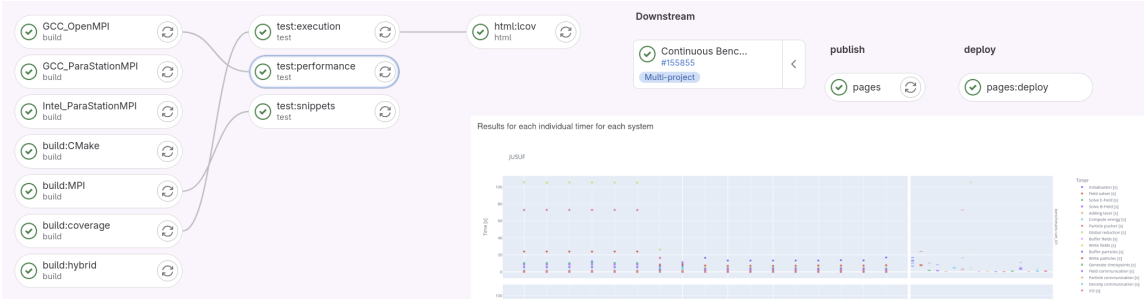


INTEGRATED CONTINUOUS BENCHMARKING

Connecting gitlab, Jacamar, and JUBE

06.03.2024 | D. Brömmel, J. Fritz, R. Speck | JSC

MOTIVATION AND GOALS

HPC is all about Performance

- Apart from testing for correct execution, HPC codes demand regular performance monitoring
- Check for regressions during development, on different systems, or toolchain changes

- May require running on larger scale
- Benefits from running on target HPC systems

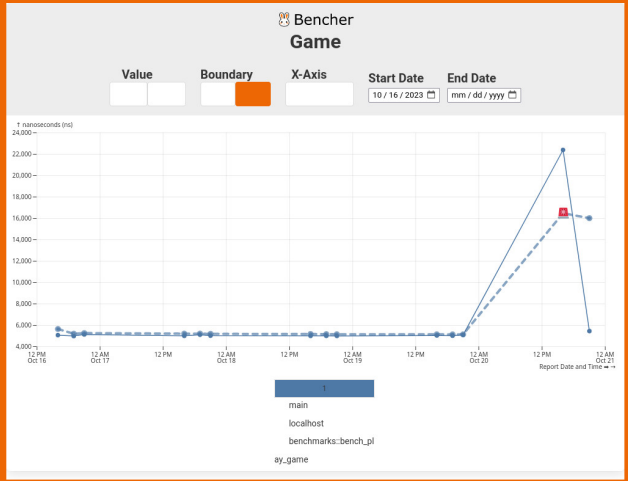
- Use CI pipelines on JSC's systems (per commit, periodically)
- Create badge and pages that display performance regressions

pipeline passed | CB score 98%

Showcase expected performance and correct setup to potential users

EXISTING APPROACHES

Bencher – a one-stop solution?



How It Works

Run your benchmarks

Run your benchmarks locally or in CI using your favorite benchmarking tools. The `bencher` CLI simply wraps your existing benchmark harness and stores its results.

Track your benchmarks

Track the results of your benchmarks over time. Monitor, query, and graph the results using the Bencher web console based on the source branch and testbed.

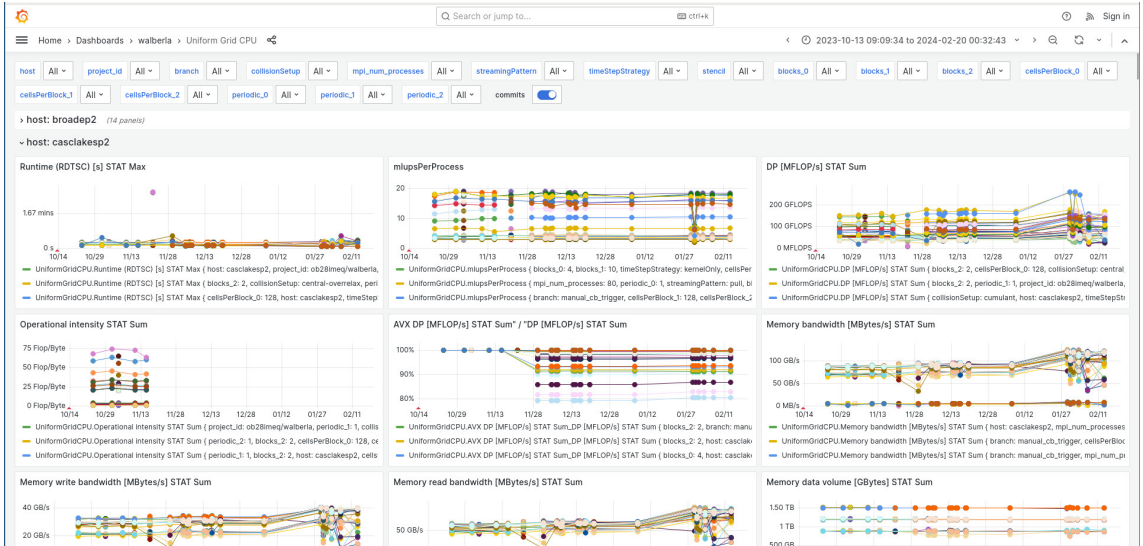
Catch performance regressions

Catch performance regressions in CI. Bencher uses state of the art, customizable analytics to detect performance regressions before they make it to production.

EXISTING APPROACHES

waLBerla – Grafana to display results

⟶ Talk by Christoph Alt and Harald Köstler



CONSTRAINTS AND THINGS TO CONSIDER

1. Want to reuse existing tools and setup as far as possible
   - `JUBE` with common scripts
   - Usable standalone outside of CI
2. Avoid dependencies on additional servers (maintenance, cost, complexity)
   - Purely `gitlab` may require additional repositories
3. HPC systems at JSC
   - Requires account and project on JSC systems and internal `gitlab`
   - Strict authorisation, shared user database → HPC runners: no leaking credentials, but need to check 'payload'
4. Pipelines are accessing a shared resource
   - Turn-around times not predictable
   - Pipeline races

JUBE

The Jülich Benchmarking Environment

⟶ www.fz-juelich.de/jsc/jube

- Makes it easy to spawn parameter ranges (think: scaling, toolchains, optimisations, . . .)
- Compiles and executes the code, generates job-scripts, performs analysis
- Stores results also in `sqlite` format

```
−<parameterset name="modeset">
    <!-- mode to run PEPC in -->
    <parameter name="mode" type="string">benchmark</parameter>
    <parameter tag="check" name="mode" type="string">test</parameter>
    <!-- strong scaling with constant number of particles -->
    <parameter name="particles" type="int">50000000</parameter>
    <parameter tag="check" name="particles" type="int">50000</parameter>
  </parameterset>
−<substituteset name="runsub">
    <iofile in="${job_script}" out="job.slurm"/>
    <sub source="##PARTITION##" dest="$partition"/>
    <sub source="##NODES##" dest="$nodes"/>
    <sub source="##TASKSPNODE##" dest="$taskspnode"/>
    <sub source="##THREADS##" dest="$threads"/>
    <sub source="##WORKERTHREADS##" dest="$workerthreads"/>
    <sub source="##SMT_THREADS##" dest="$thread_places"/>
    <sub source="##MODULES##" dest="$load_modules"/>
  </substituteset>
+<!---->
```

```
+<!---->
+<step name="compile"></step>
  <!-- run the binary, apply substitutions as necessary -->
+<step tag="!stats" name="run" depend="compile" iterations="1"></step>
−<step tag="stats" name="run" depend="compile" iterations="5">
    <use>system</use>
    <use>nodeset</use>
    <use>modeset</use>
    <use>executable</use>
    <use>runfiles</use>
    <use>inputsub</use>
    <use>runsub</use>
    <!-- load modules -->
    <do>${load_modules}</do>
    <do done_file="ready">sbatch -A ${SBATCH_ACCOUNT} job.slurm</do>
  </step>
```

JUBE

The Jülich Benchmarking Environment

⟶ www.fz-juelich.de/jsc/jube

Table: full_results — Filter in any column

	chksum	systemname	list_of_mpis	walk	nodes	taskspnode	threads	total_threads	particles	wallclock_min	wallclock_avg	wallclock_max	tree_walk_min	tree_walk_avg	tree_walk_max	tree_grow_min
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	heads/cb_testing-0-g72ae7eaa	jusuf	ompi	simple	4	8	32	1024	50000000	55.649	55.649	55.649	3.8733	4.19417	4.2355	1.0652
2	heads/cb_testing-0-gdfbf93b0	jusuf	ompi	simple	4	8	32	1024	50000000	55.66	55.66	55.66	3.8759	4.19041	4.2345	1.0637
3	heads/cb_testing-0-ge97f0eaf	jusuf	ompi	simple	4	8	32	1024	50000000	55.675	55.675	55.675	3.877	4.19183	4.2314	1.0627
4	heads/cb_testing-0-gdfbf93b0	jusuf	ompi	simple	4	8	32	1024	50000000	55.671	55.671	55.671	3.8782	4.20361	4.2969	1.0649
5	heads/cb_testing-0-ge97f0eaf	jusuf	ompi	simple	4	8	32	1024	50000000	55.744	55.744	55.744	3.8796	4.19547	4.2382	1.0626
6	heads/cb_testing-0-ge97f0eaf	jusuf	ompi	simple	4	8	32	1024	50000000	55.537	55.537	55.537	3.8709	4.19359	4.2417	1.0643

Advantages

- Code experts can generate benchmarks (without CI knowledge)
- Highly reproducible
- Logic and setup in `JUBE` , not in CI
- Independent of CI, everyone can run the same experiments (development, external testing)

GITLAB **AND** JACAMAR

gitlab runners on HPC systems

⟶ ecp-ci.gitlab.io

- Like any other CI steps in a pipeline
- Pick runner on system (timeouts!)
- Load system modules, no container images to manage
- Mind `ENV` variables for batch jobs

```
81  toolchains:
82    stage: build
83    parallel:
84      matrix:
85        - SYSTEM: [jusuf]
86          COMPILER: GCC
87          MPI: [OpenMPI, ParaStationMPI]
88        - SYSTEM: [juwels]
89          COMPILER: Intel
90          MPI: [ParaStationMPI, IntelMPI]
91    tags:
92      - ${SYSTEM}
93      - shell
94    needs: ["frontends: [pepc-essential, pthreads]"]
95    script:
96      - echo "Building PEPC (all frontends) on $SYSTEMNAME using $HOSTNAME"
97      - echo "Loading modules..."
98      - ml ${COMPILER}
99      - ml ${MPI}
100     - ml Autotools
101     - ln -s makefiles/makefile.defs.${COMPILER} makefile.defs
102     - make all
103   artifacts:
104     when: on_failure
105     paths:
106       - bin/*
107   rules:
108     # Only run when on main repository
109     - if: $CI_PROJECT_URL == "https://gitlab.jsc.fz-juelich.de/SLPP/pepc/pepc"
```

GITLAB **AND** JACAMAR

gitlab runners on HPC systems

⟶ ecp-ci.gitlab.io

- Like any other CI steps in a pipeline
- Pick runner on system (timeouts!)
- Load system modules, no container images to manage
- Mind `ENV` variables for batch jobs

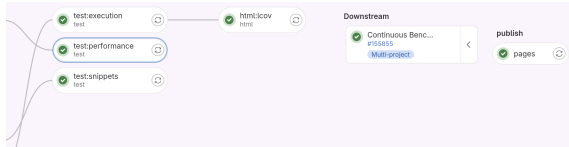
```
111 # If the 'toolchain' builds work, test PEPC correctness on a cluster
112 correctness:
113   stage: test
114   tags:
115     - jusuf
116     - shell
117   needs: ["toolchains: [jusuf, GCC, OpenMPI]"]
118   script:
119     - echo "Running pepc-benchmark on $SYSTEMNAME using $HOSTNAME"
120     - echo "Loading modules..."
121     # load JUBE, the rest will be done from there
122     - ml JUBE
123     # start JUBE
124     - echo "Running JUBE... This will take some time..."
125     - jube-autorun -o -r "-t check CI" benchmark/benchmark.xml
126     - echo "Checking for success..."
127     - jube result run/benchmark/ | grep -q failed && exit 128
128     - echo "Check seems to have passed..."
129   artifacts:
130     when: on_failure
131     paths:
132       - run
133   rules:
134     # Only run when on main repository
135     - if: $CI_PROJECT_URL == "https://gitlab.jsc.fz-juelich.de/SLPP/pepc/pepc"
```

GITLAB **AND** JACAMAR

gitlab runners on HPC systems

→ ecp-ci.gitlab.io

- Need to consider how to store results:
  - separate repository
  - specific branch
  - as artifacts
  - directly on a filesystem
- May separate permissions, possibly combine several codes, or use external codes

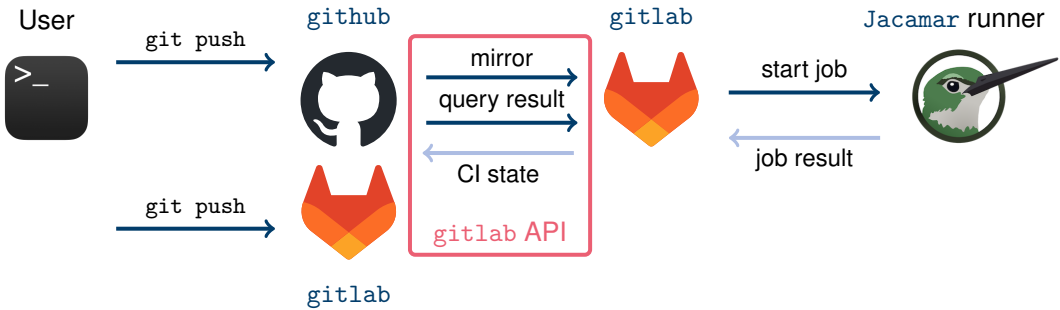


FROM GITHUB TO GITLAB

Coupling ‘external’ repositories

⟶ Talk by Jakob Fritz and Thomas Gruber

⟶ github.com/jakob-fritz/github2lab_action



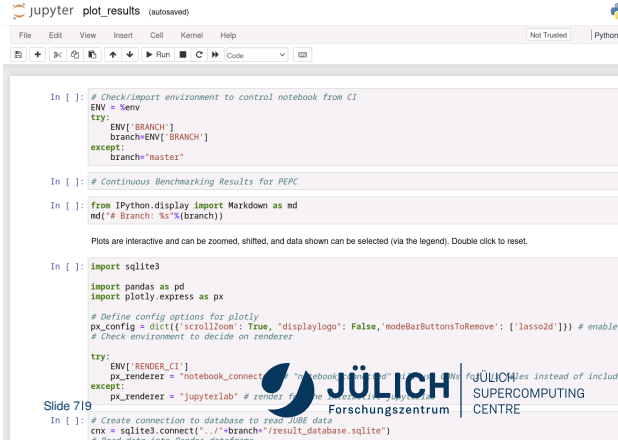
- Mind usage agreement
- Careful checks who’s allowed to pushed

JUPYTER **NOTEBOOKS...**

...to help with the analysis and review

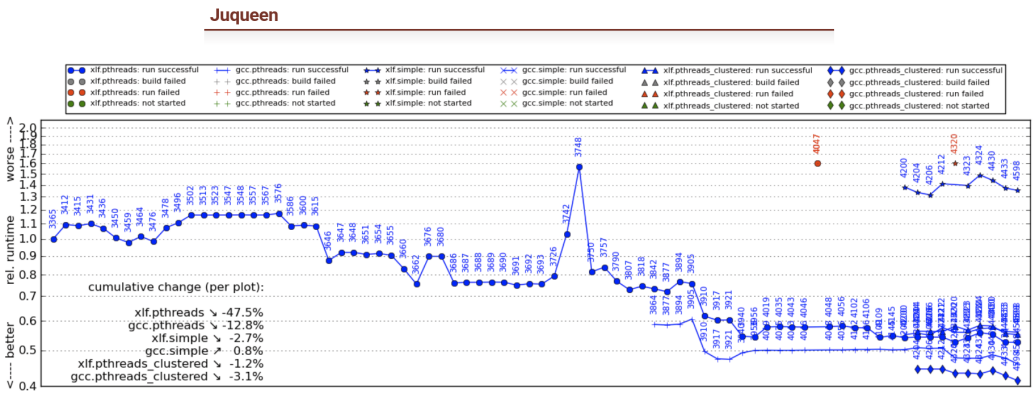
- `jupyter` notebooks ease analysis, domain expert can create and test this
- As complex as you like
- Generate badges and trigger pipeline fails

  CB score 98%
- Combine with, e.g. `mkdocs` to include in documentation, create static webpages that can be used on `gitlab` pages



OLD RESULTS – AND NOW

From manual labour by PhD students to integrated benchmarking



Done manually, quite some time of PhD students went into this.

OLD RESULTS – AND NOW

From manual labour by PhD students to integrated benchmarking

Branch: cb_testing

Gitlab ☆0 ⑂2

Continuous Benchmarking for PEPC

Branch: cb_testing

Branch: master

Code Repository

Issue Tracker

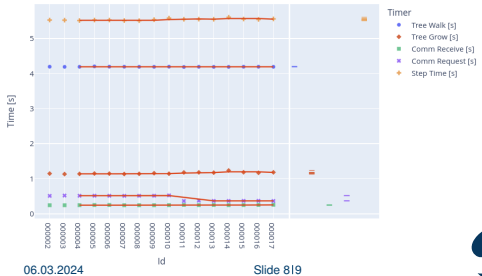
Results for individual timers

Per benchmark iteration

The solid red line represents a trend in form of a sliding average of 3 values.

Table of contents

JUSUF



SUMMARY

- ☑ Testing toolchains on target systems
- ☑ Integrated Continuous Benchmarking (CB) the easy way
- ☑ Minimal extra steps – since everyone should be using `JUBE`
- ☑ Separation of expertise: domain scientists stick to their part
- ☑ Scalable, running on HPC systems via `Jacamar` – no limitation on problem size
- ☑ Standalone with no dependencies on external services

`https://slpp.pages.jsc.fz-juelich.de/pepc/pepc/`