Making reproducible and publishable large-scale HPC experiments

Philippe SWARTVAGHER
https://ph-sw.fr

HPC, Big Data & Data Science devroom - FOSDEM’24
About me

- Assistant Professor in Computer Science at ENSEIRB-MATMECA engineering school, Bordeaux, France

- Research about HPC, MPI, runtime systems, performance analysis and visualisation, ...
  > Experimental work: writing/modifying software, evaluating performance

- PhD thesis:
  > On the Interactions between HPC Task-based Runtime Systems and Communication Libraries
  > 2019-2022

- Doing HPC experiments for several years now!
In this presentation...

- Feedbacks from making several reproducible articles
- Advertisement for Guix
- Some advice (hopefully!)

- Apply mostly to scientific publications
  > But probably also to, e.g., blog posts
  > And not only HPC!
What you can find in publications...

Philippe Swartvagher, Sascha Hunold, Jesper Larsson Träff, Ioannis Vardas.
Using Mixed-Radix Decomposition to Enumerate Computational Resources of Deeply Hierarchical Architectures.
SC 2023 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Nov 2023, Denver, CO, United States.

Cool algorithms!

Algorithm 3 Generate list of cores for --cpu-bind=map_cpu

<table>
<thead>
<tr>
<th>Inputs</th>
<th>h : hierarchy of one compute node, σ : permutation, n : number of cores to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>l : list of core physical IDs</td>
</tr>
</tbody>
</table>

1. $l \leftarrow []$
2. $N \leftarrow 1$
3. for $i = 0$ to $|h| - 1$
4. $N = N \times h[i]$
5. end for
6. for $c = 0$ to $N - 1$
7. $r \leftarrow \text{COMPUTE_NIRANK}(h, c, \sigma)$
8. if $r < n$ then
9. $l[i] \leftarrow c$
10. end if
11. end for

For applications that do not use all cores of compute nodes, the mixed-radix decomposition technique can be used for two distinct steps: (i) selecting cores and (ii) scheduling the MPI tasks. Each...

(a) With 1 NIC per compute node

Figure 8: Impact of process mapping of Splatt executions on...

in a different order are evaluated. Bars correspond to the median duration of 5 executions and error bars delimit the best and worst durations. For a given number of MPI processes, orders with bar...
What you can find in publications...

- Cool algorithms!
- Nice results!

Philippe Swartvagher, Sascha Hunold, Jesper Larsson Träff, Ioannis Vardas.
Using Mixed-Radix Decomposition to Enumerate Computational Resources of Deeply Hierarchical Architectures.
SC 2023 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Nov 2023, Denver, CO, United States.

« How did you actually implement this algorithm? »
« How did you run the program? »
« Which options did you use? »
« I don't understand, show me the code! »
What you can find in publications...

« How did I actually implement this algorithm? »

« How did I run the program? »

« Which options did I use? »

« I don’t understand, show me the code! »

« How did I compute statistics? »

Nice results!

Figure 8: Impact of process mapping of Splatt executions in a different order are evaluated. Bars correspond to the median duration of 5 executions and error bars delimit the best and worst durations. For a given number of MPI processes, orders with bars...
Reproducibility

- Many terms, slightly different meanings:
  - Reproducibility
  - Replicability
  - Repeatability
  - Availability

- Make available everything needed to reproduce something, e.g., an experiment
  - In this case: scripts and source code

https://www.acm.org/publications/policies/artifact-review-and-badging-current
Why should I care?

- Conferences/Journals require it and give you nice badges
- For you:
  > To easily come back to an experiment 6 months later
  > To have trust in your experiments
  > Some kind of open-source?
- For others:
  > To see what you actually did in practice
  > To better understand what you are talking about
  > To reproduce your experiments (to change or extend it, to compare themselves with it, ...)
  > To easily share with your colleague/collaborator/... what you did and how
- Part of the contribution!

February 3, 2024
The common workflow

Software → Build / Installation → Execution → Output data → Analysis
The common workflow

- **Software**
  - Source code, version, patches, ...

- **Build / Installation**
  - Building / Installation scripts

- **Execution**
  - Execution scripts

- **Output data**
  - Post-processing, plotting, ...

- **Analyzis**
The common workflow

Software → Build / Installation → Execution → Output data → Analyzis

Source code, version, patches, ...

Building / Installation scripts

Execution scripts

Post-processing, plotting, ...

What you need to make available (+documentation)
Different levels

Availability → Bit-to-bit reproducibility

- Availability is a minimum
- Bit-to-bit reproducibility when you manage to rebuild the exact same environment
  > Not always possible
  > Not always necessary: on a different system, with different input data or configuration, ...
Software

- Used software, version, where to download
- But also: which compiler? Which compiler version? Which version of library X and Y? ...
- The whole software environment is important!

Source code, version, patches, ...

- How software were installed? Which building flags? Which dependencies?

Building / Installation scripts
## Software

### Minimum

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Website</th>
<th>Code location</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewMadeleine</td>
<td>2021-05-21</td>
<td><a href="https://pm2.gitlabpages.inria.fr/NewMadeleine">https://pm2.gitlabpages.inria.fr/NewMadeleine</a></td>
<td><a href="https://gitlab.inria.fr/pm2/pm2.git">https://gitlab.inria.fr/pm2/pm2.git</a></td>
<td>27656f86b2c77af733778643f53e90a08e6a29715a</td>
</tr>
<tr>
<td>StarPU</td>
<td>1.3.8</td>
<td><a href="http://starpu.gforge.inria.fr">http://starpu.gforge.inria.fr</a></td>
<td><a href="https://gitlab.inria.fr/starpu/starpu.git">https://gitlab.inria.fr/starpu/starpu.git</a></td>
<td>starpu-1.3.8</td>
</tr>
<tr>
<td>Chameleon</td>
<td>1.0</td>
<td><a href="https://gitlab.inria.fr/solverstack/chameleon">https://gitlab.inria.fr/solverstack/chameleon</a></td>
<td><a href="https://gitlab.inria.fr/solverstack/chameleon">https://gitlab.inria.fr/solverstack/chameleon</a></td>
<td>4cbb899ca30d29927018d82964b69bc6517269abe1</td>
</tr>
<tr>
<td>StarVZ</td>
<td>0.5.1</td>
<td><a href="https://github.com/schnorr/starvz">https://github.com/schnorr/starvz</a></td>
<td><a href="https://github.com/schnorr/starvz">https://github.com/schnorr/starvz</a></td>
<td>b789286a90e22ae0cd7a6a58d3ff2bdff02e3</td>
</tr>
<tr>
<td>GCC</td>
<td>11.2.0</td>
<td><a href="https://gcc.gnu.org/">https://gcc.gnu.org/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slurm</td>
<td>19.05.8</td>
<td><a href="https://slurm.schedmd.com/">https://slurm.schedmd.com/</a></td>
<td><a href="https://download.schedmd.com/slurm/slurm-19.05.8.tar.gz">https://download.schedmd.com/slurm/slurm-19.05.8.tar.gz</a></td>
<td></td>
</tr>
<tr>
<td>FxT</td>
<td>0.3.14</td>
<td><a href="https://savannah.nongnu.org/projects/fxt">https://savannah.nongnu.org/projects/fxt</a></td>
<td><a href="http://download.savannah.nongnu.org/releases/fxt/fxt-0.3.14.tar.gz">http://download.savannah.nongnu.org/releases/fxt/fxt-0.3.14.tar.gz</a></td>
<td></td>
</tr>
<tr>
<td>Matplotlib</td>
<td>3.1.2</td>
<td><a href="https://matplotlib.org/">https://matplotlib.org/</a></td>
<td><a href="https://files.pythonhosted.org/packages/source/m/matplotlib/matplotlib-3.1.2.tar.gz">https://files.pythonhosted.org/packages/source/m/matplotlib/matplotlib-3.1.2.tar.gz</a></td>
<td></td>
</tr>
<tr>
<td>Pandas</td>
<td>1.3.0</td>
<td><a href="https://pandas.pydata.org">https://pandas.pydata.org</a></td>
<td><a href="https://files.pythonhosted.org/packages/source/p/pandas/pandas-1.3.0.tar.gz">https://files.pythonhosted.org/packages/source/p/pandas/pandas-1.3.0.tar.gz</a></td>
<td></td>
</tr>
</tbody>
</table>

NewMadeleine is built with the following commands:

```bash
# NewMadeleine
# git clone https://gitlab.inria.fr/pm2/pm2.git
# cd pm2
# git checkout 6e5a6b0d78d78dd06b115b3f8af22177bdf5e39
# cd scripts
# ./pm2-build-packages ./pkg/build/madpmi-mini.conf --prefix=/installationprefix
# Set environment variables as indicated at the end of installation
```

memory-contention is built with the following commands:

```bash
# memory-contention
# git clone https://gitlab.inria.fr/pswartva/memory-contention.git
# cd memory-contention
# git checkout e2d70eef718380c818f0a07e82f0df9e8c9b4870
# ./autogen.sh
# mkdir build
# cd build
# ./configure
# make
```
Software

- What about...
  - Module files?
  - Spack / Conda / Easybuild / ... environments?
  - Docker / Singularity / ... images?

Source code, version, patches, ...

Building / Installation scripts
Software

What about...

- Module files → specific to a system, don’t say how it is build, may disappear after some time

- Spack / Conda / Easybuild / ... environments → don’t fully isolate the environment, depends on what is already installed on the system

- Docker / Singularity / ... images → what is inside the image? Building the image 6 months later may contain different things

No guarantee of being always reproducible!
Even worse: give the feeling of being reproducible
Enters... Guix!

- From https://hpc.guix.info:
  - Transactional package manager
  - Create as many software environments as you like (like virtualenv but not only for Python; like module, but for every software defined in Guix)
  - The software environments created with Guix are fully reproducible: a package built from a specific Guix commit on your laptop will be exactly the same as the one built on the HPC cluster you deploy it too, usually bit-for-bit.
Software

Source code, version, patches, ...

Building / Installation scripts

module load openmpi/4.1.2
# Build chameleon
cmake .. -DFOO=BAR
make && make install
mpirun ...
guix shell --pure chameleon -- mpirun ...
Software

- All Guix package definitions are located in Guix channels
  > Actually Git repositories
  > → Commit hashes of used Guix channels pin versions of all packages
    (solves problem of pinning versions of dependencies of dependencies)
  > Everything except the kernel

Source code, version, patches, ...

Building / Installation scripts
Software

- Export currently used channels (and their versions):
  `guix describe -f channels > channels.scm`

- Explicitly use pinned channels:
  `guix time-machine --channels=./channels.scm -- shell --pure chameleon -- mpirun ...`

- Backup channels.scm: to be sure to execute the same code, even 6 months later

**Source code, version, patches, ...**

**Building / Installation scripts**
The Guix killer feature: package transformations

- Change package definition on-the-fly

- Simple package substitution:
  guix shell --pure chameleon --with-input=openblas=mkl -- mpirun ...

- Use a specific upstream branch, commit, version:
  guix shell --pure chameleon --with-commit=starpu=acae6e -- mpirun ...

- Apply a patch to package source code:
  guix shell --pure chameleon --with-patch=chameleon=./foo.patch -- mpirun ...

- And others

- Makes the installation of software much easier!
  > No need for installation scripts and instructions anymore!

Guix: what about performance?

- Should be the same
- What is not in Guix: tuning of libraries made by cluster providers
  > @Cluster providers: please share these modifications!
- What is in Guix: package transformation `--tune` to rebuild package for a specific processor architecture
- One of the goal of the Guix-HPC effort

https://hpc.guix.info
https://hpc.guix.info/blog/2022/01/tuning-packages-for-a-cpu-micro-architecture/
https://hpc.guix.info/blog/2019/12/optimized-and-portable-open-mpi-packaging/
Comment!
  > What you are doing and why

Try to separate what is specific to *your* experiment (platform, input data, ...) and experiment logic:
  > Job scheduler system
  > Used resources (number of nodes, cores, ...)
  > Paths, usernames, problem size, input data, ...
Post-processing, plotting, ... scripts

- Also executed inside a Guix environment!
- Separate post-processing (analyzing data, computing what will be plotted) from plotting
  > Ease (and accelerate!) the writing of plotting scripts
- Factorization (moving things to functions, modules, ...) is not always a good idea
  > You may need to add annotation to this specific plot
  > You may need to compute this specific value only for this kind of data
  > Scripts have to remain flexible enough
Directly generate from scripts codes of table to be included in your TeX file

Think if you need to change how you compute all the values...

\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\texttt{\%} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} & {\texttt{\%}} \\
\hline
\end{tabular}
Document things

- In a README.md

- What is this about? Link to the paper

- What do I need for described experiments?
  - Which hardware, how many nodes, ...
  - How much storage, RAM, ...
  - How much time

- Installation and execution instructions with **and** without Guix

- For each table, plot, ... in the paper:
  - Section about how it is done, folder containing scripts and other relevant resources
Where to make it available?

- A Git repository
- Dedicated repository: paper-title-reproducibility → paper-title-r13y
Where to make it available?

- A Git repository
- Dedicated repository: paper-title-reproducibility → paper-title-r13y
- Archive it forever on SoftwareHeritage!
  > Will give you a SWHID to identify your repository / snapshot / directory
  > Like a DOI but computed based on archived content (like a Git commit ID)

https://archive.softwareheritage.org/
(Almost) Last remarks

- Have reproducibility in mind from the beginning of your experiments
- If *bit-to-bit reproducibility* seems difficult, at least publish your code and scripts
  > To have at least *availability*
- Guix is not mandatory!
  > But very handy tool to get *bit-to-bit reproducibility* for (almost) free
- What about data (input, output, pre- and post-processed)?
  > Don’t have a strong opinion yet
  > Host everything on Zenodo or equivalent?
  > Mandatory for reproducibility
Some initiatives

- **ReScience C Journal**
  > Publication of articles explaining how another article was replicated (or not)
  > Open and public reviewing process

- **Ten Years Reproducibility Challenge**
  > Reproduce one of your 10 year old articles

- **Guix Past**
  > Guix channel containing old software, old versions

- **Follow the activity of Guix-HPC for blog posts and events!**

https://rescience.github.io/
https://rescience.github.io/ten-years/
https://gitlab.inria.fr/guix-hpc/guix-past
https://hpc.guix.info/
Conclusion

- Publish source code of your software and experiment scripts
- Document it

- Reproducibility adds more value to experiments, results, research
- You will be more confident about your experiments
  > Especially in case you need to run them again
- It contributes to makes a better science!
  > Spread the word!

https://hpc.guix.info/blog/2023/06/a-guide-to-reproducible-research-papers/
https://gitlab.tuwien.ac.at/philippe.swartvagher/paper-mpi-rank-reordering-r13y
https://gitlab.inria.fr/pswartva/paper-starpu-traces-r13y/
https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y