

Understanding exascale performance through benchmarking



Ilektra Christidi¹, Tuomas Koskela¹, Mose Giordano¹, Emily Dubrovskaja¹
Tom Deakin², Kaan Olgu², Chris Maynard³, Dave Case⁴

1 - Advanced Research Computing Centre, UCL

2 - Advanced Computer Systems, University of Bristol

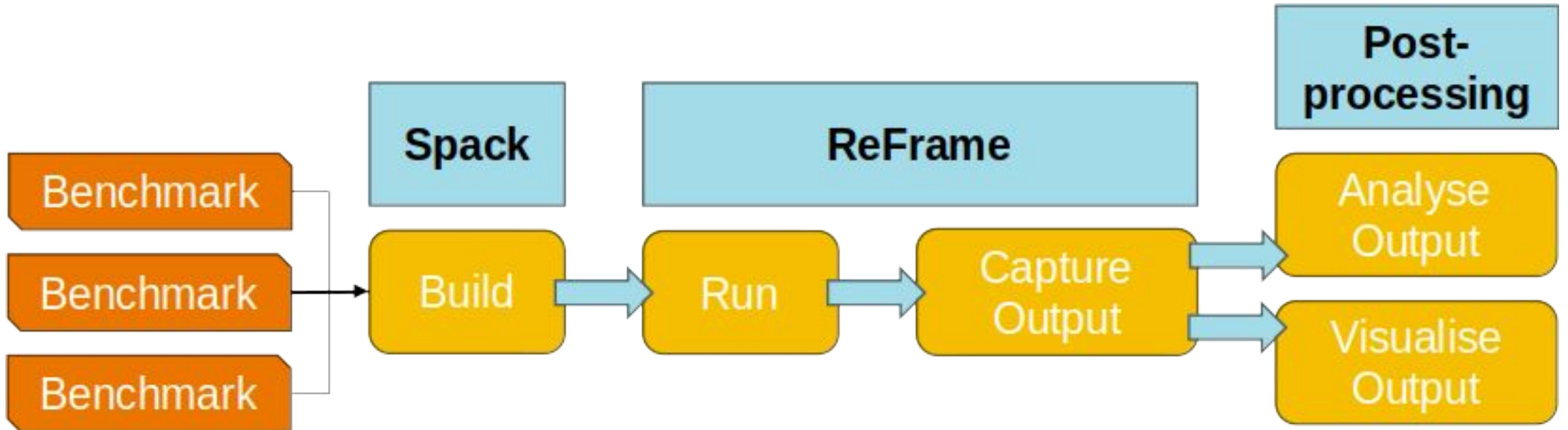
3 - Met Office

4 - University of Reading

Summary

- It's important to understand how our applications perform on different HPC architectures in order to make informed decisions about **future hardware** and **future software**
- Benchmarking, as it's traditionally done, requires a lot of manual effort, is time-consuming, **error prone and difficult to reproduce**
- We've developed a framework using **Spack** and **ReFrame** to **automate the manual effort** in portable building and running of benchmarks.
- Adopting it takes an effort, but the **increase in productivity** is worth it!
Please join our collaborators!

Building Blocks of a Benchmarking Workflow



ExCALIBUR Portable Benchmarking Framework v1.0.0



Framework for automating builds, runs and data collection of benchmarks across HPC systems

It contains

- ReFrame configuration and Spack environments for UK HPC systems
- Suite of benchmark applications from collaborators
- Analysis and Visualisation tools
- Documentation and Tutorials

It is **not**

- An authoritative collection of benchmarks
- A benchmarking campaign



<https://github.com/ukri-excalibur/excalibur-tests>

Currently Supported Systems



Cosma



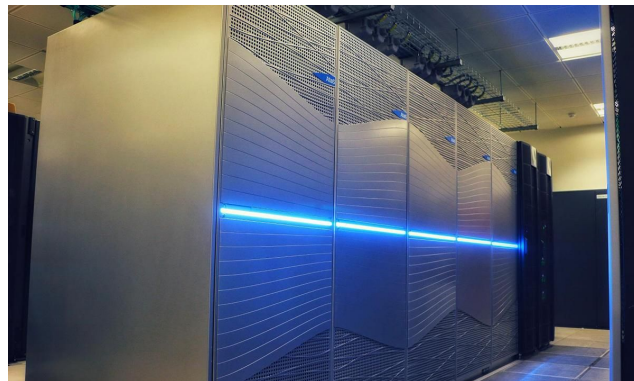
ARCHER2



CSD3



DiaL2 & DiaL3



Tursa



Myriad & Kathleen
(& Contender testbed)



[Isambard 2]

Using Spack for building benchmarks

Spack manages dependencies and automates build systems

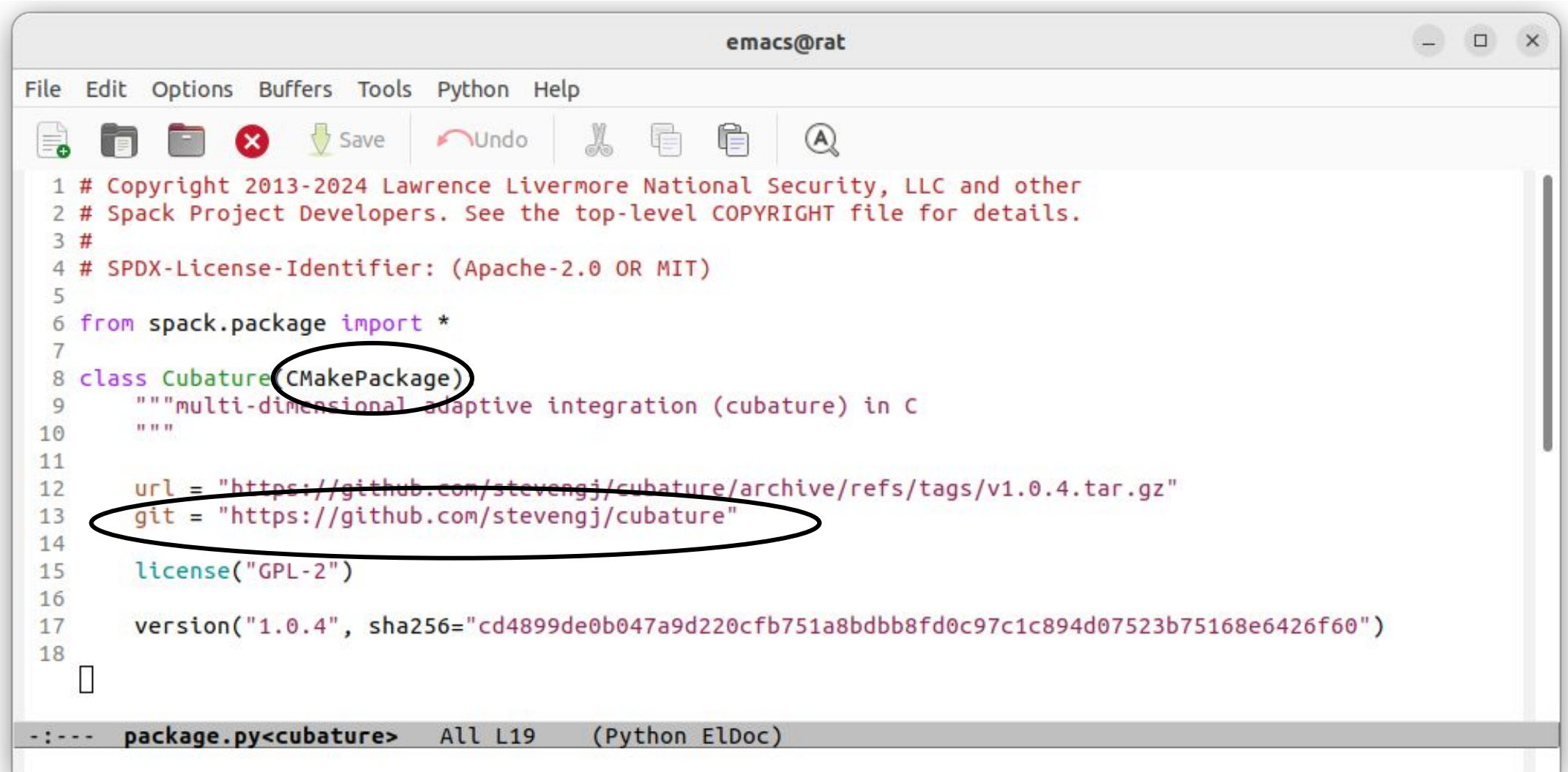
- Popular package manager for HPC in UK and US (ARCHER2, CSD3, DoE Labs)
- Large community developing and maintaining packages – Easy to contribute upstream
- Encourages reproducibility by keeping track of versions, variants and dependencies



github.com/spack/spack

A screenshot of a web browser displaying the HPSF website. The browser's address bar shows 'hpsf.io/#projects'. The page has a dark grey header with the word 'Projects' in white. Below the header, there is a grid of project logos: Spack (blue diamond), kokkos (purple and blue square), APPTAINER (orange and blue 'A'), Viskores (blue text), HPCToolkit (colorful nodes), and E4S (black text). A white sidebar on the left contains a list of navigation links: 'HPSF HAS LAUNCHED!', 'GOALS', 'WHY NOW?', 'ORGANIZATION', 'MEMBERS', 'PROJECTS', 'JOIN HPSF!', and 'RETURN TO TOP'. Below the project logos, there is a white section titled 'Join HPSF!' with text: 'If you are interested in joining HPSF as a member, you can [sign up in LFX](#), Linux Foundation's onboarding platform.' and 'Projects will need to be approved by the TAC, which is currently being formed.'

Spack recipe example (1/2)



```
emacsrat
File Edit Options Buffers Tools Python Help
Save Undo
1 # Copyright 2013-2024 Lawrence Livermore National Security, LLC and other
2 # Spack Project Developers. See the top-level COPYRIGHT file for details.
3 #
4 # SPDX-License-Identifier: (Apache-2.0 OR MIT)
5
6 from spack.package import *
7
8 class Cubature(CMakePackage)
9     """multi-dimensional adaptive integration (cubature) in C
10     """
11
12     url = "https://github.com/stevengj/cubature/archive/refs/tags/v1.0.4.tar.gz"
13     git = "https://github.com/stevengj/cubature"
14
15     license("GPL-2")
16
17     version("1.0.4", sha256="cd4899de0b047a9d220cfb751a8bdbb8fd0c97c1c894d07523b75168e6426f60")
18
19
20 -:-- package.py<cubature> All L19 (Python ElDoc)
```

Spack recipe example (2/2)

```
Spack: Preparing cubature [3/3]
Spack: Installing purify [21/21]
Terminal
Spack: Preparing cubature [3/3]

[tkoskela@rat ~]$ spack spec cubature
Input spec
-----
- cubature

Concretized
-----
[+] cubature@1.0.4%gcc@11.4.0~ipo build_system=cmake build_type=Release generator=make arch=linux-ubuntu22.04-icelake
[e]   ^cmake@3.22.1%gcc@11.4.0~doc+ncurses+ownlibs~qt build_system=generic build_type=Release arch=linux-ubuntu22.04-icelake
[+]   ^gcc-runtime@11.4.0%gcc@11.4.0 build_system=generic arch=linux-ubuntu22.04-icelake
[e]   ^glibc@2.35%gcc@12.3.0 build_system=autotools arch=linux-ubuntu22.04-icelake
[e]   ^gmake@4.3%gcc@11.4.0~guile build_system=generic patches=599f134 arch=linux-ubuntu22.04-icelake

[tkoskela@rat ~]$ spack install cubature
[+] /usr (external glibc-2.35-dlo2shftattw2ouxzdc6zhhyh5ndp)
[+] /home/tkoskela/git_repos/spack/opt/spack/linux-ubuntu22.04-icelake/gcc-11.4.0/gcc-runtime-11.4.0-swawlsnlfng36yymhcdepayjp65utrql
[+] /home/tkoskela/git_repos/spack/opt/spack/linux-ubuntu22.04-icelake/gcc-11.4.0/cubature-1.0.4-dv5ndygp7xhc3rlgoi7iatnpumbkghp
[tkoskela@rat ~]$ ls /home/tkoskela/git_repos/spack/opt/spack/linux-ubuntu22.04-icelake/gcc-11.4.0/cubature-1.0.4-dv5ndygp7xhc3rlgoi7iatnpumbkghp/lib/
libcubature.so
[tkoskela@rat ~]$
```

Using ReFrame for running performance benchmarks

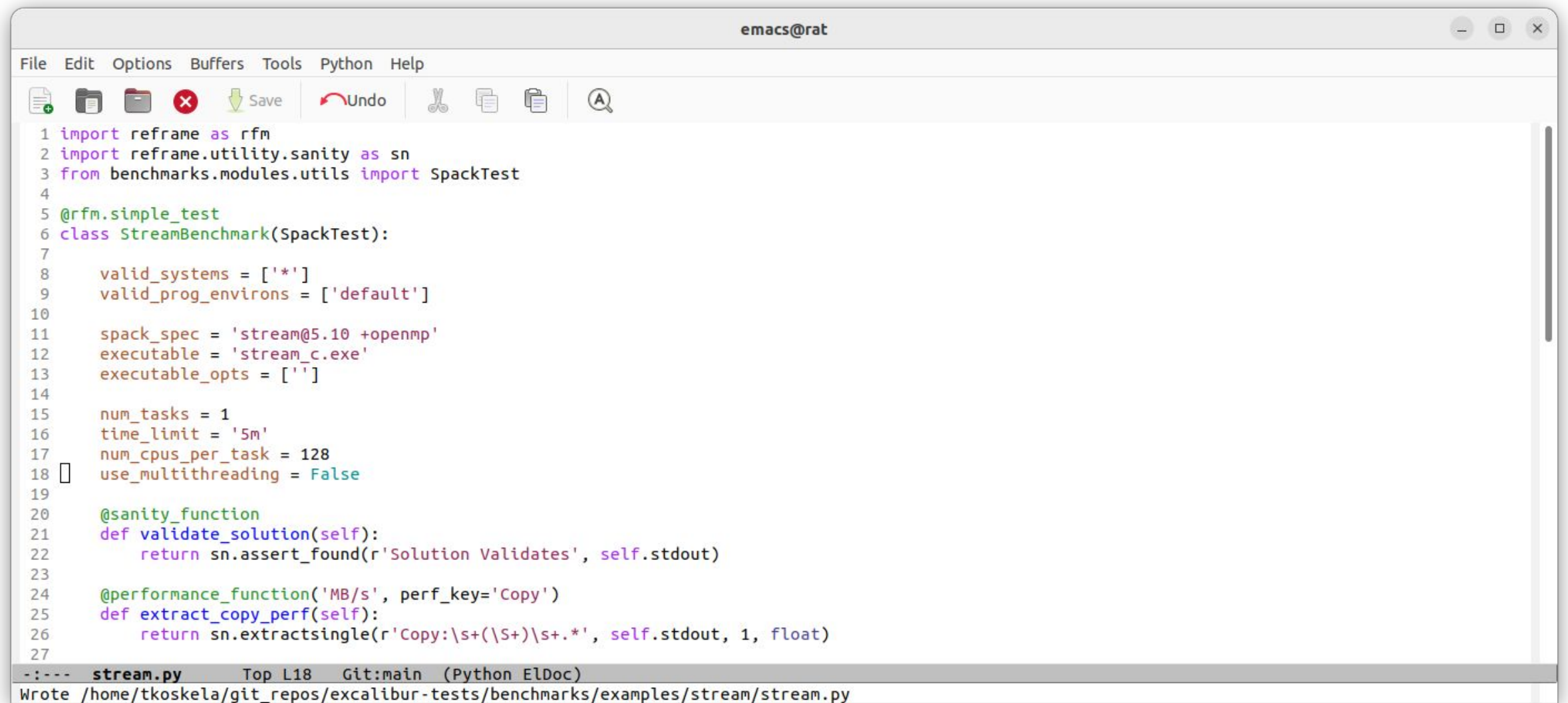
ReFrame

github.com/reframe-hpc/reframe

Open-source framework for regression tests and benchmarks for HPC systems

- Abstracts interactions with the scheduler and build system, separates benchmark and system properties.
- Interfaces with HPC concepts, such as schedulers, modules, build systems (including Spack)
- Multidimensional parametrisation for benchmarking campaigns

ReFrame class example

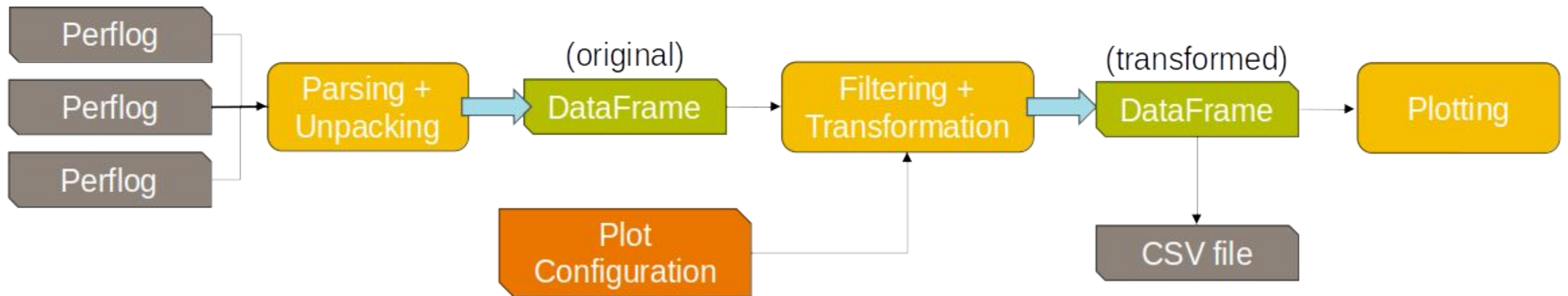


```
emacs@rat
File Edit Options Buffers Tools Python Help
Save Undo
1 import reframe as rfm
2 import reframe.utility.sanity as sn
3 from benchmarks.modules.utils import SpackTest
4
5 @rfm.simple_test
6 class StreamBenchmark(SpackTest):
7
8     valid_systems = ['*']
9     valid_prog_environs = ['default']
10
11     spack_spec = 'stream@5.10 +openmp'
12     executable = 'stream_c.exe'
13     executable_opts = []
14
15     num_tasks = 1
16     time_limit = '5m'
17     num_cpus_per_task = 128
18     use_multithreading = False
19
20     @sanity_function
21     def validate_solution(self):
22         return sn.assert_found(r'Solution Validates', self.stdout)
23
24     @performance_function('MB/s', perf_key='Copy')
25     def extract_copy_perf(self):
26         return sn.extractsingle(r'Copy:\s+(\S+)\s+.*', self.stdout, 1, float)
27
-:--- stream.py Top L18 Git:main (Python ELDoc)
Wrote /home/tkoskela/git_repos/excalibur-tests/benchmarks/examples/stream/stream.py
```

Analysis and Visualisation Workflow

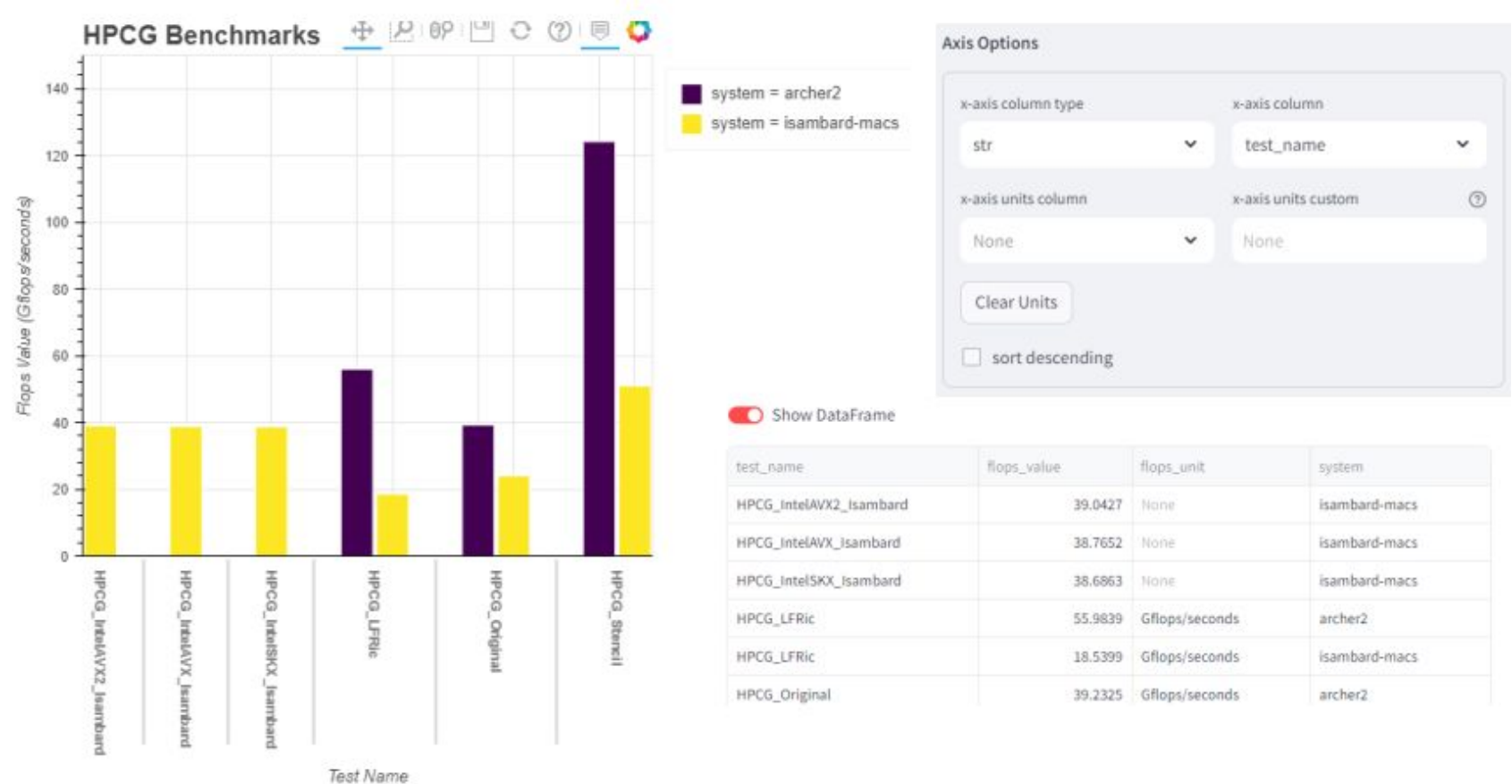
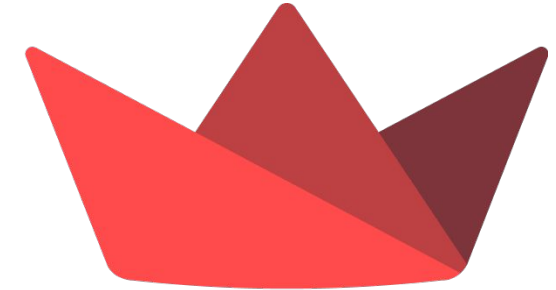


```
python post_processing.py log_path config_path
```



Visualisation GUI

- Post-processing can be run as a web app via **Streamlit**.
- Plot config is now an optional argument because it can be created from scratch in the UI.



Use Cases

System monitoring
Application development
System(s) evaluation

Use Cases

- Submitting ReFrame jobs on different nodes
- Node performance analysing

nodelist handlers

```
reframe_config.py M x
benchmarks > reframe_config.py > ...
732     ],
733     'handlers_perflong': [
734         {
735             'type': 'filelog',
736             'prefix': '%(check_system)s/%(check_partition)s',
737             'level': 'info',
738             'format': (
739                 '%(check_job_completion_time)s|'
740                 'reframe %(version)s|'
741                 '%(check_info)s|'
742                 '%(check_jobid)s|'
743                 '%(check_job_nodelist)s|'
744                 '%(check_num_tasks)s|'
745                 '%(check_num_cpus_per_task)s|'
746                 '%(check_num_tasks_per_node)s|'
747                 '%(check_num_gpus_per_node)s|'
748                 '%(check_perfvalues)s|'
749                 '%(check_spack_spec)s|'
750                 '%(check_display_name)s|'
751                 '%(check_system)s|'
752                 '%(check_partition)s|'
753                 '%(check_environs)s|'
754                 '%(check_extra_resources)s|'
755                 '%(check_env_vars)s|'
756                 '%(check_tags)s'
757             ),
```

submitting jobs on different nodes

parsing node information

```

$ SwiftDailyV2.sh x $ SwiftDailyV4.sh $ #SwiftDailyV3.sh#
cosma > home > durham > dc-kill1 > $ SwiftDailyV2.sh
1 |/bin/bash
2 |/cosma/home
3 raw_nodes=$(sinfo -h -o "%N %T" -p cosma8 | awk '!/drain|alloc|mixed|down/ {print $1}')
4 processed_nodes=""
5
6 IFS=' ' read -r -a nodes_array <<< "$raw_nodes"
7 for node in "${nodes_array[@]"; do
8   if [[ $node == *[-]* ]]; then
9     prefix=${node%*}
10    range=${prefix//-/ }
11    start=${range[0]}
12    end=${range[1]}
13    for (( i=$start; i<=$end; i++ )); do
14      processed_nodes+="$i,"
15    done
16  else
17    processed_nodes+="$node,"
18  fi
19 done
20 processed_nodes=${processed_nodes%,}
21
22 IFS=' ' read -r -a NODE_ARRAY <<< "$processed_nodes"
23
24 EXISTING_JOBS=$(queue -u dc-kill1 | grep rfm_Swif | wc -l)
25
26 if [ "$EXISTING_JOBS" -eq 0 ]; then
27   if [ ${#NODE_ARRAY[@]} -lt 4 ]; then
28     echo "Error: Not enough available nodes to submit jobs. At least 4 nodes are required."
29     exit 1
30   fi
31
32   for i in {1..20}; do
33     shuf_nodes=( $(shuf -e "${NODE_ARRAY[@]}" ) )
34     selected_nodes="m["
35     for j in {0..3}; do
36       selected_nodes+="${shuf_nodes[j]}"
37       if [ $j -ne 3 ]; then
38         selected_nodes+=", "
39       fi
40     done
41     selected_nodes+="]"
42     reframe -c excalibur-tests/benchmarks/apps/swift/swift.py -J="-A dr004 --nodelist=${selected_nodes}" -r
43   done
44 else
45   echo "Existing jobs found. No action taken."
46 fi

```

```

|4|8|1|null|40.0|seconds|50|None|0.2|swiftsim@0.9.0|SwiftBenchmark|cosma8|
|4|8|1|null|41.9|seconds|50|None|0.2|swiftsim@0.9.0|SwiftBenchmark|cosma8|
|4|8|1|null|40.4|seconds|50|None|0.2|swiftsim@0.9.0|SwiftBenchmark|cosma8|
m8360,m8442,m8444,m8477|4|8|1|null|34.0|seconds|50|None|0.2|swiftsim@0.9.0
m8360,m8442,m8444,m8477|4|8|1|null|34.5|seconds|50|None|0.2|swiftsim@0.9.0

```

BabelStream

- Benchmarks achievable (main) memory bandwidth
- Based on McCalpin STREAM, except
 - Arrays allocated on the heap
 - Problem size is known only at runtime
- Written in many programming models
- Constructed of simple vector operations
 - Copy: $c[i] = a[i]$
 - Mul: $b[i] = \text{scalar} * c[i]$
 - Add: $c[i] = a[i] + b[i]$
 - Triad: $a[i] = b[i] + \text{scalar} * c[i]$
 - Dot: $\text{sum} += a[i] * b[i]$



<https://github.com/UoB-HPC/BabelStream>

BabelStream Spack Package

Kaan Olgu, Bristol

Background Information

- CMake builds well, but we need a wrapper script for a smoother installation.
- The paper was completed in 18 months with a team of 6-7 authors contributing 2-3 months each.
- Avoid manual installation of external library dependencies.
- When using BabelStream with an older project that requires a specific version of CMake, users may need to switch between different CMake versions.



The Spack package simplified the commands required to build implementations

`spack install babelstream@develop +sycl2020%oneapi`

@develop : we could use any branch
%oneapi : we could use compiler of choice
Other flag options could be set from documentation



No Performance Penalty

Reason of variation :

Different data generated at compile time!

Function	Throughput Difference Formula	Throughput Difference
Copy	$(\text{Spack/CMake} - 1) * 100$	1.75 %
Mul		0.07 %
Add		2.13 %
Triad		1.77 %
Dot		3.43 %



No dependencies to deal with by the user

The required dependencies are installed in Spack's own specific folder not into the system

Background Information

- ReFrame package uses the Spack build system for regression tests and benchmarks designed for HPC systems.
- It simplifies the complexity of interacting with the system, focusing only on test functionality.
- Users with a single command can run BabelStream on the desired environment and get the figure of merit
- Using ReFrame with the Spack build system allowed us to maintain code easily

Run Command

```
reframe -c benchmarks/apps/babelstream -r --tag cuda --system=isambard-macs:volta -S  
build_locally=false -S spack_spec='babelstream@develop%gcc@9.2.0 +cuda cuda_arch=70'
```

--tag=cuda :

Instructs ReFrame to use "cuda" subclass for setting up environment

--system=isambard-macs:volta :

Choosing which environment to build and run the Babelstream

-Sbuild_locally=false :

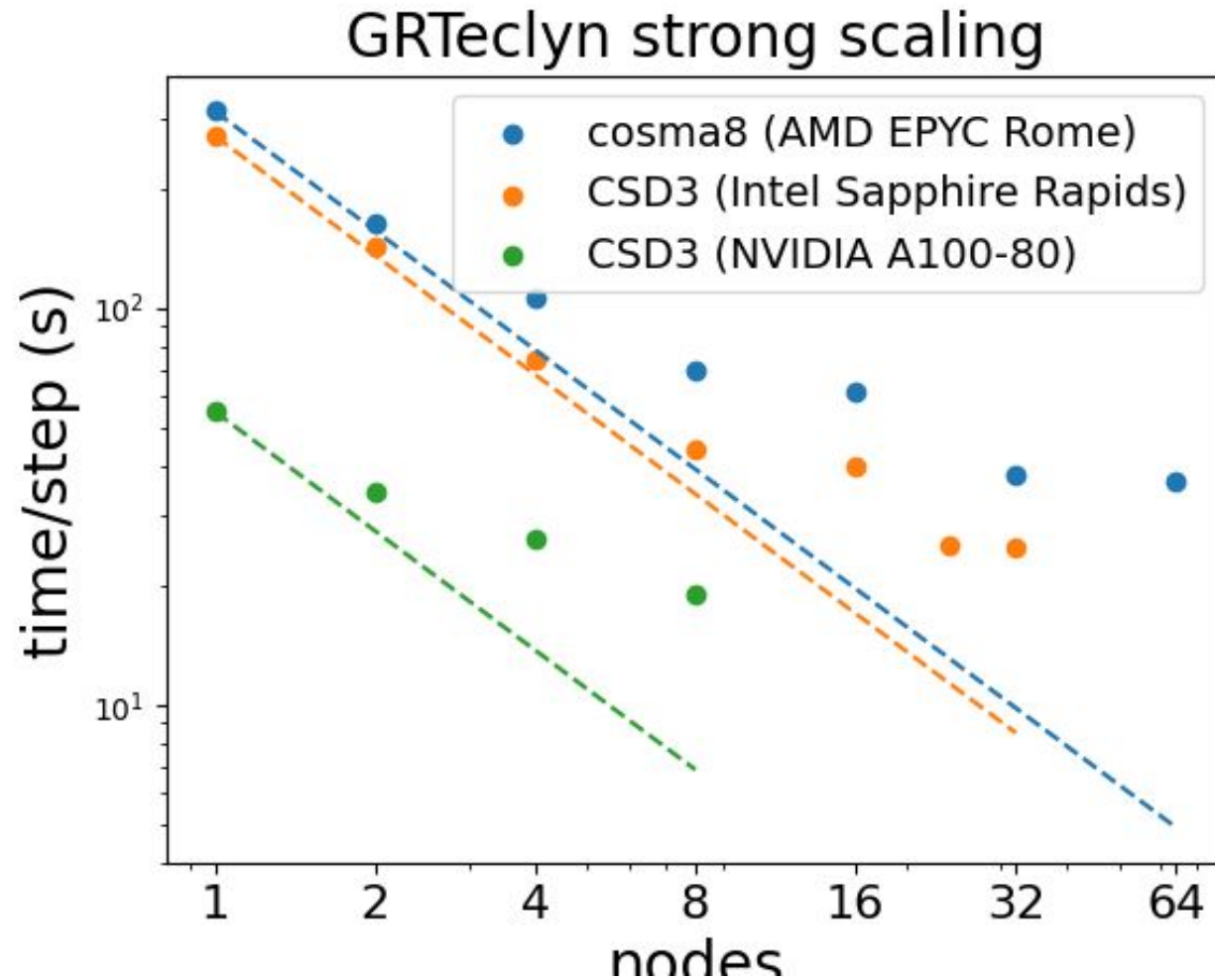
This instructs not to use login nodes and use system provided for running "*spack install babelstream*"

-Sspack_spec='babelstream%gcc +cuda cuda_arch=70' :

Spack commands we used previously

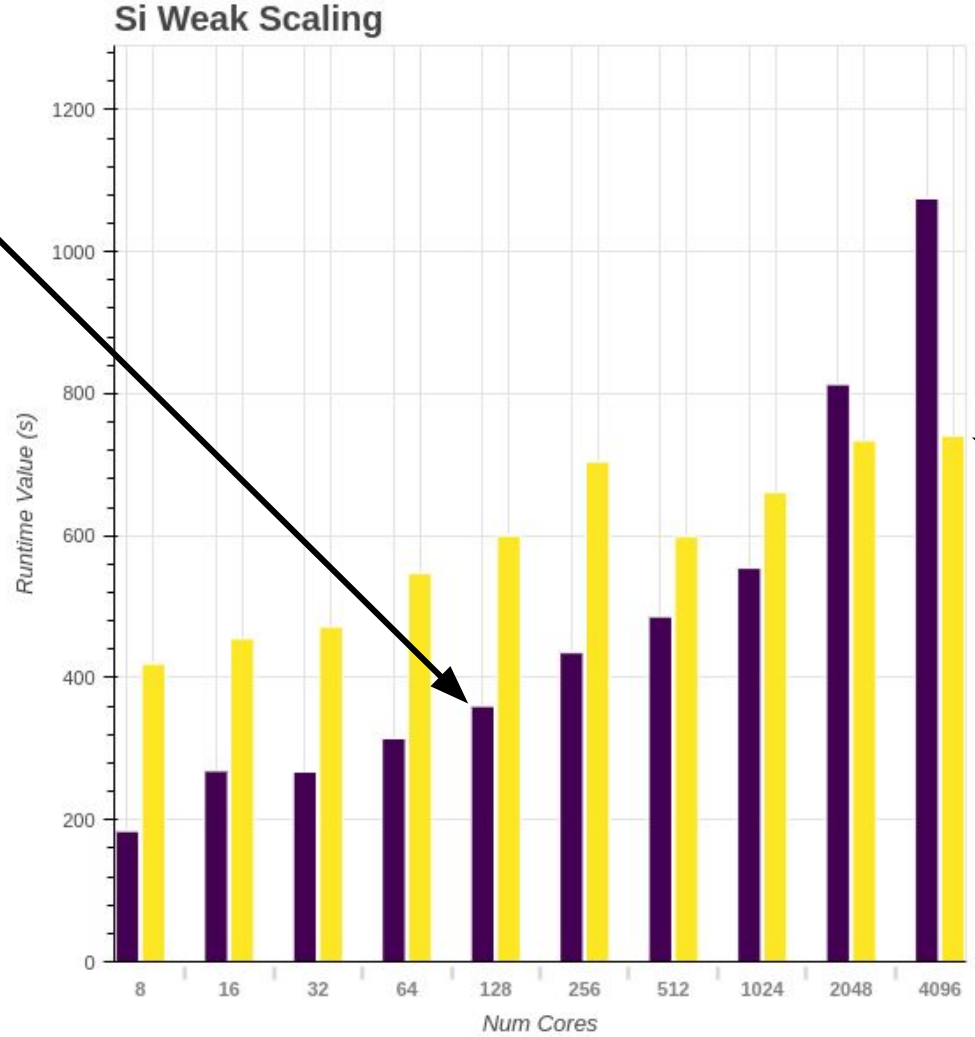
GRTEclyn: performance benchmarks

Juliana Kwan, Cambridge



Weak scaling benchmark on ARCHER2

Pure MPI is ~30% faster on one node



eCSE08 project
CONQUEST

David Bowler, UCL

Hybrid MPI+OpenMP
is ~30% faster on 32
nodes

LEXCI: Reproducibility of performance benchmarks

Jason McEwen, UCL

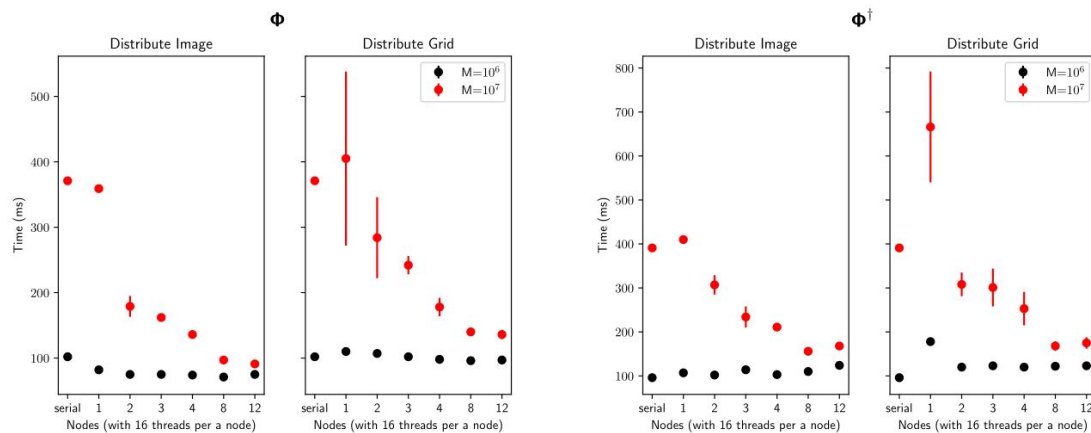
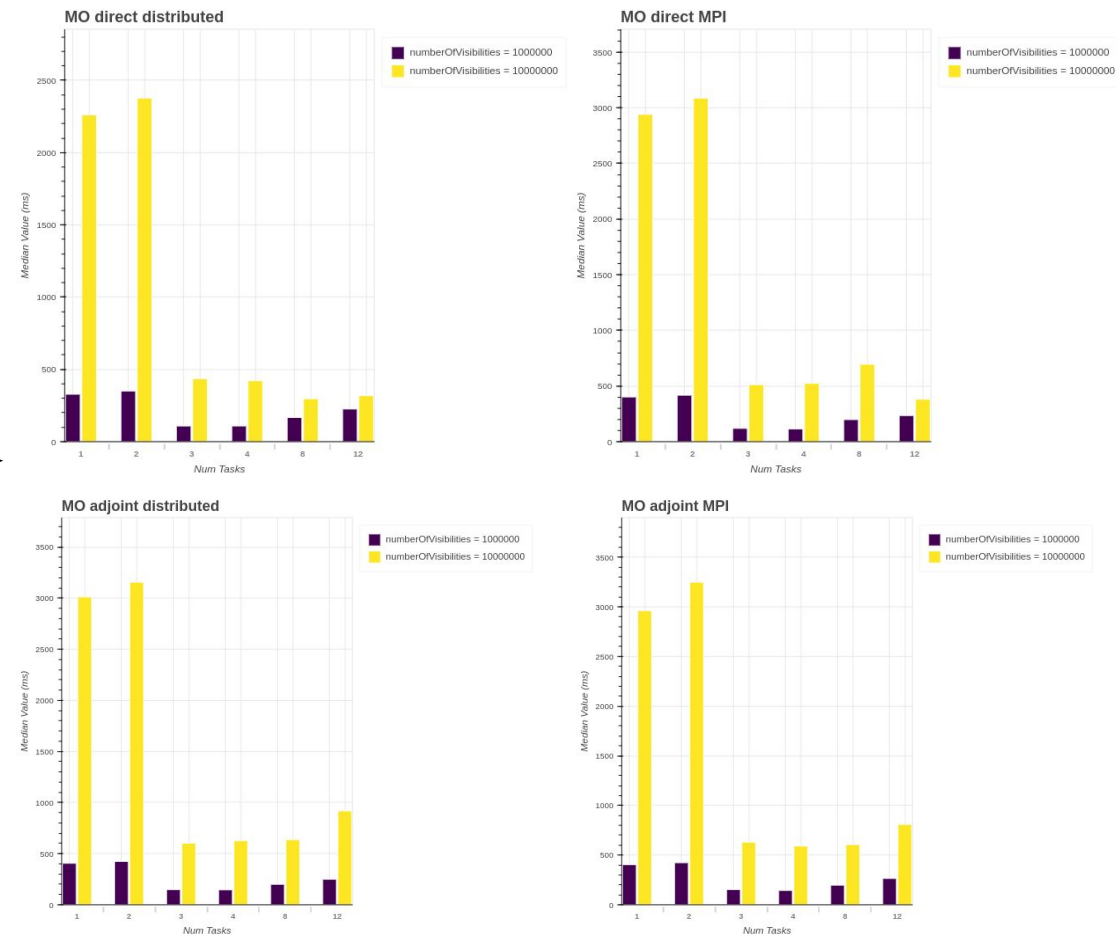


Figure 1. Time to apply forward Φ (left two plots) and adjoint Φ^\dagger (right two plots) as a function of the number of MPI nodes, benchmarked against the non MPI (serial) implementation. We fix the number of visibilities and image size at $N = 1024 \times 1024$, $M \in \{10^6, 10^7\}$. On the left the MPI implementation corresponds to using an all sum all MPI operation in the adjoint described in Section 6.3.1; on the right MPI implementation corresponds to distribution of the grid from the root node, as described in Section 6.3.2.



Pratley, McEwen et al. (2019)

Arriving now!

Project Contender

UPDATE

As Moore's
Law fails,
accelerator
cards boom.

Mainframe

Current systems:

- Nvidia ◇ Grace-Hopper ("Locust") ✓
- Ampere ◇ Altra + Nvidia ◇ A100 ("Cricket") ✓
- Graphcore ◇ POD16 ("Mandelbrot" & "Fibonacci") ✓
- Qualcomm ◇ Cloud AI100 ("Zeno of Citium") ✓
- NEC ◇ SX Aurora TSUBASA ("Terrarium") ✓
- IBM ◇ LinuxOne ("Vindicator") ✓

**Various
GPUs**

**ARM
CPUs**

**ML Training
& Inference**

Project Contender at the
**Centre for Advanced
Research Computing** is a
collection of **accelerator
cards**, and the like, for
researchers to

- compare performance,
- gain familiarity
- stimulate novel
programming approaches

To apply for system access contact i.legg@ucl.ac.uk
with a brief outline of your proposed use.

<https://www.ucl.ac.uk/advanced-research-computing/coming-soon-platforms>

Building Purify benchmark with 56 dependencies

```

- purify@4.2.0%gcc@13.1.1+benchmarks~coverage~docs~ipo+mpi~onnxrt+openmp+tests build_system=cmake build_type=Release
generator=make arch=linux-rhel9-neoverse_v2
- ^benchmark@1.8.5%gcc@13.1.1~ipo~performance_counters build_system=cmake build_type=RelWithDebInfo generator=make
arch=linux-rhel9-neoverse_v2
- ^boost@1.82.0%gcc@13.1.1
- ^catch2@3.7.1%gcc@13.1.1~ipo+pic~shared build_system=cmake build_type=Release cxxstd=17 generator=make
arch=linux-rhel9-neoverse_v2
- ^cfitsio@4.4.0%gcc@13.1.1+bzip2+shared build_system=autotools arch=linux-rhel9-neoverse_v2
- ^cubature@1.0.4%gcc@13.1.1~ipo build_system=cmake build_type=Release generator=make arch=linux-rhel9-neoverse_v2
- ^eigen@3.4.0%gcc@13.1.1~ipo build_system=cmake build_type=RelWithDebInfo generator=make arch=linux-rhel9-neoverse_v2
- ^fftw@3.3.10%gcc@13.1.1+mpi~openmp~pfft_patches+shared build_system=autotools patches=872cff9 precision=double,float
arch=linux-rhel9-neoverse_v2
[^] ^gcc-runtime@13.1.1%gcc@13.1.1 build_system=generic arch=linux-rhel9-neoverse_v2
[e] ^glibc@2.34%gcc@13.1.1 build_system=autotools arch=linux-rhel9-neoverse_v2
[^] ^gmake@4.4.1%gcc@13.1.1~guile build_system=generic arch=linux-rhel9-neoverse_v2
- ^libtiff@4.6.0%gcc@13.1.1
- ^libjpeg-turbo@3.0.3%gcc@13.1.1
- ^nasm@2.16.03%gcc@13.1.1 build_system=autotools arch=linux-rhel9-neoverse_v2
- ^openmpi@5.0.5%gcc@13.1.1
- ^pmix@5.0.3%gcc@13.1.1~munge~python~restful build_system=autotools arch=linux-rhel9-neoverse_v2
- ^sopt@4.2.0%gcc@13.1.1~benchmarks~coverage~docs~examples~ipo+mpi~onnxrt+openmp~tests build_system=cmake
build_type=Release generator=make patches=00729db arch=linux-rhel9-neoverse_v2
- ^yaml-cpp@0.8.0%gcc@13.1.1~ipo+pic+shared~tests build_system=cmake build_type=Release generator=make
arch=linux-rhel9-neoverse_v2

```

Future Ideas/Plans

- Provide a service for the UKRI community – Maintain a suite of benchmarks and support benchmark and system additions.
- Deliver training and knowledge exchange
- Develop more post-processing tools
- Support automatic profiling and system monitoring
- Systematic deployment – include framework as CI on key HPC systems
- Platform for hosting data – curate and store benchmark results

Summary

- It's important to understand how our applications perform on different HPC architectures in order to make informed decisions about **future hardware** and **future software**
- Benchmarking, as it's traditionally done, requires a lot of manual effort, is time-consuming, **error prone and difficult to reproduce**
- We've developed a framework using **Spack** and **ReFrame** to **automate the manual effort** in portable building and running of benchmarks.
- Adopting it takes an effort, but the **increase in productivity** is worth it!
Please join our collaborators!

Thanks!



<https://github.com/ukri-excalibur/excalibur-tests>

