HPC Singularity : reproduire et partager vos simulations

David Brusson
Direction du Numérique, Université de Strasbourg
brusson@unistra.fr

Michel Ringenbach
Direction du Numérique, Université de Strasbourg
mir@unistra.fr

Vincent Lucas
Direction du Numérique, Université de Strasbourg
lucas@unistra.fr

Résumé

La démarche scientifique se base sur la reproductibilité des expériences. Pour des expérimentations exigeant d'importantes ressources de calcul, l'utilisation d'un supercalculateur devient nécessaire. Se posent alors les problématiques des différences d'architecture, de système, de bibliothèques, et des communications entre nœuds de calcul qui sont des notions complexes à appréhender.

La résolution de ces problématiques présente un double intérêt :

- *permettre facilement de rejouer les simulations et par conséquent permettre la validation des résultats de recherche ;*
- *permettre l'ouverture des ressources de calcul à un plus grand nombre d'utilisateurs, grâce à une simplification de l'usage.*

Pour répondre à ces besoins, l'Université de Strasbourg offre depuis 2019, au sein de son mésocentre, un service de conteneurs basé sur la technologie Singularity [1].

Ces conteneurs incluent à la fois le système d'exploitation, les bibliothèques spécifiques, et le logiciel scientifique compilé. Ainsi, cela permet de répondre à une majorité des problématiques de reproductibilité et de simplicité de diffusion.

Dans cet article, nous présentons les technologies de conteneurs, les avantages de Singularity dans un environnement HPC [2] et l'interopérabilité de ce dernier avec Docker. Puis, nous détaillons l'architecture ainsi que les performances obtenues.

La conclusion résume ces fonctionnalités et présente les évolutions des usages attendues : une réelle facilité d'utilisation avec des simulations réutilisables et améliorables facilement.

Mots-clefs

HPC, conteneurs, Singularity, Docker, reproductibilité

1 Introduction

La démarche scientifique se base sur la reproductibilité des expériences. Dans le cas d'expérimentations par simulations, d'importantes ressources de calcul peuvent être nécessaires. Se pose alors la problématique de porter une simulation d'un ordinateur personnel vers un supercalculateur. En effet, les différences d'architecture, de système, de bibliothèques, ainsi que l'utilisation de communications entre nœuds de calcul sont des notions complexes à appréhender pour réaliser un paramétrage rigoureux et une exécution adéquate de chaque simulation.

Cette problématique de simplifier le portage d'une simulation présente un double intérêt :

- faciliter le rejeu de simulations et par conséquent la validation des résultats de recherche ;
- et par simplification de l'usage, permettre l'ouverture des ressources de calcul à un plus grand nombre d'utilisateurs.

Pour répondre à ces besoins, l'Université de Strasbourg a mis en place en février 2019 un service de conteneurs au sein de son mésocentre et basé sur la technologie *Singularity* [1] .

Or, ces conteneurs reposent sur des images pouvant inclure à la fois :

- le système d'exploitation ;
- les bibliothèques spécifiques ;
- et le logiciel scientifique compilé.

Ainsi, réaliser des simulations via les conteneurs permet de répondre à une grande majorité des problématiques de reproductibilité et de simplicité de diffusion.

De plus, pour traiter la problématique des différences d'architectures, nous avons implémenté un mécanisme totalement automatisé de régénération à la demande des images en y ajoutant des modules spécifiques : par exemple les bibliothèques MPI pour la communication entre nœuds du supercalculateur.

Cet article détaille la mise en place et le retour d'expérience de l'utilisation de ce service.

2 Singularity

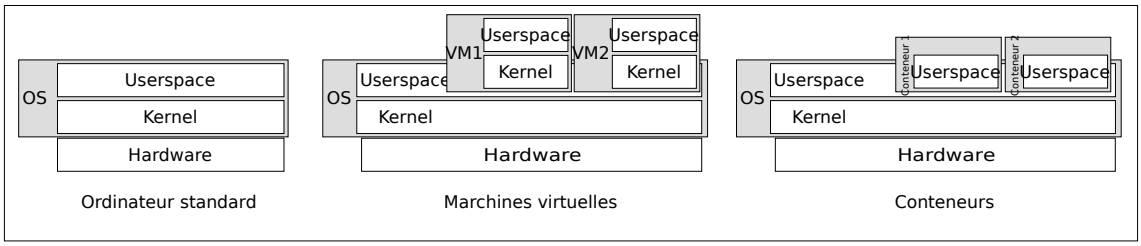

Figure 1 - Différences entre ordinateur, machines virtuelles et conteneurs

Singularity est un gestionnaire de conteneur (cf. Figure 1 -), comme *Docker* [9], qui permet de virtualiser un système d'exploitation qui :

- partage le noyau de la machine hôte ;
- permet d'installer son propre environnement de travail (OS, bibliothèques, logiciels).

Deux étapes sont nécessaires pour utiliser un conteneur [8] :

- la construction de l'image servant à installer l'environnement désiré ;
- l'exécution du contenu de l'image avec des paramètres définis par l'utilisateur.

Cependant, la philosophie de *Singularity* diffère de celle de *Docker*. En effet, *Docker* a été créé pour faire fonctionner des services nécessitant les droits d'un super-utilisateur.

De son côté *Singularity* permet à un utilisateur standard de démarrer un conteneur. De fait, les processus lancés dans le conteneur bénéficie strictement des mêmes droits que les autres processus de cet utilisateur, induisant une bonne isolation des données et des calculs de chaque utilisateur. Cette différence permet à *Singularity* d'être particulièrement adapté au monde du *HPC* où de multiples utilisateurs partagent les mêmes ressources matérielles.

2.1 Création de conteneurs

Des images déjà construites peuvent être récupérées sur des dépôts publics comme *Docker Hub* [3] ou *Singularity Hub* [4]. Ces entrepôts proposent des images basées sur des distributions ou incluant des logiciels standards. Ces images sont versionnées et utilisées comme base de création de nouvelles images, grâce à un système d'héritage.

La création d'une image est décrite par un fichier de définition (cf. Figure 2 -) qui contient toutes les commandes de génération, d'installation et de paramétrage de l'environnement :

```
Bootstrap: docker
From: ubuntu:cosmic

%runscript
echo "Hello world!"

%setup

%environment

%labels

%post
echo "Distribution update"
apt-get update
```

Figure 2 - Fichier de description singularity : hello_world.def

```
root@machine_hote$ singularity build hello_world.sif
hello_world.def
```

Figure 3 - Génération d'une image singularity : hello_world.sif

Un avantage de *Singularity* est de pouvoir utiliser des images *Docker* ou d'en hériter comme base de construction (cf. Figure 2 -).

Docker comme *Singularity* nécessitent les droits du super-utilisateur pour cette création (cf. Figure 3 -).

2.2 Exécution de conteneurs

Une fois le paquet *Singularity* installé sur la machine hôte, un utilisateur standard peut exécuter des images *Singularity* sur cette dernière.

Singularity permet deux types d'utilisation :

– un mode shell qui permet de rentrer dans le conteneur et de bénéficier de l'environnement construit (cf. Figure 4 -).

```
user@machine_hote$ python --version
Python 3.7.0 (default, Jan  1 1970, 00:00:01)
[GCC 5.5.0]

user@machine_hote$ singularity shell hello_world.sif
Singularity hello_world.sif:~> ls
hello_world.def  hello_world.sif

Singularity hello_world.sif:~> python --version
bash: python: command not found
```

Figure 4 - Exécution en mode shell d'un conteneur Singularity

– une exécution comme une application standard avec d'éventuels arguments de manière transparente pour l'utilisateur (cf. Figure 5 -).

```
user@machine_hote$ ./hello_world.sif
Hello world!
```

Figure 5 - Exécution en mode application d'un conteneur Singularity

Ce deuxième mode d'exécution permet notamment d'utiliser des conteneurs à l'aide d'ordonnanceurs (*Slurm, Torque, PBS, …*) qui sont généralement utilisés pour lancer des calculs sur des clusters *HPC*.

2.3 Support des pilotes HPC

Les machines *HPC* nécessitent des pilotes et des bibliothèques pour optimiser les communications entre nœuds de calcul. Au sein d'un conteneur, ces mêmes pilotes et bibliothèques doivent être installés pour accéder au matériels spécialisés des machines hôtes : comme les cartes réseau haut-débit et faible latence *InfiniBand* et *Omni-Path*, ou des cartes d'accélérations graphiques attachées en *PCIe*.

Pour fonctionner, les versions installées sur la machine hôte et le conteneur doivent être compatibles. Pour assurer cette compatibilité au sein du mésocentre du site Alsace, nous fournissons des images « de base » intégrant les drivers et bibliothèques réseau, notamment *OpenMPI*, qui est la bibliothèque de communication entre les nœuds de calcul.

Les utilisateurs peuvent ainsi construire des images personnalisées qui héritent de ces images de base. Néanmoins cela pose deux problèmes :

- avoir accès aux droits super-utilisateur afin de créer une image ;
- laisser la possibilité à l'utilisateur de redistribuer des images construites avec des logiciels soumis à licence, comme par exemple les compilateurs *Intel*®.

Pour pallier ces problèmes, nous avons dû déporter la génération d'images des utilisateurs sur une architecture dédiée.

3 Architecture

Comme nous l'avons vu dans les paragraphes précédents, l'utilisation de *Singularity* [10] se découpe en 2 phases : création et exécution d'une image. Si l'exécution peut être réalisée par un utilisateur standard, la création nécessite des droits super-utilisateur.

Pour des raisons de sécurité, un utilisateur du centre de calcul ne peut pas avoir ce type de droits. Toutefois, pour permettre à un utilisateur de créer des images, nous avons mis en place une machine virtuelle isolée et dédiée à cet usage.

3.1 Singularity sur le centre de calcul

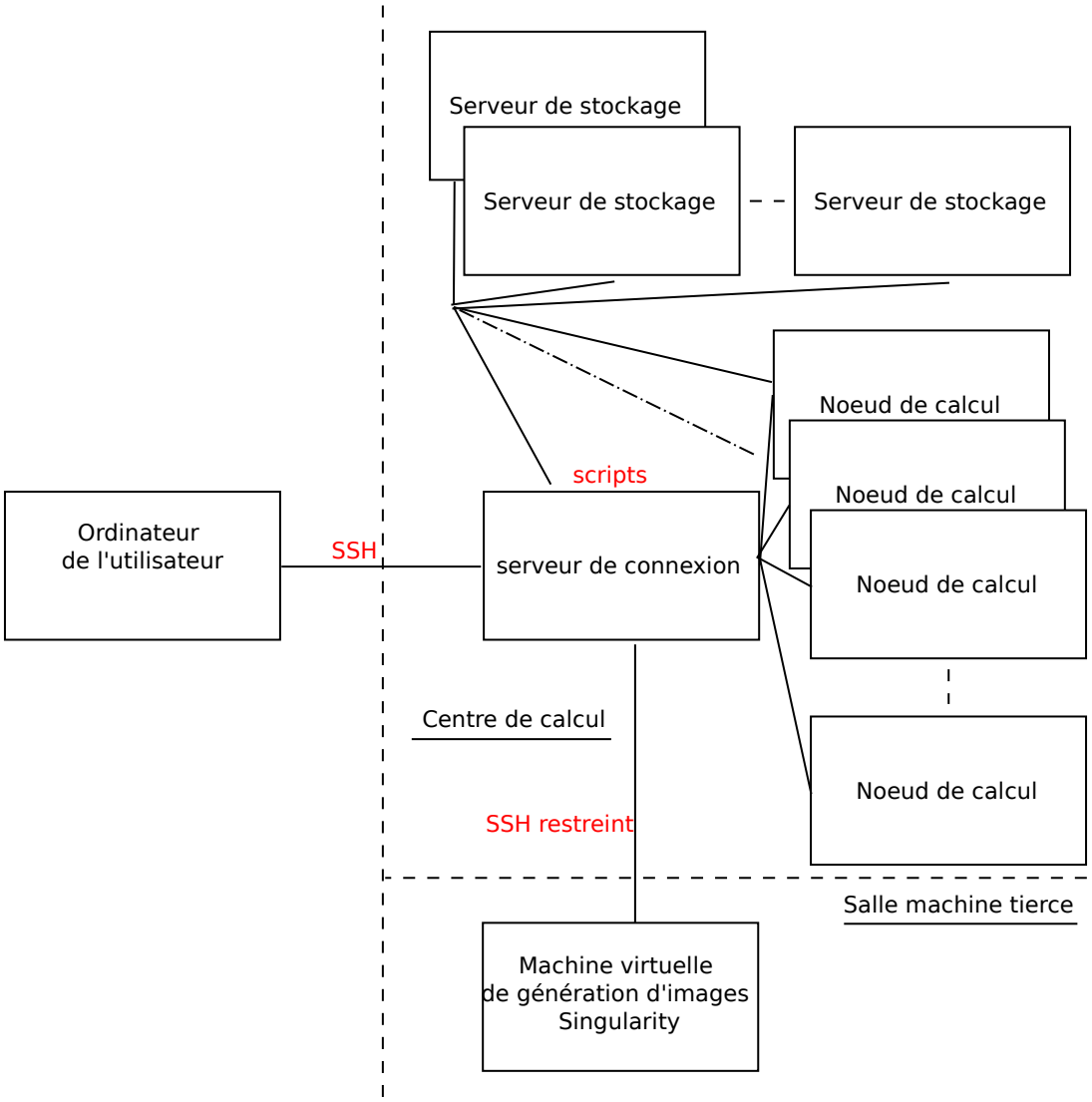

Figure 6 - Architecture

Le centre de calcul est composé d'un serveur de connexion, de nœuds de calcul et de serveurs de stockage (cf. Figure 6 -) :

- le serveur de connexion est le point d'entrée des utilisateurs dans le centre de calcul. Il permet de transférer des données, de compiler des logiciels ou des bibliothèques spécifiques et de préparer des simulations à exécuter sur les nœuds de calcul. Une fois la préparation validée, un ordonnanceur recueille et met dans une file d'attente les jobs des utilisateurs ;
- les nœuds de calcul sont réservés par un ordonnanceur et exécutent les simulations des utilisateurs ;
- les serveurs de stockage mettent à disposition un système de fichiers distribué et accessible à distance via le serveur de connexion et les nœuds de calcul.

Singularity est installé sur le système de fichiers distribué et est ainsi utilisable sur tout le cluster. Une image *Singularity* se lance comme une application standard et son intégration à l'ordonnanceur est donc transparente (cf. Figure 7 -).

```
#!/bin/bash

#SBATCH -t 00:30:00
#SBATCH -n 24

module load openmpi/openmpi-3.1.i18

mpirun mon_application


#!/bin/bash

#SBATCH -t 00:30:00
#SBATCH -n 24

module load openmpi/openmpi-3.1.i18
module load singularity/singularity

mpirun singularity run mon_application
```

Figure 7 - Ordonnancement : script Slurm intégrant un conteneur Singularity

3.2 Singularity sur la machine virtuelle

Pour permettre la génération d'images *Singularity*, notamment avec des logiciels et bibliothèques spécifiques au centre de calcul, nous avons mis en place une machine virtuelle dédiée [5]. Cette machine est isolée du centre de calcul et propose des commandes restreintes afin de permettre aux utilisateurs d'obtenir les accès privilégiés nécessaire à la génération d'une image.

Pour accéder à cette machine, le frontal de connexion du centre de calcul met à disposition des commandes spécifiques permettant de :

- copier les données nécessaire à la création de l'image, dont le fichier de description ;
- lancer la création de l'image ;
- récupérer l'image sur le frontal de connexion.

La restriction des commandes sur cette machine se fait au moyen d'une configuration ssh limitant la clé de chaque utilisateur à un seul script réalisant les actions ci-dessus (cf. Figure 8 -).

```
# /home/singularity/.ssh/authorized_keys
```

```
command="hpc_singularity_generation_script.sh <login>",no-port-
forwarding,no-x11-forwarding,no-agent-forwarding ssh-rsa
<public_key> <comment>
```

Figure 8 - Exemple de restriction SSH pour un utilisateur

```
singularity run --bind
/host_path_to_openmpi:/container_path_to_openmpi --bind
/host_path_to_intel_compilers_and_libraries:/container_path_to_in
tel_compilers_and_libraries <project_name>.sif
```

Figure 9 - Exemple d'utilisation d'un conteneur avec points de montage

Néanmoins, ce script permet de générer une image à la fois depuis un fichier de description *Singularity* ou d'une image *Docker*. Pour ce faire, ce serveur dispose non seulement d'une installation de *Singularity*, mais également d'un *repository* local *Docker*.

Pour les bibliothèques spécifiques et les logiciels soumis à licence, comme par exemple les compilateurs *Intel*®, nous mettons à disposition des images pré-construites servant de base pour compiler les images contenant les logiciels des chercheurs. Une fois l'image généré, le contenu non redistribuable est supprimé. L'image sera néanmoins exécutable sur toute machine disposant dans son système hôte de ce contenu en définissant un point de montage avec l'option ***--bind*** sur le système de fichier hôte (cf. Figure 9 -).

4 Performances

Pour les tests suivants, nous comparons les performances obtenues entre l'OS hôte des nœuds de calcul (*Scientific Linux 7.4*) et les conteneurs (utilisant une image de base *Ubuntu 18.10*).

4.1 Test CPU

Le but de ce test est d'évaluer les performances entre une application multi-threadée directement exécutée par le système d'exploitation et la même application incluse dans un conteneur Singularity. L'application sur un calcul matriciel utilisant la méthode de Jacobi, qui est une méthode itérative de résolution d'un système matriciel et hautement parallélisable. Les tests ont été effectués sur la même machine disposant de 2 processeurs avec 12 cœurs chacun, soit un total de 24 cœurs.

```
#!/bin/bash

#SBATCH -t 00:30:00
```

```
#SBATCH -N 1-1
#SBATCH --exclusive

source ~/.bashrc

module load compilers/intel18

time ./jacobi

time singularity run --bind
/opt/intel/compilers_and_libraries_2018.1.163:/opt/intel/compiler
s_and_libraries_2018.1.163 jacobi.sif
```

Figure 10 - Script de tests CPU

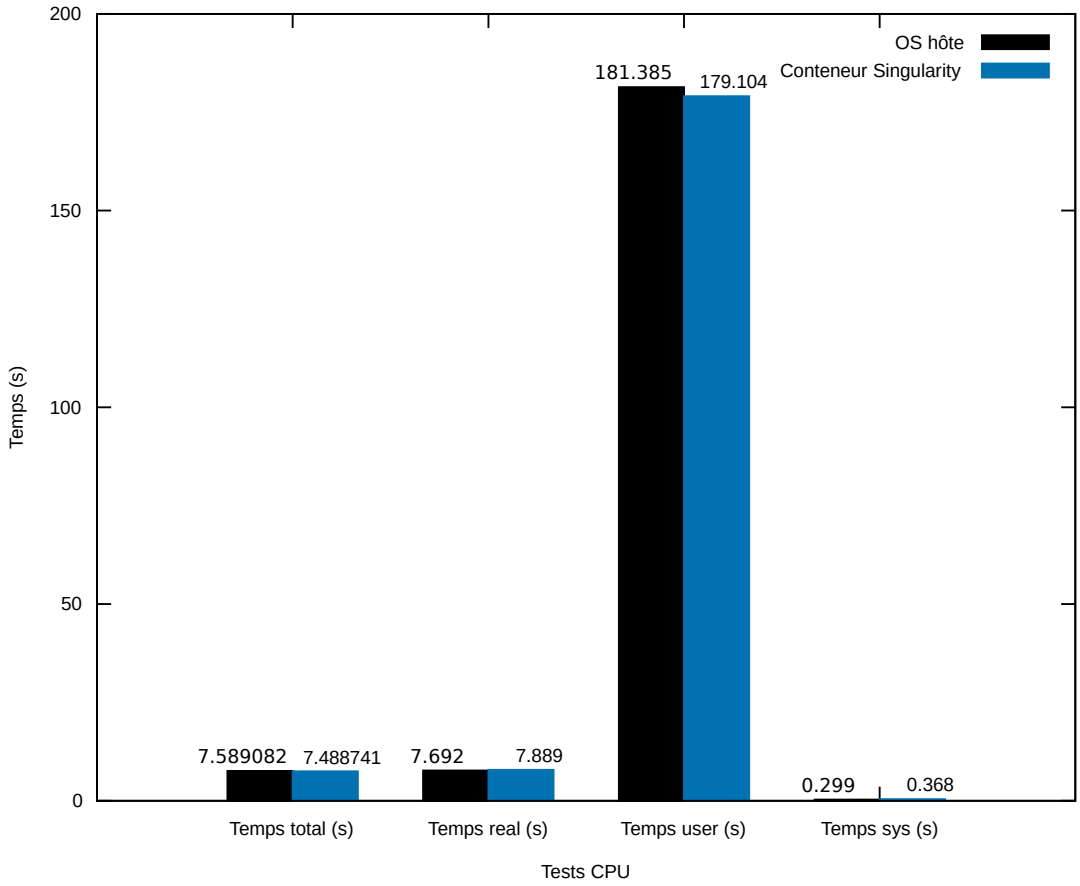

Figure 11 - Résultats : tests CPU

Pour ce test, les résultats (cf. Figure 11 -) sont similaires et montrent que l'impact de *Singularity* sur les performances est négligeable.

4.2 Test GPU

Le but de ce test est d'évaluer la capacité de *Singularity* à accéder directement aux *GPUs*. Ce test réalise une opération matricielle basée sur la fonction *SGEMM* de la bibliothèque *cuBLAS* [11] écrite en *Cuda* [12]. Le serveur utilisé pour ces expérimentations dispose de cartes *Nvidia P100*.

```
#!/bin/bash

#SBATCH -t 00:30:00
#SBATCH -p publicgpu
#SBATCH -n 1
#SBATCH --gres=gpu:1
#SBATCH --constraint=gpup100

source ~/.bashrc

module load compilers/cuda-9.1

./sgemm_gpu 13000

singularity run --nv gpu.sif 13000
```

Figure 12 - Script de tests GPU

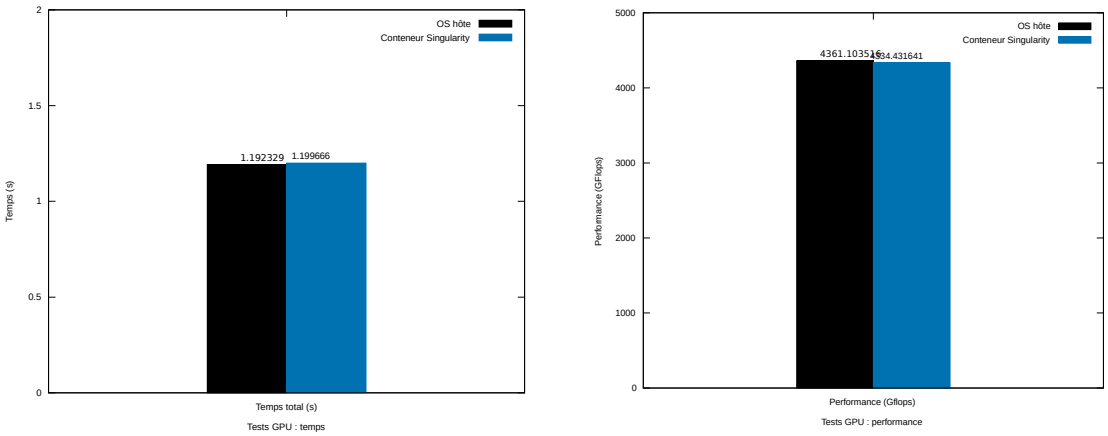

Figure 13 - Résultats : tests GPU

Nous constatons que *Singularity* a bien accès au *GPU*. De plus, nous constatons que les résultats sont similaires avec un impact de *Singularity* négligeable (cf. Figure 13 -).

4.3 Test MPI

Le but de ces tests est d'évaluer la capacité de *Singularity* à utiliser les réseaux de communication inter-nœuds. Nous mesurons ici la bande-passante sur différents types de réseaux : *Ethernet* 1G, *Ethernet* 10G, *Infiniband* 40G et *Omni-path* 100G.

L'application utilise la bibliothèque *OpenMPI* [13] (version 3.1) pour échanger des données entre 2 nœuds de calcul.

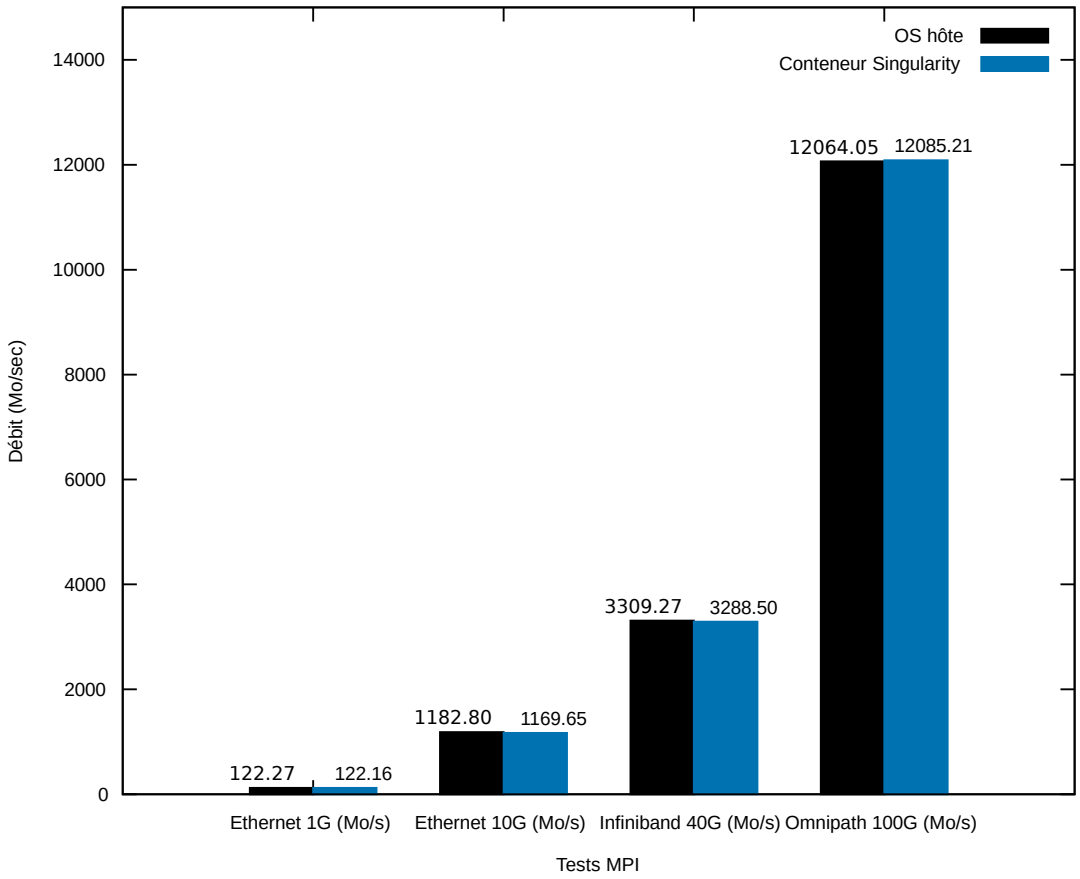

Figure 14 - Résultats : tests MPI

Les résultats (cf. Figure 14 -) montrent que *Singularity* est capable d'utiliser ces différentes technologies réseau et d'en exploiter les capacités avec les mêmes performances que si on utilisait l'OS original.

4.4 Analyse

Les différents tests effectués prouvent qu'un conteneur *Singularity* basé sur un OS différent de celui de l'hôte fonctionne sans dégrader les performances, aussi bien au niveau d'utilisation *CPU*, *GPU* et réseau.

5 Reproductibilité

Afin de rejouer avec précision une simulation, il faut que les éléments suivants soient reproductibles.

– Les données à analyser : aujourd'hui la publication d'un article de recherche est très fréquemment lié à un entrepôt contenant les données sources à analyser.

– La version du système d'exploitation, des logiciels et des bibliothèques : les conteneurs permettent à la fois de fournir des images réutilisables ainsi qu'un fichier de description qui décrit toutes les étapes pour installer et compiler les logiciels utilisés pour la simulation. Il y a deux paramètres à évaluer. Le premier est l'exactitude de conservation de version des logiciels, ce qui est apporté par l'image elle-même. Le second est de permettre à chacun d'étudier la façon dont l'image a été construite, ce qui est apporté par le fichier de description. Il faut cependant bien veiller à fixer les versions des logiciels et bibliothèques installés via les différents gestionnaires de paquets (*apt-get, yum, pip,* etc...) ainsi que la version de l'OS utilisé comme base pour l'image. À noter que les conteneurs n'intègrent pas de kernel et utilisent celui de la machine hôte.

– L'architecture de la machine : c'est le point le plus difficile à reproduire. La méthode la plus fiable est de fournir la liste des caractéristiques techniques de la machine et de son environnement (réseau, système de fichiers partagé).

Avec une publication des données et l'utilisation de *Singularity*, reproduire une expérience est donc largement facilitée. Cependant, l'exactitude des résultats est encore dépendante des versions des *kernels* et de l'architecture des machines hôtes.

6 Conclusion

Dans une optique de reproductibilité des expériences, *Singularity* facilite le rejeu de simulations. En effet, l'utilisation d'une image *Singularity* assure de pouvoir utiliser et distribuer un code sans changement de bibliothèques et de logiciels. Ainsi, hormis les arrondis dus au *kernel* et à l'architecture matérielle utilisés et couplé à la publication des données sources, le mécanisme d'images et de fichiers de description permettent de redistribuer facilement une simulation et de la rejouer sur différents ordinateurs ou centres de calcul [6]. Enfin, les performances obtenus pour les conteneurs sont similaires aux systèmes hôtes : ceci aussi bien au niveau *CPU*, *GPU* que réseau.

De plus, la génération d'images est facilitée par la mise en place d'une machine virtuelle dédiée, qui pourra évoluer notamment avec l'arrivée de nouveau services comme la génération à distance fournie par *Sylabs* [7].

Enfin, la mise en place de conteneurs sur le centre de calcul montre l'évolution possible du service : par exemple avec l'arrivée de services web pour réaliser des types de simulations prédéfinies.

Au final, ce service apporte une réelle facilité d'utilisation avec des simulations distribuables, réutilisables et améliorables facilement.

Bibliographie

[1] Singularity : https://www.sylabs.io/singularity

[2] HPC : https://fr.wikipedia.org/wiki/Superordinateur

[3] Entrepôt de container Docker : https://hub.docker.com

[4] Entrepôt de container Singularity : https://singularity-hub.org

[5] Scripts de génération pour singularity du mésocentre de l'Université de Strasbourg : https://git.unistra.fr/HPC/singularity

[6] Utilisateurs de singularity : https://www.sylabs.io/singularity/whos-using-singularity/

[7] Générateur d'images à distance : https://cloud.sylabs.io/builder

[8] Documentation officielle de singularity : https://www.sylabs.io/guides/3.0/user-guide/

[9] Docker : https://www.docker.com/

[10] Documentation d'utilisation de singularity sur le mésocentre de l'Université de Strasbourg : https://hpc.pages.unistra.fr/singularity/

[11] Documentation de cuBLAS https://docs.nvidia.com/cuda/cublas/index.html

[12] Cuda : https://developer.nvidia.com/cuda-zone

[13] OpenMPI : https://www.open-mpi.org