ReFrame: A Regression Testing and Continuous Integration Framework for HPC systems

FOSDEM’19
Victor Holanda Rusu, CSCS
February 3rd, 2019

reframe@sympa.cscs.ch
https://eth-cscs.github.io/reframe
https://github.com/eth-cscs/reframe
https://reframe-slack.herokuapp.com
Background

- CSCS had a shell-script based regression suite
  - Tests very tightly coupled to system details
  - Lots of code replication across tests
  - 15K lines of test code

- Simple changes required significant team effort
  - Porting all tests to native SLURM took several weeks

- Fixing even simple bugs was a tedious task
  - Tens of regression test files had to be fixed
What is ReFrame?

A new regression testing framework that

- allows writing portable HPC regression tests in Python,
- abstracts away the system interaction details,
- lets users focus solely on the logic of their test.

https://eth-cscs.github.io/reframe

https://github.com/eth-cscs/reframe
Timeline / ReFrame Evolution

- **ReFrame starts as a pilot project**: 03/16
- **Production ReFrame 2.0**: 12/16
  - 5x reduction in tests code; more coverage
- **First public release ReFrame 2.4**: 04/17
  - Asynchronous execution of tests
- **Development moves on Github**: 02/18
  - CSCS checks published
- **ReFrame 2.16**: 02/19
  - 18 forks
  - 35 stargazers
Design Goals

- Productivity
- Portability
- Speed and Ease of Use
- Robustness

*Write once, test everywhere!*
Key Features

- Separation of system and prog. environment configuration from test’s logic
- Support for cycling through prog. environments and system partitions
- Regression tests written in Python
  - Easy customization of tests
  - Flexibility in organizing the tests
- Support for sanity and performance tests
  - Allows complex and custom analysis of the output through an embedded mini-language for sanity and performance checking.
- Progress and result reports
- Performance logging with support for Graylog
- Clean internal APIs that allow the easy extension of the framework’s functionality
- Complete documentation (tutorials, reference guide)
- ... and more ([https://github.com/eth-cscs/reframe](https://github.com/eth-cscs/reframe))
ReFrame’s architecture

Developer of regression tests

@rfm.simple_test
class MyTest(rfm.RegressionTest):

ReFrame Frontend

Regression Test API

System abstractions

Environment abstractions

Job schedulers

Job launchers

Build systems

Environment modules

Operating System

reframe -r

Pluggable backends
import reframe as rfm
import reframe.utility.sanity as sn

@rfm.simple_test
class Example7Test(rfm.RegressionTest):
    def __init__(self):
        super().__init__()
        self descr = 'Matrix-vector multiplication (CUDA performance test)'
        self valid_systems = ['daint:gpu']
        self valid_prog_environments = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
        self sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
        self build_system = 'SingleSource'
        self build_system.cxxflags = ['-O3']
        self executable_opts = ['4096', '1000']
        self modules = ['cudatoolkit']
        self num_gpus_per_node = 1
        self sanity_patterns = sn.assert_found(r'time for single matrix vector multiplication', self.stdout)
        self perf_patterns = {'perf': sn.extractsingle(r'Performance: s+(?P<Gflops>S+); \(G\)flop/s', self.stdout, 'Gflops', float)}
        self.reference = {'daint:gpu': {'perf': (50.0, -0.1, 0.1)}}
        self maintainers = ['you-can-type-your-email-here']
        self.tags = {'tutorial'}
Writing a Regression Test in ReFrame

Use parameterized tests to create test factories!

In use inheritance to avoid redefining common functionality!

```python
@rfm.simple_test
class ArborBaseTest(rfm.RegressionTest):
    def __init__(self):
        super().__init__()
        if arch_kind == 'haswell':
            self.valid_systems = ['daint:gpu']
        elif arch_kind == 'broadwell':
            self.valid_systems = ['daint:mc', 'tresa']
        elif arch_kind == 'native':
            self.valid_systems = ['tresa']
        self.arch_kind = arch_kind
        self.build_system.config_opts += ['-DARB_VECTORIZE=ON',
                                          '-DARB_ARCH=%s' % self.arch_kind]
```

Use parameterized tests to create test factories!
The Regression Test Pipeline / How ReFrame Executes Tests

A series of well defined phases that each regression test goes through:

1. Start
2. Pick next test
3. Supports system?
   - NO: Go to next phase
   - YES: Check sanity
4. Supports prog. environment?
   - NO: Go to next phase
   - YES: Setup test
5. Compile test
6. Run test
7. Check performance
8. Cleanup test resources

Executed asynchronously
The Regression Test Pipeline / How ReFrame Executes Tests

- Tests may skip some pipeline stages
  - Compile-only tests
  - Run-only tests

- Users may define additional actions before or after every pipeline stage by overriding the corresponding methods of the regression test API.
  - E.g., override the setup stage for customizing the behavior of the test per programming environment and/or system partition.

- Frontend passes through three phases and drives the execution of the tests
  1. Regression test discovery and loading
  2. Regression test selection (by name, tag, prog. environment support etc.)
  3. Regression test listing or execution
Running ReFrame

reframe -C /path/to/config.py -c /path/to/checks -r

- ReFrame uses three directories when running:
  1. **Stage directory**: Stores temporarily all the resources (static and generated) of the tests
     - Source code, input files, generated build script, generated job script, output etc.
     - This directory is removed if the test finishes successfully.
  2. **Output directory**: Keeps important files from the run for later reference
     - Job and build scripts, outputs and any user-specified files.
  3. **Performance log directory**: Keeps performance logs for the performance tests

- ReFrame generates a summary report at the end with detailed failure information.
Running ReFrame (sample output)

---------- Running 1 check(s)
---------- Started on Fri Sep  7 15:32:50 2018

---- started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN ] Example7Test on daint:gpu using PrgEnv-cray
[ OK ] Example7Test on daint:gpu using PrgEnv-cray
[ RUN ] Example7Test on daint:gpu using PrgEnv-gnu
[ OK ] Example7Test on daint:gpu using PrgEnv-gnu
[ RUN ] Example7Test on daint:gpu using PrgEnv-pgi
[ OK ] Example7Test on daint:gpu using PrgEnv-pgi
---- finished processing Example7Test (Matrix-vector multiplication using CUDA)

[ PASSED ] Ran 3 test case(s) from 1 check(s) (0 failure(s))
---------- Finished on Fri Sep  7 15:33:42 2018
Running ReFrame (sample failure)

[==========] Running 1 check(s)
[==========] Started on Fri Sep 7 16:40:12 2018

[--------] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN ] Example7Test on daint:gpu using PrgEnv-gnu
[   FAIL ] Example7Test on daint:gpu using PrgEnv-gnu
[--------] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[   FAILED ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[==========] Finished on Fri Sep 7 16:40:22 2018

==============================================================================
SUMMARY OF FAILURES
==============================================================================

FAILURE INFO for Example7Test
* System partition: daint:gpu
* Environment: PrgEnv-gnu
* Stage directory: /path/to/stage/daint/gpu/PrgEnv-gnu/Example7Test
* Job type: batch job (id=823427)
* Maintainers: ['you-can-type-your-email-here']
* Failing phase: performance
* Reason: sanity error: 50.363125 is beyond reference value 70.0 (l=63.0, u=77.0)
Running ReFrame (examining performance logs)

- `/path/to/reframe/prefix/perflogs/<testname>.log`
  - A single file named after the test’s name is updated every time the test is run
  - Log record output is fully configurable

- ReFrame can also send logs to a Graylog server, where you can plot them with web tools.
Using ReFrame at CSCS
ReFrame @ CSCS / Tests

- Used for continuously testing systems in production
  - Piz Daint: 179 tests
  - Piz Kesch: 75 tests
  - Leone: 45 tests
  - **Total: 241 different tests (reused across systems)**

- Three categories of tests
  1. Production (90min)
     - Applications, libraries, programming environments, profiling tools, debuggers, microbenchmarks
     - Sanity and performance
     - Run nightly by Jenkins
  2. Maintenance (10min)
     - Programming environment sanity and key user applications performance
     - Before/after maintenance sessions
  3. Diagnostics
ReFrame @ CSCS / Production set-up
ReFrame @ CSCS / Production set-up
Using ReFrame with a CI service
ReFrame integration with CI service

- **CSCS CI service**
  - Based on Jenkins
  - Run on CSCS HPC systems
  - On the remote side there is a Jenkins VM that can only run `sbatch` to the compute nodes
  - Integration steps
    1. Add a `Jenkinsfile` to project
    2. Add a batch script for running ReFrame on the compute nodes
    3. Add configuration entry for the target systems
    4. Add ReFrame tests

- **Travis – Github**
  - Runs a VM on the cloud
  - Integration steps
    1. Add `.travis.yml` file
    2. Add configuration entry for the Travis VM
    3. Add ReFrame tests
ReFrame with CSCS CI service
ReFrame with Travis
Conclusions and Future Directions

ReFrame is a powerful tool that allows you to continuously test an HPC environment without having to deal with the low-level system interaction details.

- High-level tests written in Python
- Portability across HPC system platforms
- Comprehensive reports and reproducible methods

- ReFrame is being actively developed with a regular release cycle.

Future directions

- Test dependencies
- Seamless support for containers
- Benchmarking mode

- Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe
Who is running ReFrame
Acknowledgements

- Framework contributions
  - Andreas Jocksch
  - Christopher Bignamini
  - Matthias Kraushaar
  - Rafael Sarmiento
  - Samuel Omlin
  - Theofilos Manitaras
  - Vasileios Karakasis
  - Victor Holanda

- Regression tests
  - SCS and OPS team
Thank you for your attention.

reframe@sympa.cscs.ch
https://eth-cscs.github.io/reframe
https://github.com/eth-cscs/reframe
https://reframe-slack.herokuapp.com