Using Guix for scientific, reproducible, and publishable experiments

Ten Years of Guix – September 16, 2022

Philippe SWARTVAGHER
Inria Bordeaux – Sud-Ouest
Few words about me

- PhD Student @ Inria Bordeaux
- HPC: interactions between task-based runtime systems and communication libraries
- Occasional Guix user
  - For my experiments
  - Maintainer of several packages in Guix-HPC channel
Agenda

1. Software environment
2. Experiments without Guix
3. Experiments *with* Guix
4. Reproducible experiments
Experimental protocol

1. Development, tries and failures on my laptop

2. Experiments on clusters
   • Job scheduler (SLURM, OAR, …)
   • Non-interactive: submit a job executing a script, wait for its execution
   • At one point: experiments whose results will be published
My software stack

User application

Task-based runtime system

Network communications

Traces

Machine

Chameleon

StarPU

nmad

FxT

Daltons, PlaFRIM, Grid5k, Occigen, ...
My software stacks!

- Several possible combinations of building parameters
- Rebuild the whole stack!
Experimental protocol and variants

1. Development, tries and failure on my laptop

2. Experiments on clusters
   - Job scheduler (SLURM, OAR, …)
   - Non-interactive: submit a job executing a script, wait for its execution
   - At one point: experiments whose results will be published

A main variant

Comparaison of several variants of the same stack:
- nmad
- madmpi
- openmpi
Several variants simultaneously

How to switch from a variant to another one?

• Rebuild everything?
  • Too long
• Prevent using simultaneously different variants
Level 0: PATH, LD_LIBRARY_PATH, etc

Each variants installed in its own folder hierarchy

- `--prefix=$HOME/builds/nmad/` at compile-time
  - Small script to wrap all these commands: `./build.sh nmad && ./build.sh madmpi`
  - `PATH=$HOME/builds/nmad/bin LD_LIBRARY_PATH=$HOME/builds/nmad/lib` to run

- OK in scripts for non-interactive jobs
- But in interactive jobs: need to remember all variables and paths to define, need to type them...
Level 1: modules

Each variants installed in its own folder hierarchy

- `--prefix=$HOME/builds/nmad/` at compile time
- Small script to wrap all these commands: `./build.sh nmad && ./build.sh madmpi`

- `PATH`, `LD_LIBRARY_PATH`, ... defined in modules files

  module load nmad
  module unload nmad
  module load madmpi

- OK in scripts for non-interactive jobs
- OK in interactive scripts

<table>
<thead>
<tr>
<th>set</th>
<th>name</th>
<th>nmad</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>prefix</td>
<td>$HOME/builds/nmad/</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prepend-path</th>
<th>PATH</th>
<th>$prefix/bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepend-path</td>
<td>LIBRARY_PATH</td>
<td>$prefix/lib</td>
</tr>
<tr>
<td>prepend-path</td>
<td>LD_LIBRARY_PATH</td>
<td>$prefix/lib</td>
</tr>
<tr>
<td>prepend-path</td>
<td>INCLUDE</td>
<td>$prefix/include</td>
</tr>
<tr>
<td>prepend-path</td>
<td>C_INCLUDE_PATH</td>
<td>$prefix/include</td>
</tr>
<tr>
<td>prepend-path</td>
<td>CPLUS_INCLUDE_PATH</td>
<td>$prefix/include</td>
</tr>
<tr>
<td>prepend-path</td>
<td>PKG_CONFIG_PATH</td>
<td>$prefix/lib/pkgconfig</td>
</tr>
</tbody>
</table>
Final boss of level 1

Harder:
- Comparaisons between branches of the same library
- Comparaisons between commits of the same library
- Comparaisons with and without a patch applied to a library

- Different folders, module files, ... again?
  - In this case, source code is modified, not the result of its compilation!

- How to know which source code was used to build the software we are using...
  - Right now?
  - 6 months ago?
Guix!

- No package installed with Guix (no `guix install`)
- Use of `guix shell` instead
  - Required packages are built on-the-fly

```
./build.sh --starpu --chameleon openmpi
module load openmpi
module load starpu-openmpi
module load chameleon-openmpi
mpirun ...
guix shell --pure chameleon -- mpirun ...
```
Several variants simultaneously

openmpi variant:
- `guix shell --pure chameleon -- mpirun ...
- Chameleon depends on StarPU, which depends on OpenMPI

Default variant

nmad variant:
- `guix shell --pure chameleon --with-input=openmpi=nmad -- mpirun ...

madmpi variant:
- `guix shell --pure chameleon --with-input=openmpi=nmad-mini -- mpirun ...

Variant with fxt:
- `guix shell --pure chameleon --with-input=starpu=starpu+fxt -- mpirun ...

Existing packages in Guix-HPC
Package transformations


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

- Use a specific upstream Git branch:
  - `guix shell --pure chameleon --with-branch=starpu=coop-mcast -- mpirun ...

- Use a specific upstream commit:
  - `guix shell --pure chameleon --with-commit=starpu=acae6e78df7a9475bfbfd26e33fe324b1f7bedce -- mpirun ...

- Apply a patch to package source code:
  - `guix shell --pure chameleon --with-patch=chameleon=./wait-graph.patch -- mpirun ...
Package transformations

- Combinations of several transformations!
- Be careful to transformation order:
  - `--with-input=openmpi=nmad --with-branch=nmad=master`: OK, master branch of nmad
  - `--with-branch=nmad=master --with-input=openmpi=nmad`: version specified in nmad package

- Visualize applied transformations with:
  - `guix graph -M 4 chameleon --with-input=openmpi=nmad --with-branch=nmad=master | xdot -`
Final boss of level 1

with Guix!

Harder:
- Comparaisons between branches of the same library
- Comparaisons between commits of the same library
- Comparaisons with and without a patch applied to a library

- Different folders, module files, ... again?
  - In this case, source code is modified, not the result of its compilation!

- How to know which source code was used to build the software we are using...
  - Right now?
  - 6 months ago?
Final boss of level 1

with Guix!

Harder:
• Comparaisons between branches of the same library
• Comparaisons between commits of the same library
• Comparaisons with and without a patch applied to a library

• Different folders, module files, … again?
  • In this case, source code is modified, not the result of its compilation!

• How to know which source code was used to build the software we are using...
  • Right now?
  • 6 months ago?
Final boss of level 1

Harder:
• Comparaisons between branches of the same library
• Comparaisons between commits of the same library
• Comparaisons with and without a patch applied to a library

• Different folders, module files, ... again?
  • In this case, source code is modified, not the result of its compilation!

• How to know which source code was used to build the software we are using...
  • Right now?
  • 6 months ago?
Reproducibility: the problem

```
guix shell --pure chameleont -- mpirun ...
...
6 months later ...
guix pull
```

- `chameleon` != `chameleon`
- Different package version of `chameleon`
- Different versions of `chameleon`'s dependencies
Reproducibility: the solution

- 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
Reproducibility: making scripts available

• Making experiments (scripts) with reproducibility in mind
• Public Git repository with scripts and instructions for reproducibility:
  • Detailed README to understand what is done, how, where, ...
  • Contains channels.scm
  • Instructions to use also without Guix
• Examples:
  • https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y
  • https://gitlab.inria.fr/pswartva/paper-starpu-traces-r13y
Reproducibility: in papers

• Ask SoftwareHeritage for a snapshot of your repository
  • Repository available forever
  • https://archive.softwareheritage.org/save/
  • Provide a unique identifier, to find the saved repository

• In the paper:

  A public companion contains the instructions to reproduce our study:
  https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y,
  archived on https://www.softwareheritage.org/ with the ID
  swh:1:snp:306f7c10cf69a5860587e5aad62b76070b798ecd.
Conclusion: Guix’s advantages

- Very easy to move from a machine to another one*,**,
  - No wasted time to reinstall, recompile, look for appropriate modules, ...
  - *As long as the job scheduler is the same
  - **Require to parametrize/factorize scripts from the beginning

- More confidence in experiment executions
  - Especially if I need to run them again (with different parameter, ...)
Conclusion: future work

- Use manifest files
  - Put all parameters of `guix shell` in a file
  - Good way to factorize code?
- Use on a machine without Guix
  - `guix pack`