Guix

Utilisation de Guix pour la gestion d'environnements reproductibles, flexibles et collaboratifs dans le cadre d'une thèse

7 novembre 2024
Workshop Guix HPC

Antoine Gicquel
antoine.a.gicquel@inria.fr

Qui suis-je ?

- Doctorant dans l'équipe CONCACE (depuis novembre 2023)
- Formation en calcul scientifique (à Rennes)
- Utilisateur du gestionnaire de paquets `guix` [2]
- Pas d'expérience avec `guix` avant mon arrivée à l'Inria
- Première expérience avec `guix` lors de mon stage M2

Activités (autour de ma thèse)

- Composabilité des méthodes hiérarchiques et abstraction des produits de matrices (FMM, $\mathcal{H}$ -matrices, etc.)
- Langages/Outils: C++, CMake, Python (un peu de Rust)
- Bibliothèques utilisées:
  - `ScalFMM`: Méthode des multipoles rapide (FMM)
  - `composyx`: Algèbre linéaire, HPC, composabilité
- Utilisation de PlaFRIM [9] (avec `guix`)
  - CPU, GPU, calcul distribué, hétérogène, etc.

Au programme

Retour d'expérience utilisateur

- Pourquoi j'utilise `guix` ?
- Reproductibilité, collaboration et flexibilité
- Quelques exemples de *workflows*

Contenu

- Commande `guix shell` [6]
- Commande `guix time-machine` [7]
- Options de transformation des paquets [8]
  - *e.g.*, `--with-input`, `--with-source`, `--tune`, etc.

Contexte - projet ANR [1] TensorVim


CES0 - Sciences de base pour l'énergie

Décomposition tensorielle de bas rang pour la modélisation électromagnétique rapide des dispositifs en génie électrique – TensorVim

Résumé de soumission

Ce projet vise à développer des outils de calculs performants pour la mise en œuvre rapide de simulations électromagnétiques de dispositifs du génie électrique par des méthodes intégrales volumiques. Pour cela nous mettrons en œuvre des techniques

- Utilisation de la bibliothèque `ScalFMM` [3] (accélération de produits matrice-vecteur)
- Solveur écrit en C++ / Fortran 90 + Création d'un module Python (*binding* via `pybind11`)

[1] Agence National de Recherche

Création d'environnements logiciels "à la volée"

Installation des paquets dans mon profil

```
$ guix install <mes paquets> ...
```

Création d'un environnement de développement

```
$ guix shell <mes paquets> ...
```

Remarque: Pas d'interférence avec le système (cf: `/gnu/store/`)

`guix shell` en bref

- Création d'environnements de développement logiciels sans interférer avec mon profil utilisateur.

```
$ guix shell --help
Usage: guix shell [OPTION] PACKAGES... [-- COMMAND...]
Build an environment that includes PACKAGES and execute COMMAND or an
interactive shell in that environment.
[...]
  -m, --manifest=FILE    create environment with the manifest from FILE
[...]
      --pure             unset existing environment variables
[...]
  -C, --container        run command within an isolated container
[...]
```

Différents niveaux d'isolation de l'environnement de développement

Basique

```
$ guix shell <mes paquets> ...
```

Environnement "pure"

```
$ guix shell --pure <mes paquets> ...
```

Avec conteneurisation

```
$ guix shell --container <mes paquets> ...
$ # guix shell -C <mes paquets> ...              # version courte
```

Identification de l'environnement

- Utilisation de `guix search` pour trouver les paquets

```
guix search gcc-toolchain
```

- Environnement complet (exemple pour TensorVim)

```
guix shell -C scalfmm gcc-toolchain@11 gfortran-toolchain@11 \
           bash ncurses diffutils cmake make openblas fftw \
           fftwf gdb grep findutils sed gawk ninja file gmsh less \
           evince pkg-config python python-matplotlib python-numpy \
           python-scipy python-colorama pybind11 vim which coreutils
```

Exportation de l'environnement

```
$ guix shell --export-manifest scalfmm gcc-toolchain@11 [...] \
             coreutils > manifest.scm
```

Contenu du fichier manifeste

```
(specifications->manifest
(list "scalfmm"
      "gcc-toolchain@11"
      [...]
      "coreutils"))
```

Utilisation

```
guix shell -C -m manifest.scm
```

D'où viennent les paquets ? (différents canaux)

Dépôt "principal"

- guix: `https://git.savannah.gnu.org/git/guix.git`

Dépôts additionnels (exemples)

- guix-hpc: `https://gitlab.inria.fr/guix-hpc/guix-hpc.git`
- guix-science: `https://codeberg.org/guix-science/guix-science.git`
- et bien d'autres...

Exemple de la recette du paquet ScalFMM (guix-hpc)

```
(define-public scalfmm
  (package (name "scalfmm")
           (version "3.0")
             (home-page "https://gitlab.inria.fr/solverstack/ScalFMM.git")
             (synopsis "Fast Multipole Method Framework")
                    [...]
             (build-system cmake-build-system)
             (arguments
             '(#:configure-flags '("-Dscalfmm_BUILD_EXAMPLES=ON"
                                   "-Dscalfmm_BUILD_TOOLS=ON"
                                  "-Dscalfmm_BUILD_UNITS=ON")
                    [...]
```

Que faire si je veux modifier la recette ?

```
            [...]
(build-system cmake-build-system)
  (arguments
  '(#:configure-flags '("-Dscalfmm_BUILD_EXAMPLES=ON"
                        "-Dscalfmm_BUILD_TOOLS=ON"
                        "-Dscalfmm_BUILD_UNITS=ON")
            [...]
```

- Exemple pour `ScalFMM`: que faire si on veut activer une option ? (par exemple l'option CMake `scalfmm_BUILD_PBC`)

```
cd ScalFMM
cmake -B build -Dscalfmm_BUILD_PBC=ON [...]
cmake --build build --target install
```

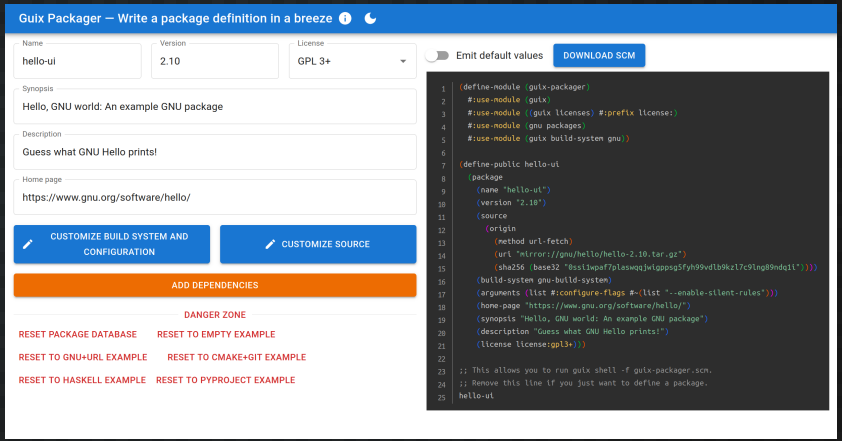
Définition d'une nouvelle variante ?

```
(define-public scalfmm-pbc
    (package
    (inherit scalfmm)
    (name "scalfmm-pbc")
    (arguments
        (substitute-keyword-arguments
            (package-arguments scalfmm)
            ((#:configure-flags flags)
                #~(append (list "-Dscalfmm_BUILD_PBC=ON")
                                #$flags))))))
```

Utilisation

```
guix shell -C scalfmm-pbc [...] -L path/to/scalfmm-pbc.scm
```

Guix packager [4]



Guix packager: aide précieuse pour écrire ses premiers paquets

Options de transformation [8]

Option --with-configure-flags

- Ajout d'un flag de configuration à la définition du paquet

```
guix shell -C scalfmm [...] \
           --with-configure-flags=scalfmm="-Dscalfmm_USE_PBC=ON"
```

- Transformation du paquet "à la volée"
- Ne modifie pas la définition originale du paquet
- Pas besoin de définir de nouveaux paquets !

Changement des sources [8]

Option --with-source

```
guix shell -C scalfmm [...] --with-source=scalfmm=$HOME/src/ScalFMM
```

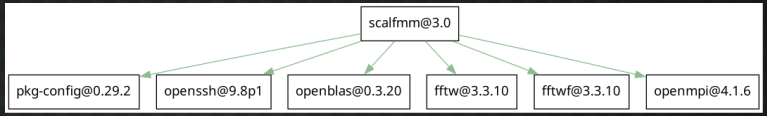
Option --with-branch

```
guix shell -C scalfmm [...] --with-branch=scalfmm="experimental-feature"
```

Option --with-commit

```
guix shell -C scalfmm [...] --with-commit=scalfmm="fd8841b"
```

Changement dans la chaîne de dépendance d'un paquet [8]



Option --with-input (or --with-graft)

- Remplacement d'une dépendance d'un paquet (*e.g.*, remplacement de `openblas` par la `MKL`)

```
guix shell -C scalfmm [...] --with-input=openblas=mkl
```

⚠ avec l'option `--with-graft` ⚠

Exportation de l'environnement (avec transformations)

```
$ guix shell --export-manifest scalfmm [...] \
    --with-input=openblas=mkl > transformed-manifest.scm
```

Contenu du fichier manifeste

```
(use-modules (guix transformations))
  (define transform1
        (options->transformation
        '((with-input . "openblas=mkl"))))
  (packages->manifest
        (list (transform1 (specification->package "scalfmm"))
            (transform1 (specification->package "coreutils"))
            [...]
            (transform1 (specification->package "bash"))))
```

Bilan

- `guix shell` créé des environnements logiciels "jetables" [6]
- `guix` me propose un large choix de paquets (+ canaux additionnels)
- Flexibilité grâce aux options de transformation
  - autres options: `--with-debug-info`, `--with-c-toolchain`, `--with-version`, `=--with-latest` etc. [8]
- Exportation des environnements (fichiers `manifest.scm`)

Stockage (et partage) des environnements (1/2)

```
$ cd tensorvim
$ ls .guix
tensorvim-manifest-clang.scm
tensorvim-manifest-gcc-mkl.scm
[...]
```

Intérêts

- Contrôle de versions, facilement partageable, description centralisée

Utilisation

```
$ guix shell -C -m .guix/tensorvim-manifest-gcc.scm
```

- Quid de la reproductibilité ?

`guix time-machine` en bref

Voyage dans le temps (dans le passé) [7]

- `guix time-machine` donne accès à d'autres révisions de `guix` et lance une commande `guix` indiquée après `--` dans cette révision

```
$ guix time-machine --help
Usage: guix time-machine [OPTION] [-- COMMAND ARGS...]
Execute COMMAND ARGS... in an older version of Guix.
[...]
  -C, --channels=FILE    deploy the channels defined in FILE
[...]
```

Utilisation en pratique

Aujourd'hui (novembre 2024)

```
$ guix time-machine --no-channel-files --commit=8964dfd \
        -- shell -C gcc-toolchain -- gcc --version

[...]
gcc (GCC) 14.2.0
```

Il y a un an (novembre 2023)

```
$ guix time-machine --no-channel-files --commit=204f08c \
          -- shell -C gcc-toolchain -- gcc --version

[...]
gcc (GCC) 13.2.0
```

Quels sont les canaux que j'utilise ? (1/2)

Description de la génération courante (3 novembre 2024)

```
$ guix describe

Generation 93 nov. 03 2024 14:47:48 (current)
  guix 8964dfd
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: 8964dfdb84f7d21dbc89c217ca4f4546a15990af
    [...]
```

Quels sont les canaux que j'utilise ? (2/2)

Téléchargement des derniers changements des paquets

```
$ guix pull
```

Nouvelle génération (5 novembre 2024)

```
$ guix describe

Generation 94 nov. 05 2024 11:49:11 (current)
  guix 254f465
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: 254f465b611383d80c08786bc3a8ae32b72066ab
    [...]
```

"Fixer" des canaux

Récupération de l'état courant

```
$ guix describe --format=channels > channels.scm
```

```
(list (channel (name 'guix)
               (url "https://git.savannah.gnu.org/git/guix.git")
               (branch "master")
               (commit "254f465b611383d80c08786bc3a8ae32b72066ab"))
      (channel (name 'guix-hpc)
               (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
               (branch "master")
               (commit "971f60b2854afb8b7dff77096c0f0f3149ee5070"))
               [...]
              )
```

Stockage (et partage) des environnements (2/2)

```
$ cd tensorvim
$ ls .guix
tensorvim-channels.scm
tensorvim-manifest-gcc.scm
tensorvim-manifest-clang.scm
tensorvim-manifest-gcc-mkl.scm
[...]
```

Utilisation

```
$ guix time-machine -C .guix/tensorvim-channels.scm -- \
    shell -C -m .guix/tensorvim-manifest-gcc.scm
```

Bilan

- Définition de l'environnement (`guix search`, `guix shell` + options de transformation)
- Création d'un fichier `manifest.scm` (option `--export-manifest`)
- Récupération de l'état courant des canaux dans `channels.scm` (via `guix describe`)

Utilisation

```
$ guix time-machine -C channels.scm -- shell -C -m manifest.scm
```

- Ressources: *"A guide to reproducible research papers"* [1] pour plus de détails

Encadrement d'un stage sur Kokkos

- Aurélien Gauthier, stagiaire M1 chez CONCACE (été 2024)
- Exploration de l'écosystème Kokkos (versions portables pour des noyaux de `ScalFMM`)
- Utilisation similaire de `guix time-machine` et de `guix shell`:
  - Fichiers `channels.scm` et `manifest.scm`
  - Transition "facile" sur PlaFRIM
  - Aide au débuggage
  - Poursuite des travaux

"Réglage" des paquets [8]

Option --tune=[arch]

- Optimisation du paquet pour le CPU utilisé
- Vectorisation SIMD (`--tune=skylake`, `--tune=haswell`, `--tune=native`, etc.)
- Les paquets visés doivent avoir la propriété `tunable?`

```
(define-public kokkos
  (package
    (name "kokkos")
     [...]
    (properties '((tunable? . #t)))
     [...]
```

"Réglage" des paquets [8] (exemple)

Sur PlaFRIM (noeud bora):

- CPU: 2x 18-core Cascade Lake Intel Xeon Skylake Gold 6240 @ 2.6 GHz

Construction du paquet (via guix build [5])

```
$ guix build --tune=native kokkos \
       --with-configure-flag=kokkos="-DKokkos_ARCH_NATIVE=ON"

guix build: tuning kokkos@4.4.00 for CPU skylake-avx512
[...]
starting phase `configure'
[...]
-- SIMD: AVX512 detected
[...]
```

Obtenir toutes les dépendances d'un paquet

- Création d'un environnement de développement "sur-mesure"

Dépendances de `ScalFMM` (niveau 1 seulement)


```
guix shell -C -D scalfmm [...]
```

Dépendances de `composyx` (niveau 1 seulement)


```
guix shell -C -D composyx [...]
```

Conclusion

Mon utilisation de `guix`

- `guix shell` + `guix time-machine` + options de transformation
- Collaboration, reproductibilité et flexibilité
- Gain de temps sur la gestion de mes environnements logiciels: plus de temps pour la recherche

Déploiement

- Déploiement "facile" sur PlaFRIM (`guix` est installé !)
- Utilisation de `guix pack` pour "exporter" ses environnements logiciels sur des clusters où `guix` n'est pas installé

Bibliographie

[1] *A guide to reproducible research papers.* https://hpc.guix.info/blog/2023/06/a-guide-to-reproducible-research-papers/.

[2] Ludovic Courtès and Ricardo Wurmus. “Reproducible and User-Controlled Software Environments in HPC with Guix”. In: *Euro-Par 2015: Parallel Processing Workshops*. Lecture Notes in Computer Science. Aug. 2015, pp. 579–591.

[3] Leslie Greengard and Vladimir Rokhlin. “A fast algorithm for particle simulations”. In: *Journal of computational physics* 73.2 (1987), pp. 325–348.

[4] *Guix Packager - Write a package definition in a breeze.* https://guix-hpc.gitlabpages.inria.fr/guix-packager/.

[5] *Invoquer guix build (Manuel de référence de GNU Guix).* https://guix.gnu.org/manual/devel/fr/html_node/Invoquer-guix-build.html.

[6] *Invoquer guix shell (Manuel de référence de GNU Guix).* https://guix.gnu.org/manual/devel/fr/html_node/Invoquer-guix-shell.html.

[7] *Invoquer guix time-machine (Manuel de référence de GNU Guix).* https://guix.gnu.org/en/manual/devel/fr/html_node/Invoquer-guix-time_002dmachine.html.

[8] *Options de transformation de paquets.* https://guix.gnu.org/manual/devel/fr/html_node/Options-de-transformation-de-paquets.html.

[9] *PlaFRIM: Plateforme fédérative pour la recherche en informatique et mathématiques.* https://plafrim.fr/.