Co-funded by the European Union

MultiXscale

EuroHPC Joint Undertaking

EuroHPC JU Centre of Excellence

EESSI test suite

10th EasyBuild User Meeting@ Juelich

Thu 27 March 2025

EESSI
EUROPEAN ENVIRONMENT FOR SCIENTIFIC SOFTWARE INSTALLATIONS

Caspar van Leeuwen (SURF)
Lara Peeters (Ugent)
Sam Moors (VUB)

cecam
Centre Européen de Calcul Atomique et Moléculaire

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and countries participating in the project. Neither the European Union nor the granting authority can be held responsible for them.

The EESSI test suite

Goal of the EESSI test suite

- To test the functionality and performance of the EESSI software stack on a wide range of systems

The opportunity

- Same software (modules) on every system

The challenge

- Every system is different! Need tests that are *portable*

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Writing portable tests is challenging…

- EESSI test suite is based on ReFrame
- ReFrame tests are *typically* very system specific, example attributes:
  - `valid_systems`: on which systems this test can/should run
  - `num_cpus_per_task`, `num_tasks`, `num_gpus_per_node`: typically chosen to match the system
  - And many more …
- ReFrame offers *amazing* fine-grained control, but at the cost of portability

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

How we make EESSI tests portable

- All system-specific information goes into ReFrame config file
- Make the test do something sensible *based on the config file*, examples:
  - Launch one rank per available (physical) CPU core / numa node / socket / GPU
  - Skip a test if the system has insufficient memory to run it
  - Skip a test that requires GPUs if the system doesn't have any

N.B. Tests ≠ benchmarks! These portable tests are *not* guaranteed to get the best performance from your system for a particular use case, they are meant to spot performance changes.

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

MPI4PY example

```
@rfm.simple_test
class EESSI_MPI4PY(rfm.RunOnlyRegressionTest, EESSI_Mixin):
    device_type = DEVICE_TYPES[CPU]
    compute_unit = COMPUTE_UNIT[CPU]

    module_name = parameter(find_modules('mpi4py'))

    n_iterations = variable(int, value=1000)
    n_warmup = variable(int, value=100)

    executable = 'python3'
    executable_opts = ['mpi4py_reduce.py', '--n_iter', f'{n_iterations}', '--n_warmup', f'{n_warmup}']

    time_limit = '5m00s'

    bench_name = 'mpi4pi'
    bench_name_ci = 'mpi4pi'

    readonly_files = ['mpi4py_reduce.py']

    def required_mem_per_node(self):
        return self.num_tasks_per_node * 100 + 250

    @sanity_function
    …
    @performance_function('s')
    …
```

Requires ‘CPU’ feature

Launch one task per core

Create tests for all modules called mpi4py/<something>

Automatically tags a test instance with this bench_name with ‘CI’ tag

Request sufficient memory, and skip if nodes don’t have enough

Multiscale

See https://www.eessi.io/docs/test-suite/writing-portable-tests/#as-portable-reframe-test

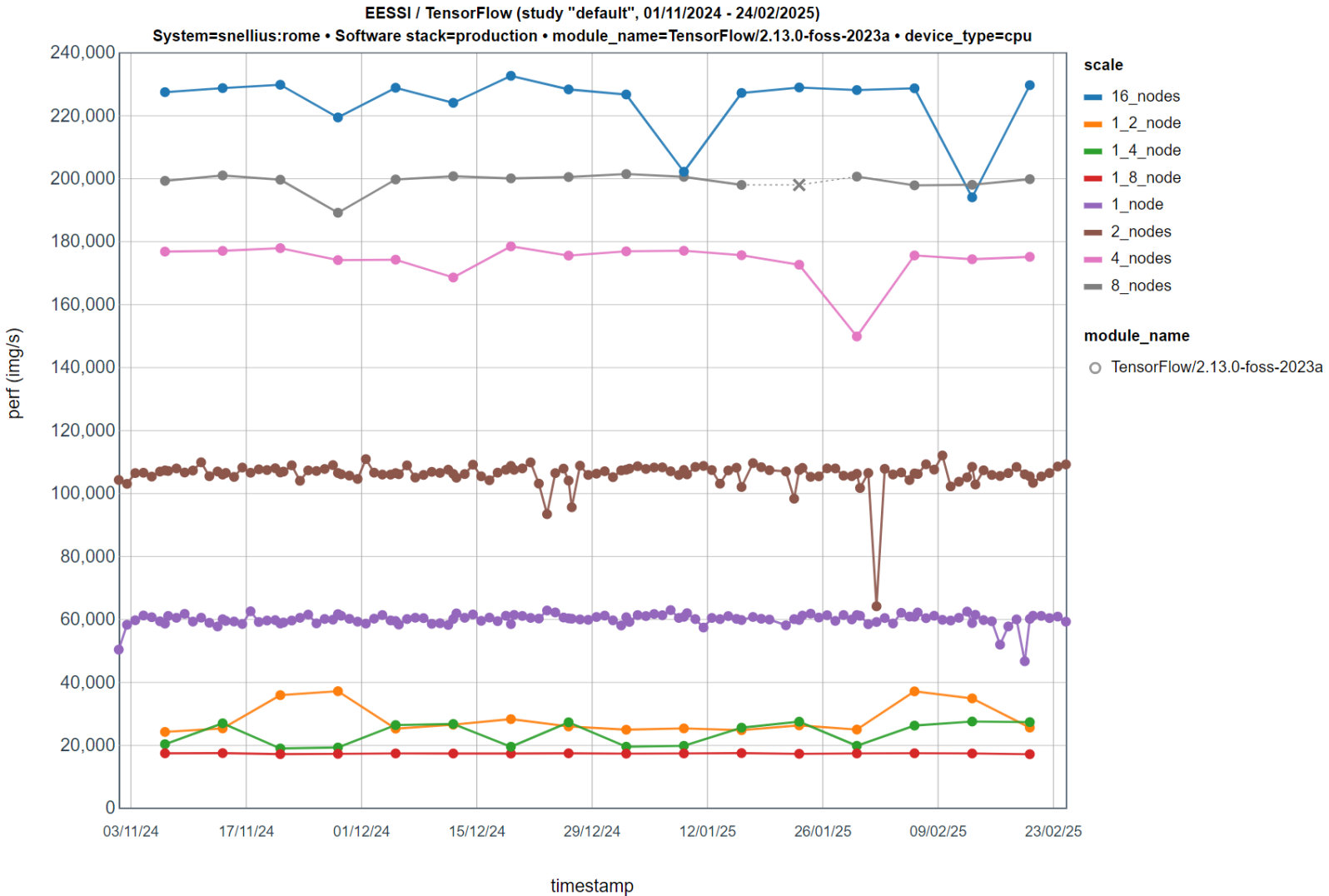
This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

EuroHPC Joint Undertaking

How we use the EESSI test suite

- We run the test suite periodically (daily/weekly) on about 5 different systems
- We run selected (single node) tests when building new software for EESSI (before deployment)
- Some of us also use the test suite to test local module stacks!

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169



Writing an EESSI test suite configuration

Goal: For everyone to have run the EESSI test suite on your HPC cluster (or laptop) by the end of EUM'25!

- Step 1: install ReFrame & the EESSI test suite
- Step 2: create a ReFrame configuration file
- Step 3: run reframe --list -t CI
- Step 4: run reframe --dryrun -t CI -n /<somehash>
- Step 5: run reframe --run -t CI -n /<somehash>

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Find presentation online

- Detailed steps on subsequent slides
- Quickest way: copy-paste from slides at https://github.com/casparvl/EUM25

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 1: Install ReFrame & EESSI test suite

```
module purge  # Use system python
python3 -m venv $HOME/eessi_testsuite/eessi_testsuite_venv
source $HOME/eessi_testsuite/eessi_testsuite_venv/bin/activate
pip install reframe-hpc
pip install eessi-testsuite
# Check we can use things from ReFrame's hpctestlib
python3 -c 'import hpctestlib.sciapps.gromacs'
# Check we can use things from the EESSI testsuite
python3 -c 'import eessi.testsuite.eessi_mixin'
```

See: https://www.eessi.io/docs/test-suite/installation-configuration/

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 2a: create ReFrame config file

```
cd $HOME/eessi_testsuite/
wget https://raw.githubusercontent.com/casparvl/test-suite/refs/heads/update_example_settings/config/settings_example.py
export RFM_CONFIG_FILES=$HOME/eessi_testsuite/settings_example.py
export RFM_PREFIX=$HOME/eessi_testsuite/reframe_runs
export RFM_CHECK_SEARCH_PATH=$HOME/eessi_testsuite/eessi_testsuite_venv/lib/python3.9/site-packages/eessi/testsuite/tests/
export RFM_CHECK_SEARCH_RECURSIVE=1
```


This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 2b: create ReFrame config file

Now, modify `settings_example.py` to match your system

- Define a `stagedir` on a shared filesystem
- Select the matching `scheduler` https://reframe-hpc.readthedocs.io/en/stable/config_reference.html#config.systems.partitions.scheduler (set `local` if you are doing this on your laptop)
- Select the matching parallel `launcher` https://reframe-hpc.readthedocs.io/en/stable/config_reference.html#config.systems.partitions.launcher (`mpirun` *should* work for everyone, but you can use e.g. `srun`)
- Modify the `access` field to define arguments to be passed to the scheduler, etc. It should define a homogeneous set of nodes

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 2c: create ReFrame config file

Now, modify `settings_example.py` to match your system

- Under `resources` set the flag that should be passed to your scheduler to define required memory per node and pass `{size}` as argument
  - Slurm users: '`--mem={size}`'
  - Local spawner: '`--whatever={size}`' (unused)
- Define the max available memory per node under the `EXTRAS.MEM_PER_NODE` item (in MiB).
  - SLURM users: check `scontrol show node <nodename>` for the `RealMemory` on your nodes.
  - Local spawner: put anything (unused)

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 2d: create ReFrame config file

Now, modify `settings_example.py` to match your system

- Under `features` specify what FEATURES (CPU/GPU) and SCALES your system support
  - CPU partition: `'features': [FEATURES.CPU],`
  - GPU partition where you don't want to run CPU-only tests: `'features': [FEATURES.GPU],`
  - GPU partition where you also want to run CPU-only tests: `'features': [FEATURES.CPU, FEATURES.GPU],`
  - To run all scales (up to 16 nodes): `'features': [FEATURES.XYZ] + list(SCALES.keys())`
  - To run only single (full) node (e.g. local laptop): `'features': [FEATURES.XYZ] + [key for key, value in SCALES.items() if value.get("num_nodes") == 1]`
- GPU partitions only: under extras define `EXTRAS.GPU_VENDOR: GPU_VENDORS.NVIDIA`

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 3: run reframe --list -t CI

Run `reframe --list -t CI`

- You may get things like "`WARNING: skipping test 'EESSI_TensorFlow': the following parameters are undefined: module_name`". That's ok, it simply means you don't have the software(module) needed to run this test

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 3: run reframe --list -t CI

Run `reframe --list -t CI`

- If you get "`WARNING: failed to retrieve remote processor info: command 'sbatch rfm-detect-job.sh' failed with exit code 1:`", ReFrame's automatic CPU detection failed.
  - Check the ReFrame log ("`Log file(s) saved in '/path/to/log'`")
  - You might be missing `access` arguments
  - If it keeps failing, you could try 'manually' running `reframe --detect-host-topology` on the relevant node https://www.eessi.io/docs/test-suite/ReFrame-configuration-file/#create-topology-file . Then copy to `~/.reframe/topology/<system>-<partition>/processor.json`

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 3: run reframe --list -t CI

Run `reframe --list -t CI`

- You'll need to have at least one module available for which we have a test ☺
  - If you don't, simply install e.g. a CPU version of OSU-MicroBenchmarks with EasyBuild
- Expected output:

```
...
- EESSI_TensorFlow %scale=2_nodes %module_name=TensorFlow/2.13.0-foss-2023a %device_type=cpu /cbc475c5
- EESSI_TensorFlow %scale=1_node %module_name=TensorFlow/2.13.0-foss-2023a %device_type=cpu /9864d0f5
```


Test hash

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 4: run reframe --dryrun -t CI

Run `reframe --dryrun -t CI -n /<testhash>` to just run an individual test as an example

- Check the jobscript ReFrame will generate & submit in `<stagedir>/<system_name>/<partition_name>/default/<testname_testhash>/rfm_job.sh`
- If you have issues, that job script is (probably) your first place to look!

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Step 5: run reframe --run -t CI

Run `reframe --run -t CI -n /<testhash>` to just run an individual test as an example

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

Summary

- Writing the ReFrame config requires some knowledge specific to the EESSI test suite, but is documented https://www.eessi.io/docs/test-suite/ReFrame-configuration-file/
- Apart from the ReFrame config, the EESSI test suite is ‘plug-and-play’!
- Number of supported applications is could be bigger – open to new contributions (see https://www.eessi.io/docs/test-suite/writing-portable-tests/)
- EESSI test suite is application-focused: no replacement for system-specific (e.g. scheduler or filesystem) tests
- Credit to ReFrame devs: EESSI test suite is possible because they spent time on our bug reports & feature requests ☺

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No. 101093169

MultiXscale

Web page: multixscale.eu

Facebook: MultiXscale

X: @MultiXscale

LinkedIn: multixscale

YouTube: @MultiXscale

Co-funded by the European Union

EuroHPC Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and countries participating in the project under grant agreement No 101093169.