Expériences reproductibles avec Guix

Café Guix – 11 mars 2022

Philippe SWARTVAGHER
Inria Bordeaux – Sud-Ouest

À propos

- Doctorant à l’Inria Bordeaux
- HPC : interactions entre les runtimes à tâches et les bibliothèques réseau
- Utilisateur occasionnel de Guix
  - Pour les expériences
  - Mainteneur de quelques paquets dans Guix-HPC

Plan

1. Environnement logiciel
2. Expériences sans Guix
3. Expériences *avec* Guix
4. Expériences reproductibles

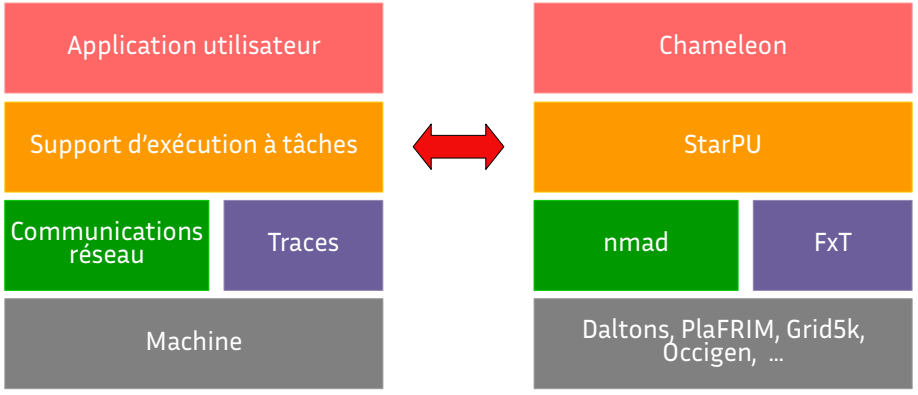
Protocole expérimental

1. Développement, essais en local (portable)

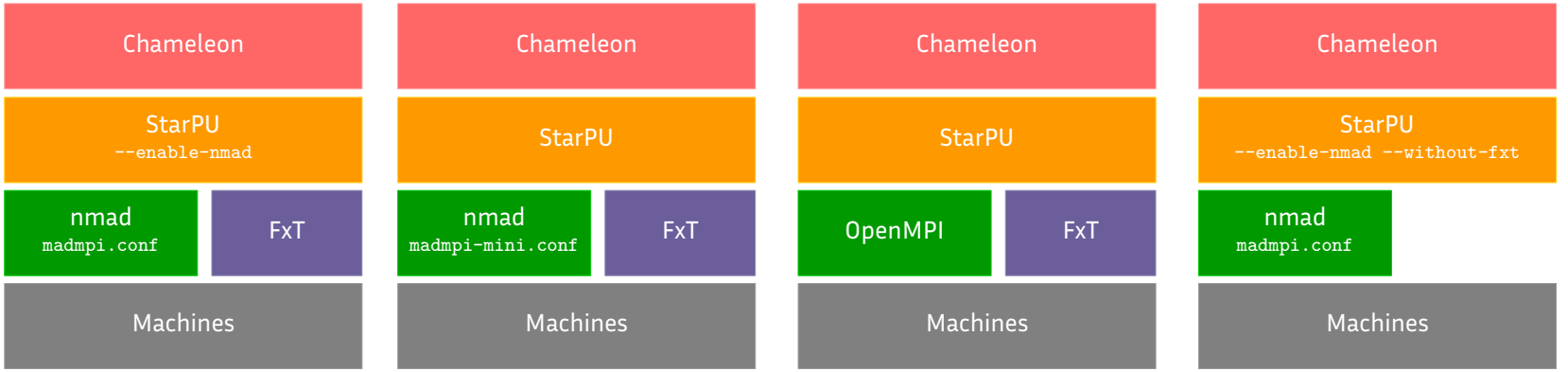
2. Expériences sur clusters

   - Gestionnaire de travaux (SLURM, OAR, …)
   - Non-interactif
   - À un moment : expériences pour résultats à publier

Ma pile logicielle



Mes piles logicielles !



- Plusieurs combinaisons de paramètres de compilation possibles
- Recompilation de toute la pile !

Protocole expérimental et variantes

1. Développement, essais en local (portable)

Une variante principale

2. Expériences sur clusters

- Gestionnaire de travaux (SLURM, OAR, …)
- Non-interactif
- À un moment : expériences pour résultats à publier

Comparaisons des différentes variantes de la pile :

- `nmad`
- `madmpi`
- `openmpi`

Plusieurs variantes en même temps

Comment passer d’une variante à l’autre ?

- Recompiler à chaque fois ?
  - Trop long
  - Empêche l’utilisation simultanée de différentes variantes

Niveau 0 : `PATH`, `LD_LIBRARY_PATH`, *etc*

Chaque variante installée dans son dossier dédié !

- `--prefix=$HOME/builds/nmad/` à la compilation
  - Petit script pour enrober tout ça : `./build.sh nmad && ./build.sh madmpi`
- `PATH=$HOME/builds/nmad/bin LD_LIBRARY_PATH=$HOME/builds/nmad/lib` à l'exécution

- OK dans les scripts pour les jobs non-interactifs
- Mais dans des jobs interactifs...

Niveau 1 : modules

Chaque variante installée dans son dossier dédié !

- `--prefix=$HOME/builds/nmad/` à la compilation
  - Petit script pour enrober tout ça : `./build.sh nmad && ./build.sh madmpi`
- `PATH`, `LD_LIBRARY_PATH`, … déclarés dans des **fichiers modules**

```
module load nmad
module unload nmad
module load madmpi
```

```
set     name            nmad
set     prefix          $HOME/builds/nmad/

prepend-path    PATH                $prefix/bin
prepend-path    LIBRARY_PATH        $prefix/lib
prepend-path    LD_LIBRARY_PATH     $prefix/lib
prepend-path    INCLUDE             $prefix/include
prepend-path    C_INCLUDE_PATH      $prefix/include
prepend-path    CPLUS_INCLUDE_PATH  $prefix/include
prepend-path    PKG_CONFIG_PATH     $prefix/lib/pkgconfig
```

- OK dans les scripts pour les jobs non-interactifs
- OK dans les jobs interactifs

Monstre final du niveau 1

Plus difficile :

- Comparaisons entre branches d’une bibliothèque
- Comparaisons entre commits d’une bibliothèque
- Comparaison avec ou sans patch dans une bibliothèque

- Dossiers et modules… ?
  - Là, les sources sont modifiées !

- Comment savoir à quelles sources correspond ce qui est compilé...
  - Actuellement ?
  - Il y a 6 mois ?

Guix !

- Aucun paquet installé avec Guix (pas de `guix install`)
- Utilisation des environnements `guix shell`
  - Construction des paquets nécessaires à la volée

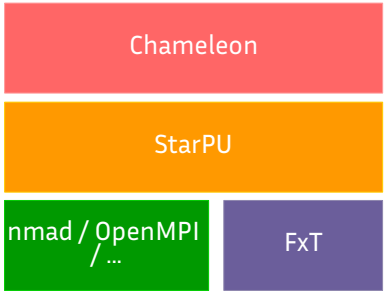
```
./build.sh --starpu --chameleon openmpi
module load openmpi
module load starpu-openmpi
module load chameleon-openmpi
mpirun ...
```

```
guix shell --pure chameleon -- mpirun ...
```

Plusieurs variantes en même temps

avec Guix !

- Variante `openmpi` :
  - `guix shell --pure chameleon -- mpirun …`
  - Chameleon dépend de StarPU, qui dépend de OpenMPI
  - Variante « par défaut »
- Variante `nmad` :
  - `guix shell --pure chameleon` **`--with-input=openmpi=nmad`** `-- mpirun …`
- Variante `madmpi` :
  - `guix shell --pure chameleon --with-input=openmpi=`**`nmad-mini`** `-- mpirun …`
- Variante avec `fxt` :
  - `guix shell --pure chameleon --with-input=`**`starpu=starpu+fxt`** `-- mpirun …`



Paquet déjà existants dans Guix-HPC

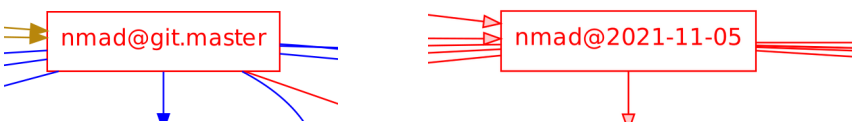
Transformations de paquets

- https://guix.gnu.org/manual/fr/html_node/Options-de-transformation-de-paquets.html
- Simple substitution de paquet :
  - `guix shell --pure chameleon` **`--with-input=openmpi=nmad`** `-- mpirun …`
  - `guix shell --pure chameleon` **`--with-input=openblas=mkl`** `-- mpirun …`
- Utilisation d'une branche particulière :
  - `guix shell --pure chameleon` **`--with-branch=starpu=coop-mcast`** `-- mpirun …`
- Utilisation d'un commit particulier :
  - `guix shell --pure chameleon` **`--with-commit=starpu=acae6e78df7a9475bbfbd26e33fe324b1f7bedce`** `-- mpirun …`
- Application d'un patch aux sources d'un paquet :
  - `guix shell --pure chameleon` **`--with-patch=chameleon=./wait-graph.patch`** `-- mpirun …`

Transformations de paquets

- Combinaisons de transformations possibles !
  - Attention à l'ordre des options de transformations :
    - `--with-input=openmpi=nmad --with-branch=nmad=master` : OK, branche `master` de `nmad`
    - `--with-branch=nmad=master --with-input=openmpi=nmad` : version du paquet `nmad`

  - Visualisation des transformations réalisées avec :
    - `guix graph -M 4 chameleon --with-input=openmpi=nmad --with-branch=nmad=master | xdot -`



Monstre final du niveau 1

avec Guix !

Plus difficile :

- Comparaisons entre branches d’une bibliothèque
- Comparaisons entre commits d’une bibliothèque
- Comparaison avec ou sans patch dans une bibliothèque

- Dossiers et modules… ?
  - Là, les sources sont modifiées !

- Comment savoir à quelles sources correspond ce qui est compilé...
  - Actuellement ?
  - Il y a 6 mois ?

Monstre final du niveau 1

avec Guix !

Plus difficile :

- Comparaisons entre branches d'une bibliothèque
- Comparaisons entre commits d'une bibliothèque
- Comparaison avec ou sans patch dans une bibliothèque

Super facile !

- Dossiers et modules… ?
  - Là, les sources sont modifiées !

- Comment savoir à quelles sources correspond ce qui est compilé...
  - Actuellement ?
  - Il y a 6 mois ?

Monstre final du niveau 1

avec Guix !

Plus difficile :

- Comparaisons entre branches d'une bibliothèque
- Comparaisons entre commits d'une bibliothèque
- Comparaison avec ou sans patch dans une bibliothèque

Super facile !

- Dossiers et modules… ?
  - Là, les sources sont modifiées !

Pas besoin !

- Comment savoir à quelles sources correspond ce qui est compilé...
  - Actuellement ?
  - Il y a 6 mois ?

Reproductibilité : le problème

```
guix shell --pure chameleon -- mpirun …
```

… 6 mois plus tard …

```
guix pull
guix shell --pure chameleon -- mpirun …
```

- `chameleon` != `chameleon`
  - Version du paquet `chameleon` différente
  - Versions des dépendances de `chameleon` différentes

Reproductibilité : la solution

- Exporter les canaux (**et leurs versions**) actuellement utilisés :

  ```
  guix describe -f channels > channels.scm
  ```

- Utiliser explicitement les versions des canaux :

  ```
  guix time-machine --channels=./channels.scm --
  shell --pure chameleon -- mpirun …
  ```

- En conservant `channels.scm` : sûr d'exécuter le même code, même 6 mois après

```
(list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (branch "master")
        (commit
          "ec66f84824198f380d20126d3e4b2ea795fd205a")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"))))
      (channel
        (name 'guix-hpc-non-free)
        (url "https://gitlab.inria.fr/guix-hpc/guix-hpc-non-free.git")
        (branch "master")
        (commit
          "58aaac8c18773d900511d441e935145d73cdfc5e"))
      (channel
        (name 'guix-hpc)
        (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
        (branch "master")
        (commit
          "74840c47b744ad7342e7a86852831009a2831630")))
```

Reproductibilité : dépôt des scripts

- Faire les expériences en ayant en tête la publication des scripts
- Dépôt Git public dédié aux scripts et instructions de reproductibilité :
  - `README` suffisamment détaillé pour comprendre ce qui est fait, comment, où, …
  - Contient `channels.scm`
  - Instructions aussi pour une utilisation sans Guix
- Exemples :
  - https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y
  - https://gitlab.inria.fr/pswartva/paper-starpu-traces-r13y

Reproductibilité : dans les papiers

- Soumettre le dépôt à SoftwareHeritage
  - Dépôt disponible pour toujours
  - https://archive.softwareheritage.org/save/
  - Fournit un identifiant unique, pour retrouver le dépôt
- Dans le papier :

> 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 : avantages de Guix

- Très facile de passer d’une machine à l’autre*,**
  - Pas de temps passé à tout réinstaller, recompiler, trouver quels modules utiliser, ...
  - **Si le gestionnaire de travaux est le même*
  - ***Demande de bien factoriser / paramétriser les scripts tout de suite*

- Gain en confiance dans l’exécution des expériences
  - Surtout s’il faut les refaire (en changeant un paramètre de l’application…) !

Conclusion : points à creuser

- Utilisation des manifests
  - Place tous les paramètres de `guix shell` dans un fichier
  - Moyen de factoriser
- Utilisation avec une machine qui n’a pas Guix
  - `guix pack`