Ressources pour l'IA au Mésocentre

Journée Audaces 2025
jeudi 5 juin 2025

David Grimbichler
david.grimbichler@uca.fr

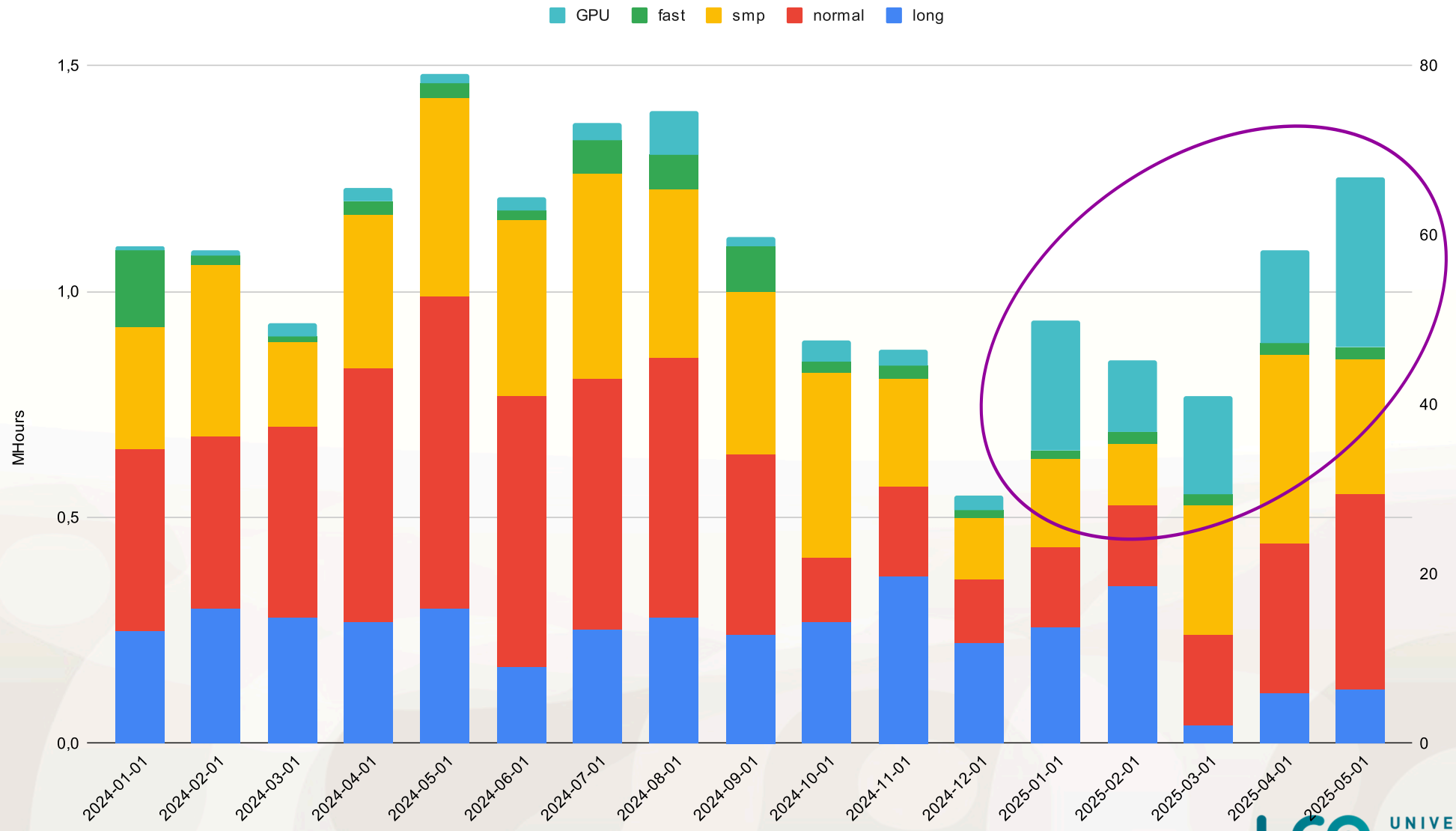
MÉSOCENTRE
UNIVERSITÉ
Clermont Auvergne

Ressources GPU au Mésocentre

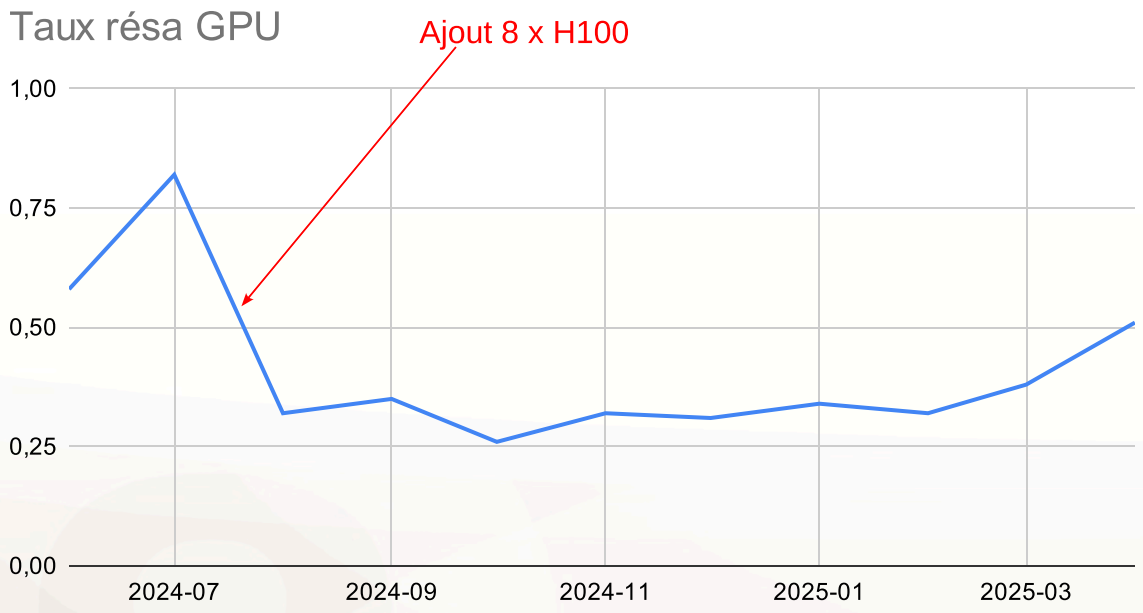
Les GPU au Mésocentre sont accessibles via :

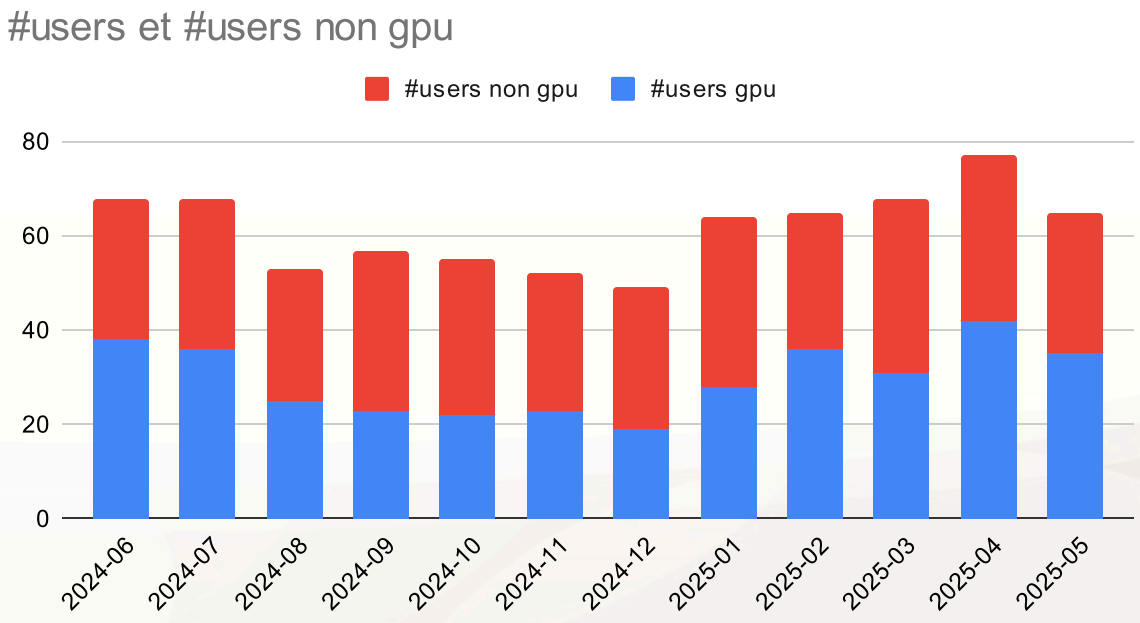
- des noeuds de calcul sur le cluster HPC2
- autrefois, sur le serveur frontal du cluster de calcul (GPU déplacé depuis)
- prochainement :
  - des noeuds de calcul sur le cluster HPC3 (8 x H100)
  - et les frontaux HPC3 (2 x L4)
- pas encore sur le cloud OSCAR, mais maquettage prévu

Quel usage des GPU ?



Quel usage des GPU ?





GPU sur HPC2

Liste des GPU disponibles sur le cluster de calcul HPC2 :

Modèle GPU	Nombre	Mémoire par GPU	Coeurs (TFlops FP32)
NVIDIA Tesla T4	1	16 Go (GDDR6, 300Go/s)	2560 (8 TFlops, 320 Tensor)
NVIDIA Tesla P40	2	24 Go (GDDR5, 300Go/s)	3840 (12 TFlops)
NVIDIA Tesla P100	3	16 Go (HBM2, 700Go/s)	3584 (9 TFlops)
NVIDIA Tesla V100	8	32 Go (HBM2, 900Go/s)	5120 (14 TFlops, 640 Tensor)
NVIDIA A100	3	82 Go (HBM2e, 1.5To/s)	6912 (20 TFlops, 432 Tensor)
AMD Instinct MI210	1	64 Go (HBM2e, 1.6To/s)	6656 (23 TFlops)
NVIDIA H100	10	96 Go (HBM3, 3.3To/s)	16896 (62 TFlops, 528 Tensor)

Accès aux GPU du cluster HPC2 par job

Accès par job SLURM :

- soumission d'un job sur la file `gpu`
- spécification de la ressource gpu demandée via `gres`

Job interactif demandant un GPU de n'importe quel type :

```
srun --partition=gpu --ntasks=1 --cpus-per-task=4 --mem=16G --time=12:00:00 --gres=gpu:1 --pty bash
```

Job interactif demandant un GPU de type H100 :

```
srun --partition=gpu --ntasks=1 --cpus-per-task=4 --mem=16G --time=12:00:00 --gres=gpu:h100:1 --pty bash
```

Accès aux GPU du cluster HPC2 par job

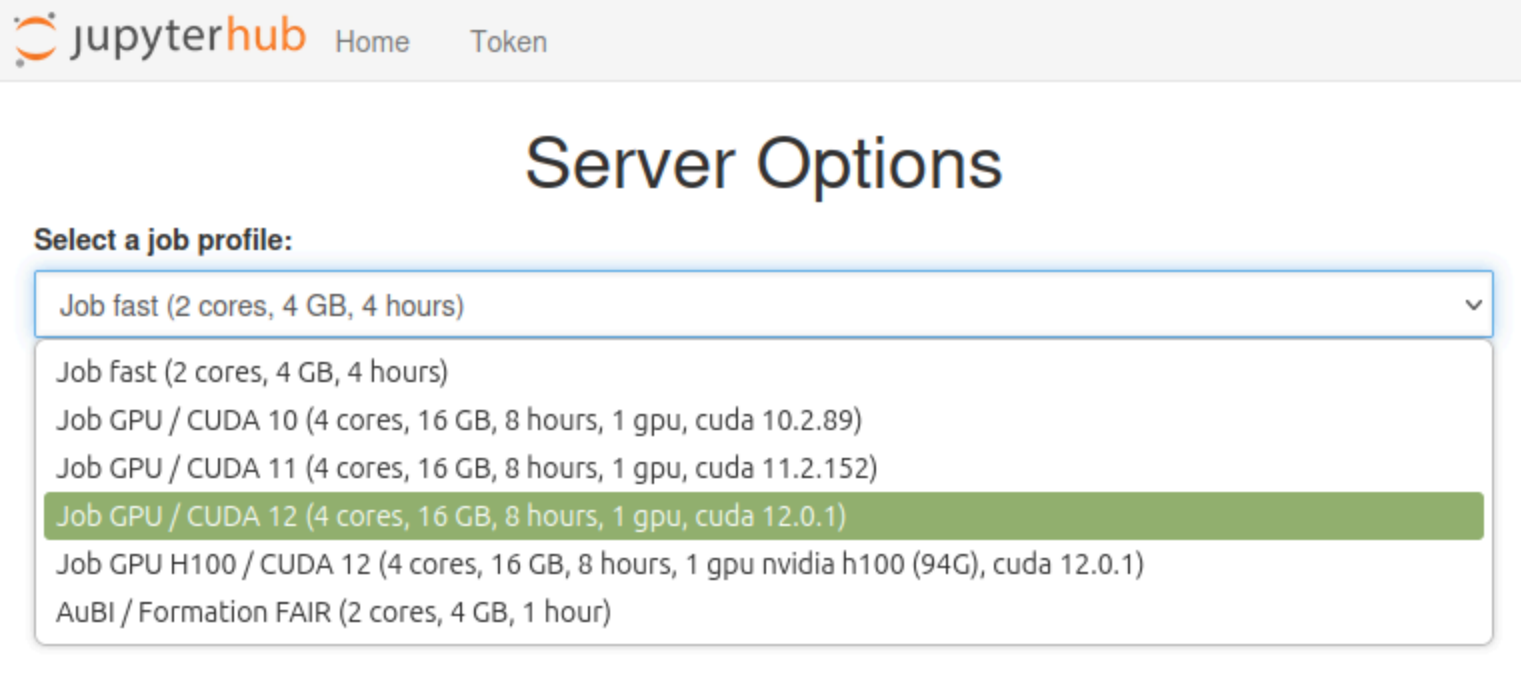
Job demandant un GPU de n'importe quel type **parmi V100, A100, H100** :

```
#!/bin/bash
#SBATCH ...
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=16G
#SBATCH --time=12:00:00
#SBATCH --gres=gpu:1
#SBATCH --constraint='v100|a100|h100'
...
```

Accès aux GPU par Jupyter Hub sur cluster HPC2

Accès par JupyterHub :

- sur https://hub.mesocentre.uca.fr/ : `Accéder au JupyterHub`
- choix du profil :



- soumission transparente d'un job sur le cluster HPC2 pour le notebook

GPU sur cluster HPC2 : conseils

> Faites du checkpointing pendant les entraînements de modèles

- jobs à durée limitée
- jobs pouvant être stoppés pour diverses raisons
  - out of memory
  - problème serveur ou réseau
  - coupure électrique
  - ...
- checkpointing == ne pas reprendre un entraînement de zéro

GPU HPC2 : checkpointing pytorch

Exemple en pytorch :

```
model = MyModel()
optimizer = optim.Adam(model.parameters(), lr = 0.001)
criterion = nn.CrossEntropyLoss()
num_epochs = 10
checkpoint_every = 3

if args.load_model:
    checkpoint = torch.load("checkpoint.pth")
    model.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])

for epoch in range(num_epochs):
    losses = []
    for batch_idx, (data, targets) in enumerate(train_loader):
        [...]

    if epoch % save_every == 0:
        torch.save( { 'state_dict': model.state_dict(),
                      'optimizer': optimizer.state_dict() },
                    f"checkpoint_epoch_{epoch}.pth")
```

GPU sur cluster HPC2 : conseils

> Utiliser le multi-gpu facilement

- facile à mettre en place avec plusieurs GPU sur le même noeud de calcul, peu de changement de code

Exemple PyTorch :

```
model = MyModel()
model = torch.nn.DataParallel(model)

# En spécifiant les device id GPU :
model = MyModel()
model = torch.nn.DataParallel(
          model,
          device_ids = [2, 3])
```

Exemple TensorFlow :

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    model = MyModel()
    model.compile(...)

model.fit(...)

# Avec les device id GPU :
strategy = tf.distribute.MirroredStrategy(
                  ["GPU:2", "GPU:3"])
```

GPU sur cluster HPC2 : conseils

> Utiliser le multi-gpu facilement

- plus difficile à mettre en place avec plusieurs GPU sur plusieurs noeuds de calcul
  - changements importants dans le code
  - nécessité de faire de la communication inter-noeuds
  - utilisation de bibliothèques de passage de messages

Conseil : utiliser la bibliothèque `horovod`
https://github.com/horovod/horovod
`Distributed training framework for TensorFlow, Keras, PyTorch`

Exemple complet avec SLURM :
http://www.idris.fr/jean-zay/gpu/jean-zay-gpu-hvd-tf-multi.html

Environnement logiciel pour l'utilisation du GPU

Solution 1 : préparer un environnement python (venv ou conda, par exemple)

Exemple (https://hub.mesocentre.uca.fr/docs/cluster/applis/exempleGpuTensorflowVenv/) :

Préparation :

```
$ module load gcc/10.2.0
$ module load python/3.10.10
$ module load cuda/12.4.1
$ virtualenv --python=python3.10 ~/cuda12-py310-venv
$ cd ~/cuda12-py310-venv
$ source bin/activate
...
$ pip install tensorflow[and-cuda] torch ...
$ pip install -r requirements.txt
```

Dans le job :

```
#!/bin/bash
#SBATCH ...
...
module load gcc/10.2.0
module load python/3.10.10
module load cuda/12.4.1
cd ~/cuda12-py310-venv
source bin/activate
...
```

Environnement logiciel pour l'utilisation du GPU

Solution 2 : construire une image conteneur et utiliser singularity sur le cluster

Exemple (https://hub.mesocentre.uca.fr/docs/cluster/applis/singularity/#dans-un-job-avec-gpu) :

```
Bootstrap: docker
From: nvidia/cuda:12.2.2-cudnn8-runtime-ubuntu22.04

%post
    apt -y update
    apt -y install python3-dev python3-pip python3-venv git curl wget zip vim cuda-compiler-12-2
    apt autoremove -y
    git clone https://github.com/XXX/YYY.git && cd YYY && pip install -r requirements.txt && pip install .

%environment
    export LC_ALL=C

%runscript
    bash
```

```
$ singularity build YYY.sif YYY.def
```

Environnement logiciel pour l'utilisation du GPU

Solution 3 : utiliser une image déjà préparée et optimisée par le constructeur du GPU (proposée sur son registry) pour le logiciel dont on a besoin

Exemple (https://catalog.ngc.nvidia.com/containers) : GROMACS

```
# export GMX_ENABLE_DIRECT_GPU_COMM=1
# export GROMACS_TAG=2023.2

# SINGULARITY="singularity run --nv -B ${PWD}:/host_pwd --pwd /host_pwd
                              docker://nvcr.io/hpc/gromacs:${GROMACS_TAG}"

# ${SINGULARITY} gmx mdrun -ntmpi 8 -ntomp 16 -nb gpu -pme gpu -npme 1
                           -update gpu -bonded gpu
                           -nsteps 100000 -resetstep 90000 -noconfout
                           -dlb no -nstlist 300 -pin on -v -gpu_id 0123
```

Cluster de calcul HPC2 : GPU AMD

Constatons :

- beaucoup de dépendance à NVIDIA
- registry NVIDIA
- images docker de base NVIDIA
- tensorflow et torch sur PyPi compilés pour CUDA

Idée : proposer à AMD un **prêt de GPU** au Mésocentre dans le but de maquetter, préfigurer, démontrer l'utilisabilité, évangéliser.

- stack logicielle GPU AMD existante et prête (ROCM au lieu de CUDA, versions de pytorch, tensorflow, onnxruntime, ...)
- délais d'approvisionnement des GPU AMD bien meilleur que NVIDIA
- rapport FLOPS/€ plus avantageux chez AMD
- stack logicielle opensource (pilote amdgpu, rocm, hip, ...)
- outils d'adaptation de code CUDA vers ROCM (HIP)

UNIVERSITÉ Clermont Auvergne

Cluster de calcul HPC2 : GPU AMD

Réception du GPU AMD en août 2024 :

- le Mésocentre prépare les environnements techniques et logiciels
- des tests internes sont effectués
- des mises en pratique sont proposées à quelques équipes/chercheurs/chercheuses
  - nnUNetv2 avec Laurent Sarry, Institut Pascal
    - venv python
  - Helixer avec Hélène Rimbert, GDEC
    - construction image singularity
  - GROMACS avec Julien Devémy, ICCF
    - utilisation image conteneur toute prête
- des rapports sont publiés par AMD et par l'UCA

Cluster de calcul HPC2 : GPU AMD, cas 1

Cas 1 avec Laurent Sarry, Institut Pascal :

- départ : IA (nnUNetv2) pour segmentation auto d'images du myocarde (catégoriser intérieur, myocarde, extérieur) https://doi.org/10.1016/j.compmedimag.2025.102546
- entraînement effectué sur les infra NVIDIA du Mésocentre
- proposition : refaire le travail avec le GPU AMD, et vérifier l'égalité des résultats
- mise en place des environnements techniques (venv python, bibliothèques, ROCM)
- lancement de l'entraînement (mêmes données qu'avec NVIDIA)
- vérification sur les jeux de tests et validation : ok
- utilisation du modèle entraîné sur NVIDIA en inférence sur GPU AMD : ok
- comparaison des résultats avec version NVIDIA (pixel par pixel) : ok

```
$ idiff rocm/2dpp/patient001_frame013.png cuda/2dpp/patient001_frame013.png
Comparing "rocm/2dpp/patient001_frame013.png" and "cuda/2dpp/patient001_frame013.png"
PASS
```

Cluster de calcul HPC2 : GPU AMD, cas 1

> Qu'est-ce qui change ?

```
module load gcc/10.2.0
module load python/3.10.10
virtualenv --python=python3.10 rocm-6.1.2-pytorch
cd rocm-6.1.2-pytorch/
source bin/activate

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm6.1
pip install nnunetv2
```

- la seule ligne de différence par rapport à la création du venv CUDA est l'installation de torch
- tout documenté sur https://pytorch.org/get-started/locally/

> effort de déploiement : minuscule

Cluster de calcul HPC2 : GPU AMD, cas 2

Cas 2 Hélène Rimbert, GDEC :

- départ : utilisation du logiciel Helixer sur le cluster de calcul (utilisation du deep learning pour prédire les annotations sur les gènes)
- proposition : vérifier le fonctionnement avec le GPU AMD
- mise en place des environnements techniques :
  - test 1 : venv python + install tensorflow
  - test 2 : préparation d'une image conteneur singularity à façon (pour calquer l'usage initial)
- publié sur https://github.com/meso-uca/helixer-docker-rocm
- tests et validations :
  - lancement sur le cluster : `singularity run --rocm helixer.sif ...`
  - résultats obtenus à moins de 0.001% d'écart par rapport à version NVIDIA

Cluster de calcul HPC2 : GPU AMD, cas 2

> Qu'est-ce qui change ?

- test 1 : venv python + install tensorflow :
  - `pip install tensorflow-rocm -f https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1/`
  - doc : https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/3rd-party/tensorflow-install.html#install-tensorflow-wheels
- test 2 : construction image conteneur :
  - au lieu de choisir `nvidia/cuda:12.2.2-cudnn8-runtime-ubuntu22.04` comme base
  - on prend `rocm/tensorflow:rocm6.1-py3.10-tf2.15-runtime`
  - doc : https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/3rd-party/tensorflow-install.html#install-tensorflow-prebuilt-docker

> effort de déploiement : minuscule

Cluster de calcul HPC2 : GPU AMD, cas 3

Cas 3 Julien Devémy, ICCF :

- départ : utilisation du logiciel GROMACS sur le cluster de calcul avec GPU
- proposition : vérifier le fonctionnement avec le GPU AMD
- environnements techniques : utilisation d'une image conteneur singularity préparée par AMD
- tests : utilisation GPU AMD ok sur données de test, à poursuivre pour les cas d'usage réels

> Qu'est-ce qui change ?

- on utilise AMD InfinityHub pour trouver le tag de l'image à utiliser (https://www.amd.com/en/developer/resources/infinity-hub.html#q=gromacs)
- et on récupère depuis le registry docker : `docker pull amdih/gromacs:2022.3.amd1_174`
- le Dockerfile utilisé est public : https://github.com/amd/InfinityHub-CI/tree/main/gromacs

> effort de déploiement : plus petit que minuscule

Cluster de calcul HPC2 : GPU AMD

Visibilité :

- Rapports techniques de comparaison GPU AMD, publié par AMD : https://www.linkedin.com/posts/philippe-gregoire-5794716_amd-gpu-customer-testimonial-from-uca-university-activity-7312412521158594561-U-1f/
- Sur le site du Mésocentre aussi : https://mesocentre.uca.fr/actualites/nouvel-article-utilisant-les-ressources-mesocentre-dans-computerized-medical-imaging-and-graphics-halsr-net-improving-cnn-segmentation-of-cardiac-left-ventricle-mri-with-hybrid-attention-and-latent-space-reconstruction

Merci de votre attention 😄

https://mesocentre.uca.fr/

MÉSOCENTRE

UNIVERSITÉ
Clermont Auvergne