
mpi_nxn_completion              1978 0.0000040 0.0001135 0.0000000008 0.0000607473␣
↪0.0078960137 0.0000000001 0.0000378494 0.0001892469
```

reframechecks.common.sphexa.sanity_scalasca.**rpt_tracestats_omp**(*obj*)

Reports OpenMP statistics by reading the trace.stat file: - omp_ibarrier_wait: OMP Wait at Implicit Barrier (sec)

in Cube GUI - omp_lock_contention_critical: OMP Critical Contention (sec) in Cube GUI Each column (Count Mean Median Minimum Maximum Sum Variance Quartil25 and Quartil75) is read, only Sum is reported here.

reframechecks.common.sphexa.sanity_scalasca.**scalasca_mpi_pct**(*obj*)
MPI % reported by Scalasca (scorep.score, notice no hits column)

```
type max_buf[B]  visits time[s] time[%] time/visit[us]  region
ALL  6,529,686 193,188   28.13   100.0         145.63  ALL
OMP  6,525,184 141,056   27.33    97.1         193.74  OMP
MPI      4,502      73    0.02     0.1         268.42  MPI
                              *****
```

reframechecks.common.sphexa.sanity_scalasca.**scalasca_omp_pct**(*obj*)
OpenMP % reported by Scalasca (scorep.score, notice no hits column)

```
type max_buf[B]  visits time[s] time[%] time/visit[us]  region
ALL  6,529,686 193,188   28.13   100.0         145.63  ALL
OMP  6,525,184 141,056   27.33    97.1         193.74  OMP
                              *****
MPI      4,502      73    0.02     0.1         268.42  MPI
```

8.1.6 Extrae

extrae.py

Sanity checks

reframechecks.common.sphexa.sanity_extrae.**create_sh**(*obj*)
Create a wrapper script to insert Extrae libs (with LD_PRELOAD) into the executable at runtime

reframechecks.common.sphexa.sanity_extrae.**extrae_version**(*obj*)
Checks tool's version. As there is no `--version` flag available, we read the version from extrae_version.h and compare it to our reference

```
> cat $EBROOTEXTRAE/include/extrae_version.h
#define EXTRAE_MAJOR 3
#define EXTRAE_MINOR 7
#define EXTRAE_MICRO 1
returns: True or False
```

reframechecks.common.sphexa.sanity_extrae.**rpt_mpistats**(*obj*)
Reports statistics (histogram of MPI communications) from the comms.dat file

```
#_of_comms %_of_bytes_sent # histogram bin
466         0.00           # 10 B
3543        0.25           # 100 B
11554      11.69           # 1 KB
29425      88.05           # 10 KB
0           0.00           # 100 KB
0           0.00           # 1 MB
0           0.00           # 10 MB
0           0.00           # >10 MB
```

reframechecks.common.sphexa.sanity_extrae.**tool_reference_scoped_d**(*obj*)
Sets a set of tool perf_reference to be shared between the tests.

8.1.7 mpiP

mpip.py

Sanity checks

class reframechecks.common.sphexa.sanity_mpip.**MpipBaseTest**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

mpip_sanity_patterns()
Checks tool's version:

```
> cat ./sqpatch.exe.6.31820.1.mpiP
@ mpiP
@ Command : sqpatch.exe -n 62 -s 1
@ Version : 3.4.2  <-- 57fc864
```

set_basic_perf_patterns()
A set of basic perf_patterns shared between the tests

set_mpip_perf_patterns()
More perf_patterns for the tool

```
-----------------------------------
@--- MPI Time (seconds) -----------
-----------------------------------
Task    AppTime    MPITime     MPI%
   0        8.6      0.121     1.40 <-- min
   1        8.6      0.157     1.82
   2        8.6       5.92    68.84 <-- max
   *       25.8        6.2    24.02 <---

=> NonMPI= AppTime - MPITime
```

Typical performance reporting:

```
* mpip_avg_app_time: 8.6 s  (= 25.8/3mpi)
* mpip_avg_mpi_time: 2.07 s (=  6.2/3mpi)
* %mpip_avg_mpi_time: 24.02 %
* %mpip_avg_non_mpi_time: 75.98 %
```

reframechecks.common.sphexa.sanity_mpip.**mpip_perf_patterns**(*obj*, *reg*)
More perf_patterns for the tool

```
-----------------------------------
@--- MPI Time (seconds) -----------
-----------------------------------
Task    AppTime    MPITime     MPI%
   0        8.6      0.121     1.40 <-- min
   1        8.6      0.157     1.82
   2        8.6       5.92    68.84 <-- max
```

(continues on next page)

(continued from previous page)

```
   *       25.8        6.2    24.02 <---

=> NonMPI= AppTime - MPITime
```

Typical performance reporting:

```
* mpip_avg_app_time: 8.6 s  (= 25.8/3mpi)
* mpip_avg_mpi_time: 2.07 s (=  6.2/3mpi)
* %mpip_avg_mpi_time: 24.02 %
* %max/%min
* %mpip_avg_non_mpi_time: 75.98 %
```

8.1.8 Perftools

patrun.py

Sanity checks

class reframechecks.common.sphexa.sanity_perftools.**PerftoolsBaseTest**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

patrun_energy_power()

This table shows HW performance counter data for the whole program, averaged across ranks or threads, as applicable.

```
Table 8:  Program energy and power usage (from Cray PM)

   Node |     Node |   Process | Node Id
 Energy |    Power |      Time |  PE=HIDE
    (J) |      (W) |           |

  7,891 |  692.806 | 11.389914 | Total    <---
|-- ------------------------------------
|  2,076 |  182.356 | 11.384319 | nid.7
|  1,977 |  173.548 | 11.391657 | nid.4
|  1,934 |  169.765 | 11.392220 | nid.6
|  1,904 |  167.143 | 11.391461 | nid.5
|====================================
```

Typical output:

- patrun_avg_power: 692.806 W

patrun_hotspot1_mpi()

```
Table 1:  Profile by Function

  Samp% |    Samp |  Imb. |  Imb. | Group
        |         |  Samp | Samp% |  Function
```

(continues on next page)

(continued from previous page)

```
        |         |       |       |   PE=HIDE

 100.0% | 1,126.4 |    -- |    -- | Total
...
||=================================================
|   9.9% |   111.4 |    -- |    -- | MPI
||-------------------------------------------------
||   5.2% |    58.2 | 993.8 | 95.5% | MPI_Allreduce <--
||   3.6% |    40.9 | 399.1 | 91.7% | MPI_Recv
```

patrun_hwpc()

This table shows HW performance counter data for the whole program, averaged across ranks or threads, as applicable.

```
Table 4:  Program HW Performance Counter Data
  ...
  Thread Time                                          11.352817 secs
  UNHALTED_REFERENCE_CYCLES                       28,659,167,096
  CPU_CLK_THREAD_UNHALTED:THREAD_P                34,170,540,119
  DTLB_LOAD_MISSES:WALK_DURATION                      61,307,848
  INST_RETIRED:ANY_P                              22,152,242,298
  RESOURCE_STALLS:ANY                             19,793,119,676
  OFFCORE_RESPONSE_0:ANY_REQUEST:L3_MISS_LOCAL        20,949,344
  CPU CLK Boost                                             1.19 X
  Resource stall cycles / Cycles  -->                      57.9%
  Memory traffic GBytes           -->       0.118G/sec      1.34 GB
  Local Memory traffic GBytes               0.118G/sec      1.34 GB
  Memory Traffic / Nominal Peak                             0.2%
  DTLB Miss Ovhd                      61,307,848 cycles  0.2% cycles
  Retired Inst per Clock          -->                       0.65
===============================================================================
```

Typical output:

- patrun_memory_traffic: 1.34 GB
- patrun_ipc: 0.65
- %patrun_stallcycles: 57.9 %

patrun_imbalance()

Load imbalance from csv report

```
Table 1:  load Balance with MPI Message Stats
```

patrun_memory_bw()

This table shows memory traffic to local and remote memory for numa nodes, taking for each numa node the maximum value across nodes.

```
Table 9:  Memory Bandwidth by Numanode

  Memory |   Local |    Thread |  Memory |  Memory | Numanode
 Traffic |  Memory |      Time | Traffic | Traffic |  Node Id
```

(continues on next page)

(continued from previous page)

```
  GBytes | Traffic |           |  GBytes |       / |   PE=HIDE
         |  GBytes |           |   / Sec | Nominal |
         |         |           |         |    Peak |
|-------------------------------------------------------------
|   33.64 |   33.64 | 11.360701 |    2.96 |    4.3% | numanode.0
||------------------------------------------------------------
||   33.64 |   33.64 | 11.359413 |    2.96 |    4.3% | nid.4
||   33.59 |   33.59 | 11.359451 |    2.96 |    4.3% | nid.6
||   33.24 |   33.24 | 11.360701 |    2.93 |    4.3% | nid.5
||   28.24 |   28.24 | 11.355006 |    2.49 |    3.6% | nid.7
|=============================================================

2 sockets:
Table 10:  Memory Bandwidth by Numanode

  Memory |   Local |  Remote | Thread |  Memory |  Memory | Numanode
 Traffic |  Memory |  Memory |   Time | Traffic | Traffic |  Node Id
  GBytes | Traffic | Traffic |        |  GBytes |       / |   PE=HIDE
         |  GBytes |  GBytes |        |   / Sec | Nominal |
         |         |         |        |         |    Peak |
|-------------------------------------------------------------------
|   11.21 |   10.99 |    0.22 | 3.886926 |  2.88 | 3.8% | numanode.0
||-----------------------------------------------------------------
||   11.21 |   10.99 |    0.22 | 3.886926 | 2.88 |3.8% | nid.407
||   10.47 |   10.27 |    0.20 | 3.886450 | 2.69 |3.5% | nid.416
||=================================================================
|   11.29 |   11.06 |    0.23 | 3.889932 |  2.90 | 3.8% | numanode.1
||-----------------------------------------------------------------
||   11.29 |   11.06 |    0.23 | 3.889932 | 2.90 |3.8% | nid.407
||   10.09 |    9.88 |    0.20 | 3.885858 | 2.60 |3.4% | nid.416
|=================================================================
```

Typical output:

- patrun_memory_traffic_global: 33.64 GB
- patrun_memory_traffic_local: 33.64 GB
- %patrun_memory_traffic_peak: 4.3 %

patrun_num_of_compute_nodes()

Extract the number of compute nodes to compute averages

```
> ls 96mpi/sqpatch.exe+8709-4s/xf-files/:
  000004.xf
  000005.xf
  000006.xf
  000007.xf
```

Typical output:

- patrun_cn: 4

patrun_samples()
Elapsed time (in samples) reported by the tool:

```
Table 1:  Profile by Function

  Samp% |  Samp |  Imb. |  Imb. | Group
        |       |  Samp | Samp% |  Function
        |       |       |       |   PE=HIDE

 100.0% | 382.8 |    -- |    -- | Total
 TODO:
  Experiment:  samp_cs_time
  Sampling interval:  10000 microsecs
```

patrun_version()
Checks tool's version:

```
> pat_run -V
CrayPat/X:  Version 20.08.0 Revision 28ef35c9f
```

patrun_walltime_and_memory()
This table shows total wall clock time for the ranks with the maximum, mean, and minimum time, as well as the average across ranks.

```
Table 10:  Wall Clock Time, Memory High Water Mark

   Process |   Process | PE=[mmm]
      Time |     HiMem |
           | (MiBytes) |

 11.389914 |      76.3 | Total    <-- avgt
|---------------------------------
| 11.398188 |      57.7 | pe.24    <-- maxt
| 11.389955 |      98.9 | pe.34
| 11.365630 |      54.0 | pe.93    <-- mint
|=================================
```

Typical output:

- patrun_wallt_max: 11.3982 s
- patrun_wallt_avg: 11.3899 s
- patrun_wallt_min: 11.3656 s
- patrun_mem_max: 57.7 MiBytes
- patrun_mem_min: 54.0 MiBytes

perftools_lite_memory()
20.10.0 / AMD High Memory: 85,743.7 MiBytes 669.9 MiBytes per PE # More –> pat_report -O himem exe+141047-1002s/index.ap2 > rpt.mem

set_tool_perf_patterns()
More perf_patterns for the tool

Typical performance reporting:

```
      * patrun_wallt_max: 18.7552 s
      * patrun_wallt_avg: 18.7445 s
      * patrun_wallt_min: 18.7213 s
      * patrun_mem_max: 60.1 MiBytes
      * patrun_mem_min: 53.8 MiBytes
      * patrun_memory_traffic_global: 53.95 GB
      * patrun_memory_traffic_local: 53.95 GB
      * %patrun_memory_traffic_peak: 4.2 %
      * patrun_memory_traffic: 2.15 GB
      * patrun_ipc: 0.64
      * %patrun_stallcycles: 58.0 %
      * %patrun_user: 84.7 % (slow: 1677.0 smp [pe14] / mean:1570.2 median:1630.
↪0 / fast:26.0 [pe95])
      * %patrun_mpi: 11.1 % (slow: 1793.0 smp [pe94] / mean:205.9 median:146.0 /
↪ fast:91.0 [pe56])
      * %patrun_etc: 4.2 % (slow: 97.0 smp [pe63] / mean:78.3 median:78.5 /␣
↪fast:38.0 [pe93])
      * %patrun_total: 100.0 % (slow: 1862.0 smp [pe92] / mean:1854.4␣
↪median:1854.0 / fast:1835.0 [pe5])
      * %patrun_user_slowest: 90.5 % (pe.14)
      * %patrun_mpi_slowest: 5.6 % (pe.14)
      * %patrun_etc_slowest: 3.9 % (pe.14)
      * %patrun_user_fastest: 1.4 % (pe.95)
      * %patrun_mpi_fastest: 96.3 % (pe.95)
      * %patrun_etc_fastest: 2.3 % (pe.95)
      * %patrun_avg_usr_reported: 84.5 %
      * %patrun_avg_mpi_reported: 11.1 %
      * %patrun_avg_etc_reported: 4.4 %
      * %patrun_hotspot1: 34.7 % (sphexa::sph::computeMomentumAndEnergyIADImpl<>
↪)
      * %patrun_mpi_h1: 6.6 % (MPI_Allreduce)
      * %patrun_mpi_h1_imb: 94.1 % (MPI_Allreduce)
      * patrun_avg_energy: 3274.0 J
      * patrun_avg_power: 174.665 W
```

CHAPTER

NINE

NVIDIA® TOOLS

The NVIDIA Developer Tools are a collection of applications, spanning desktop and mobile targets, which enable developers to build, debug, profile, and develop class-leading and cutting-edge software that utilizes the latest visual computing hardware from NVIDIA.

The performance tools from the VI-HPS Institute are also supported on NVIDIA gpus.

9.1 Nsight™ Systems

NVIDIA® Nsight™ Systems is a system-wide performance analysis tool designed to visualize an application's algorithms, help you identify the largest opportunities to optimize, and tune to scale efficiently across any quantity or size of CPUs and GPUs.

9.1.1 CUDA

Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/nvidia

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--performance-report \
--keep-stage-files \
-c ./nsys_cuda.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----------] started processing sphexa_nsyscuda_sqpatch_002mpi_001omp_100n_0steps (Tool␣
↪validation)
[ RUN      ] sphexa_nsyscuda_sqpatch_002mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
```

(continues on next page)

(continued from previous page)

```
[       OK ] sphexa_nsyscuda_sqpatch_002mpi_001omp_100n_0steps on daint:gpu using PrgEnv-
↪gnu
[----------] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the *Class* shows how to setup and run the code with the tool.

`self.executable_opts` sets a list of arguments to pass to the tool. The report will be generated automatically at the end of the job.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the *Class*. For example, the information (%) about the data transfers from host to device (`[CUDA memcpy HtoD]`) is extracted with the *nsys_report_HtoD_pct* method. Looking at the report with the tool gives more insight into the performance of the code:

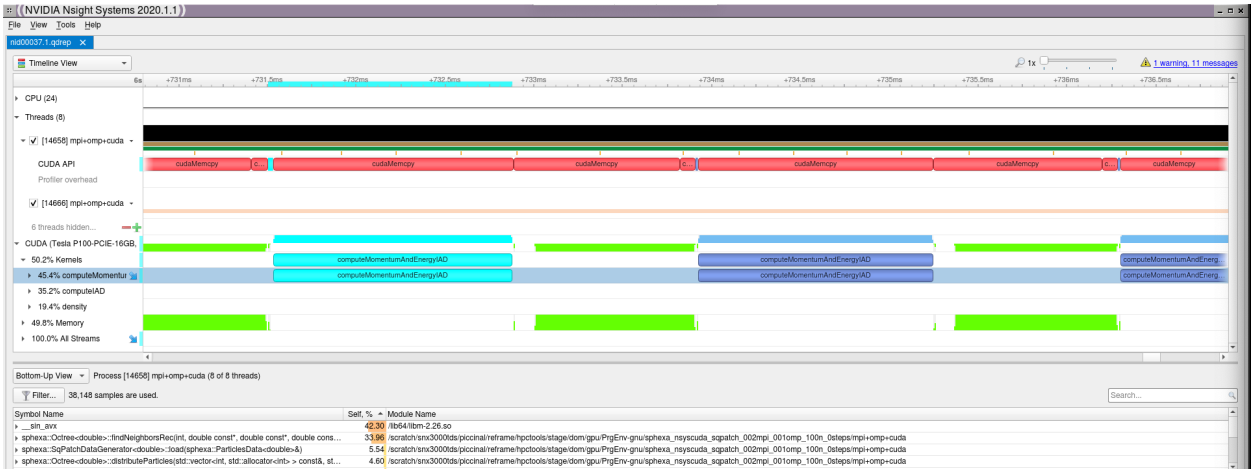

Fig. 1: Nsight Cuda (launched with: nsight-sys nid00036.0.qdrep)

9.2 VI-HPS tools

9.2.1 OPENACC

Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/openacc

~/reframe.git/reframe.py \
```

(continues on next page)

(continued from previous page)

```
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-pgi \
--performance-report \
--keep-stage-files \
-c ./scorep_openacc.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 2.22
Launched on host: daint101

[----] waiting for spawned checks to finish
[ OK ] openacc_scorepT_sqpatch_001mpi_001omp_100n_1steps_0cycles_ru_maxrss,ru_utime on␣
↪daint:gpu using PrgEnv-pgi
[ OK ] openacc_scorepT_sqpatch_002mpi_001omp_126n_1steps_0cycles_ru_maxrss,ru_utime on␣
↪daint:gpu using PrgEnv-pgi
[ OK ] openacc_scorepT_sqpatch_004mpi_001omp_159n_1steps_0cycles_ru_maxrss,ru_utime on␣
↪daint:gpu using PrgEnv-pgi
[ OK ] openacc_scorepT_sqpatch_008mpi_001omp_200n_1steps_0cycles_ru_maxrss,ru_utime on␣
↪daint:gpu using PrgEnv-pgi
[ OK ] openacc_scorepT_sqpatch_016mpi_001omp_252n_1steps_0cycles_ru_maxrss,ru_utime on␣
↪daint:gpu using PrgEnv-pgi
[----] all spawned checks have finished

[  PASSED  ] Ran 5 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the *Class* shows how to setup and run the code with the tool. This check is based on the `PrgEnv-pgi` programming environment and the `Score-P` and `otf2_cli_profile` performance tools. Set `self.build_system.cxx` to instrument the code and set the SCOREP runtime variables with `self.variables` to trigger the (OpenACC and rusage) *tracing* analysis. A text report will be generated using the otf_profiler method at the end of the job.

Performance reporting

A typical output from the `--performance-report` flag will look like this:

```
openacc_scorepT_sqpatch_016mpi_001omp_252n_1steps_0cycles_ru_maxrss,ru_utime
   - PrgEnv-pgi
      * num_tasks: 16
      * Elapsed: 28.3313 s
      * _Elapsed: 51 s
      * domain_distribute: 4.1174 s
      * mpi_synchronizeHalos: 0.8513 s
      * BuildTree: 0 s
      * FindNeighbors: 14.6041 s
      * Density: 0.5958 s
      * EquationOfState: 0.0139 s
      * IAD: 0.2347 s
      * MomentumEnergyIAD: 0.3717 s
      * Timestep: 0.1205 s
      * UpdateQuantities: 0.3322 s
```

(continues on next page)

(continued from previous page)

```
    * EnergyConservation: 0.0222 s
    * SmoothingLength: 0.0759 s
    * %MomentumEnergyIAD: 1.31 %
    * %Timestep: 0.43 %
    * %mpi_synchronizeHalos: 3.0 %
    * %FindNeighbors: 51.55 %
    * %IAD: 0.83 %
    * otf2_serial_time: 4155615
    * otf2_parallel_time: 118870912344
    * otf2_func_compiler_cnt: 176081961
    * otf2_func_compiler_time: 109542525102
    * otf2_func_mpi_cnt: 18112
    * otf2_func_mpi_time: 4931595527
    * otf2_func_openacc_cnt: 63504
    * otf2_func_openacc_time: 4400947330
    * otf2_messages_mpi_cnt: 17072
    * otf2_messages_mpi_size: 10356574336 Bytes
    * otf2_coll_mpi_cnt: 1664
    * otf2_coll_mpi_size: 1577926656 Bytes
    * otf2_rusage: 0 (deactivated)
```

This report is generated from the data collected from the tool and processed in the `self.perf_patterns` part of the *`Class`*. For example, the total size (in Bytes) of MPI communication the data transfers (`otf2_messages_mpi_size`) is extracted with the *`otf2cli_perf_patterns`* method.

Looking at the report with the tool gives more insight into the performance of the code:

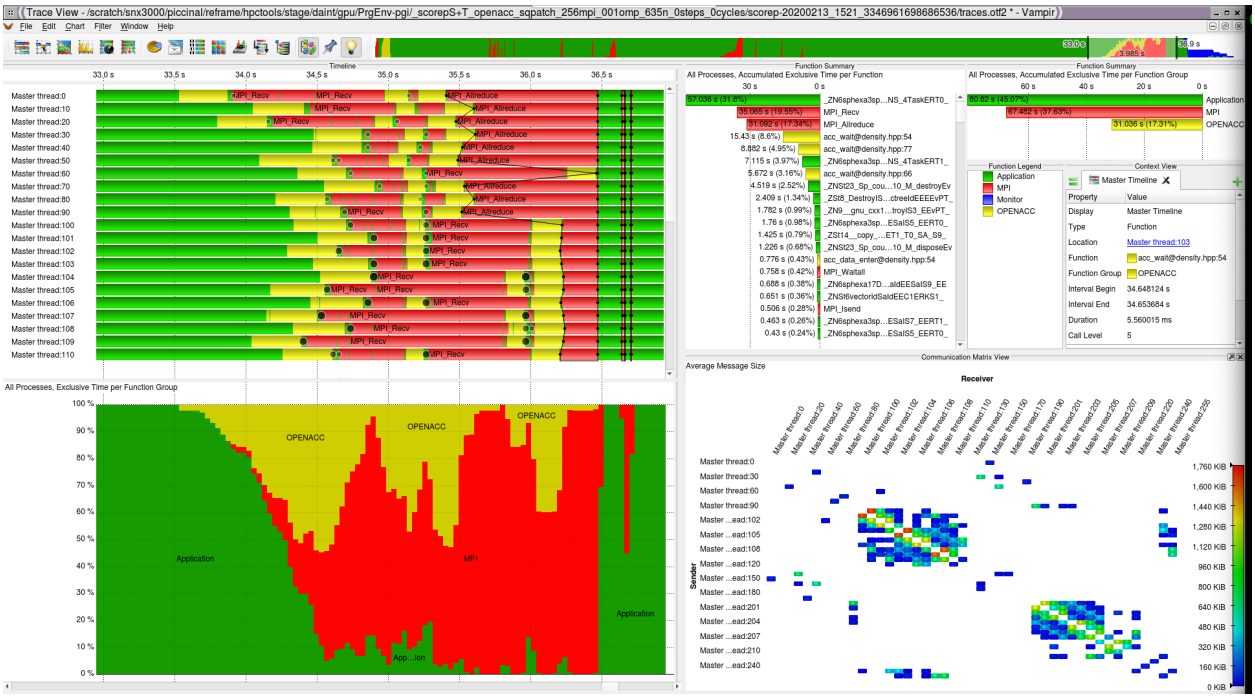

Fig. 2: Score-P Vampir OpenACC (launched with: vampir scorep-/traces.otf2)

CHAPTER

TEN

GPU REFERENCE GUIDE

10.1 Regression tests

10.1.1 nsys_cuda.py

class reframechecks.nvidia.nsys_cuda.**SphExaNsysCudaCheck**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

This class runs the test code with Nvidia nsys systems (2 mpi tasks min) https://docs.nvidia.com/nsight-systems/index.html

Available analysis types are: `nsys profile -help`

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

Typical performance reporting:

Versions:

- cudatoolkit/10.2.89 has nsys/2019.5.2.16-b54ef97
- nvhpc/2020_207-cuda-10.2 has nsys/2020.3.1.54-2bd2a65
- nvidia-nsight-systems/2020.3.1.72 has nsys/2020.3.1.72-e5b8014 <–

class reframechecks.nvidia.nsys_cuda.**SphExaNsysCudaCheck**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

This class runs the test code with Nvidia nsys systems (2 mpi tasks min) https://docs.nvidia.com/nsight-systems/index.html

Available analysis types are: `nsys profile -help`

2 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.

Typical performance reporting:

Versions:

- cudatoolkit/10.2.89 has nsys/2019.5.2.16-b54ef97
- nvhpc/2020_207-cuda-10.2 has nsys/2020.3.1.54-2bd2a65
- nvidia-nsight-systems/2020.3.1.72 has nsys/2020.3.1.72-e5b8014 <–

Sanity checks

reframechecks.common.sphexa.sanity_nvidia.**nsys_perf_patterns**(*obj*)
Dictionary of default nsys_perf_patterns for the tool

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_DtoH_KiB**(*self*)
Reports `[CUDA memcpy DtoH]` Memory Operation (KiB) measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Memory Operation Statistics (KiB)
#
#              Total      Operations             Average             Minimum
# -----------------  --------------  ------------------  ------------------
#         1530313.0             296              5170.0               0.055
#           16500.0              84               196.4              62.500
#           ********
# ...
#             Maximum  Name
#  -----------------  --------------------
#             81250.0  [CUDA memcpy HtoD]
#               250.0  [CUDA memcpy DtoH]
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_DtoH_pct**(*self*)
Reports `[CUDA memcpy DtoH]` Time(%) measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Memory Operation Statistics (nanoseconds)
#
# Time(%)      Total Time  Operations         Average  ...
# -------  --------------  ----------  --------------  ...
#     0.9         1385579          84         16495.0  ...
#    ****
#
#             Minimum         Maximum  Name
#      --------------  --------------  --------------------
#                6144           21312  [CUDA memcpy DtoH]
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_HtoD_KiB**(*self*)
Reports `[CUDA memcpy HtoD]` Memory Operation (KiB) measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Memory Operation Statistics (KiB)
#
#              Total      Operations             Average             Minimum
# -----------------  --------------  ------------------  ------------------
```

(continues on next page)

(continued from previous page)

```
#          1530313.0              296              5170.0              0.055
#          **********
#            16500.0               84               196.4             62.500
# ...
#             Maximum  Name
#  -----------------  --------------------
#             81250.0  [CUDA memcpy HtoD]
#               250.0  [CUDA memcpy DtoH]
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_HtoD_pct**(*self*)

Reports `[CUDA memcpy HtoD]` Time(%) measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Memory Operation Statistics (nanoseconds)
#
# Time(%)      Total Time  Operations         Average  ...
# -------  --------------  ----------  --------------  ...
#    99.1       154400354         296        521622.8  ...
#    ****
#
#             Minimum         Maximum  Name
#      --------------  --------------  --------------------
#                 896         8496291  [CUDA memcpy HtoD]
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_computeIAD_pct**(*self*)

Reports `CUDA Kernel` Time (%) for computeIAD measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Kernel Statistics (nanoseconds)
#
# Time(%)      Total Time   Instances         Average         Minimum
# -------  --------------  ----------  --------------  --------------
#    49.7        69968829           6      11661471.5        11507063
#    26.4        37101887           6       6183647.8         6047175
#    ****
#    24.0        33719758          24       1404989.9         1371531
# ...
#         Maximum  Name
#  --------------  ------------------
#        11827539  computeMomentumAndEnergyIAD
#         6678078  computeIAD
#         1459594  density
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_cudaMemcpy_pct**(*self*)

Reports `CUDA API` Time (%) for cudaMemcpy measured by the tool and averaged over compute nodes

```
> job.stdout

# CUDA API Statistics (nanoseconds)
#
# Time(%)      Total Time       Calls         Average         Minimum
# -------  --------------  ----------  --------------  --------------
#    44.9       309427138         378        818590.3            9709
```

(continues on next page)

(continued from previous page)

```
#    ****
#    40.6       279978449           2     139989224.5           24173
#     9.5        65562201         308        212864.3             738
#     4.9        33820196         306        110523.5            2812
#     0.1          704223          36         19561.8            9305
# ....
#          Maximum  Name
#  --------------  -------------------
#         11665852  cudaMemcpy
#        279954276  cudaMemcpyToSymbol
#          3382747  cudaFree
#           591094  cudaMalloc
#            34042  cudaLaunch
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_report_momentumEnergy_pct**(*self*)

Reports `CUDA Kernel` Time (%) for MomentumAndEnergyIAD measured by the tool and averaged over compute nodes

```
> job.stdout
# CUDA Kernel Statistics (nanoseconds)
#
# Time(%)      Total Time   Instances         Average         Minimum
# -------  --------------  ----------  --------------  --------------
#    49.7        69968829           6      11661471.5        11507063
#    ****
#    26.4        37101887           6       6183647.8         6047175
#    24.0        33719758          24       1404989.9         1371531
# ...
#          Maximum  Name
#  --------------  -------------------
#         11827539  computeMomentumAndEnergyIAD
#          6678078  computeIAD
#          1459594  density
```

reframechecks.common.sphexa.sanity_nvidia.**nsys_version**(*obj*)

Checks tool's version:

```
> nsys --version
NVIDIA Nsight Systems version 2020.1.1.65-085319d
returns: True or False
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_perf_patterns**(*obj*)

Dictionary of default nsys_perf_patterns for the tool

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_DtoH_KiB**(*self*)

Reports `[CUDA memcpy DtoH]` Memory Operation (KiB) measured by the tool and averaged over compute nodes (TODO)

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_DtoH_pct**(*self*)

Reports `[CUDA memcpy DtoH]` Time(%) measured by the tool and averaged over compute nodes

```
> job.stdout (Name: [CUDA memcpy DtoH])
# Time(%)    Time   Calls       Avg       Min       Max
# 2.80%  1.3194ms      44  29.986us  29.855us 30.528us [CUDA memcpy DtoH]
```

(continues on next page)

(continued from previous page)

```
# 1.34%  1.7667ms      44  40.152us  39.519us 41.887us [CUDA memcpy DtoH]
# ^^^^
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_HtoD_KiB**(*self*)
Reports [CUDA memcpy HtoD] Memory Operation (KiB) measured by the tool and averaged over compute nodes (TODO)

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_HtoD_pct**(*self*)
Reports [CUDA memcpy HtoD] Time(%) measured by the tool and averaged over compute nodes

```
> job.stdout (Name: [CUDA memcpy HtoD])
#             Type  Time(%)      Time Calls       Avg   Min       Max
#  GPU activities:   48.57%  22.849ms   162  141.04us 896ns  1.6108ms
#  GPU activities:   56.12%  74.108ms   162  457.45us 928ns  5.8896ms
#                    ^^^^^
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_computeIAD_pct**(*self*)
Reports CUDA Kernel Time (%) for computeIAD measured by the tool and averaged over compute nodes

```
> job.stdout
# (where Name = sphexa::sph::cuda::kernels::computeIAD)
# Time(%)     Time  Calls       Avg       Min       Max  Name
# 12.62%  5.9380ms      4  1.4845ms  1.3352ms  1.6593ms  void ...
# 10.54%  13.915ms      4  3.4788ms  3.3458ms  3.7058ms  void ...
# ^^^^^
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_cudaMemcpy_pct**(*self*)
Reports CUDA API Time (%) for cudaMemcpy measured by the tool and averaged over compute nodes

```
> job.stdout (where Name = cudaMemcpy|cudaMemcpyToSymbol)
#           Time(%) Total Time Calls   Average   Minimum   Maximum  Name
# API calls: 74.37%   219.93ms     2  109.96ms  20.433us  219.90ms  ...
#            18.32%   54.169ms   204  265.53us  11.398us  3.5624ms  ...
# API calls: 54.65%   222.03ms     2  111.02ms  20.502us  222.01ms  ...
#            34.88%   141.73ms   204  694.76us  21.168us  7.5486ms  ...
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_report_momentumEnergy_pct**(*self*)
Reports CUDA Kernel Time (%) for MomentumAndEnergyIAD measured by the tool and averaged over compute nodes

```
> job.stdout
# (where Name = sphexa::sph::cuda::kernels::computeMomentumAndEnergyIAD)
# Time(%)     Time  Calls   Avg       Min       Max  Name
# 28.25%  13.288ms  4  3.3220ms  3.1001ms  3.4955ms  void ...
# 21.63%  28.565ms  4  7.1414ms  6.6616ms  7.4616ms  void ...
# ^^^^^
```

reframechecks.common.sphexa.sanity_nvidia.**nvprof_version**(*obj*)
Checks tool's version:

```
> nvprof --version
nvprof: NVIDIA (R) Cuda command line profiler
Copyright (c) 2012 - 2019 NVIDIA Corporation
```

(continues on next page)

(continued from previous page)

```
Release version 10.2.89 (21)
                ^^^^^^^
returns: True or False
```

10.1.2 scorep_openacc.py

class reframechecks.openacc.scorep_openacc.**SphExaNativeCheck**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

This class runs the test code with Score-P (MPI+OpenACC):

4 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.
- **cycles** – Compiler-instrumented code is required for OpenACC (regions obtained via sampling/unwinding cannot be filtered) => cycles is set to 0.
- **rumetric** – Record Linux Resource Usage Counters to provide information about consumed resources and operating system events such as user/system time, maximum resident set size, and number of page faults: man getrusage

otf_profiler()

class reframechecks.openacc.scorep_openacc.**SphExaNativeCheck**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

This class runs the test code with Score-P (MPI+OpenACC):

4 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks; the size of the cube in the 3D square patch test is set with a dictionary depending on mpitask, but cubesize could also be on the list of parameters,
- **steps** – number of simulation steps.
- **cycles** – Compiler-instrumented code is required for OpenACC (regions obtained via sampling/unwinding cannot be filtered) => cycles is set to 0.
- **rumetric** – Record Linux Resource Usage Counters to provide information about consumed resources and operating system events such as user/system time, maximum resident set size, and number of page faults: man getrusage

otf_profiler()

Sanity checks

reframechecks.common.sphexa.sanity_scorep_openacc.**otf2cli_metric_flag**(*obj*)
If SCOREP_METRIC_RUSAGE is defined then return the `otf-profiler` flags so that it will not segfault.

reframechecks.common.sphexa.sanity_scorep_openacc.**otf2cli_metric_name**(*obj*)
If SCOREP_METRIC_RUSAGE is defined then return the metric name.

reframechecks.common.sphexa.sanity_scorep_openacc.**otf2cli_perf_patterns**(*obj*)
Dictionary of default `perf_patterns` for the tool

reframechecks.common.sphexa.sanity_scorep_openacc.**otf2cli_read_json_rpt**(*obj*)
Reads the json file reported by `otf_profiler`, needed for perf_patterns.

reframechecks.common.sphexa.sanity_scorep_openacc.**otf2cli_tool_reference**(*obj*)
Dictionary of default `reference` for the tool

CHAPTER

ELEVEN

DEBUGGING TOOLS

CSCS provides several debugging tools:

- Arm Forge DDT
- Cray ATP
- GNU gdb
- Nvidia Cuda gdb
- Arm Forge Cuda DDT

11.1 Arm Forge DDT

Arm Forge DDT is a licensed tool that can be used for debugging serial, multi-threaded (OpenMP), multi-process (MPI) and accelerator based (Cuda, OpenACC) programs running on research and production systems, including the CRAY Piz Daint system. It can be executed either as a graphical user interface (ddt `--connect` mode or just ddt) or from the command-line (ddt `--offline` mode).

11.1.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/debug/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--keep-stage-files \
-c ./arm_ddt.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev6 (rev: e0f8d969)
Launched on host: daint101

[---] waiting for spawned checks to finish
[ OK ] (1/1) sphexa_ddt_sqpatch_024mpi_001omp_35n_2steps on daint:gpu using PrgEnv-gnu
```

(continues on next page)

(continued from previous page)

```
[---] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool. In this case, the code is knowingly written in order that the mpi ranks other than 0, 1 and 2 will call `MPI::COMM_WORLD.Abort` thus making the execution to crash.

11.1.2 Bug reporting

An overview of the debugging data will typically look like this:

```
Tracepoints

#   Time      Tracepoint    Processes                                          Values
          main(int,                  domain.clist: std::vector of length 0, capacity␣
↪1786 domain.clist[0]: Sparkline
1 0:07.258 char**)          0-23      from 0 to 19286
          (sqpatch.cpp:75)
          main(int,                  domain.clist: std::vector of length 0, capacity␣
↪1786 domain.clist[0]: Sparkline
2 0:07.970 char**)          0-23      from 0 to 26171
          (sqpatch.cpp:75)
          main(int,                  domain.clist: std::vector of length 0, capacity␣
↪1786 domain.clist[0]: Sparkline
3 0:08.873 char**)          0-23      from 0 to 19097
          (sqpatch.cpp:75)
```

The same data can be viewed with a web browser:

In the same way, using DDT gui will give the same result and more insight about the crash of the code:

11.2 Cray ATP

Cray ATP (Abnormal Termination Processing) is a tool that monitors user applications, and should an application take a system trap, performs analysis on the dying application. All of the stack backtraces of the application processes are gathered into a merged stack backtrace tree and written to disk as the file `atpMergedBT.dot`.

11.2.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/debug/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
```

(continues on next page)

23	⚠	0:16.923	3-23	Process stopped in raise from /lib64/libc.so.6 with signal SIGABRT (Aborted). Reason/Origin: tkill Check the Input/Output view for error messages.
24				Additional Information

▼ Stacks

Processes	Function	Source	Variables
3-23	main (sqpatch.cpp:91)	▶ if (d.rank > 2 && d.iteration > 1) { MPI::COMM_WORLD.Abort(0); }	▶ Rank 13, thread 1
3-23	MPI::Comm::Abort (mpicxx.h:1236)	▶ MPIX_CALLREF(this, MPI_Abort((MPI_Comm) the_real_comm, v2));	▶ Rank 13, thread 1
3-23	PMPI_Abort		
3-23	MPID_Abort		
3-23	abort		
3-23	* raise		

▼ Current Stack

```
#5 main (argc=5, argv=0x7fffffff41e8) at /scratch/snx3000tds/piccinal/reframe/stage/dom/gpu/PrgEnv-gnu/sphexa_ddt_sqpatch_024mpi_001
#4 MPI::Comm::Abort (this=0x63f380, v2=0) at /opt/cray/pe/mpt/7.7.10/gni/mpich-gnu/8.2/include/mpicxx.h:1236 (at 0x0000000000405670)
#3 PMPI_Abort () from /opt/cray/pe/mpt/7.7.10/gni/mpich-gnu/8.2/lib/libmpich_gnu_82.so.3 (at 0x00002aaaab1f15e6)
#2 MPID_Abort () from /opt/cray/pe/mpt/7.7.10/gni/mpich-gnu/8.2/lib/libmpich_gnu_82.so.3 (at 0x00002aaaab2e4268)
#1 abort () from /lib64/libc.so.6 (at 0x00002aaaac44eb01)
#0 raise () from /lib64/libc.so.6 (at 0x00002aaaac44d520)
```

Messages | Tracepoints | Output

Tracepoints

#	Time	Tracepoint	Processes	Values
1	0:07.258	main(int, char**) (sqpatch.cpp:75)	0-23	domain.clist: std::vector of length 0, capacity 1786 domain.clist[0]: ⎯ from 0 to 19286
2	0:07.970	main(int, char**) (sqpatch.cpp:75)	0-23	domain.clist: std::vector of length 0, capacity 1786 domain.clist[0]: ⎯ from 0 to 26171
3	0:08.873	main(int, char**) (sqpatch.cpp:75)	0-23	domain.clist: std::vector of length 0, capacity 1786 domain.clist[0]: ⎯ from 0 to 19097

Fig. 1: ARM Forge DDT html report (created with `--offline --output=rpt.html`)

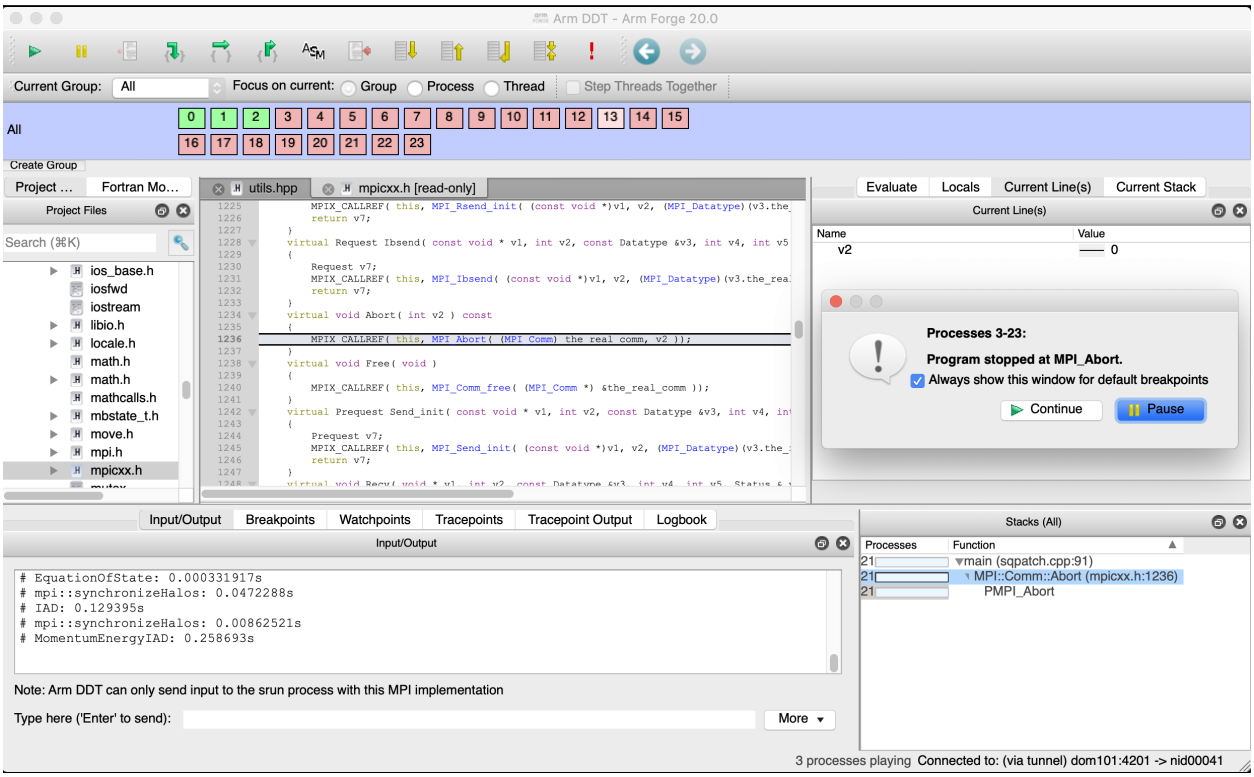

Fig. 2: ARM Forge DDT (All mpi ranks (except 0, 1 and 2) aborted)

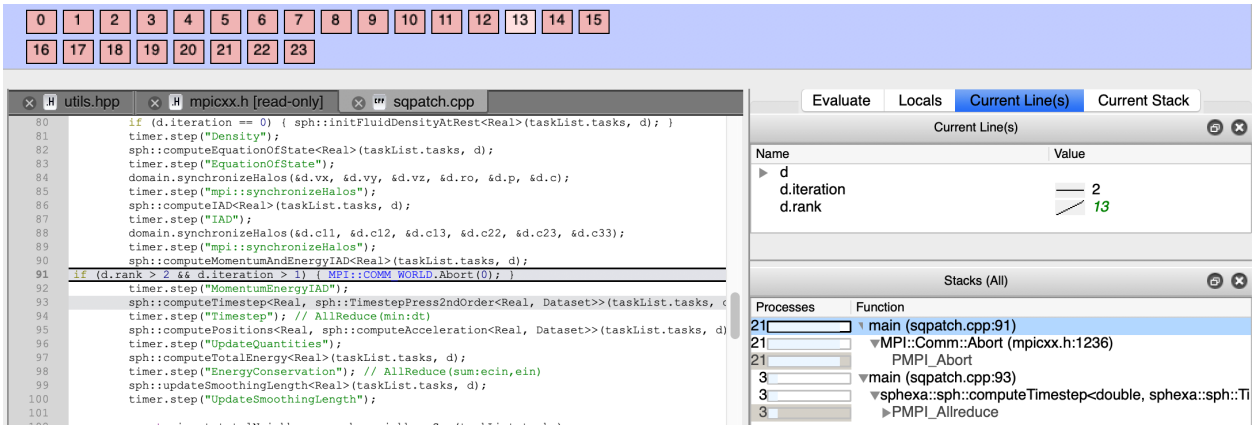

Fig. 3: ARM Forge DDT (callstack)

(continued from previous page)

```
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--keep-stage-files \
-c ./cray_atp.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev6 (rev: e0f8d969)
Launched on host: daint101

[----] waiting for spawned checks to finish
[ OK ] (1/1) sphexa_atp_sqpatch_024mpi_001omp_50n_1steps on daint:gpu using PrgEnv-gnu
[----] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the `Class` shows how to setup and run the code with the tool. In this case, the code is knowingly written in order that the mpi ranks other than 0, 1 and 2 will call `MPI::COMM_WORLD.Abort` thus making the execution to crash.

11.2.2 Bug reporting

An overview of the debugging data will typically look like this:

```
MPI VERSION    : CRAY MPICH version 7.7.10 (ANL base 3.2)
...
Rank 1633 [Tue May  5 19:30:24 2020] [c9-2c0s1n2] application called MPI_Abort(MPI_COMM_
↪WORLD, 7) - process 1633
Rank 1721 [Tue May  5 19:30:24 2020] [c9-2c0s3n1] application called MPI_Abort(MPI_COMM_
↪WORLD, 7) - process 1721
...
Rank 757 [Tue May  5 19:30:24 2020] [c7-1c0s4n1] application called MPI_Abort(MPI_COMM_
↪WORLD, 7) - process 757
Application 22398835 is crashing. ATP analysis proceeding...
```

(continues on next page)

(continued from previous page)

```
ATP Stack walkback for Rank 1743 starting:
  _start@start.S:120
  __libc_start_main@0x2aaaac3ddf89
  main@sqpatch.cpp:85
  MPI::Comm::Abort(int) const@mpicxx.h:1236
  PMPI_Abort@0x2aaaab1f15e5
  MPID_Abort@0x2aaaab2e4267
  __GI_abort@0x2aaaac3f4740
  __GI_raise@0x2aaaac3f3160
ATP Stack walkback for Rank 1743 done
Process died with signal 6: 'Aborted'
Forcing core dumps of ranks 1743, 0
View application merged backtrace tree with: stat-view atpMergedBT.dot
You may need to: module load stat

srun: error: nid04079: tasks 1344-1355: Killed
srun: Terminating job step 22398835.0
srun: error: nid03274: tasks 672-683: Killed
srun: error: nid04080: tasks 1356-1367: Killed
...
srun: error: nid03236: tasks 216-227: Killed
srun: error: nid05581: tasks 1716-1727: Killed
srun: error: nid05583: task 1743: Aborted (core dumped)
srun: Force Terminated job step 22398835.0
```

Several files are created:

```
atpMergedBT.dot
atpMergedBT_line.dot
core.atp.22398835.0.5324
core.atp.22398835.1743.23855
```

These files contains useful information about the crash:

- `atpMergedBT.dot`: File containing the merged backtrace tree at a simple, function-level granularity. This file gives the simplest and most-collapsed view of the application state.
- `atpMergedBT_line.dot`: File containing the merged backtrace tree at a more-complex, source-code line level of granularity. This file shows a denser, busier view of the application state and supports modest source browsing.
- `core.atp.apid.rank`: These are the heuristically chosen core files named after the application ID and rank of the process from which they came.

The corefile contains an image of the process's memory at the time of termination. This image can be opened in a debugger, in this case with gdb:

```
            f'echo ATP_VERSION2='
            f'`pkg-config --modversion libAtpSigHandler` >> {version_rpt}',
            f'echo ATP_HOME=$ATP_HOME >> {version_rpt}',
            f'pkg-config --variable=exec_prefix libAtpSigHandler &>{which_rpt}'
        ]
```

A typical report for rank 0 (or 1) will look like this:

```
Program terminated with signal SIGQUIT, Quit.
#0  0x00002aaaab2539bc in MPIDI_Cray_shared_mem_coll_tree_reduce () from /opt/cray/pe/
↪lib64/libmpich_gnu_82.so.3
#0  0x00002aaaab2539bc in MPIDI_Cray_shared_mem_coll_tree_reduce () from /opt/cray/pe/
↪lib64/libmpich_gnu_82.so.3
#1  0x00002aaaab2653f7 in MPIDI_Cray_shared_mem_coll_reduce () from /opt/cray/pe/lib64/
↪libmpich_gnu_82.so.3
#2  0x00002aaaab265fdd in MPIR_CRAY_Allreduce () from /opt/cray/pe/lib64/libmpich_gnu_82.
↪so.3
#3  0x00002aaaab1756b4 in MPIR_Allreduce_impl () from /opt/cray/pe/lib64/libmpich_gnu_82.
↪so.3
#4  0x00002aaaab176055 in PMPI_Allreduce () from /opt/cray/pe/lib64/libmpich_gnu_82.so.3
#5  0x00000000004097e3 in ?? ()
```

and for other ranks:

```
Program terminated with signal SIGABRT, Aborted.
#0  0x00002aaaac3f7520 in raise () from /lib64/libc.so.6
#0  0x00002aaaac3f7520 in raise () from /lib64/libc.so.6
#1  0x00002aaaac3f8b01 in abort () from /lib64/libc.so.6
#2  0x00002aaaab2e4638 in MPID_Abort () from /opt/cray/pe/lib64/libmpich_gnu_82.so.3
#3  0x00002aaaab1f19a6 in PMPI_Abort () from /opt/cray/pe/lib64/libmpich_gnu_82.so.3
#4  0x0000000000405664 in ?? ()
#5  0x0000000000857bb8 in ?? ()
```

The `atpMergedBT.dot` files can be viewed with `stat-view`, a component of the STAT package (module load stat). The merged stack backtrace tree provides a concise, yet comprehensive, view of what the application was doing at the time of the crash.

11.3 GNU GDB

GDB is the fundamental building block upon which other debuggers are being assembled. GDB allows to see what is going on inside another program while it executes — or what another program was doing at the moment it crashed (core files).

11.3.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/debug/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--keep-stage-files \
-c ./gdb.py
```

A successful ReFrame output will look like the following:

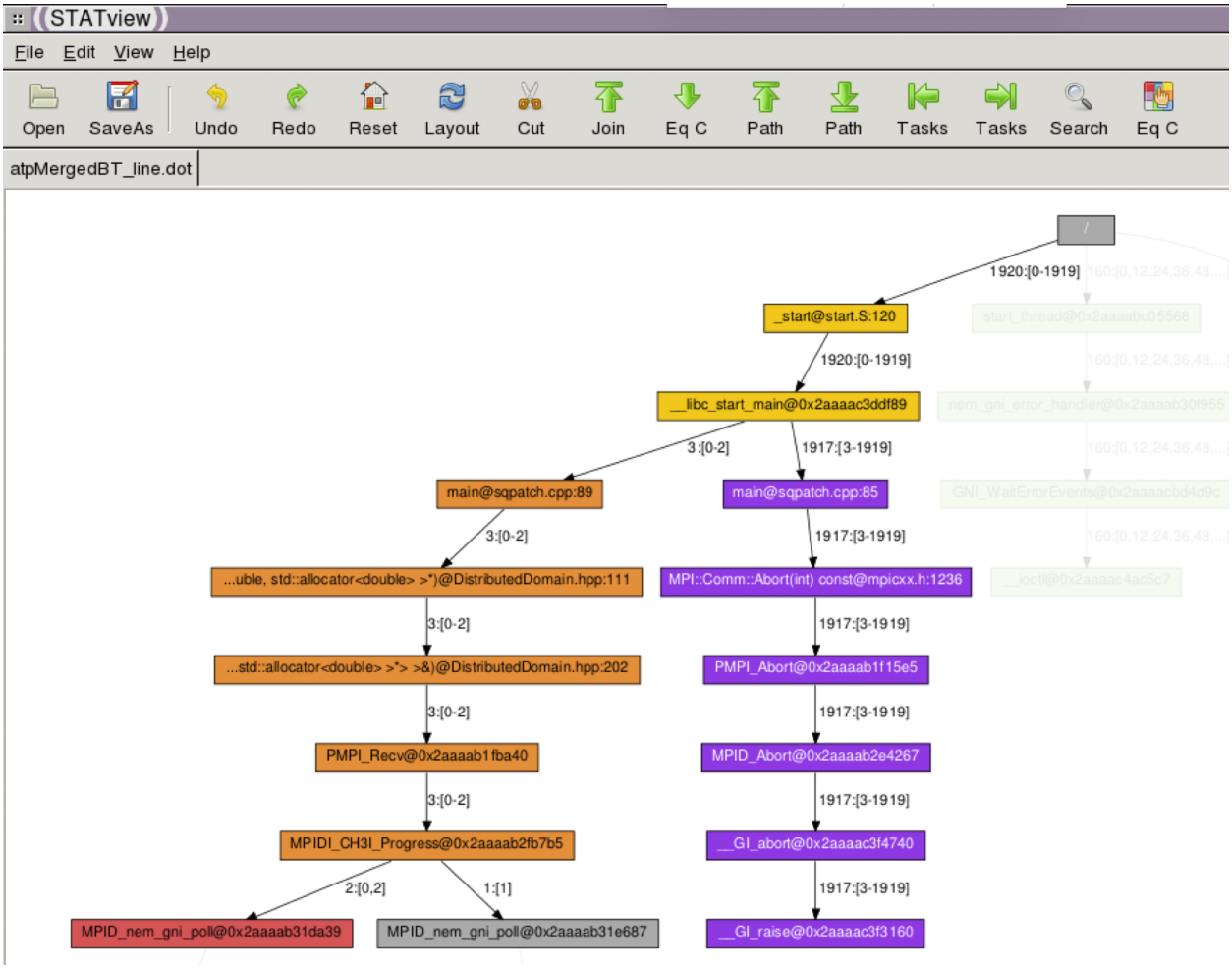

Fig. 4: ATP/STAT (launched with stat-view atpMergedBT_line.dot, 1920 mpi ranks)

```
Reframe version: 3.0-dev6 (rev: e0f8d969)
Launched on host: daint101

[-----] started processing sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps (Tool validation)
[ RUN ] sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps on dom:gpu using PrgEnv-cray
[ RUN ] sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps on dom:gpu using PrgEnv-gnu
[ RUN ] sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps on dom:gpu using PrgEnv-intel
[ RUN ] sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps on dom:gpu using PrgEnv-pgi
[-----] finished processing sphexa_gdb_sqpatch_001mpi_001omp_15n_0steps (Tool validation)

[  PASSED  ] Ran 4 test case(s) from 1 check(s) (0 failure(s))
```

Looking into the *Class* shows how to setup and run the code with the tool. In this example, the code is serial.

11.3.2 Bug reporting

Running gdb in non interactive mode (batch mode) is possible with a input file that specify the commands to execute at runtime:

```
break 75
run -s 0 -n 15
# pretty returns this:
# $1 = std::vector of length 3375, capacity 3375 = {0, etc...
# except for PGI:
#   Dwarf Error: wrong version in compilation unit header
#   (gdb) p domain.clist[0]
#   No symbol "operator[]" in current context.print domain.clist
print domain.clist
print domain.clist[1]
# pvector returns this:
# ------------------------------------------------------------------------
# elem[2]: $3 = 2
# elem[3]: $4 = 3
# elem[4]: $5 = 4
# Vector size = 3375
# Vector capacity = 3375
# Element type = std::_Vector_base<int, std::allocator<int> >::pointer
# ------------------------------------------------------------------------
pvector domain.clist 2 4
continue
quit
```

An overview of the debugging data will typically look like this:

```
Breakpoint 1, main (argc=5, argv=0x7fffffff5f28) at sqpatch.cpp:75
75                  taskList.update(domain.clist);
$1 = std::vector of length 3375, capacity 3375 = {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, 28, 29, 30, 31, 32, 33,
↪ 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,␣
↪55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
↪ 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,␣
↪98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
↪ 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,␣
↪133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,␣
↪150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,␣
↪167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,␣
↪184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199...}
```

(continues on next page)

(continued from previous page)

```
$2 = 1
elem[2]: $3 = 2
elem[3]: $4 = 3
elem[4]: $5 = 4
Vector size = 3375
Vector capacity = 3375
Element type = std::_Vector_base<int, std::allocator<int> >::pointer

# Total execution time of 0 iterations of SqPatch: 0.305378s
[Inferior 1 (process 19366) exited normally]
```

11.4 NVIDIA CUDA GDB

CUDA-GDB is the NVIDIA tool for debugging CUDA applications running on GPUs.

11.4.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/debug/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--keep-stage-files \
-c ./cuda_gdb.py
```

A successful ReFrame output will look like the following:

```
Reframe version: 3.0-dev6 (rev: 3f0c45d4)
Launched on host: daint101

[----------] started processing sphexa_cudagdb_sqpatch_001mpi_001omp_30n_0steps (Tool␣
↪validation)
[ RUN      ] sphexa_cudagdb_sqpatch_001mpi_001omp_30n_0steps on daint:gpu using PrgEnv-
↪gnu
[----------] finished processing sphexa_cudagdb_sqpatch_001mpi_001omp_30n_0steps (Tool␣
↪validation)

[----------] waiting for spawned checks to finish
[       OK ] (1/1) sphexa_cudagdb_sqpatch_001mpi_001omp_30n_0steps on daint:gpu using␣
↪PrgEnv-gnu
[----------] all spawned checks have finished
```

Looking into the *Class* shows how to setup and run the code with the tool.

11.4.2 Bug reporting

Running cuda-gdb in batch mode is possible with a input file that specify the commands to execute at runtime:

```
break main
run -s 0 -n 15
break 75
info br
continue
p domain.clist
# $1 = std::vector of length 1000, capacity 1000 = {0, 1, 2, ...
ptype domain.clist
# type = std::vector<int>
print "info cuda devices"
set logging on info_devices.log
print "info cuda kernels"
set logging on info_kernels.log
show logging
print "info cuda threads"
set logging on info_threads.log
```

cuda-gdb supports user-defined functions (via the define command):

```
# set trace-commands off
define mygps_cmd
set trace-commands off
printf "gridDim=(%d,%d,%d) blockDim=(%d,%d,%d) blockIdx=(%d,%d,%d) threadIdx=(%d,%d,
↪%d) warpSize=%d thid=%d\n", gridDim.x, gridDim.y, gridDim.z, blockDim.x, blockDim.y,␣
↪blockDim.z, blockIdx.x, blockIdx.y, blockIdx.z, threadIdx.x, threadIdx.y, threadIdx.z,␣
↪warpSize, blockDim.x * blockIdx.x + threadIdx.x
```

You can also extend GDB using the Python programming language. An example of GDB's Python API usage is:

```
import re
txt=gdb.execute('info cuda devices', to_string=True)
regex = r'\s+sm_\d+\s+(\d+)\s+'
res = re.findall(regex, txt)
gdb.execute('set $sm_max = %s' % res[0])
```

An overview of the debugging data will typically look like this:

```
PERFORMANCE REPORT
------------------------------------------------------------------------------
sphexa_cudagdb_sqpatch_001mpi_001omp_30n_0steps
- daint:gpu
   - PrgEnv-gnu
      * num_tasks: 1
      * info_kernel_nblocks: 106
      * info_kernel_nthperblock: 256
      * info_kernel_np: 27000
      * info_threads_np: 27008
      * SMs: 56
      * WarpsPerSM: 64
      * LanesPerWarp: 32
```

(continues on next page)

(continued from previous page)

```
      * max_threads_per_sm: 2048
      * max_threads_per_device: 114688
      * best_cubesize_per_device: 49
      * cubesize: 30
      * vec_len: 27000
      * threadid_of_last_sm: 14335
      * last_threadid: 27007
--------------------------------------------------------------------------------
```

It gives information about the limits of the gpu device:

cuda	thread	warp	sm	P100
threads	1	32	2'048	114'688
warps	x	1	64	3'584
sms	x	x	1	56
P100	x	x	x	1

It can be read as: one P100 gpu leverages up to 32 threads per warp, 2048 threads per sm, 114'688 threads per device, 64 warps per sm, 3'584 warps per device, 56 sms per device and so on.

11.5 NVIDIA CUDA and ARM Forge DDT

Arm Forge DDT can be used for debugging GPU parallel codes.

11.5.1 Running the test

The test can be run from the command-line:

```
module load reframe
cd hpctools.git/reframechecks/debug/

~/reframe.git/reframe.py \
-C ~/reframe.git/config/cscs.py \
--system daint:gpu \
--prefix=$SCRATCH -r \
-p PrgEnv-gnu \
--keep-stage-files \
-c ./arm_ddt_cuda.py
```

A successful ReFrame output will look like the following:

```
[----------] started processing sphexa_cudaddt_sqpatch_001mpi_001omp_30n_0steps (Tool␣
↪validation)
[ RUN      ] sphexa_cudaddt_sqpatch_001mpi_001omp_30n_0steps on daint:gpu using PrgEnv-
↪gnu
[----------] finished processing sphexa_cudaddt_sqpatch_001mpi_001omp_30n_0steps (Tool␣
↪validation)

[----------] waiting for spawned checks to finish
```

(continues on next page)

(continued from previous page)

```
[       OK ] (1/1) sphexa_cudaddt_sqpatch_001mpi_001omp_30n_0steps on daint:gpu using␣
↪PrgEnv-gnu
[----------] all spawned checks have finished

[  PASSED  ] Ran 1 test case(s) from 1 check(s) (0 failure(s))
==============================================================================
PERFORMANCE REPORT
------------------------------------------------------------------------------
sphexa_cudaddt_sqpatch_001mpi_001omp_30n_0steps
- daint:gpu
   - PrgEnv-gnu
      * num_tasks: 1
      * elapsed: 113 s
------------------------------------------------------------------------------
```

Looking into the `Class` shows how to setup and run the code with the tool.

11.5.2 Bug reporting

DDT will automatically set a breakpoint at the entrance of cuda kernels.

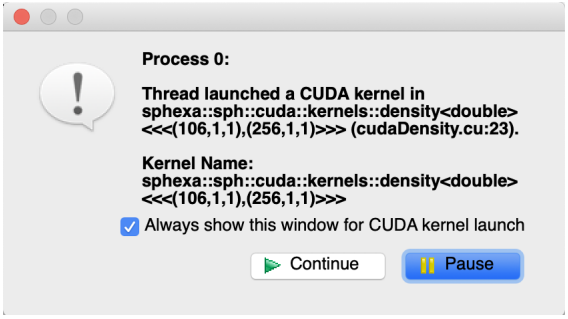

Fig. 5: Arm Forge DDT break on cuda kernel launch

In this example, the first cuda kernel to be launched is the density kernel:

The Thread Selector allows to select a gpu thread and/or threadblock.

Arm DDT also includes a GPU Devices display that gives information about the gpu device:

Table 1: gpu device info

cuda	thread	warp	sm	P100
threads	1	32	2'048	114'688
warps	x	1	64	3'584
sms	x	x	1	56
P100	x	x	x	1

It can be read as: one NVIDIA Pascal P100 gpu leverages up to 32 threads per warp, 2048 threads per sm, 114'688 threads per device, 64 warps per sm, 3'584 warps per device, 56 sms per device and so on.

As usual, it is possible to inspect variables on the cpu and on the gpu:

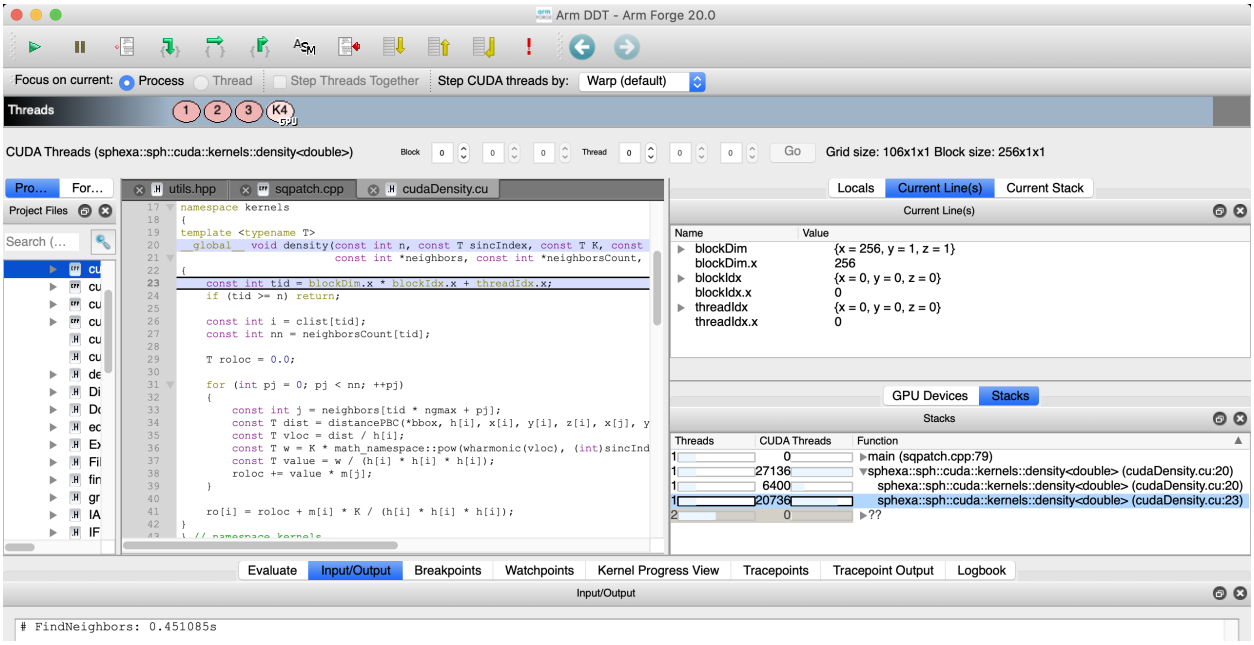

Fig. 6: Arm Forge DDT density kernel (block 0, thread 0)

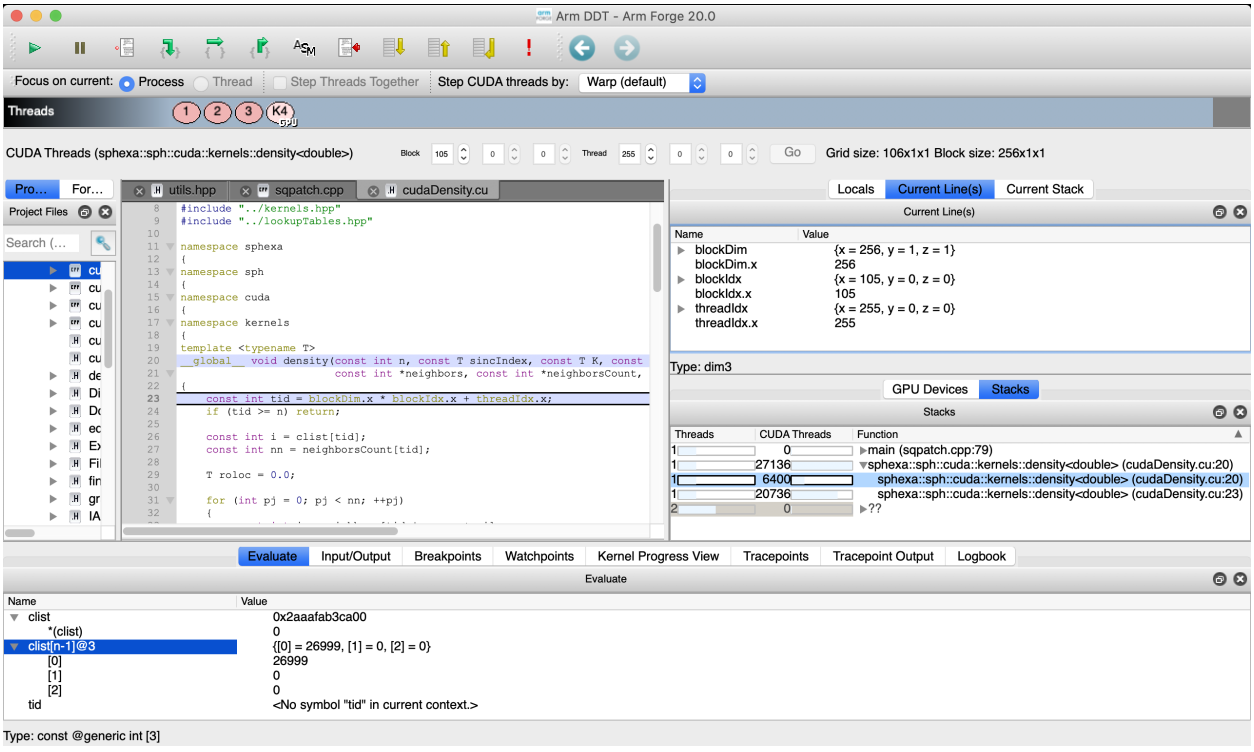

Fig. 7: Arm Forge DDT density kernel (last block, last thread)

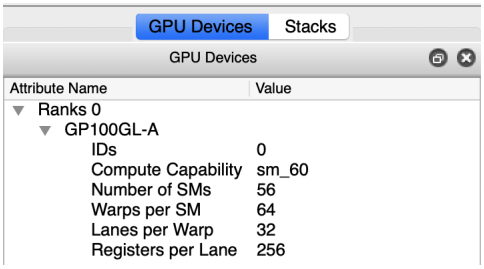

Fig. 8: Arm Forge DDT gpu devices info

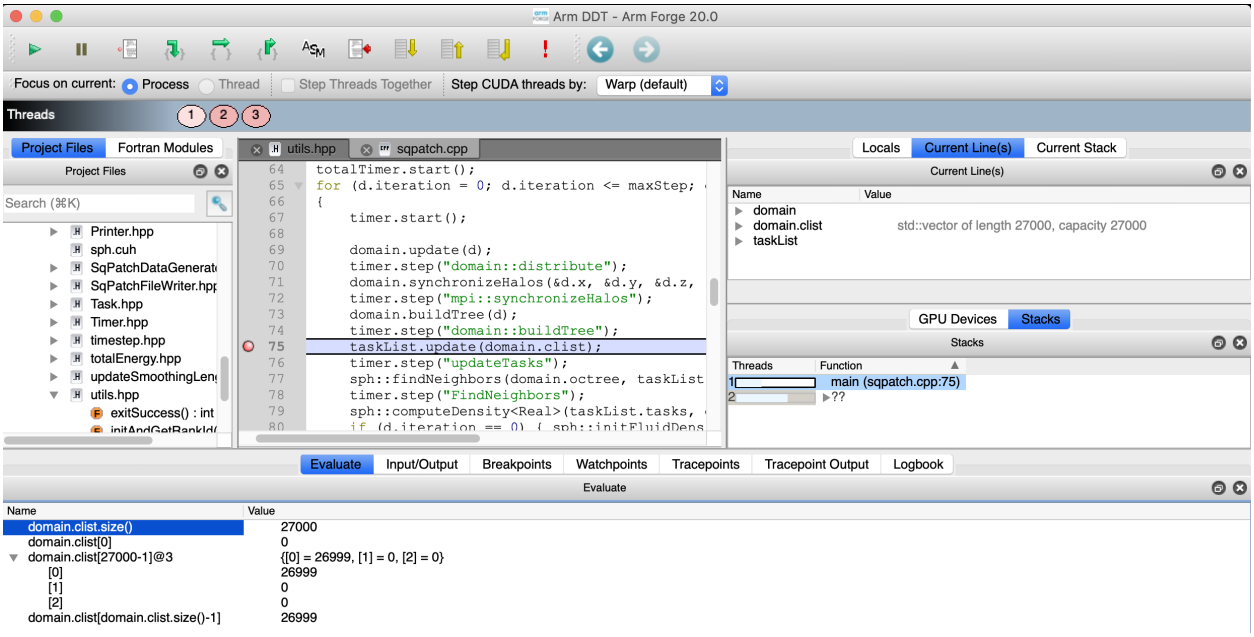

Fig. 9: Arm Forge DDT variables (cpu)

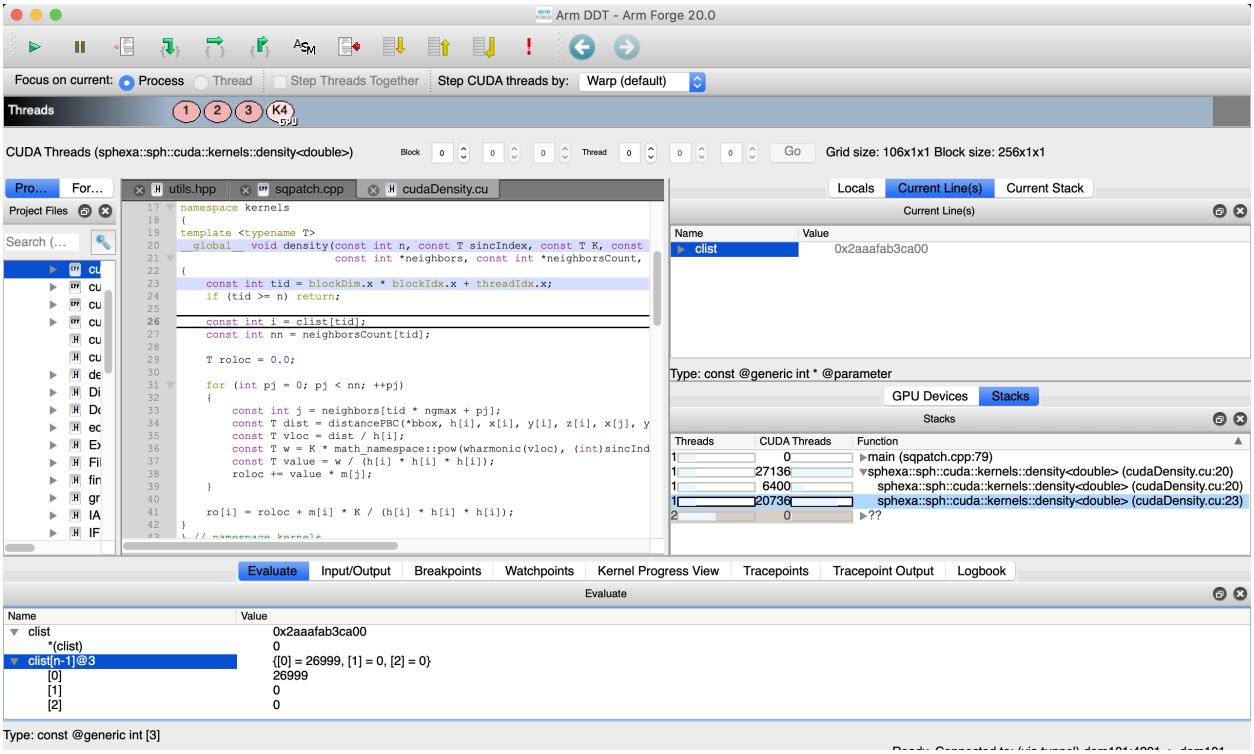

Fig. 10: Arm Forge DDT variables (gpu)

Note: GPU execution under the control of a debugger is not as fast as running without a debugger.

Running ddt with a tracepoint allows to specify the variables to record at runtime in batch mode. This is done in the *set_launcher* method. An overview of the debugging data will typically look like this in the html report:

Tracepoints

#	Time	Tracepoint	Processes	Values
1	0:19.039	sphexa::sph::cuda::kernels::density<double>(int, double, double, int, sphexa::BBox<double> const*, int const*, int const*, int const*, double const*, double const*, double const*, double const*, double const*, double*) (cudaDensity.cu:26)	0	clist[27000-1]@3: {[0] = 26999, [1] = 0, [2] = 0} clist: — 0x2aaafab3ca00

Fig. 11: Arm Forge DDT html report (tracepoints)

and similarly in the txt report:

```
Tracepoints

#   Time              Tracepoint                Processes                 ␣
↪     Values
          sphexa::sph::cuda::kernels::density
          <double>(int, double, double, int,
          sphexa::BBox<double> const*, int
1 0:17.610 const*, int const*, int const*,      0          clist[27000-1]@3: {[0] = 26999,␣
↪[1] = 0, [2] = 0} clist: Sparkline
          double const*, double const*,                    0x2aaafab3ca00
          double const*, double const*,
```

(continues on next page)

(continued from previous page)

```
          double const*, double*)
          (cudaDensity.cu:26)
          sphexa::sph::cuda::kernels::density
```

CHAPTER

TWELVE

DEBUGGING REFERENCE GUIDE

12.1 Regression tests

12.1.1 arm_ddt.py

12.1.2 cray_atp.py

12.1.3 gdb.py

class reframechecks.debug.gdb.**SphExaGDBCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with gdb (serial), 3 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks,
- **cubesize** – size of the cube in the 3D square patch test,
- **steps** – number of simulation steps.

12.1.4 cuda_gdb.py

class reframechecks.debug.cuda_gdb.**SphExaCudaGdbCheck**(**args: Any*, ***kwargs: Any*)

Bases: reframe.

This class runs the test code with cuda-gdb, 3 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks,
- **cubesize** – size of the cube in the 3D square patch test,
- **steps** – number of simulation steps.

set_sanity_gpu()

12.1.5 arm_ddt_cuda.py

class reframechecks.debug.arm_ddt_cuda.**SphExaCudaDDTCheck**(**args: Any*, ***kwargs: Any*)
Bases: reframe.

This class runs the test code with Arm Forge DDT (cuda+mpi), 3 parameters can be set for simulation:

Parameters

- **mpitask** – number of mpi tasks,
- **cubeside** – set to small value as default,
- **steps** – number of simulation steps.

elapsed_time_from_date()

set_compiler_flags()

set_launcher()

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

N

O

P

R

S

T

V