EVIDEN

HPCW

A Framework for Reproducible Benchmarks

David Guibert – CEPP EVIDEN

Nov. 9, 2023

an atos business

Outline

EVIDEN

1. Introduction

What, Why a benchmark

- Benchmarks
  - allow to score things
    - e.g. clusters (top500, green500), cpus, interconnects
  - allow to compare methods
    - mathematical methods
    - hardware solutions
    - compiler optimisations
    - etc.

- Also Benchmarks are vehicles
  - to collaborate inside a community
    - like the CMIP Projet
  - most importantly, to collaborate with technology providers
    - once the benchmark is adopted by the community

That's why it is critical to have a benchmark within a community to have impacts on technologies.

EVIDEN

HPC Benchmarks

- Historical Benchmarks in HPC
  - HPL: High Performance Linpack (1979), pure compute bounds
    - solving a dense system of linear equations $Ax = b$
    - LU decomposition with partial pivoting
    - provide the score of the TOP500 list (since June 1993)
  - Stream: pure memory bandwidth benchmarks
  - HPCG: High Performance Conjugate Gradients (since June 2014)
    - Conjugate Gradient method
    - "kinf of" balanced memory bandwidth benchmark

- not adequate, nor representative of the weather and climate community needs

That's why we propose HPCW benchmark.

EVIDEN

Introducing HPCW

- High Performance Climate and Weather (HPCW) benchmark for (pre)-exascale application of climate and weather codes
  - set of relevant and realistic, near-operational weather and climate workloads
  - developed in European projects: ESCAPE2, ESiWACE2 and, now ESiWACE3.
  - W&C tuned vehicle to collaborate with technology providers

HPCW description

- Current models available in HPCW (9 currently)
  - ICON - Ocean and Atmosphere
  - IFS inside RAPS
  - NEMO
  - Mini-applications (dwarfs):
    - IFS atmosphere FV dwarf IFS-FVM
    - Radiation dwarf ACRANEB2
    - ICON ocean advection dwarf
    - ECRad
    - ECTrans
    - Dwarf-P-CloudSC

- Verification procedures for automatic correctness checking
- 3 test-cases per model of increasing size (small, medium, big)
  - for scalability study
- Automatic performance metrics extraction
  - e.g. time to solution

EVIDEN

Results of HPCW benchmark
Automatically generated results at EVIDEN (v1.0)

- Hardware setup
  - BullSequana XH2000
    - CPU: 2x AMD EPYC 7763 64-Core Processor (MILAN)
    - + GPU: 4x Nvidia A100 GPU when needed
  - Memory: 256GB RAM DDR4 3200 MT/s
  - Interconnect: HDR 100 (fat tree)
- Software setup
  - OS: RHEL 8.3

HPCW_component-test_case-type	revision	time (s)	time_app
dwarf-p-radiation-acraneb2-lonlev-0.91-small	v1.0	0.16	
ecrad-small	v1.0	0.42	0.19
icon-atmo-small	v1.0	48.97	
icon-coupled-small-n24	v1.0	365.54	
icon-ocean-big	v1.0	1775.12	
icon-ocean-medium	v1.0	1046.89	
icon-ocean-small	v1.0	18.61	
icon-atmo-gpu-small	v1.0	31.66	
ifs-fvm-big	v1.0	8925.28	
ifs-fvm-medium	v1.0	2597.70	
ifs-fvm-small	v1.0	1102.11	
ifs-tco1999-big	v1.0	1263.78	1213.02
ifs-tco639-medium	v1.0	346.12	340.36
ifs-tl159-small	v1.0	20.25	16.52
nemo-orca25-medium	v1.0	1750.42	
nemo-bench-orca1-like-small	v1.0	56.16	

- Also HPCW has been validated on
  - differents architectures: AMD Milan/Genoa, Intel, Arm Ampere Altra, Nvidia GPU
  - different sites: Atos (Internal clusters), DKRZ (Levante), ECMWF (HPCF), CSC (LUMI)

EVIDEN

2. Presentation of the HPCW framework

What is a benchmark ?

- Benchmark
  1. a code
  2. one test-case
  3. verification procedure
  4. scoring metrics

- The results of the benchmark are its metrics

- The benchmark allows to compare
  - hardware
  - software stack
    - library optimisations
    - compilers, compiler flags

EVIDEN

A reproducible benchmark?

- Benchmark
  1. a code
     - a specific version
     - a specific configuration
     - its specific dependencies
  2. a test-case
     - specific input files (compatible with the code version)
     - the reference output files
  3. verification procedure
     - numerical error checking
  4. scoring metrics
     - time to solution
     - gflops
     - energy to solution

=> How the HPCW technical choices try to get reproducibility

Framework possibilities

The needs of a framework

- Each Weather and Climate model
  - comes with its build system (Makefile, CMake, ...)
  - needs different libraries and tools as dependencies

- We want a framework
  - to build all models and their dependencies the same way
  - to bench all models with their test cases
  - to report the results

- We want a framework
  - Simple, easy to use, to maintain and to extend
  - Agnostic to
    - each model build system
    - each cluster environment
    - each scheduler system
  - Customizable
    - adapt and change dependencies
    - change compilers and flags
    - allow optimizations at all levels

Framework possibilities
The existing frameworks

- Many build frameworks already exist:
  - CMake
  - Spack
  - Nix (or Nixpack=Nix+Spack)
  - EasyBuild
  - PkgSrc, Portage...

- Many bench framework already exist:
  - CTest
  - JUBE Benchmarking Environment (@Jülich SC)
  - Reframe (@CSCS)

Framework possibilities
The existing frameworks chosen

- Build frameworks under consideration for HPCW
  - CMake: SuperBuild approach (as VTK, Paraview)
    - first choice in early 2019
  - SPACK
    - since V2.0 of HPCW
    - a lot of dependencies already packaged
    - development and support in ESiWACE

- Bench framework under consideration for HPCW
  - CTest
  - shell script for automatic result extraction

EVIDEN

HPCW overview
Agnostic, Easy to use framework

- HPCW is a CMake-based framework
  - able to compile all the “models” on top of their own build system
    1. CMake SuperBuild
    2. SPACK recipes (optional usage but recommended)
    3. CTest
  - agnostic to
    - each code build system (autotools, Makefile, CMake, …)
    - each cluster environment (compilers/libraries version…)
    - each scheduler system (slurm, …) to launch test cases

- HPCW have a strong separation of concerns:
  - specificities are managed separately (in files described later on)
  - stored in the Git repository as well.

- The advantages of CMake and Spack are
  - to deal with the dependencies
  - and to deal with the dependencies of the dependencies
  - their scripting capabilities for automation
- we need tools to re-do, to reproduce benchmark results

HPCW overview

Multi model and multi dependencies management

HPCW requirements:

- CMake (>=3.19)
- Python (for SPACK repices)

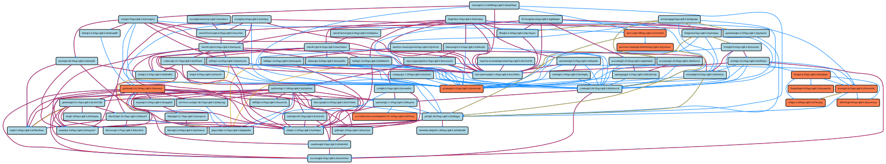

Figure: Overview the installed packages by HPCW for the 9 models

For CMake-recipes:

- C/C++ & Fortran compilers (gcc/gfortran, Intel icc/ifort)
- Python 2 (for IFS RAPS 18)
- Perl (FCM build system for NEMO)
- Optional MPI vendor optimized library

EVIDEN

3. Reproducibility "workflow"

Reproducibility workflow

- Workflow with HCPW full-CMake approach

```
source $env.sh
cmake −DENABLE_$model \
   −DCMAKE_TOOLCHAIN_FILE=$toolchain_file.cmake \
   −DHPCW_JOB_LAUNCHER=$job_launcher.sbatch \
   −S $HPCW_src −B build−$model
cmake −−build $build−$model
cd build−$model
ctest −VV |& tee tests.log
./analyse.sh tests.log
```

- $env.sh defines shell variables, modulefiles to be loaded, . . .
- $toolchain_file.cmake defines CMake variables
- $job_launcher.sbatch wraps the runs of the test-cases

- Note about the inputs files
  - manual download is required (with checksum verification available)
  - automatic exprimental download support with CMake ExternalData or datalad/git-annex tool

EVIDEN

Reproducibility workflow

- Workflow with HCPW using SPACK recipes and CMake approach
  - Use SPACK environment for reproducibility

```
git  clone https://github.com/spack/spack
git  reset --hard $SHA1
source spack/share/spack/setup-env.sh
spack -e env-$model install
spack -e env-$model load $model
```

  - Tell HPCW that the $model is already available (USE_SYSTEM_$model=ON)

```
source $env.sh
cmake -DENABLE_$model  \
   -DUSE_SYSTEM_$model=ON \
   -DCMAKE_TOOLCHAIN_FILE=$toolchain_file.cmake \
   -DHPCW_JOB_LAUNCHER=$job_launcher.sbatch \
   -S $HPCW_src -B build-$model
cmake --build $build-$model
cd build-$model
ctest -VV |& tee tests.log
./analyse.sh tests.log
```

EVIDEN

The SPACK environment

- Spack automates
  - the build
  - the installation of scientific software

- Packages are *parametrized* using specifications (specs)
  - for tweaking and tuning configuration
  - check the good documentation:
    `https://spack.rtfd.io`

- Environments are used to group together a set of specs for
  - building
  - rebuilding
  - deploying

Simple spack.yaml file

```yaml
spack:
  # include external configuration
  include:
  - ../special-config-directory/
  - ./config-file.yaml

  # add package specs to the `specs` list
  specs:
  - hdf5
  - libelf
  - openmpi
```

Concrete spack.lock file (generated)


```json
{
 "concrete_specs": {
  "6s63so2kstp3zyvjezglndmavy6l3nul": {
   "hdf5": {
       "version": "1.10.5",
       "arch": {
           "platform": "darwin",
           "platform_os": "mojave",
           "target": "x86_64"
       },
       "compiler": {
           "name": "clang",
           "version": "10.0.0-apple"
       },
```


EVIDEN

What about containers

- A great way to reproduce and distribute an already-built software stack

- Someone needs to build the container!
  - This isn't trivial
  - Containerized applications still have hundreds of dependencies

- Using the OS package manager inside a container is insufficient
  - generic binaries, not optimized for specific architectures
  - some binaries can have illegal instructions
    - e.g. for ARM systems, SVE application on NEON processor

- May need to be *rebuilt* to support many different hosts
  - not clear that we can ever build one container for all facilities
  - likely won't solve the N-platforms problem in HPC

4. Conclusion

Take aways

- Technology providers need trusted metrics
- Trusted metrics need Reproducibility
- Repoducibiliy needs trusted tools
- HPCW benchmark provides
  - Weather and Climate models (applications, mini-applications) from the European community
  - an easy way to deploy the sofware stack to build, verify and bench W&C models
  - tables to compare trusted and reproducible metrics
- We are open for collaboration: deployment and/or adding new components
  - david.guibert@eviden.com
  - open source version coming soon

EVIDEN

Questions

EVIDEN

Thank You!