University of Basel

Reproducible Stencil Compiler Benchmarks Using PROVA!

Danilo Guerrera, Helmar Burkhart, Antonio Maffia

University of Basel

danilo.guerrera@unibas.ch

PMMA 16

Thu, Jun 23, 2016

Overview

1. Reproducibility in Science
   - Problem, System, Method
   - Reproducibility Levels
2. PROVA!
   - Current Version
   - Environment Management
3. Use Cases
   - Wave
   - Molecular Dynamics
4. Conclusions

Reproducibility - A Science Principle

"Non-reproducible single occurrences are of no significance to science." (Karl Popper The Logic of Scientific Discovery 1934/1959)

Goals of the PROVA! Project

try, prove, convince

- Taxonomy for Reproducibility Levels

Goals of the PROVA! Project

try, prove, convince

- Taxonomy for Reproducibility Levels
- Performance Engineering Support

Goals of the PROVA! Project

try, prove, convince

- Taxonomy for Reproducibility Levels
- Performance Engineering Support
- Best Practice Demonstrator

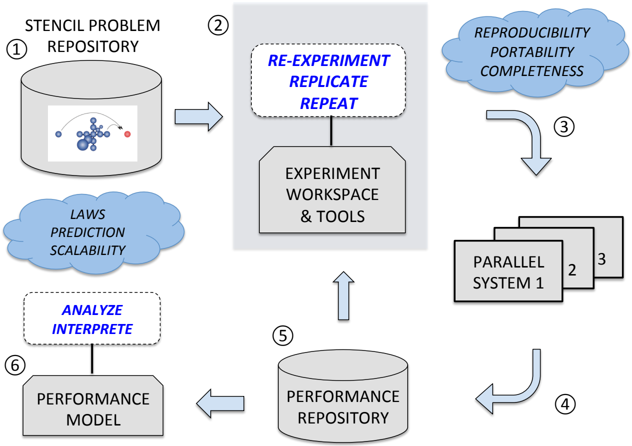
Goals of the PROVA! Project

try, prove, convince

- Taxonomy for Reproducibility Levels
- Performance Engineering Support
- Best Practice Demonstrator
- Modern HPC Education

Goals of the PROVA! Project

try, prove, convince



Problem / Method / System

A computational problem is solved by an algorithmic method on a compute system.

Problem / Method / System

A computational problem is solved by an algorithmic method on a compute system.

- **Problem**: specification of the problem including characteristic parameters.

Problem / Method / System

A computational problem is solved by an algorithmic method on a compute system.

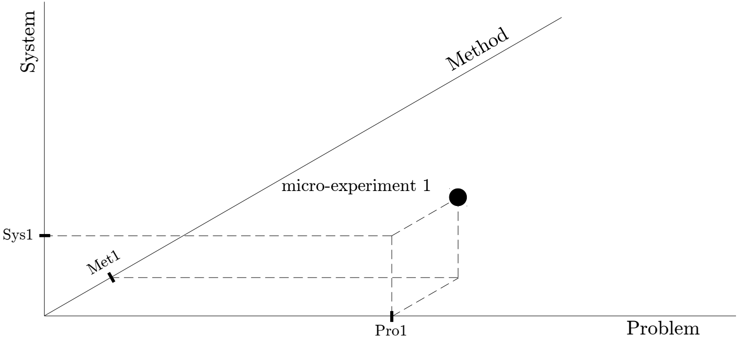
- **Problem**: specification of the problem including characteristic parameters.
- **Method**: description of the algorithmic approach used to tackle the problem.

Problem / Method / System

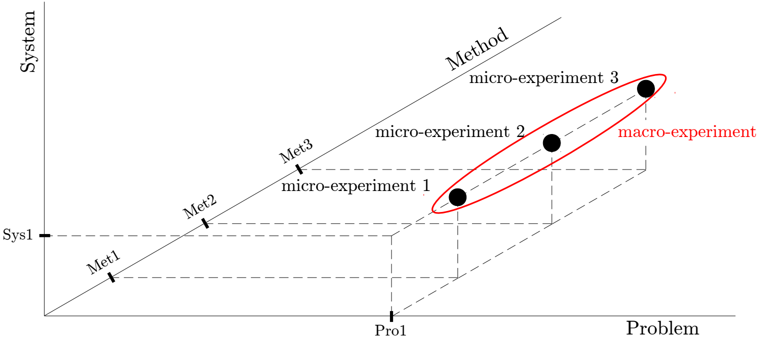
A computational problem is solved by an algorithmic method on a compute system.

- **Problem**: specification of the problem including characteristic parameters.
- **Method**: description of the algorithmic approach used to tackle the problem.
- **System**: representation of the compute environment (both hardware and software), on which an experiment is run.

Micro- and Macro-Experiments



Micro- and Macro-Experiments



Reproducibility Levels

- **Repetition**: re-running the original *micro-* or *macro-experiment* without any variation of the parameters, should drive to the same results and a certain level of credibility is guaranteed (**completeness** of documentation)

Reproducibility Levels

- **Repetition**: re-running the original *micro-* or *macro-experiment* without any variation of the parameters, should drive to the same results and a certain level of credibility is guaranteed (**completeness** of documentation)
- **Replication**: is related to the system hosting an experiment. An experiment should not be bound to a specific compute environment (**portability**)

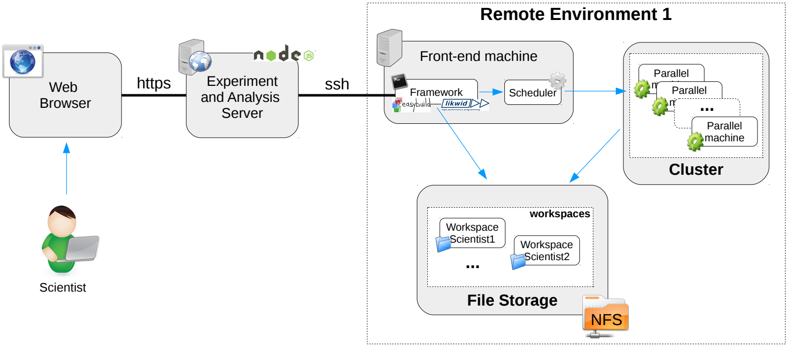
Reproducibility Levels

- **Repetition**: re-running the original *micro-* or *macro-experiment* without any variation of the parameters, should drive to the same results and a certain level of credibility is guaranteed (**completeness** of documentation)
- **Replication**: is related to the system hosting an experiment. An experiment should not be bound to a specific compute environment (**portability**)
- **Re-experimentation**: if changing the methods drives to the same outputs, the scientific approach is proven (**correctness** of the approach)

Functionalities Needed - Support Given

- Collaboration Support: git
- Software Management: EasyBuild, LMod
- Experiment Reproduction
- Experiment Portability
- Performance Modeling Support
- Visualization

PROVA! - Current Version of the Architecture



Lmod mudules

- developed at TACC[1]
- user's environment can be changed dynamically through modulefiles
- manages the PATH
- a modulefile contains information on how to run a particular application or provide access to a particular library

[1] https://www.tacc.utexas.edu/research-development/tacc-projects/lmod

Scientific Software Management and Build Via EasyBuild[2]

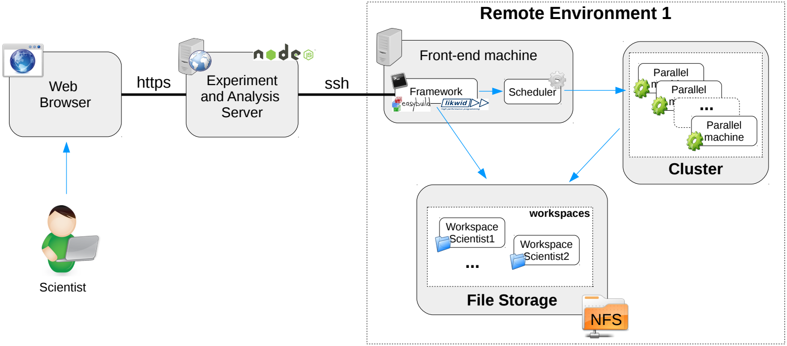
- a flexible framework for building/installing (scientific) software
- fully automates software builds
- keeps track of the versions
- consistent software stack
- allows for easily reproducing previous builds
- keep the software build recipes/specifications simple and human-readable
- supports co-existence of versions/builds via dedicated installation prefix and module files
- enables sharing with the HPC community

[2] http://hpcugent.github.io/easybuild/

Lmod + EasyBuild

- Easily install new software as module
- Clean environment for all of the users
- Keep track of the software installed
- Possibility to attach to an experiment a subset of modules
    - Export source code + environment

Web Application



Simple 3D Wave

University of Basel 2016

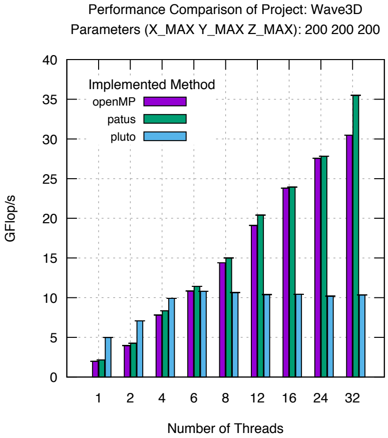
Problem	Calculate a 3-D wave equation of 200^3 elements (IEEE single precision arithmetic) in 100 timesteps
System	**SW:** OpenMP 4.0, GCC 4.9.2, PATUS 0.1.4, PLUTO 0.10 **HW:** 1 node • CPU: 2x AMD Opteron 6274 "Bulldozer" 16-Core, 2.2 GHz, 12 MiB L3 cache, 4 NUMA domains • RAM: 256 GiB • OS: Ubuntu 14.04.4, Kernel 3.8.0-38
Method	1. Naive OpenMP implementation with NUMA aware initialization (16 FLOPS) 2. DSL + auto-tuning with PATUS (20 FLOPS) 3. Polyhedral model with PLUTO (16 FLOPS)

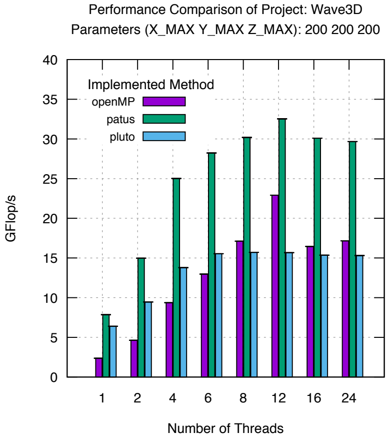
Simple 3D Wave

University of Erlangen 2016

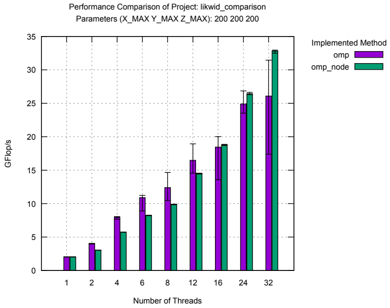
Problem	Calculate a 3-D wave equation of 200^3 elements (IEEE single precision arithmetic) in 100 timesteps
System	**SW:** OpenMP 4.0, GCC 4.9.2, PATUS 0.1.4, PLUTO 0.10 **HW:** 1 node • CPU: 2x Xeon 5650 "Westmere" 6 cores + SMT, 2.66 GHz, 12 MiB Shared Cache per chip, 2 NUMA domains • RAM: 24 GB (DDR3-1333) • OS: CentOS 6.7, Kernel 2.6.32-573.7.1.el6
Method	1. Naive OpenMP implementation with NUMA aware initialization (16 FLOPS) 2. DSL + auto-tuning with PATUS (20 FLOPS) 3. Polyhedral model with PLUTO (16 FLOPS)

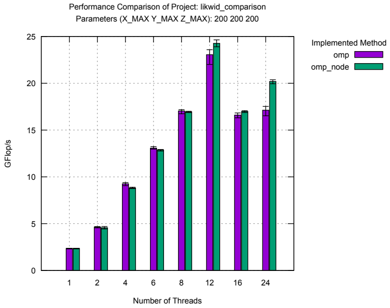
Simple 3D Wave (2)





Simple 3D Wave (3)





Demo

https://repro-hpc.dmi.unibas.ch

PROVA!: Adaptation Effort

- How to create a methodType?
- What is the effort?
  - What knowledge is needed?
  - How much time to invest in it?

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor

```
{
        "name":"GROMACS-5.0.5",
        "eb_modules": [
                "GROMACS/5.0.5-foss-2016a-hybrid-noGPU"
        ],
        "version":"5.0.5",
        "comment":"GROMACS, a molecular dynamics package primarily
                   designed for biomolecular systems such as proteins
                   and lipids, based on the foss-2016a toolchain, compiled
      with hybrid OpenMP and Open MPI, without CUDA support"
}
```

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor
- Create (if not existing) the easyconfigs for the needed modules

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor
- Create (if not existing) the easyconfigs for the needed modules

```
name = 'GROMACS'
version = '5.0.5'
versionsuffix = '-hybrid-noGPU'

homepage = 'http://www.gromacs.org'
description = """GROMACS is a versatile package to perform molecular
    dynamics, i.e. simulate the Newtonian equations of motion for systems
    with hundreds to millions of particles."""

toolchain = {'name': 'foss', 'version': '2016a'}
toolchainopts = {'openmp': True, 'usempi': True}

source_urls = ['ftp://ftp.gromacs.org/pub/gromacs/']
sources = [SOURCELOWER_TAR_GZ]

builddependencies = [
    ('CMake', '3.4.3'),
    ('libxml2', '2.9.2')
]

dependencies = [('Boost', '1.59.0', '-Python-2.7.11')]
# explicitely disable CUDA support
configopts = ' -DGMX_GPU=OFF'
moduleclass = 'bio'
```

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor
- Create (if not existing) the easyconfigs for the needed modules
- Create compile (Makefile) and run scripts

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor
- Create (if not existing) the easyconfigs for the needed modules
- Create compile (Makefile) and run scripts
- Install it in the tool

PROVA!: Adaptation Effort - Molecular Dynamics

- Create a method descriptor
- Create (if not existing) the easyconfigs for the needed modules
- Create compile (Makefile) and run scripts
- Install it in the tool
- Create a project and use it! [3]

[3] Thanks to Florent Hedin `http://www.chemie.unibas.ch/~hedin/`

PROVA!: Adaptation Effort - Molecular Dynamics

```
     R E A L   C Y C L E   A N D   T I M E   A C C O U N T I N G

On 2 MPI ranks, each using 8 OpenMP threads

 Computing:          Num   Num      Call    Wall time         Giga-Cycles
                     Ranks Threads  Count      (s)         total sum    %
-----------------------------------------------------------------------------
 Domain decomp.        2    8        960       7.755        272.988   2.2
 DD comm. load         2    8         39       0.001          0.019   0.0
 Neighbor search       2    8        961      10.492        369.350   2.9
 Comm. coord.          2    8      18240       2.674         94.139   0.7
 Force                 2    8      19201     229.409       8075.973  64.3
 Wait + Comm. F        2    8      19201       3.283        115.568   0.9
 PME mesh              2    8      19201      67.117       2362.756  18.8
 NB X/F buffer ops.    2    8      55681       8.287        291.731   2.3
 Write traj.           2    8          1       0.029          1.035   0.0
 Update                2    8      19201       4.215        148.368   1.2
 Constraints           2    8      19201      18.922        666.110   5.3
 Comm. energies        2    8        961       0.039          1.372   0.0
 Rest                                          4.639        163.315   1.3
-----------------------------------------------------------------------------
 Total                                       356.862      12562.724 100.0
-----------------------------------------------------------------------------
```

PROVA!: Adaptation Effort - Molecular Dynamics

```
-----------------------------------------------------------------------------
 Breakdown of PME mesh computation
-----------------------------------------------------------------------------
 PME redist. X/F        2    8      38402      13.482        474.601   3.8
 PME spread/gather      2    8      38402      32.375       1139.695   9.1
 PME 3D-FFT             2    8      38402      12.703        447.171   3.6
 PME 3D-FFT Comm.       2    8      38402       7.780        273.874   2.2
 PME solve Elec         2    8      19201       0.623         21.918   0.2
-----------------------------------------------------------------------------

               Core t (s)   Wall t (s)        (%)
       Time:    5701.124      356.862     1597.6
                 (ns/day)    (hour/ns)
Performance:        9.298        2.581
Finished mdrun on rank 0 Sat Jun 18 11:53:30 2016
```

Conclusions

- Reproducibility needs to be emphasized in the performance modeling.
- Repeatability of an experiment only possible if precise description of experiment is given: Problem, System, and Method.
- Repeatability: World-wide access to experiments through Internet feasible (security and authentication mechanisms essential).
- Replication and re-experimentation: harder to achieve but not impossible.

Future Work

Short term:

- Jobs: no clue about when the job finishes its execution
- Homogeneity of nodes: libraries and sw are installed on a shared NFS so all the nodes must be homogeneous to run such sw
- Experiment is run as a block: bad resource usage
- Installation of the modules is simply delegated to EasyBuild
- Visualization is not so powerful

Mid term:

- Provenance of the experiments
- Collaborative Performance Engineering
- Integrate Performance Models to Evaluate Performance Outputs
- Towards a Science Gateway

Interested?

https://prova.io