How to deal with validation as an HPC software?
An approach to power software testing at scale

Julien Adam — julien.adam@paratools.com

FOSDEM'23
February 5th, 2023
Background

❖ Back in 2012, from a team developing an MPI runtime

❖ Shell-script-only regression base
  ✦ Highly dependent on test environments

❖ Maintenance was a costly task
  ✦ When extending the validation process with new tools
  ✦ When integrating even minor changes from the project
What to expect

❖ A simple tool
  ✦ Basically, a CLI & minimal configuration files

❖ Highly customizable tool
  ✦ Test Scenarios may be complex
  ✦ Adaptable to future tech without a complete rewriting

❖ See Also:
  ✦ ReFrame
  ✦ JuBE
  ✦ Pavilion2
Parallel Computing Validation System

- Shorten as PCVS
- A CLI + YAML-based configuration files

Testing framework designed to:
- Make test design portable
- Retarget benchmarks for comparison
- Autoscale benchmarks to test environments
Features

❖ Split environment & test design
   ✦ Test specifications are carried with projects/benchmarks
   ✦ Environment are stored on clusters

❖ Adapt tests to new environments
   ✦ Auto-retargeting tests to compilers/runtimes
   ✦ Auto-scaling tests to exec environment

❖ Integrate tools for in-place reporting

❖ Stand-alone execution (=sessions)

❖ Simple format: YAML

❖ Store runs over time

❖ Run analysis to create trend/stats over time
PCVS Architecture

- Job descriptions expose resource requirements
- Environment provides resources
- => the intersection constitutes the combinatory matrix
- Test workload depends on both these information
PCVS Architecture

- Job descriptions expose resource requirements
- Environment provides resources
- \( \Rightarrow \) the intersection constitutes the combinatory matrix
- Test workload depends on both these information

```
1  compiler:
2    · commands:
3      · cc: mpicc
4  criterion:
5    · iterators:
6      · n_mpi:
7      · values: [1, 2, 3, 4]
8  runtime:
9    · iterators:
10   · n_mpi:
11   · option: '-np '
12  · program: mpirun
```

```
node:
 · run:
   · program: "/prog"

mpirun -np 1 <prog>
mpirun -np 2 <prog>
mpirun -np 3 <prog>
mpirun -np 4 <prog>
```
How to write a compilation test

- Alongside with tests/benchmarks
- Static file: `pcvs.yml`
- Multiple build systems supported

Compilation test

Lang. Autodetect

```
program_description:
  tag: ["MPI"]
  build:
    files: "*.c" # relative to YML file
    sources:
      binary: "myprogram"
      depends_on: ["deps/program_description"]
      cflags: "extra cflags"
      ldflags: "extra ldflags"
      cwd: "dir/to/build"
      variants:
        - openmp

autotools:
  params: ['--disable-bootstrap']
cmake:
  vars: ['CMAKE_VERBOSE_MAKEFILE=ON']
make:
  target: all
```
How to write a run test

❖ Alongside with tests/benchmarks

❖ Static file: pcvs.yml

❖ Many validation triggers

```
program_description:
  tag: ['MPI']
  run:
    program: './a.out'
    iterate:
      n_mpi:
        values: [2, 4]
      program:
        arg:
          option: '-iter'
          values: [10, 100, 1000]
```

Alter global scenario
Create your own pattern
How to write a run test

❖ Alongside with tests/benchmarks

❖ Static file: `pcvs.yml`

❖ Many validation triggers

Program description:

```yaml
validate:
  expect_exit: 0
  time:
    mean: 10.0
    tolerance: 2.0
    kill_after: 20
  match:
    label:
      expr: '^\d+(\.\d+) received$'
      expect: true|false
    label2: 'Total Elapsed: \d+\.\d+ sec.$'
  analysis:
    method: "<method>"
  script:
    path: "/path/to/script"
```

Option:

```yaml
values: [10, 100, 1000]
```

Alter global scenario

Create your own pattern

ParaTools
PCVS relies on a single output directory (defaults to $PWD/.pcvs-build) containing:

- Configurations
- Build artefacts (programs, temp files...)
- PCVS cache (pre-compiled python files)
- **JSON-formatted Results**
- Previous results run in the same directory (compressed)

Now Run!

```bash
$ pcvs profile create mympi --base mpi
$ pcvs run --profile mympi ./MPI/IMB/check
```
Now Run!

PCVS can be executed in the background, detached from the current shell.

Test results are packed when displayed for compactness.

I/O are managed thanks to rich

$ pcvs exec --list
$ pcvs exec IMB/MPI_Barrier_n2 [--show]
Goal: preserve benchmarks

- To ease the benchmark update process, no modifications should be applied directly to sources.

- => Prepare/generate dynamically job descriptions
  - Run a script loading the run environment
  - Output the actual YAML script
  - `pcvs.setup`
Now Report!

- Need: review the results to get direct feedback
  - Ex: Quick rerun

- Tests may be uniquely rerun in the same condition
  - `$ pcvs exec IMB/MPI_BARRIER_n2`

- Tests may be GUI-reviewed (Flask Server)
  - From any build or generated archive
  - `$ pcvs report`
Now Report!

- Need: review the results to get direct feedback
  - Ex: Quick rerun

- Tests may be uniquely rerun in the same condition
  - $ pcvs exec IMB/MPI_Barrier_n2

- Tests may be GUI-reviewed (Flask Server)
  - From any build or generated archive
  - $ pcvs report

<table>
<thead>
<tr>
<th>Progress</th>
<th>Name</th>
<th>Test Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IMB</td>
<td>293</td>
</tr>
</tbody>
</table>
### Test Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Elapsed time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMB/Ibarrier_args=Ibarrier_n1</td>
<td>SUCCESS</td>
<td>3.36</td>
</tr>
<tr>
<td>IMB/Unidir_Get_args=Unidir_Get_n1</td>
<td>SUCCESS</td>
<td>3.40</td>
</tr>
<tr>
<td>IMB/Ibarrier_args=Ibarrier_n2</td>
<td>SUCCESS</td>
<td>3.55</td>
</tr>
<tr>
<td>IMB/Open_Close_args=Open_Close_n2</td>
<td>SUCCESS</td>
<td>3.67</td>
</tr>
<tr>
<td>IMB/S_Read_Indv_args=S_Read_Indv_n1</td>
<td>SUCCESS</td>
<td>3.71</td>
</tr>
<tr>
<td>IMB/P_Read_Priv_args=P_Read_Priv_n1</td>
<td>SUCCESS</td>
<td>3.72</td>
</tr>
<tr>
<td>IMB/P_Read_Expl_args=P_Read_Expl_n1</td>
<td>SUCCESS</td>
<td>3.75</td>
</tr>
<tr>
<td>IMB/S_IWrite_Expl_args=S_IWrite_Expl_n1</td>
<td>FAILURE</td>
<td>3.75</td>
</tr>
<tr>
<td>IMB/C_Write_Expl_args=C_Write_Expl_n1</td>
<td>FAILURE</td>
<td>3.77</td>
</tr>
<tr>
<td>IMB/S_Write_Expl_args=S_Write_Expl_n2</td>
<td>FAILURE</td>
<td>3.80</td>
</tr>
<tr>
<td>IMB/P_IWrite_Priv_args=P_IWrite_Priv_n1</td>
<td>FAILURE</td>
<td>3.83</td>
</tr>
<tr>
<td>IMB/S_Read_Expl_args=S_Read_Expl_n1</td>
<td>SUCCESS</td>
<td>3.83</td>
</tr>
<tr>
<td>IMB/S_Read_Expl_args=S_Read_Expl_n2</td>
<td>SUCCESS</td>
<td>3.97</td>
</tr>
<tr>
<td>IMB/S_Read_Indv_args=S_Read_Indv_n2</td>
<td>SUCCESS</td>
<td>4.05</td>
</tr>
<tr>
<td>IMB/Scatterv_args=Scatterv_n1</td>
<td>SUCCESS</td>
<td>4.07</td>
</tr>
</tbody>
</table>
Need: review the results to get direct feedback

Ex: Quick rerun

Tests may be uniquely rerun in the same condition

Tests may be GUI-reviewed (Flask Server)

From any build or generated archive

```sh
$ pcvs exec IMB/MPI_Barrier_n2
$ pcvs report
```

```
#---------------------------
# Intel (R) MPI Benchmarks 2017, MPI-IO part
#---------------------------
# Date : Tue Jan 31 09:00:38 2023
# Machine : x86_64
# System : Linux
# Release : 4.18.0-305.62.1.el8_4.x86_64
# Version : #1 SMP Thu Aug 11 12:07:27 EDT 2022
# MPI Version : 3.1
# MPI Thread Environment:

# Calling sequence was:
#
# /pcvs-benchmarks/.pcvs-build/test_suite/IMB/IMB-IO S_IWrite_Expl

# Minimum io portion in bytes: 0
# Maximum io portion in bytes: 16777216
#
# List of Benchmarks to run:
#
# S_IWrite_Expl

# For nonblocking benchmarks:

# Function CPU_Exploit obtains an undisturbed
# performance of 2671.55 MFlops
```

```
Setup configuration for your site

- A profile, YAML-based, carries information relative to environment.
  - Nodes, resources, parameters...

- These profiles are stored under a scope for easy access (« a la Git »: global, user & local)

- Base profiles for common use cases are available

- Profiles may even be split up: Config blocks, composable from multiple environments sharing the same filesystem

```
compiler:
  commands:
    cc: mpicc,
    cxx: mpicxx,
    fc: mpif90

criterion:
  iterators:
    n_mpi:
      subtitle: n
      values: [1, 2, 3, 4]

machine:
  concurrent_run: 4,
  cores_per_node: 4,
  name: localhost,
  nodes: 1

runtime:
  program: mpirun
  iterators:
    n_mpi:
      option: '-'np ' "
```
A run and beyond

- More than a single run, PCVS store results under a « database »: banks
  - Simplified Git repository
  - Commit = a whole run
  - Branch = run series

- Highlight progression in software developments
  - Easy to use from third-party tools

```bash
pcvs bank show demo
```

---

Projects contained in bank '/home/adamj/mnt/work/pcvs/demo.git':
- mpc  : 2 distinct testsuite(s)
  * mpc/d5d3468e3e9a8ec9bba9eb2885434292: 17 run(s)
  * mpc/b6ffe2123be606eab75f75b7dec00eba7d943461: 100 run(s)
- test  : 1 distinct testsuite(s)
  * test/d5d3468e3e9a8ec9bba9eb2885434292: 2 run(s)
- master : 1 distinct testsuite(s)
  * master: 0 run(s)
- myproject: 1 distinct testsuite(s)
  * myproject/d5d3468e3e9a8ec9bba9eb2885434292: 9 run(s)
A run and beyond

- Per-run test results are relevant but not enough
  - Execution Noise (load, disks..)
  - Binary interpretation
- Analysing test results over multiple runs offer more depth to the « big picture ».

- PCVS provide a DSL to query banks about save runs.
  - Build trends to see the actual progression in development
  - Requalify job status by adding/recompute test results based on measurements.

- Multiple analyses are provided by PCVS
  - Custom can be inserted as plugins
A run and beyond

- PCVS provides a DSL to query banks about save runs.
- Build trends to see the actual progression in development.
- Requalify job status by adding/recomputing test results based on measurements.
- Multiple analyses are provided by PCVS.
- Custom can be inserted as plugins.
- Per-run test results are relevant but not enough.
- Execution Noise (load, disks..)
- Analysing test results over multiple runs offer more depth to the "big picture".

Custom can be inserted as plugins.
Spack & PCVS

- PCVS is not a Spack package (yet)

- PCVS can interact with Spack in three manners:
  - Install/load packages as deps
  - Validate Spack recipes according to variant matrix
  - Run any test-defined packages (=translating to « spack test »)
PCVS is not a Spack package (yet)

PCVS can interact with Spack in three manners:

- Install/load packages as deps
- Validate Spack recipes according to variant matrix
- Run any test-defined packages (=translating to «spack test»)

Paratools

FOSDEM'23 — February 5th, 2023
Future Work

❖ Scheduled

✦ CLI-based GUI (through Textualize)
✦ Allow auto-parametrization over compilation tests
✦ Job packing into a single allocation
✦ Capturing metrics

❖ Wishlist

✦ Job workflow & visualization (conditional run path)
✦ Third-party exporters (Prometheus, Graylog)
✦ Better Spack CI/test integration
✦ Spack & Easybuild support for deployment
✦ Users & feedback :)

FOSDEM’23 — February 5th, 2023
Thank you for your attention

✓ https://pcvs.io/
✓ https://pcvs.readthedocs.io/
✓ https://github.com/cea-hpc/pcvs
✓ julien.adam@paratools.com