RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

Guix vs Spack : déploiement de bibliothèques d'algèbre linéaire Retour d'expérience et comparaison.

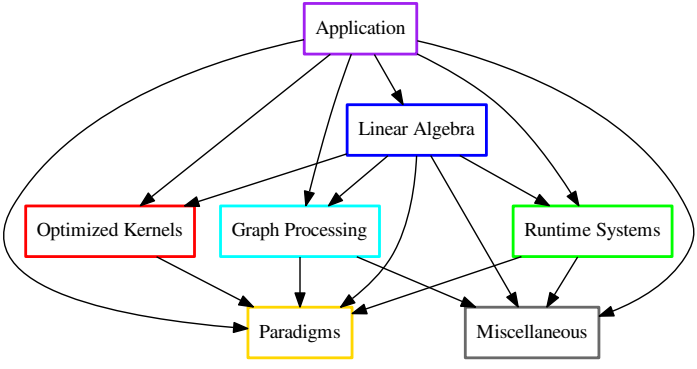
Café Guix

Florent Pruvost

Table des matières

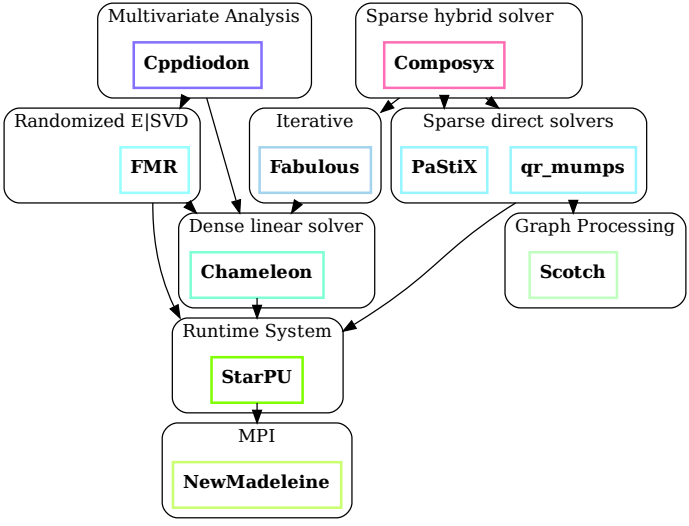
01
Contexte

Faciliter l'installation d'une pile logicielle scientifique



- modulaire, plusieurs langages, différents outils de compilation
- sur différents systèmes, environnements
- difficile d'être expert sur toute la chaine

Pile logicielle d'algebre linéaire Inria Bordeaux

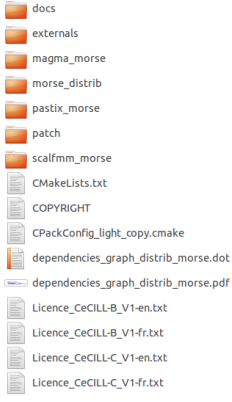


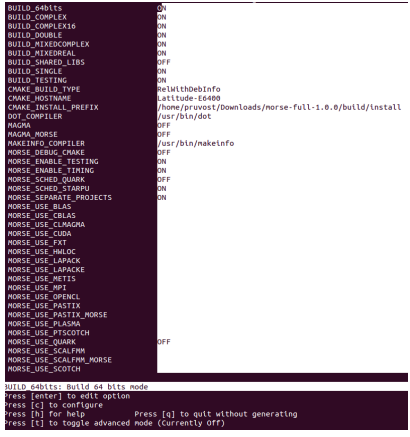
Contraintes

- Besoin de certains paquets critiques : BLAS, LAPACK, FFTW, CUDA/ROCM, HDF5, MPI
- Déployer sur des machines où l'on est pas ROOT
- Modifier certaines propriétés, e.g. variantes : l'origine des sources, options ON/OFF, options de compilation, modifier une dépendance, ...
- Reproduire l'installation d'une machine à une autre

02
Historique

[2013] Projet “superbuild” CMake





- *ExternalProject_Add* : téléchargement des sources, sources locales
- Fragile : n’installe que les dépendances de niveau 1
- “Usine à gaz” : beaucoup de code CMake écrit spécifiquement
- CMake n'est pas un gestionnaire de paquet

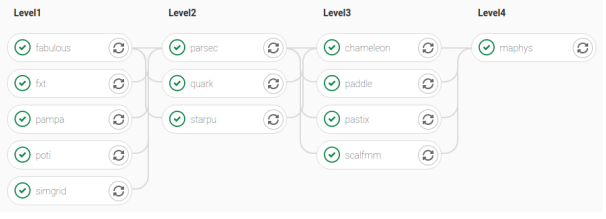
[2014] État des lieux d'outils existants

Packager	Non Root	HPC	Flexible	Robuste
Easybuild	■	■	■	■
Nix/Guix	■	■	■	■
Spack	■	■	■	■

- **Easybuild/Spack** focus sur les besoins HPC, mais dépend de paquets externes (compilateur, libc) = fragile.
- **Nix/Guix** focus sur la reproductibilité, tout le système est maitrisé, aucune dépendance externe. Pas d'installation possible en local (HOME), pas de support Intel, CUDA etc.

[2014-2016] Développement de paquets Spack

Pipeline Gitlab-CI : test d'installation de releases :



Simple d'utilisation et flexible :

```
git clone https://github.com/llnl/spack.git
git clone https://gitlab.inria.fr/solverstack/spack-repo.git
./spack/bin/spack repo add ./spack-repo
spack install chameleon@master~simgrid+mpi+starpu %gcc ^openblas ^openmpi
```

Grande combinatoire explorable :

- aspect positif → corrections de nombreux bugs d'installation dans nos bibliothèques
- aspect négatif → long, fastidieux, beaucoup de temps passer à recompiler pendant 3 ans

[2014-2016] Utilisation de Spack sur supercalculateur

- sans accès au web mondial, besoin de télécharger les sources sur sa machine :

```
spack mirror create -D -d ./mirror/ hwloc
tar cvf mirror.tar.gz mirror
```

- copier le tarball des sources sur le supercalculateur cible :

```
scp mirror.tar.gz plafrim:
ssh plafrim
tar xvf mirror.tar.gz
spack mirror add local file:///home/pruvost/mirror
```

- dépendance aux modules existant (icc, intelmpi, cuda, etc)

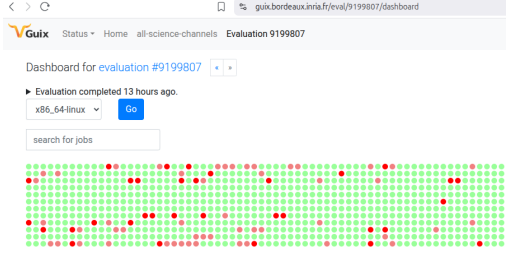
Résultat : à part 2, 3 experts, personnes n'a été capable d'installer nos logiciels sur une nouvelle machine sans guide d'installation spécifique !

[2017-2020] Guix-HPC

Ludovic Courtès est missionné pour se consacrer à Guix :

- nouveaux paquets HPC “non free” : MKL, CUDA, ...
- flexibilité de la ligne de commande : origine des sources, substitution, options de compilation
- déploiement sur supercalculateur : image singularity, tarball binaires relocalisables
- Guix disponible sur tous les moyens de calcul Inria

Développement de nos paquets Guix



Canaux guix-hpc : `https://gitlab.inria.fr/guix-hpc/guix-hpc`
guix-hpc-non-free : `https://gitlab.inria.fr/guix-hpc/guix-hpc-non-free`
Aujourd'hui dans guix-science : `https://codeberg.org/guix-science/guix-science`

03
Édition des paquets

Exemple Spack – Python

```
spack edit chameleon
```

```
version("master", branch="master", submodules=True)
version("1.4.0", sha256="7b65a6168ebdfc11a1c9994454ee474673fc92348d5aa8724e83c34fd5010aaf")
version("1.3.0", sha256="2725d2d2a9885e619e0c8d41306b9b9dc6d5df635b710cf8d077a14803ea26cd")
version("1.2.0", sha256="b8988ecbff19c603ae9f61441653c21bba18d040bee9bb83f7fc9077043e50b4")
version("1.1.0", sha256="e64d0438dfaf5effb3740e53f3ab017d12744b85a138b2ef702a81df559126df")

# cmake's specific
variant("shared", default=True, description="Build chameleon as a shared library")

# chameleon's specific
variant(
    "runtime",
    default="starpu",
    description="Runtime support",
    values=("openmp", "starpu"),
    multi=False,
)
variant("mpi", default=True, when="runtime=starpu", description="Enable MPI")
variant("cuda", default=False, when="runtime=starpu", description="Enable CUDA")
```

Versions, variantes (ON/OFF ou choix multiple)

Exemple Spack – Python

```
spack edit chameleon
```

```
# dependencies
depends_on("c", type="build")  # generated
depends_on("cxx", type="build")  # generated
depends_on("fortran", type="build")  # generated

depends_on("pkgconfig", type="build")

with when("runtime=starpu"):
    depends_on("starpu@1.3", when="@1.1.0")
    depends_on("starpu")
    depends_on("starpu~mpi", when="~mpi")
    depends_on("starpu+mpi", when="+mpi")
    depends_on("starpu~cuda", when="~cuda")
    depends_on("starpu+cuda", when="+cuda")
    with when("+simgrid"):
        depends_on("simgrid+msg")
        depends_on("starpu+simgrid")
        depends_on("starpu+mpi~shared+simgrid", when="+mpi")
        conflicts("^simgrid@:3.31", when="@:1.1.0")
        conflicts("+shared", when="+simgrid")
    with when("~simgrid"):
        depends_on("mpi", when="+mpi")
        depends_on("cuda", when="+cuda")
    with when("+fxt"):
        depends_on("fxt")
        depends_on("starpu+fxt")

with when("~simgrid"):
    depends_on("blas")
    depends_on("lapack")
```

Le graphe de dépendances découle aussi des variantes

Exemple Spack – Python

```
spack edit chameleon
```

```
def cmake_args(self):
    spec = self.spec
    args = [
        "-Wno-dev",
        self.define("CMAKE_COLOR_MAKEFILE", "ON"),
        self.define("CMAKE_VERBOSE_MAKEFILE", "ON"),
        self.define("CHAMELEON_ENABLE_EXAMPLE", "ON"),
        self.define("CHAMELEON_ENABLE_TESTING", "ON"),
        self.define_from_variant("BUILD_SHARED_LIBS", "shared"),
        self.define_from_variant("CHAMELEON_USE_MPI", "mpi"),
        self.define_from_variant("CHAMELEON_USE_CUDA", "cuda"),
        self.define_from_variant("CHAMELEON_SIMULATION", "simgrid"),
    ]

    if spec.satisfies("runtime=openmp"):
        args.extend([self.define("CHAMELEON_SCHED", "OPENMP")])
    if spec.satisfies("runtime=starpu"):
        args.extend([self.define("CHAMELEON_SCHED", "STARPU")])

    if spec.satisfies("+mpi +simgrid"):
        args.extend(
            [
                self.define("CMAKE_C_COMPILER", self.spec["simgrid"].smpicc),
                self.define("CMAKE_CXX_COMPILER", self.spec["simgrid"].smpicxx),
                self.define("CMAKE_Fortran_COMPILER", self.spec["simgrid"].smpifc),
            ]
        )

    if spec.satisfies("+mpi ~simgrid"):
        args.extend(
            [
                self.define("MPI_C_COMPILER", self.spec["mpi"].mpicc),
                self.define("MPI_CXX_COMPILER", self.spec["mpi"].mpicxx),
                self.define("MPI_Fortran_COMPILER", self.spec["mpi"].mpifc),
            ]
        )

    if spec.satisfies("^[virtuals=blas,lapack] intel-oneapi-mkl threads=none"):
        args.extend([self.define("BLA_VENDOR", "Intel10_64lp_seq")])
    elif spec.satisfies("^[virtuals=blas,lapack] intel-oneapi-mkl"):
        args.extend([self.define("BLA_VENDOR", "Intel10_64lp")])
    elif spec.satisfies("^[virtuals=blas,lapack] netlib-lapack"):
        args.extend([self.define("BLA_VENDOR", "Generic")])
    elif spec.satisfies("^[virtuals=blas,lapack] openblas"):
        args.extend([self.define("BLA_VENDOR", "OpenBLAS")])
```

Options de compilations pilotées par les variantes

Exemple Spack – Python

```
spack edit chameleon
```

- Une grande combinaison de possibilités via un seul paquet à écrire
- Paquet simple à écrire

Exemple Guix – Guile

```
guix edit chameleon
```

```
(define-public chameleon
  (package
    (name "chameleon")
    (version "1.4.0")
    (source
     (origin
       (method git-fetch)
       (uri (git-reference
             (url "https://gitlab.inria.fr/solverstack/chameleon")
             (commit (string-append "v" version))
             ;; We need the submodule in 'CMakeModules/morse_cmake'.
             (recursive? #t)))
       (file-name (string-append name "-" version "-checkout"))
       (sha256
        (base32 "0wssqbs5n9wz3g50dcpxalrsp3z39p8nvmc1rvgg3jn860vlbbj0"))
       (build-system cmake-build-system)
       (outputs '("debug" "out"))
       (arguments
        (list #:configure-flags
              #~(list "-DBUILD_SHARED_LIBS=ON"
                      "-DCHAMELEON_USE_MPI=ON")

              ;; Restrict tests to MPI/simple precision for CI resource
              ;; savings.
              #:test-exclude "test_shm_d|test_shm_z|test_mpi_d|test_mpi_z|getrf_ppiv"))
       (inputs (list openblas starpu openmpi))
       (native-inputs (list pkg-config gfortran python))
       (home-page "https://gitlab.inria.fr/solverstack/chameleon")
       (synopsis "Dense linear algebra solver")
       (description
        "Chameleon is a dense linear algebra solver relying on sequential
task-based algorithms where sub-tasks of the overall algorithms are submitted
to a run-time system.  Such a system is a layer between the application and
```

1 paquet = une version précise, peut être modifié à la ligne de commande

Exemple Guix – Guile

```
guix edit chameleon
```

```
(define-public chameleon+cuda
  (package/inherit chameleon
   (name "chameleon-cuda")
   (arguments
    (substitute-keyword-arguments (package-arguments chameleon)
      ((#:configure-flags flags '())
       #~(cons "-DCHAMELEON_USE_CUDA=ON" #$flags))))
   (inputs
    (modify-inputs (package-inputs chameleon)
                   (prepend cuda)
                   (replace "starpu" starpu+cuda)
                   (replace "openmpi" openmpi-cuda)))))
```

Chaque variante est écrite dans un nouveau paquet, héritage d'un paquet de référence

Exemple Guix – Guile

```
guix edit chameleon
```

- Chaque version/variante d'un paquet est commitée et donc validée via intégration continue
- Écriture plus difficile que Python pour les non initiés au Guile (Scheme)

04
Utilisation en local

Installation naive

```
time guix install hwloc
# 5 sec
time spack install hwloc
# 5 min
```

Par défaut Spack recompile tout à partir des sources (cache de binaires à configurer)

```
spack location -i hwloc
# /home/florent/git/spack/opt/spack/linux-icelake/hwloc-2.11.1...

guix package -I hwloc
# hwloc        2.12.2        out        /gnu/store/vfk1s1vhrlp6lbzn1pj4ddmgmma4csy2-hwloc-2.12.2
```

Par défaut Spack installe dans `/home/user`, Guix dans `/gnu/store`.
Modifiable `.spack/config.yaml` :

```
config:
  install_tree:
    root: /opt/spack
```

Binaires pré-compilés

Spack se base sur les compilateurs + la libc présents sur le système (fragilité)

```
spack find -I
[e]  gcc@13.3.0          [e]  glibc@2.39
```

Paramétrage Spack du cache de binaires :

```
spack repo update builtin --tag develop
spack mirror add develop https://binaries.spack.io/develop
spack buildcache keys --install --trust
spack install [--use-buildcache only] hwloc
```

ça reste très long (plusieurs minutes) → lenteur de lecture du “build cache”

Déploiement d'environnements

```
guix shell --container chameleon coreutils which
which chameleon_stesting
#/gnu/store/7hfh35rg6z93cq0fmxlxksxcxjp4wjmp-profile/bin/chameleon_stesting
ls /
# bin  dev  etc         gnu  home  proc        run  sys  tmp
exit

guix shell --pure chameleon coreutils grep sed which
which chameleon_stesting
#/gnu/store/vzzi9bwwx5gk1ar6q1sf2dwpy5cza8ak-profile/bin/chameleon_stesting
ls /
# bin  boot  dev        etc  gnu  grid5000  home  initrd.img  initrd.img.old  lib  lib64  ...
exit

spack env create chameleon
spack env activate chameleon
spack add chameleon
spack concretize
spack install
which chameleon_dtesting
#/home/florent/git/spack/var/spack/environments/chameleon/.spack-env/view/bin/chameleon_dtesting
spack env deactivate
```

Changement de versions, de chaine de compilation

```
guix install chameleon --with-commit=c7715a64f0fc288fc27f560f41b492e5ff30e632
guix install chameleon --with-branch=chameleon=master
guix install chameleon --with-commit=chameleon=v1.3.0 --with-commit=starpu=starpu-1.4.7

spack install chameleon@c7715a64f0fc288fc27f560f41b492e5ff30e632
spack install chameleon@master
spack install chameleon@1.3.0 ^starpu@1.4.7
```

```
guix install chameleon --with-c-toolchain=starpu=clang-toolchain@19.1.7

spack install chameleon %llvm@18.1.3
```

Variantes, options et substitutions

```
guix install chameleon-cuda-nompi

spack install chameleon +cuda -mpi
```

```
guix install chameleon --with-configure-flag=chameleon="-DCHAMELEON_STARPU_USE_INSERT=ON" \
                       --with-configure-flag=chameleon="-DCMAKE_C_FLAGS=-O0 -g"

spack install chameleon cflags="-O0 -g"
# cette dernière ligne ne fonctionne pas
```

```
guix install chameleon --with-input=openblas=intel-oneapi-mkl --with-input=openmpi=mpich

spack install chameleon ^intel-oneapi-mkl ^mpich
```

05
Déploiement sur un supercalculateur

Déploiement avec Guix disponible

```
ssh plafrim

# on fixe une version de guix
guix pull
guix describe -f channels > $( date +%Y%m%d-%H%M ).scm
# 20260224-1559.scm
ln -sf 20260224-1559.scm ~/channels-fixed.scm

# on fixe un environnement
guix shell --export-manifest bash-minimal slurm@23 chameleon-cuda-mkl \
           > chameleon-cuda-mkl.scm
cat chameleon-cuda-mkl.scm
#(specifications->manifest
#  (list "bash-minimal"
#        "slurm@23.11.11"
#        "chameleon-cuda-mkl"))

# on valide le déploiement
guix time-machine -C ~/channels-fixed.scm -- shell --pure --preserve=SLURM \
                  -m chameleon-cuda-mkl.scm
```

Déploiement avec Guix disponible – Runs

```
# bora: cpus intel cascadelake, ommipath
guix time-machine -C ~/channels-fixed.scm -- shell --pure -m chameleon-cuda-mkl.scm -- \
     srun --nodes=2 --ntasks-per-node=2 --cpus-per-task=18 --exclusive --constraint=bora \
     bash -c 'chameleon_dtesting -o potrf -g 0 -n 50000 -b 500 -P 2 -w'
# dpotrf;17;0;2;2;1;500;121;50000;50000;1804289383;0.000000e+00;1.255878e+01;3.317831e+03

# sirocco: 2 gpus nvidia p100
guix time-machine -C ~/channels-fixed.scm -- shell --pure -m chameleon-cuda-mkl.scm -- \
     srun --nodes=2 --ntasks-per-node=1 --cpus-per-task=32 --exclusive --constraint=p100 \
     bash -c 'export LD_PRELOAD=/usr/lib64/libcuda.so; \
              export STARPU_MPI_GPUDIRECT=0; \
              chameleon_dtesting -o potrf -g 2 -n 100000 -b 1000 -P 1 -w'
# dpotrf;29;2;1;2;1;1000;121;100000;100000;846930886;0.000000e+00;2.733976e+01;1.219244e+04
```

Spack sur un supercalculateur

```
ssh plafrim

# on charge les modules (gcc, cmake, cuda, openmpi)
source ~/set_env.sh
. spack/share/spack/setup-env.sh

cd /beegfs/pruvost/
git clone --depth=2 https://github.com/spack/spack.git

# on tente de détecter les composants déjà installés
spack compiler add /cm/shared/modules/intel/skylake/compiler/gcc/11.2.0/bin
spack external find
spack external find cuda
spack external find openmpi

spack install chameleon+cuda ^intel-oneapi-mkl ^cuda@12.3
```

Could not fetch manifest from https://ghcr.io/v2/spack/bootstrap-buildcache-v2.2/manifests/index.spack, due to : GET https://ghcr.io/v2/spack/bootstrap-buildcache-v2.2/manifests/index.spack errored with : Tunnel connection failed : 403 Forbidden

Spack sur un supercalculateur

- Spack est installé/utilisé au niveau utilisateur (ex. HOME)
- Philisophie : utiliser les modules déjà installés assurant les bonnes performances (CUDA, MPI)
- Le web est largement filtré depuis les noeuds frontaux/calcul

Spack – Miroir des sources

1. Récupérer l'ensemble des sources sur sa propre machine
2. Copier sur le supercalculateur
3. Compiler à partir des sources
4. Mettre à jour au fur et à mesure la liste des sources, les versions = fastidieux

```
spack mirror create -D -d /tmp/chameleon chameleon+cuda ^intel-oneapi-mkl ^cuda@12.9
```

Error : Unable to parse extension from https://curl.se/ca/cacert-2025-08-12.pem.

```
spack mirror create -D -d /tmp/chameleon chameleon+cuda ^intel-oneapi-mkl ^cuda@12.9 \
                                                        ^openssl certs=system
```

Error : Unable to parse extension from
https://developer.download.nvidia.com/compute/cuda/12.9.0/local_installers/cuda_12.9.1_linux.run.

Spack – Build Cache

1. Compiler sur sa machine en ciblant une architecture plus ou moins générique (ex. `target=x86_64`)
2. Téléverser sur un registry Docker ou OCI

```
spack install chameleon+cuda ^intel-oneapi-mkl ^cuda@12.9 ^openmpi
spack mirror add --oci-username-variable REGISTRY_USER \
                 --oci-password-variable REGISTRY_TOKEN \
                 inria-fpruvost oci://registry.gitlab.inria.fr/fpruvost/chameleon/spack

spack buildcache push --base-image ubuntu:24.04 inria-fpruvost chameleon+cuda \
                 ^intel-oneapi-mkl ^cuda@12.9 ^openmpi
```

```
docker run -it registry.gitlab.inria.fr/fpruvost/chameleon/spack:chameleon-1.4.0-....spack
chameleon_dtesting --version
```

chameleon_dtesting : error while loading shared libraries : libmkl_gf_lp64.so.2 → problème de licence ?

Guix – Image Singularity sur PlaFRIM

```
ssh plafrim

guix shell --export-manifest slurm@23 chameleon-cuda-mkl starpu-cuda openmpi-cuda hwloc \
                                      bash binutils coreutils grep inetutils vim which \
                                      > chameleon-cuda-mkl.scm

CHAM_PACK=`guix time-machine -C ~/channels-fixed.scm -- pack -f squashfs \
                             -m ~/chameleon-cuda-mkl.scm -S /bin=bin \
                             --entry-point=/bin/bash`
cp $CHAM_PACK /beegfs/pruvost/cham-pack.gz.sif

srun --nodes=2 --ntasks-per-node=1 --cpus-per-task=32 --constraint=p100 --mpi=pmi2 \
     singularity exec --nv /beegfs/pruvost/cham-pack.gz.sif \
       bash -c "export LD_PRELOAD=/.singularity.d/libs/libcuda.so; \
                export STARPU_HOSTNAME=p100; \
                export STARPU_MPI_GPUDIRECT=0; \
                chameleon_dtesting -o potrf -g 2 -n 100000 -b 1000 -P 1 -w -l 3"
#1;dpotrf;29;2;1;2;1;1000;121;100000;100000;846930886;0.000000e+00;2.778466e+01;1.199721e+04
```

Guix – Image Singularity sur Jean Zay

```
scp -3 plafrim:/beegfs/pruvost/cham-pack.gz.sif jzay:/lustre/fswork/projects/rech/hyb/uzl87rf/

ssh jzay
module load singularity
cd $WORK
idrcontmgr cp cham-pack.gz.sif

# 2 noeuds avec 4 GPUs V100
srun -N 2 -c 40 --ntasks-per-node=1 --gres=gpu:4 --mpi=pmi2 -A hyb@v100 -C v100-16g \
     singularity exec --nv $SINGULARITY_ALLOWED_DIR/cham-pack.gz.sif \
       bash -c "export LD_PRELOAD=/.singularity.d/libs/libcuda.so; \
                export STARPU_HOSTNAME=v100; \
                export STARPU_NWORKER_PER_CUDA=2; \
                export STARPU_MPI_GPUDIRECT=0; \
                chameleon_dtesting -o potrf -g 4 -n 150000 -b 2000 -P 1"
#0;dpotrf;35;4;1;2;1;2000;121;150000;150000;846930886;0.000000e+00;2.346585e+01;4.794248e+04
```

Conclusions

- Spack contient beaucoup de paquets avec des variantes utiles pour le HPC.
- Pratique pour un administrateur système sur un supercalculateur.
- Sans accès web mondial Spack est fastidieux en espace utilisateur.
- Difficile de reproduire son installation entre machines avec Spack.
- Guix nous assure de reproduire l'installation d'une machine à l'autre.
- La fonctionnalité d'accès aux binaires pré-compilés est plus mature dans Guix.

Merci pour votre attention !