

# Guix-HPC Tutorial

Emmanuel Agullo

April 29, 2024

## Contents

<b>1</b>	<b>Home</b>	<b>2</b>
1.1	Environmmment . . . . .	2
<b>2</b>	<b>Single-node</b>	<b>3</b>
2.1	Réservation d'un noeud miriel . . . . .	3
2.2	Produit de matrice (GEMM) . . . . .	3
2.2.1	via votre code d'is328 alias mini-chameleon . . . . .	3
2.2.2	via chameleon . . . . .	4
2.3	Factorisation QR via <code>chameleon</code> . . . . .	5
2.3.1	Ressources . . . . .	5
2.3.2	Exécution basique . . . . .	5
2.3.3	Exécution avec un arbre de réduction . . . . .	5
2.3.4	Dessiner (et observer les arbres de réduction) . . . . .	5
2.4	Libérer le noeud . . . . .	5
<b>3</b>	<b>Multi-node</b>	<b>6</b>
3.1	Réservation multi-noeuds . . . . .	6
3.2	Dîtes "bonjour le monde" . . . . .	6
3.2.1	<code>slurm</code> système lance l'environnement <code>guix</code> . . . . .	6
3.2.2	<code>slurm</code> système depuis l'environnement <code>guix</code> . . . . .	6
3.2.3	<code>slurm guix</code> depuis l'environnement <code>guix</code> . . . . .	6
3.3	Vérifiez l'environnement <code>srun</code> sur chaque noeud . . . . .	7
3.4	Ping Pong . . . . .	7
3.5	Hello world MPI . . . . .	7
3.6	Exemple de produit de matrice . . . . .	8
3.6.1	via votre code ( <code>mini-chameleon</code> ) . . . . .	8
3.6.2	via <code>chameleon</code> . . . . .	8
3.7	Libérez les noeuds . . . . .	9

3.8	With <code>sbatch</code> . . . . .	9
<b>4</b>	<b>Singularity</b>	<b>10</b>
4.1	Création de l'image <code>singularity</code> . . . . .	10
4.2	Réservation multi-noeuds . . . . .	10
4.3	Bonjour le monde . . . . .	10
4.4	Hello World MPI . . . . .	10
4.5	Libérez les noeuds . . . . .	11
4.6	Hello World MPI via son propre MPI hôte . . . . .	11
4.6.1	Installation d'OpenMPI . . . . .	11
4.6.2	Test de l'installation . . . . .	11
4.6.3	Réservation multi-noeuds . . . . .	11
4.6.4	Hello World MPI . . . . .	11
4.6.5	Libérez les noeuds . . . . .	12
<b>5</b>	<b>Alternative MPI (nmad)</b>	<b>12</b>
5.1	Réservation multi-noeuds . . . . .	12
5.2	Hello World MPI . . . . .	12
5.3	Produit de matrice . . . . .	12
5.4	Libérez les noeuds . . . . .	13
<b>6</b>	<b>GPU (Cuda)</b>	<b>13</b>
6.1	Node reservation . . . . .	13
6.2	(single-node) GPU execution of <code>chameleon</code> . . . . .	13
6.3	Development environment for (single-node) GPU execution of <code>mini-chameleon</code> . . . . .	13
6.4	Releasing node reservation . . . . .	14

## 1 Home

This document can be browsed in html or pdf (sources available there).

### 1.1 Environment

Nous nous basons entièrement sur l'environnement guix. Nous faisons l'hypothèse que, comme `plafirm` au 25 janvier 2024, `slurm@23` est utilisé.

## 2 Single-node

### 2.1 Réserveation d'un noeud miriel

Nous allons ici réserver un noeud miriel en exclusif (`--exclusive`) (à utiliser avec parcimonie, pour les études de performance).

```
salloc -N 1 --exclusive
ssh <node id>
```

Nous pouvons vérifier la topologie de la machine:

```
guix shell --pure coreutils hwloc -- hwloc-ls
```

Combien de coeurs sont-ils vus par `hwloc`? Est-ce consistant avec la documentation `plafrim`?

Que voit `starpu`?

```
guix shell --pure coreutils starpu -- starpu_machine_display
```

### 2.2 Produit de matrice (GEMM)

#### 2.2.1 via votre code d'is328 alias mini-chameleon

On peut directement obtenir le code de base de mini-chameleon via:

```
guix shell --pure mini-chameleon openssh -- perf_dgemm -v seq
```

C'est bien sûr plus intéressant de faire de même avec votre propre code que vous aurez enrichi:

```
export HOME_IS328=/path/to/my/great/is328 # no '/' at the end
guix shell --pure mini-chameleon openssh
↪ --with-source=mini-chameleon=$HOME_IS328 -- perf_dgemm -v
↪ seq
```

Avec un gcc récent (gcc 11.2):

```
guix shell --pure mini-chameleon openssh
↪ --with-source=mini-chameleon=$HOME_IS328
↪ --with-c-toolchain=mini-chameleon=gcc-toolchain --
↪ perf_dgemm -v seq
```

Notez qu'il est possible d'entrer dans une session. Par exemple:

```

guix shell --pure mini-chameleon coreutils openssh
↪ --with-source=mini-chameleon=$HOME_IS328 # on entre dans
↪ la session
ls $GUIX_ENVIRONMENT/bin/ # ex: on affiche les binaires
↪ exécutables installés dans la session
exit # on quitte la session

```

Pour rappel, on peut tester le blas de l'environnement via l'option `-v vendor`. Ici on teste ainsi `openblas` puis `mkl`:

```

guix shell --pure mini-chameleon openssh -- perf_dgemm -v
↪ vendor # c'est openblas ici
guix shell --pure mini-chameleon openssh
↪ --with-input=openblas=mkl -- perf_dgemm -v vendor # et mkl
↪ là

```

On peut jouer avec les tailles de matrice (options `-M`, `-N`, `-K`) et également le nombre de threads utilisés via les variables d'environnement `MKL_NUM_THREADS` et `OMP_NUM_THREADS`.

### 2.2.2 via chameleon

Exécuter un produit de matrice:

```

guix shell --pure chameleon openssh -- chameleon_dtesting -H
↪ -o gemm --check

```

Pour obtenir de l'aide:

```

guix shell --pure coreutils openssh chameleon --
↪ chameleon_dtesting -h

```

Exécuter un produit de matrice en choisissant les dimensions  $m = 4000$ ,  $n = 2000$  et  $k = 1000$ :

```

guix shell --pure coreutils openssh chameleon --
↪ chameleon_dtesting -H -o gemm -m 4000 -n 2000 -k 1000
↪ --check

```

Comment les `workers starpu` se sont-ils répartis le travail (voir On-line feedback):

```

STARPU_PROFILING=1 STARPU_WORKER_STATS=1 guix shell --pure
↪ --preserve=~STARPU coreutils openssh chameleon --
↪ chameleon_dtesting -H -o gemm -m 4000 -n 2000 -k 1000
↪ --check

```

Remplaçons openblas par mkl

```
guix shell --pure coreutils openssh chameleon  
↪ --with-input=openblas=mkl -- chameleon_dtesting -H -o gemm  
↪ -m 4000 -n 2000 -k 1000 --check
```

## 2.3 Factorisation QR via chameleon

### 2.3.1 Ressources

Transparents & article.

### 2.3.2 Exécution basique

```
guix shell --pure coreutils openssh chameleon --  
↪ chameleon_dtesting -H -o geqrf --check
```

### 2.3.3 Exécution avec un arbre de réduction

```
guix shell --pure coreutils openssh chameleon --  
↪ chameleon_dtesting -H --qra 4 -o geqrf --check
```

### 2.3.4 Dessiner (et observer les arbres de réduction)

On peut dessiner un arbre de réduction pour une matrice de  $16 \times 8$  tuiles (-M 16 -N 8), par groupes de 4 tuiles par colonne (-a 4), et en réduisant entre les groupes avec un arbre binaire (-l 3):

```
guix shell --pure coreutils openssh chameleon sed grep --  
↪ draw_hqr -M 16 -N 8 -a 4 -l 3
```

Pour l'observer, rapatrier le fichier sur votre machine (ou connectez vous avec la transmission d'X sur plafrim) et utilisez inkscape (p. ex.):

```
guix shell --pure inkscape -- inkscape hqr.svg
```

Vous pouvez aussi retrouver le fichier généré ici (attention, utiliser plutôt inkscape que votre navigateur pour le visualiser).

## 2.4 Libérer le noeud

```
squeue -u <username>  
scancel <jobid>
```

## 3 Multi-node

### 3.1 Réserveation multi-noeuds

Dans l'exemple suivant, on fait une réserveation exclusive sur deux noeuds.

```
salloc -N 2 --exclusive
```

### 3.2 Dîtes "bonjour le monde"

La réserveation précédente est prévue pour deux noeuds (-N 2) avec un processus par noeud (on pourrait autrement demander --ntasks-per-node=24, cf. the nomenclature slurm, auquel cas il y aurait 48 processus).

Voici trois façon de vérifier que les deux processus s'annoncent correctement en donnant leur hostname.

#### 3.2.1 slurm système lance l'environnement guix

Ici nous utilisons le `slurm` du système (`which srun`) pour lancer l'environnement:

```
srun -l guix shell --pure inetutils -- hostname
```

#### 3.2.2 slurm système depuis l'environnement guix

On peut alternativement, depuis un environnement `guix`, utiliser le `slurm` système en donnant son chemin en dur (`/usr/bin/srun`) et en préservant les variables d'environnement `slurm` (`--preserve=~SLURM`) qui ont été positionnées par `salloc` plus tôt:

```
guix shell --pure --preserve=~SLURM inetutils -- /usr/bin/srun  
↪ -l hostname
```

#### 3.2.3 slurm guix depuis l'environnement guix

Une troisième possibilité, sans doute la plus élégante, est d'utiliser le client `slurm` fourni par `guix`. Il faut s'assurer de fournir une version compatible avec le `slurmd` du système, en l'occurrence `slurm@23`, en sus de préserver les variables d'environnement `slurm` (`--preserve=~SLURM`):

```
guix shell --pure --preserve=~SLURM inetutils slurm@23 -- srun  
↪ -l hostname
```

### 3.3 Vérifiez l'environnement srun sur chaque noeud

```
guix shell --pure --preserve=~SLURM coreutils openssh slurm@23
↪ --with-input=slurm=slurm@23 -- srun -l env
```

### 3.4 Ping Pong

```
guix shell --pure --preserve=~SLURM intel-mpi-benchmarks
↪ openssh slurm@23 --with-input=slurm=slurm@23 -- srun -l
↪ IMB-MPI1 Pingpong
```

Notez que sur plafrim actuellement c'est la version 19 de `slurm` qui est déployée (`srun --version`). Par défaut, dans votre environnement `guix`, c'est la version 20 (`guix shell --pure slurm -- srun --version`). Il est donc préférable de demander explicitement un `slurm@23` (`--with-input=slurm=slurm@23`).

### 3.5 Hello world MPI

On peut effectuer un petit "hello world" pour vérifier si on arrive à utiliser correctement `mpi` (notez qu'on évite d'utiliser `=ucx` via `OMPI_MCA_pml='^ucx'` puis `--preserve="^OMPI"`) et si les processus parlent bien de là où ils sont sensés parler:

```
OMPI_MCA_pml='^ucx' guix shell --pure
↪ --preserve="^SLURM|^OMPI" slurm hello-mpi
↪ --with-input=slurm=slurm@23 -- srun -l hello-mpi
```

Notez qu'on peut également utiliser directement `mpiexec`. `openmpi` étant lié à `slurm` lors de sa construction, il pourra accéder à la liste de noeuds réservés par `slurm`. Il faut toutefois lui préciser le nombre de processus qu'on souhaite lancer (`-n 2`) et le fait de les associer par noeud (`--map-by node`):

```
OMPI_MCA_pml='^ucx' guix shell --pure
↪ --preserve="^SLURM|^OMPI" slurm hello-mpi
↪ --with-input=slurm=slurm@23 -- mpiexec --tag-output -n 2
↪ --map-by node hello-mpi
```

À noter que si la réservation `salloc` a été faite en précisant explicitement le nombre de "tâches" `slurm` (via `salloc -N 2 -n 2 --exclusive` dans notre cas), `mpiexec` n'a pas besoin de ces options supplémentaires:

```

OMPI_MCA_pml='^ucx' guix shell --pure
↪ --preserve="^SLURM|^OMPI" slurm hello-mpi
↪ --with-input=slurm=slurm@23 -- mpiexec --tag-output
↪ hello-mpi

```

## 3.6 Exemple de produit de matrice

### 3.6.1 via votre code (mini-chameleon)

On positionne le pointeur vers le source:

```
export HOME_IS328=/path/to/my/great/is328 # no '/' at the end
```

On peut jouer p. ex. avec la version starpu:

```

guix shell --pure --preserve="^OMPI|^SLURM|^STARPU" coreutils
↪ gdb mini-chameleon openmpi openssh slurm xterm
↪ --with-input=slurm=slurm@23
↪ --with-source=mini-chameleon=$HOME_IS328 -- srun
↪ check_dgemm -v starpu

```

Et on peut déboguer avec gdb via xterm -e (il faut s'être connecté à plafrim avec l'option -Y d'ssh):

```

guix shell --pure --preserve="^OMPI|^SLURM|^STARPU" coreutils
↪ gdb mini-chameleon openmpi openssh slurm xterm
↪ --with-input=slurm=slurm@23
↪ --with-debug-info=mini-chameleon --with-debug-info=starpu
↪ --with-source=mini-chameleon=$HOME_IS328 -- srun xterm -e
↪ gdb --args check_dgemm -v starpu

```

### 3.6.2 via chameleon

```

guix shell --pure --preserve=^SLURM chameleon openssh slurm
↪ --with-input=slurm=slurm@23 -- srun chameleon_dtesting -H
↪ -o gemm --check

```

Remplaçons openblas par mkl:

```

guix shell --pure --preserve=^SLURM chameleon openssh slurm
↪ --with-input=slurm=slurm@23 --with-input=openblas=mkl --
↪ srun chameleon_dtesting -H -o gemm --check

```

Sans les erreurs ucx:



```

OMPI_MCA_pml='^ucx' guix shell --pure
↪ --preserve="^SLURM|^OMPI" chameleon openssh slurm
↪ --with-input=slurm=slurm@23 --with-input=openblas=mkl --
↪ srun chameleon_dtesting -H -o gemm --check

```

### 3.7 Libérez les noeuds

```

squeue -u <username>
scancel <jobid>

```

### 3.8 With sbatch

Here is an sbatch example with mini-chameleon. We create the following mini-chameleon.batch file:

```

#!/bin/bash
#SBATCH --nodes=2 # Number of nodes
#SBATCH --ntasks-per-node=24 # Number of tasks (MPI processes)
↪ per node: you will probably want more than 1
#SBATCH --cpus-per-task=1 # Number of threads per task
#SBATCH --exclusive # Non shared usage of resources
#SBATCH -C miriel # Constraint running on a miriel
↪ node
#SBATCH --time=10:00 # Time limit
#SBATCH --job-name=mini-chameleon
#SBATCH --output=out_mini-chameleon_%j
## Run the reference mini-chameleon
OMPI_MCA_pml='^ucx' exec guix shell --pure
↪ --preserve="^OMPI|^SLURM|^STARPU" mini-chameleon slurm
↪ --with-input=slurm=slurm@23 -- srun check_dgemm -v mpi

## Uncomment the line below to run your own mini-chameleon
↪ with sources located in /path/to/my/mini-chameleon
# OMPI_MCA_pml='^ucx' exec guix shell --pure --tune
↪ --preserve="^OMPI|^SLURM|^STARPU" mini-chameleon slurm
↪ --with-input=slurm=slurm@23
↪ --with-source=mini-chameleon=/path/to/my/mini-chameleon --
↪ srun check_dgemm -v mpi

```

We can then batch it:

```

sbatch mini-chameleon.batch

```

## 4 Singularity

Il est possible de tourner une application MPI via singularity.

### 4.1 Création de l'image singularity

Nous pouvons créer une image `singularity` par exemple ainsi:

```
guix pack -f squashfs bash coreutils grep hello hello-mpi
↪ intel-mpi-benchmarks inetutils openssh sed slurm which
↪ --with-input=slurm=slurm@23 -S /bin=bin
↪ --entry-point=/bin/bash
```

### 4.2 Réservation multi-noeuds

Dans l'exemple suivant, on fait une réservation exclusive sur deux noeuds.

```
salloc -N 2 --exclusive
```

### 4.3 Bonjour le monde

Il est possible d'utiliser `srun` sur une application *non MPI* telle que `hostname`:

```
srun -l singularity exec
↪ /gnu/store/f96nn7ck8pn9g3pmp83g4mhppzxlyqx7-bash-coreutils
↪ -grep-hello-hello-mpi-squashfs-pack.gz.squashfs
↪ hostname
```

### 4.4 Hello World MPI

En revanche, pour tourner une application `mpi`, il faut charger explicitement le module `mpi` correspondant à la version utilisée dans l'image. Celui sera utilisé pour que le lanceur `mpiexec`, à l'extérieur de l'image `singularity` soit compatible avec le binaire (ici `hello-mpi`) qui lui-même a été lié à `mpi` lors de la construction de l'image:

```
module load mpi/openmpi/4.1.1
OMPI_MCA_pml='^ucx' mpiexec -n 2 --map-by node singularity
↪ exec /gnu/store/f96nn7ck8pn9g3pmp83g4mhppzxlyqx7-bash-core
↪ utils-grep-hello-hello-mpi-squashfs-pack.gz.squashfs
↪ hello-mpi
```

## 4.5 Libérez les noeuds

```
squeue -u <username>
scancel <jobid>
```

## 4.6 Hello World MPI via son propre MPI hôte

### 4.6.1 Installation d'OpenMPI

```
export OMPI_DIR=/path/to/openmpi/install/dir/
mkdir -p $OMPI_DIR
guix build -S openmpi
tar xJf /gnu/store/47hmvcd4qpn2f799lp6r4iq2yxxkxpx6-openmpi-4.1.1.tar.xz
./configure --with-slurm --prefix=$OMPI_DIR
sruntime -N 1 --exclusive make -j all install
```

### 4.6.2 Test de l'installation

```
export PATH=$OMPI_DIR/bin:$PATH
mpiexec -n 2 --map-by node hostname
```

Remarquez les instructions recommandées mais pas nécessaire en pratique:

```
export MANPATH=$OMPI_DIR/share/man:$MANPATH
export LD_LIBRARY_PATH=$OMPI_DIR/lib:$LD_LIBRARY_PATH
```

### 4.6.3 Réservez multi-noeuds

On fait une réservation sur deux noeuds:

```
salloc -N 2 --exclusive
```

### 4.6.4 Hello World MPI

```
OMPI_MCA_pml='^ucx' mpiexec -n 2 --map-by node singularity
↪ exec /gnu/store/f96nn7ck8pn9g3pmp83g4mhppzxlyqx7-bash-core
↪ utils-grep-hello-hello-mpi-squashfs-pack.gz.squashfs
↪ hello-mpi
```

#### 4.6.5 Libérez les noeuds

```
squeue -u <username>  
scancel <jobid>
```

## 5 Alternative MPI (nmad)

### 5.1 Réserveation multi-noeuds

```
salloc -N 2 --exclusive
```

### 5.2 Hello World MPI

Dans le cas d'nmad, mpiexec (fourni donc par nmad) appelle *in fine* srun (en lui transmettant les bonnes options). On peut donc faire l'appel ainsi:

```
guix shell --pure --preserve="^SLURM" slurm hello-mpi  
↪ --with-input=slurm=slurm@23 --with-input=openmpi=nmad --  
↪ mpiexec hello-mpi
```

Il est à noter que l'nmad fourni par guix n'est pas lui-même lié avec slurm et on peut donc se passer de --with-input=slurm=slurm@23 (du moment qu'on charge slurm@23):

```
guix shell --pure --preserve="^SLURM" hello-mpi openmpi  
↪ slurm@23 --with-input=openmpi=nmad -- mpiexec hello-mpi
```

Notez qu'on aurait pu mettre explicitement nmad à la place d'openmpi du fait de la transformation --with-input=openmpi=nmad.

### 5.3 Produit de matrice

Dans le cas d'nmad, mpiexec (fourni donc par nmad) appelle *in fine* srun (en lui transmettant les bonnes options). On peut donc faire l'appel ainsi:

```
guix shell --pure --preserve="^SLURM" chameleon openmpi slurm  
↪ --with-input=slurm=slurm@23 --with-input=openmpi=nmad  
↪ --with-input=openblas=mkl -- mpiexec chameleon_dtesting -H  
↪ -o gemm --check
```

Comme ci-dessus, puisque nmad fourni par guix n'est pas lui-même lié avec slurm, on peut se passer de --with-input=slurm=slurm@23 (du moment qu'on charge slurm@23):

```
guix shell --pure --preserve="^SLURM" chameleon openmpi
↪ slurm@23 --with-input=openmpi=nmad
↪ --with-input=openblas=mkl -- mpiexec chameleon_dtesting -H
↪ -o gemm --check
```

## 5.4 Libérez les noeuds

```
squeue -u <username>
scancel <jobid>
```

# 6 GPU (Cuda)

## 6.1 Node reservation

We reserve a (single) sirocco node (see plafrim reference)

```
salloc -C sirocco
```

## 6.2 (single-node) GPU execution of chameleon

We can launch the execution from the frontal node through srun:

```
LD_PRELOAD=/usr/lib64/libcuda.so guix shell --pure
↪ --preserve=~LD_PRELOAD chameleon-cuda openssh slurm
↪ --with-input=slurm=slurm@23 -- srun chameleon_dtesting -o
↪ gemm -H -g 2 -t 2 -m 4000 -n 4000 -k 4000 --check
```

It is also possible to explicitly log onto the computational node (assuming we obtained sirocco06 during the salloc step). In this case, slurm is not necessary:

```
ssh sirocco06 # log onto the computational node
LD_PRELOAD=/usr/lib64/libcuda.so guix shell --pure
↪ --preserve=~LD_PRELOAD chameleon-cuda openssh --
↪ chameleon_dtesting -o gemm -H -g 2 -t 2 -m 4000 -n 4000 -k
↪ 4000 --check # execution
exit # log out from the computational node
```

## 6.3 Development environment for (single-node) GPU execution of mini-chameleon

Log onto the computational node:

```
ssh sirocco06 # log onto the computational node
```

Enter the guix shell environment for developing mini-chameleon:

```
export LD_PRELOAD=/usr/lib64/libcuda.so
guix shell --pure -D chameleon-cuda
↪ --with-input=slurm=slurm@23 bash gcc-toolchain emacs nano
↪ vim --tune -- bash --norc
```

You are now in a shell ready to develop your project with cuda support.

```
# compile, test, ...
```

Once done, you can exit the guix shell environment:

```
exit # log out from the guix shell environment
```

And exit the computational node:

```
exit # log out from the computational node
```

You should now be back onto the frontal node.

## 6.4 Releasing node reservation

```
squeue -u <username>
scancel <jobid>
```