SJSU CoE HPC Access Tutorial

CMPE249 Summer 2022

Kaikai Liu, Associate Professor

Email: kaikai.liu@sjsu.edu

# Introduction of HPC

* You can access our coe HPC access information from the main website: <http://coe-hpc-web.sjsu.edu>.
  + A total of 36 nodes, 15 include 1 NVIDIA Tesla P100 12 GB GPUs, and 1 has 2 NVIDIA Tesla P100 GPUs
  + 20 nodes (compute nodes) have 128 GB of RAM, and 16 nodes (GPU and condo nodes) feature 256 GB.
  + The HPC has 110 TB of home directory and data storage, which is available from all nodes via /home and /data. Additionally, the HPC has a high-throughput Lustre parallel file system with a usable space of 0.541 PB, available via /scratch. Each group will have a sub-directory in /data and /scratch that they can write to.
* You already have access to the HPC via your SJSU account. The accessible group folder is “/data/cmpe249-fa22” and “/scratch/cmpe249-fa22”. Please create sub-directories in one of those group directories for your project (your name or group name as the folder name). You should put your dataset in “/data/cmpe249-fa22” and put your working code in “/scratch/cmpe249-fa22” (/scratch has better performance).
* You will have access to your private home directory in the head node. **The head node is not designed for heavy computation**. You can request a GPU/CPU node for heavy computation.
* Your requested GPU/CPU node is connected to your head node internally, and **you do not have internet access when you login to your GPU/CPU node**. You should download the data or install any software in the HPC host machine, not the GPU node.
* You can use the “**scp**” command or the built-in file browser inside the JupyterLab to upload or download the files from HPC.

# Remote access

1. SSH into the HPC host machine (you should VPN to the Campus).
   1. <https://www.sjsu.edu/it/services/network/vpn/index.php>
   2. asm-vpn.sjsu.edu is the new VPN server, Group select Student, enter your SJSU password. The Second password is “PUSH” if you utilize DUO (this may not be needed in the new version).
2. Your SJSUID should be added to the hpc access via your instructor or project advisor. If you cannot access, you should request the access first via your advisor (replace SJSUID with your own SJSU ID).

$ ssh [SJSUID@coe-hpc1.sjsu.edu](mailto:SJSUID@coe-hpc.sjsu.edu)

E.g., ssh [0107960xx@coe-hpc.sjsu.edu](mailto:0107960xx@coe-hpc.sjsu.edu). The password is the same as your SJSU account. If you cannot access the HPC, change it to [SJSUID@coe-hpc1.sjsu.edu](mailto:SJSUID@coe-hpc1.sjsu.edu)

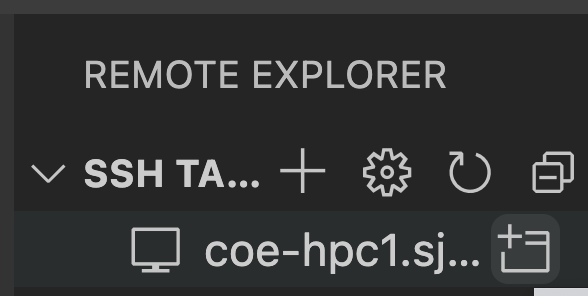
* 1. Option1: Your can use this command to prevent ssh timeout: ssh -o "ServerAliveInterval 60" -o "ServerAliveCountMax 480" *SJSUID*[@coe-hpc1.sjsu.edu](mailto:010796032@coe-hpc.sjsu.edu)
  2. Option2: (**Do not use this method if you are not doing the long training**) Create Persistent SSH connection via screen (install screen) if you do not want to have ssh timeout. You can create one persistent session via $screen -S main\_session (*if you do not perform long lasting training, do not use the persistent session, it may cause problems*)
     1. You can use $ screen -ls to view current screen session. If you want to reconnect to a previous screen session after ssh timeout, you can use

screen –r session\_name (the name you get from screen -ls)

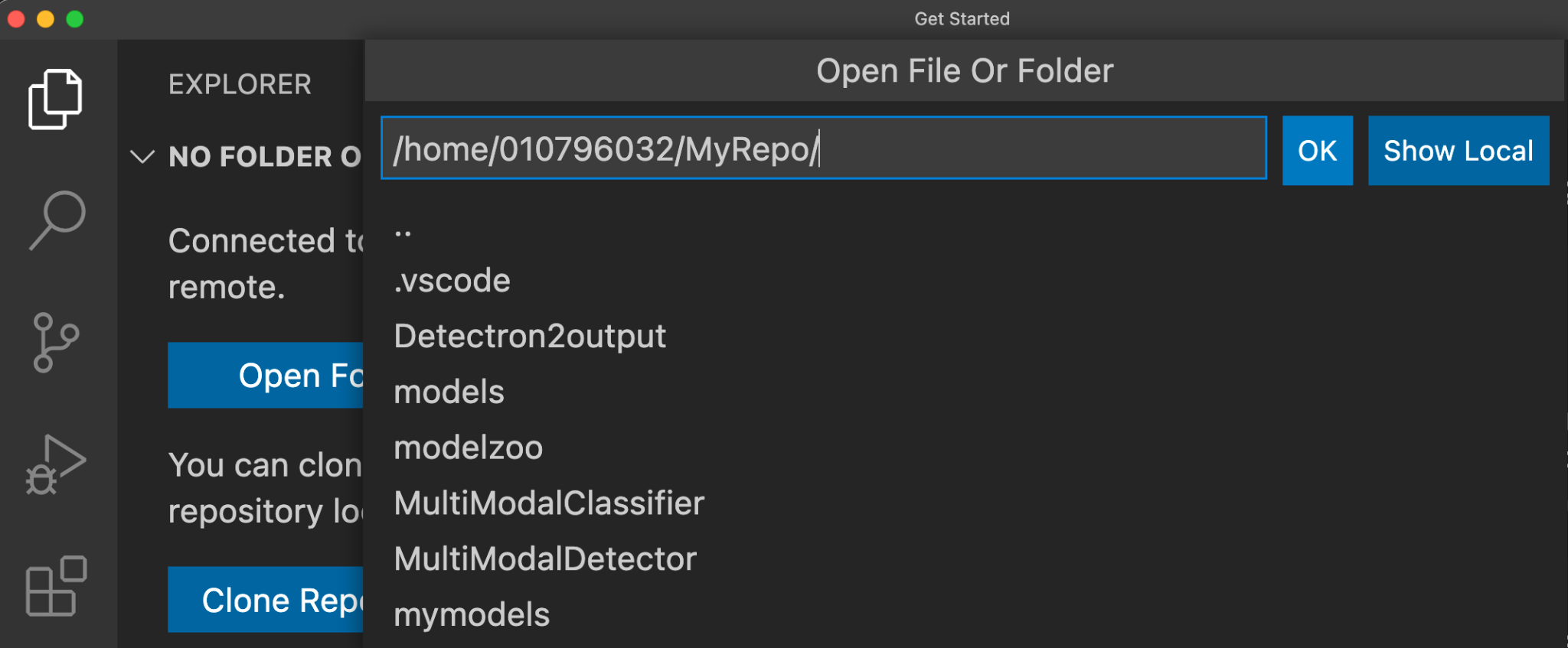
screen -S session\_name -X quit to terminate a detached session (Ctrl+a d to detach a session)

You can use Control+A, then ESC to enable the scroll up. Move up/down with the arrow keys ( ↑ and ↓ ). When you're done, hit q or Escape to get back to the end of the scroll buffer.

1. Utilize Visual Studio Code Remote Debug: <https://code.visualstudio.com/docs/remote/ssh>
   1. Install Remote - SSH extension in Visual Studio Code
   2. Add a new ssh connect in Remote, then click to open the coe-hpc1 server

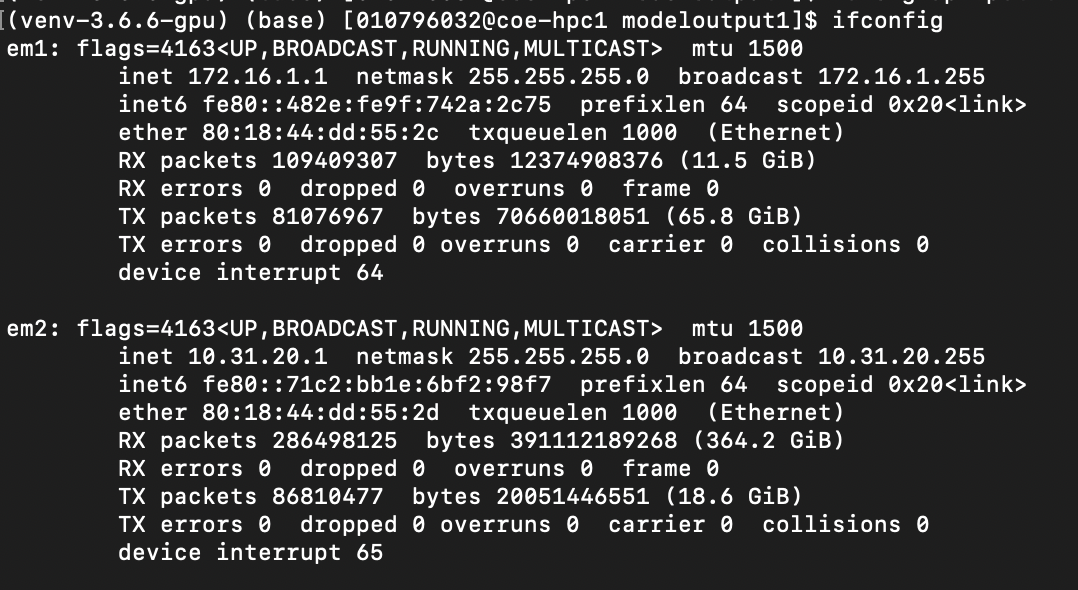


* 1. Click to open the folder as your working directory



* 1. You can also open the Jupyter notebook file in VSCode, you can also use git to sync with Github. **Note: you can edit code but cannot run code in the VSCode as this setup is only connected to the head node not the compute node (GPU node)**

1. You also can check the IP address of your head node via $ifconfig

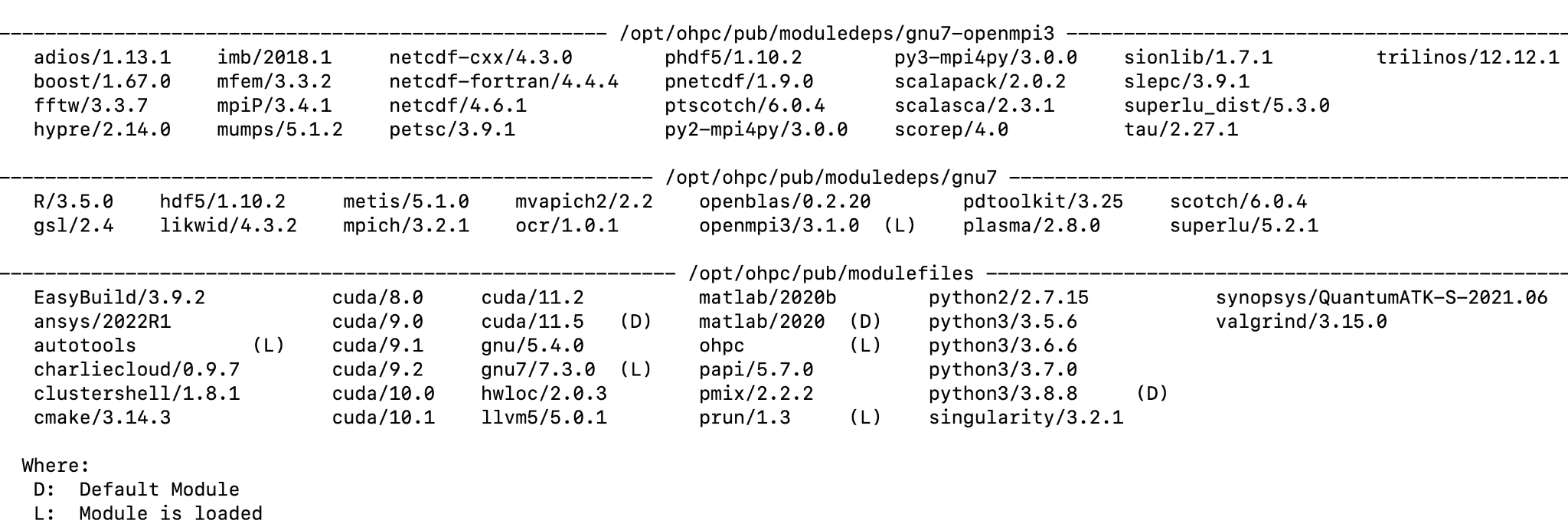


1. The em2 inet address is your IP. You can note this down for future access.

# Software Installation based on Cuda11.2 (under construction)

## Python and Cuda setup

1. Check available software modules via [@coe-hpc1 ~]$ module avail



1. Load the python3.8.8 and cuda11.2 module.

First check current python version

[@coe-hpc1 ~]$ python -V

Python 2.7.5

[@coe-hpc1 ~]$ python3 -V

Python 3.6.8

[@coe-hpc1 ~]$ module load python3/3.8.8 cuda/11.2

[@coe-hpc1 ~]$ python3 -V

Python 3.8.8

[@coe-hpc1 ~]$ pip3 -V

pip 22.0.4 from /opt/ohpc/pub/apps/python3/3.8.8/lib/python3.8/site-packages/pip (python 3.8)

1. Create python virtual environment in your selected folder

[@coe-hpc1 instructor]$ pwd

/scratch/cmpe249-fa22/instructor

[@coe-hpc1 instructor]$ ls

venvpy38cu112

[010796032@coe-hpc1 instructor]$ python3 -m venv ./venvpy38cu112/

1. Activate the python virtual environment

[010796032@coe-hpc1 instructor]$ source ./venvpy38cu112/bin/activate

(venvpy38cu112) [010796032@coe-hpc1 instructor]$ which python

/scratch/cmpe249-fa22/instructor/venvpy38cu112/bin/python

1. Install Jupyterlab:

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$ pip install jupyterlab

Collecting jupyterlab

Downloading jupyterlab-3.4.3-py3-none-any.whl (8.8 MB)

* 1. Jupyter Notebook makes sure that the IPython kernel is available, but you have to manually add a kernel with a different version of Python or a virtual environment. First, you need to activate your virtual environment. Next, install ipykernel which provides the IPython kernel for Jupyter: pip install ipykernel
  2. Show existing Jupyter kernel list $ jupyter kernelspec list
  3. Next you can add your virtual environment to Jupyter by typing:
  4. Add a new kernel to the Jupyter lab: python -m ipykernel install --user --name=venvpy38cu112
     1. venvpy38cu112 is your environment name

## Install latest tensorflow

Install tensorflow ref: <https://www.tensorflow.org/install/pip>

(venvpy38cu112) [@coe-hpc1 instructor]$pip install tensorflow

Collecting tensorflow

Downloading tensorflow-2.9.1-cp38-cp38-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (511.7 MB)

You can now test tensorflow (make sure the virtual environment (venvpy37cu10) is still there)

$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2.9.1

GPU Available: False (note: there is no GPU in the head node)

$pip show tensorflow

Name: tensorflow

Version: 2.9.1

## Install Pytorch LTS

Install pytorch ref: <https://pytorch.org/get-started/locally/> (select LTS version 1.8.2, Linux, Python, Pip, Cuda11.1)

(venvpy38cu112) [@coe-hpc1 instructor]$ pip3 install torch==1.8.2 torchvision==0.9.2 torchaudio==0.8.2 --extra-index-url <https://download.pytorch.org/whl/lts/1.8/cu111>

(venvpy38cu112) [@coe-hpc1 instructor]$ python -c 'import torch; print(torch.\_\_version\_\_); print("GPU Available: ", torch.cuda.is\_available()); print("Torch cuda version:",torch.version.cuda)'

1.8.2+cu111

GPU Available: False

Torch cuda version: 11.1

(note: there is no GPU in the head node)

## Request GPU node

1. To request GPU node and get the interactive bash, we need to use srun in the host machine: $ srun -p gpu --gres=gpu --pty /bin/bash
   1. If you see srun: job 26773 queued and waiting for resources, that means there is no available GPUs for you to use in HPC, you need to wait until you see: srun: job 26773 has been allocated resources. You will be automatically log into the allocated GPU
   2. (Do not use this if you are not running a long training job) If you want a longer GPU time (not timeout), you can request one GPU node with the following command: srun -p gpu --time=47:55:00 --gres=gpu --pty /bin/bash

( --time is the specify time limit for the job. However, the maximum GPU time is around 48 hours based on the HPC design)

1. You should already in the GPU node: (venvpy38cu112) [@g7 instructor]$

Note: g7 is your GPU node name. You do not have internet access in the GPU node, you cannot install anything in your GPU node due to lack of internet connection. All your installation should be in your head node.

1. You can check the cuda version:

(venvpy38cu112) [@g7 instructor]$ module load python3/3.8.8 cuda/11.2

(venvpy38cu112) [@g7 instructor]$ which python

/scratch/cmpe249-fa22/instructor/venvpy38cu112/bin/python

(venvpy38cu112) [@g7 instructor]$ nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2021 NVIDIA Corporation

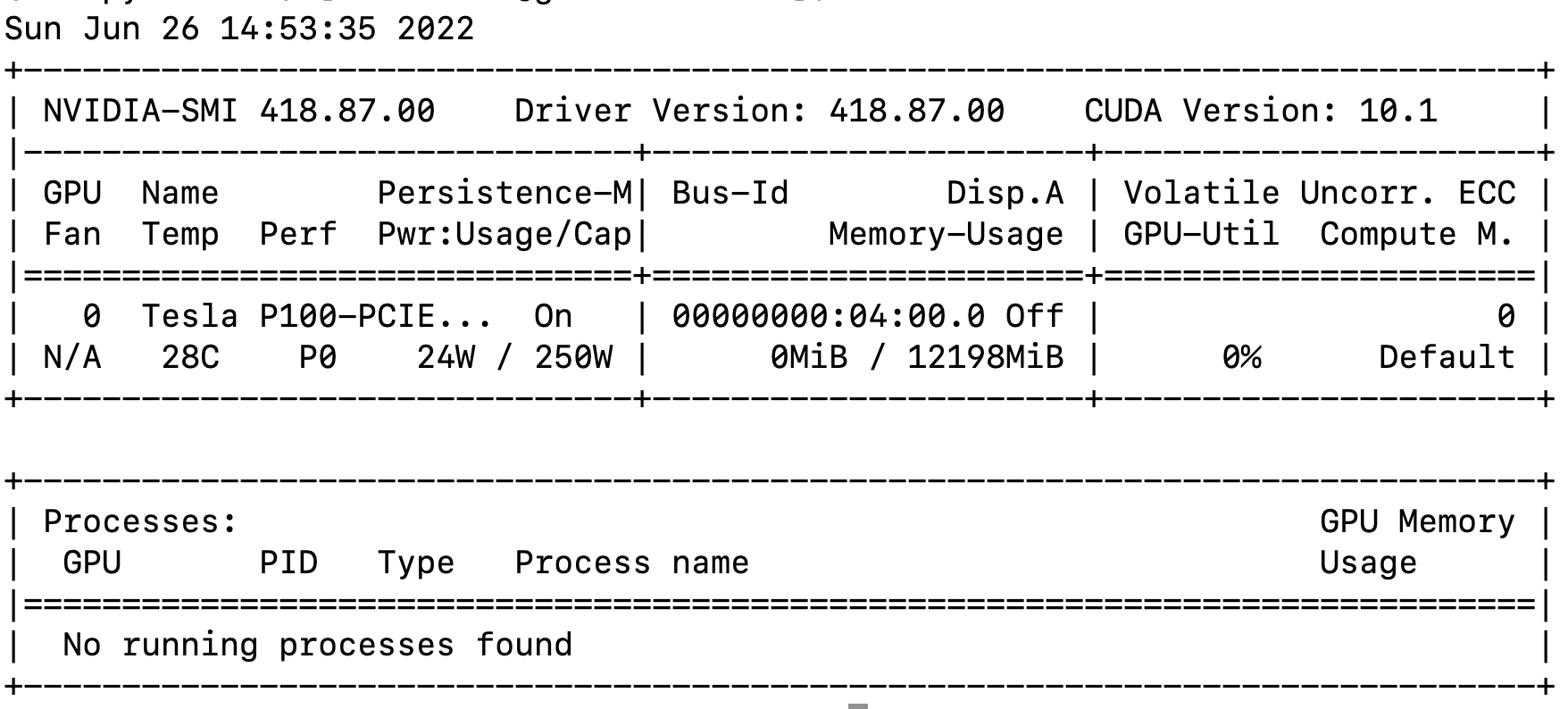
Built on Sun\_Feb\_14\_21:12:58\_PST\_2021

Cuda compilation tools, release 11.2, V11.2.152

Build cuda\_11.2.r11.2/compiler.29618528\_0

1. You can check the GPU information:

(venvpy38cu112) [010796032@g7 instructor]$ nvidia-smi



1. Check PATH and LD\_LIBRARY\_PATH, make sure cuda-11.2 is in the path

(venvpy38cu112) [@g9 deviceQuery]$ echo $PATH

/usr/local/cuda-11.2/samples-bin:/usr/local/cuda-11.2/bin:/usr/local/cuda-11.2/NsightCompute-1.0:/opt/ohpc/pub/apps/python3/3.8.8/bin:/scratch/cmpe249-fa22/instructor/venvpy38cu112/bin:/…..

(venvpy38cu112) [@g9 deviceQuery]$ echo $LD\_LIBRARY\_PATH

/usr/lib64/nvidia:/usr/local/cuda-11.2/lib64:/opt/ohpc/pub/apps/python3/3.8.8/lib:.....

1. Install Cudnn8.1.1

(venvpy38cu112) [@g9 instructor]$ tar -zvxf cudnn-11.2-linux-x64-v8.1.1.33.tgz (created one cuda folder in local directory)

(venvpy38cu112) [@g9 instructor]$ export LD\_LIBRARY\_PATH=/scratch/cmpe249-fa22/instructor/cuda/lib64:$LD\_LIBRARY\_PATH

Error: RuntimeError: cudaGetDevice() failed. Status: initialization error

1. Run CUDA sample: copy the samples folder from /usr/local/cuda-11.2 to one local directory

(venvpy38cu112) [@g9 samples]$ cd 1\_Utilities/deviceQuery

(venvpy38cu112) [@g9 deviceQuery]$ make clean

(venvpy38cu112) [@g9 deviceQuery]$ make

(venvpy38cu112) [@g9 deviceQuery]$ ./deviceQuery

./deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

cudaGetDeviceCount returned 3

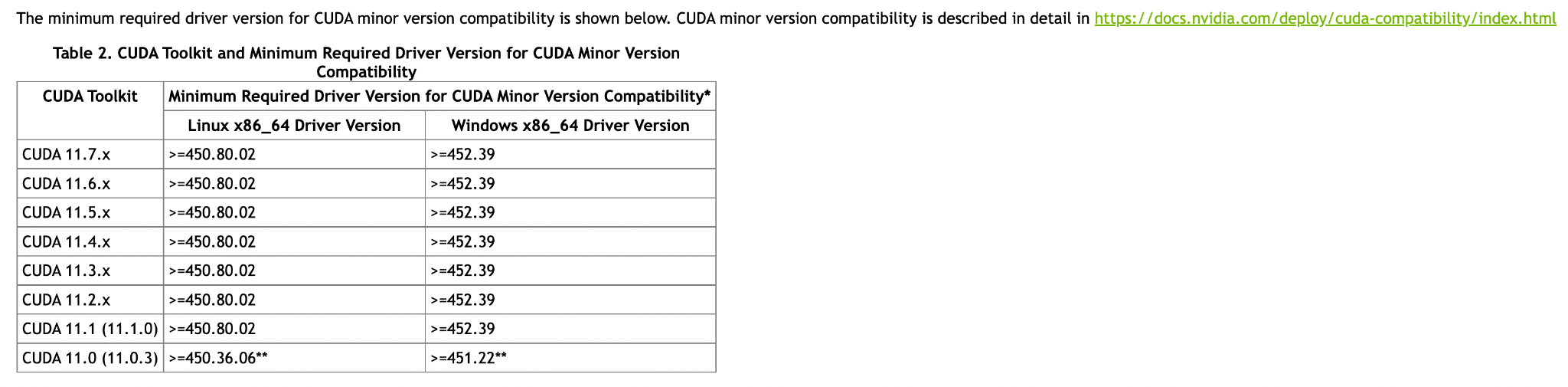
-> initialization error

Result = FAIL

**Problem reason?:**

**https://docs.nvidia.com/deploy/cuda-compatibility/index.html**

<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>

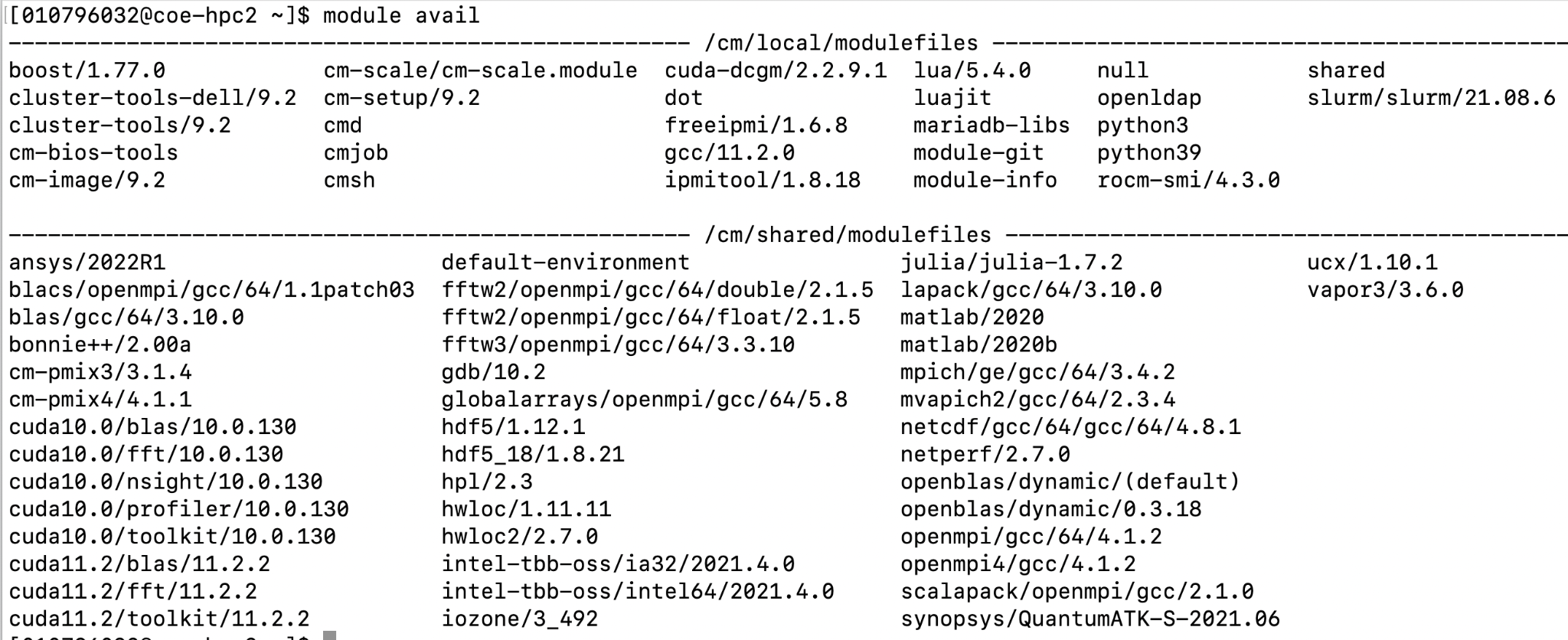


## COE HPC2 Server

[@coe-hpc1 ~]$ ssh coe-hpc2

Last login: Mon Jun 27 14:53:30 2022 from 172.16.1.1

Welcome to Bright release 9.2 Based on CentOS Linux 7



[@coe-hpc2 ~]$ module add python39

[@coe-hpc2 ~]$ python3 -m venv ./venvpy39cu112/

[@coe-hpc2 ~]$ source ./venvpy39cu112/bin/activate

(venvpy39cu112) [@coe-hpc2 ~]$ pip install jupyterlab

(venvpy39cu112) [@coe-hpc2 ~]$ pip install matplotlib

(venvpy39cu112) [@coe-hpc2 ~]$ pip install configargparse

(venvpy39cu112) [@coe-hpc2 ~]$ pip install tensorflow

Collecting tensorflow

Downloading tensorflow-2.9.1-cp39-cp39-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (511.7 MB)

(venvpy39cu112) [010796032@coe-hpc2 ~]$ pip3 install torch==1.8.2 torchvision==0.9.2 torchaudio==0.8.2 --extra-index-url https://download.pytorch.org/whl/lts/1.8/cu111

Looking in indexes: https://pypi.org/simple, https://download.pytorch.org/whl/lts/1.8/cu111

Collecting torch==1.8.2

[010796032@coe-hpc2 ~]$ module load slurm/slurm/21.08.6

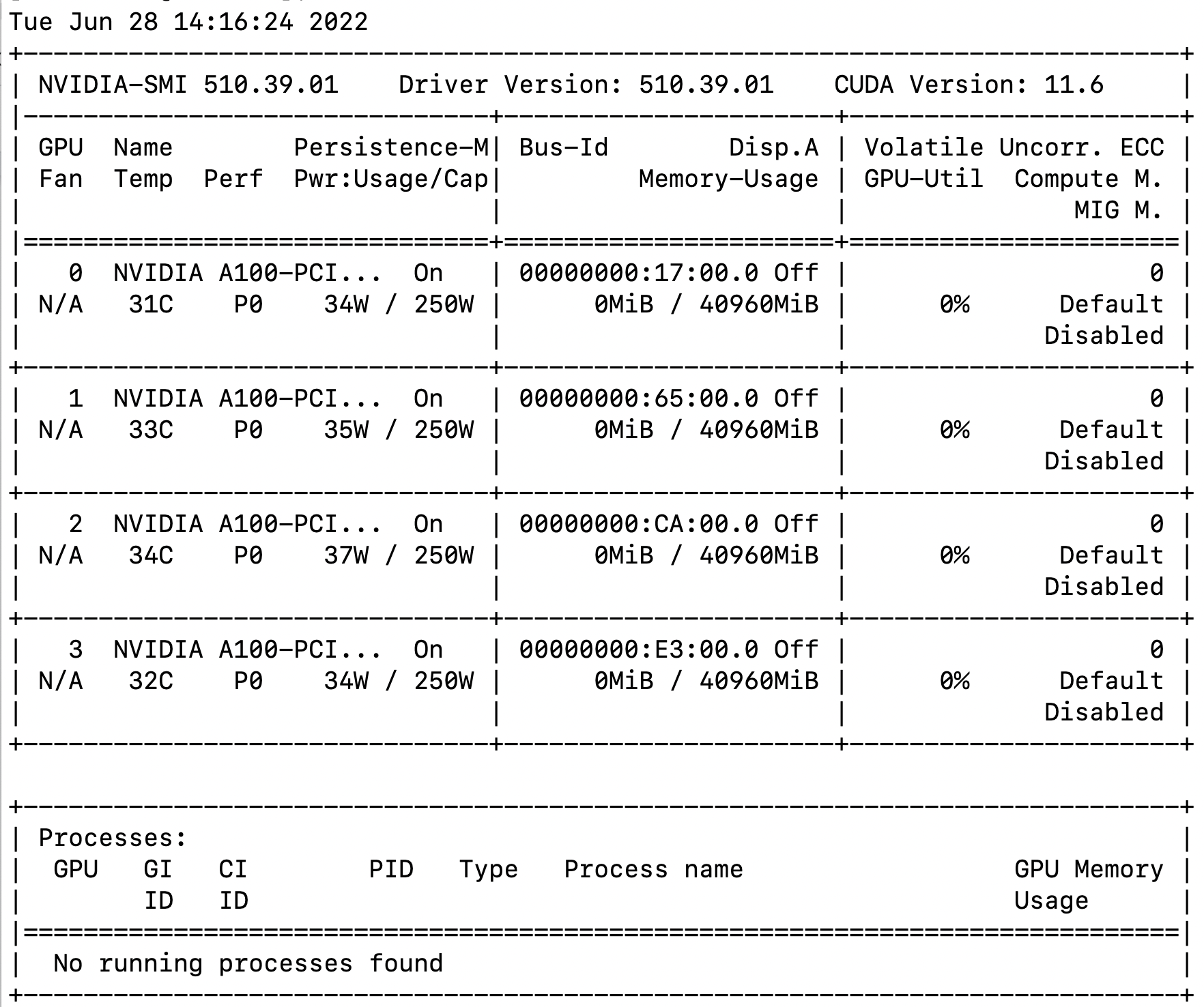
[010796032@coe-hpc2 ~]$ srun -p gpu --gres=gpu --pty /bin/bash

srun: error: Unable to allocate resources: Requested node configuration is not available

srun: Force Terminated job 263

[010796032@coe-hpc2 ~]$ srun --pty /bin/bash

[010796032@cs002 ~]$ nvidia-smi



[010796032@cs002 ~]$ module add cuda11.2/toolkit/11.2.2

[010796032@cs002 ~]$ nvcc -V

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2021 NVIDIA Corporation

Built on Sun\_Feb\_14\_21:12:58\_PST\_2021

Cuda compilation tools, release 11.2, V11.2.152

Build cuda\_11.2.r11.2/compiler.29618528\_0

[010796032@cs002 ~]$ module add python39

[010796032@cs002 ~]$ python3 -V

Python 3.9.10

[010796032@cs002 ~]$ python3 -m venv ./venvpy39cu112/

[010796032@cs002 ~]$ source ./venvpy39cu112/bin/activate

(venvpy39cu112) [010796032@cs002 ~]$ which python

/lts/home/010796032/venvpy39cu112/bin/python

(venvpy39cu112) [010796032@cs001 ~]$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2022-06-28 15:35:15.838647: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

2.9.1

WARNING:tensorflow:From <string>:1: is\_gpu\_available (from tensorflow.python.framework.test\_util) is deprecated and will be removed in a future version.

Instructions for updating:

Use `tf.config.list\_physical\_devices('GPU')` instead.

2022-06-28 15:35:21.700075: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 AVX512F AVX512\_VNNI FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-06-28 15:35:22.274435: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open shared object file: No such file or directory; LD\_LIBRARY\_PATH: /cm/local/apps/cuda/libs/current/lib64:/cm/shared/apps/cuda11.2/toolkit/11.2.2/targets/x86\_64-linux/lib:/cm/shared/apps/slurm/current/lib64/slurm:/cm/shared/apps/slurm/current/lib64::/home/010796032:/home/010796032

2022-06-28 15:35:22.274476: W tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

GPU Available: False

(venvpy39cu112) [010796032@cs001 ~]$ python -c 'import torch; print(torch.\_\_version\_\_); print("GPU Available: ", torch.cuda.is\_available()); print("Torch cuda version:",torch.version.cuda)'

1.8.2+cu111

GPU Available: True

Torch cuda version: 11.1

Cuda sample location:

[010796032@cs001 11.2.2]$ ls

0\_Simple 2\_Graphics 4\_Finance 6\_Advanced bin EULA.txt Makefile verify\_cuda11.2.sh

1\_Utilities 3\_Imaging 5\_Simulations 7\_CUDALibraries common LICENSE opencl verify\_opencl.sh

[010796032@cs001 11.2.2]$ pwd

/cm/shared/apps/cuda11.2/sdk/11.2.2

[010796032@cs001 11.2.2]$ ./verify\_cuda11.2.sh

[010796032@cs001 11.2.2]$ cd /tmp/cuda11.2/1\_Utilities/deviceQuery/

[010796032@cs001 deviceQuery]$ ls

deviceQuery deviceQuery.cpp deviceQuery.o Makefile NsightEclipse.xml readme.txt

[010796032@cs001 deviceQuery]$ ./deviceQuery

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.6, CUDA Runtime Version = 11.2, NumDevs = 4

Result = PASS

Add CUDNN:

[010796032@cs001 cuda]$ pwd

/home/010796032/mycuda/cuda

[010796032@cs001 cuda]$ export LD\_LIBRARY\_PATH=/home/010796032/mycuda/cuda/lib64:$LD\_LIBRARY\_PATH

[010796032@cs001 cuda]$ echo $LD\_LIBRARY\_PATH

/home/010796032/mycuda/cuda/lib64:/cm/local/apps/cuda/libs/current/lib64:/cm/shared/apps/cuda11.2/toolkit/11.2.2/targets/x86\_64-linux/lib:/cm/shared/apps/slurm/current/lib64/slurm:/cm/shared/apps/slurm/current/lib64::/home/010796032:/home/010796032

[010796032@cs001 ~]$ module add python39

[010796032@cs001 ~]$ source ./venvpy39cu112/bin/activate

(venvpy39cu112) [010796032@cs001 ~]$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2022-06-29 13:19:53.968220: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

2.9.1

WARNING:tensorflow:From <string>:1: is\_gpu\_available (from tensorflow.python.framework.test\_util) is deprecated and will be removed in a future version.

Instructions for updating:

Use `tf.config.list\_physical\_devices('GPU')` instead.

2022-06-29 13:19:56.084802: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 AVX512F AVX512\_VNNI FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-06-29 13:20:00.174870: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1532] Created device /device:GPU:0 with 38418 MB memory: -> device: 0, name: NVIDIA A100-PCIE-40GB, pci bus id: 0000:17:00.0, compute capability: 8.0

2022-06-29 13:20:00.175502: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1532] Created device /device:GPU:1 with 38418 MB memory: -> device: 1, name: NVIDIA A100-PCIE-40GB, pci bus id: 0000:65:00.0, compute capability: 8.0

2022-06-29 13:20:00.176014: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1532] Created device /device:GPU:2 with 38418 MB memory: -> device: 2, name: NVIDIA A100-PCIE-40GB, pci bus id: 0000:ca:00.0, compute capability: 8.0

2022-06-29 13:20:00.176486: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1532] Created device /device:GPU:3 with 38418 MB memory: -> device: 3, name: NVIDIA A100-PCIE-40GB, pci bus id: 0000:e3:00.0, compute capability: 8.0

GPU Available: True

​​[010796032@cs002 data]$ cd cmpe249-fa22/

bash: cd: cmpe249-fa22/: Permission denied

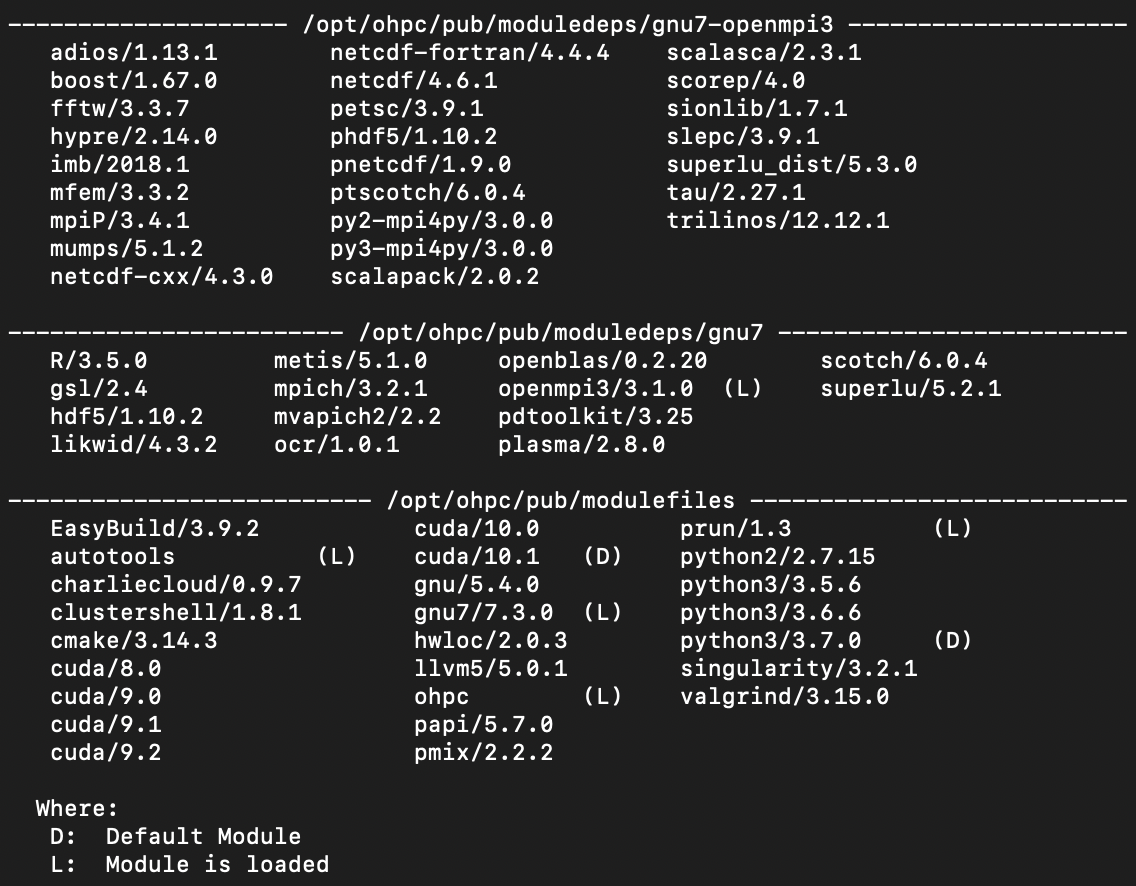
[010796032@cs002 data]$ cd cmpe295-liu

[010796032@cs002 cmpe295-liu]$

# Software Installation based on Cuda10.1 (working)

## Tensorflow2 installation

1. Check available software modules via [@coe-hpc1 ~]$ module avail



1. Load the python3 and cuda10.1 module.

First check current python version

[@coe-hpc1 ~]$ python -V

Python 2.7.5

[@coe-hpc1 ~]$ module load python3/3.7.0 cuda/10.1

[@coe-hpc1 ~]$ python -V

Python 3.7.0

1. Create python virtual environment in your home directory of the host machine

[0107xxxxx@coe-hpc1 ~]$ virtualenv --system-site-packages -p python3.7 ./venvpy37cu10

Running virtualenv with interpreter /opt/ohpc/pub/apps/python3/3.7.0/bin/python3.7

Using base prefix '/opt/ohpc/pub/apps/python3/3.7.0'

New python executable in /home/0107xxxxx/venvpy37cu10/bin/python3.7

Also creating executable in /home/0107xxxxx/venvpy37cu10/bin/python

Installing setuptools, pip, wheel...

Done.

(If the virtualenv is not available, you can also use *python3 -m venv ./xxx* to create the virtual environment)

1. Activate the python virtual environment

$ source ./venvpy37cu10/bin/activate

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$

1. Install Jupyterlab: (venvpy37cu10) [sjsuid@coe-hpc1 ~]$ pip install jupyterlab
   1. Jupyter Notebook makes sure that the IPython kernel is available, but you have to manually add a kernel with a different version of Python or a virtual environment. First, you need to activate your virtual environment. Next, install ipykernel which provides the IPython kernel for Jupyter: pip install --user ipykernel
   2. Show existing Jupyter kernel list $ jupyter kernelspec list
   3. Next you can add your virtual environment to Jupyter by typing:
   4. Add a new kernel to the Jupyter lab: python -m ipykernel install --user --name=venvpy37cu10
      1. venvpy37cu10 is your environment name
2. Install latest tensorflow, ref: https://www.tensorflow.org/install/pip

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$ pip install tensorflow==2.3.0

Note: TensorFlow supports CUDA® 11 (TensorFlow >= 2.4.0). You can choose to install tensorflow 2.3.0, which supports cuda10.1. You can now test tensorflow (make sure the virtual environment (venvpy37cu10) is still there)

$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2.3.0

GPU Available: False (note: there is no GPU in the head node)

$pip show tensorflow

Name: tensorflow

Version: 2.3.0

1. To request GPU node and get the interactive bash, we need to use srun in the host machine: $ srun -p gpu --gres=gpu --pty /bin/bash
   1. If you see srun: job 26773 queued and waiting for resources, that means there is no available GPUs for you to use in HPC, you need to wait until you see: srun: job 26773 has been allocated resources. You will be automatically log into the allocated GPU
   2. (Do not use this if you are not running a long training job) If you want a longer GPU time (not timeout), you can request one GPU node with the following command: srun -p gpu --time=47:55:00 --gres=gpu --pty /bin/bash

( --time is the specify time limit for the job. However, the maximum GPU time is around 48 hours based on the HPC design)

1. You should already in the GPU node: (venv-3.6.6-gpu) [sjsuid@g1 ~]$

Note: g1 is your GPU node name. You do not have internet access in GPU node, you cannot install anything in your GPU node due to lack of internet connection. All your installation should be in your head node.

1. You can check the cuda version:

(venvpy37cu10) [sjsuid@g1 ~]$ nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2019 NVIDIA Corporation

Built on Sun\_Jul\_28\_19:07:16\_PDT\_2019

Cuda compilation tools, release 10.1, V10.1.243

1. You can check the GPU information:

(venvpy37cu10) [sjsuid@g1 ~]$ nvidia-smi



1. You can now test tensorflow (make sure the virtual environment (venvpy37cu10) is still there)

(venvpy37cu10) [sjsuid@g1 ~]$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2.3.0

Could not load dynamic library 'libcublas.so.10'; dlerror: libcublas.so.10: cannot open shared object file: No such file or directory; LD\_LIBRARY\_PATH: /usr/lib64/nvidia:/usr/local/cuda-10.1/lib64:/opt/ohpc/pub/apps/python3/3.6.6/lib:/opt/ohpc/pub/mpi/openmpi3-gnu7/3.1.0/lib:/opt/ohpc/pub/compiler/gcc/7.3.0/lib64

GPU Available: False (You will still see the GPU is not available if you are using the latest Tensorflow 2.3 and cuda10.1, GPU is available if you are using tensorflow1.x or tensorflow2.0 with cuda10.0)

1. To solve the GPU library problem, copy the libcublas.10.0 library to your home directory (replace the *SJSUID with your own SJSU ID*): (venvpy37cu10)[*SJSUID*@g1 ~]$ cp /usr/local/cuda-10.0/targets/x86\_64-linux/lib/libcublas.so.10.0 /home/*SJSUID*
2. Rename it to libcublas.so.10, since the system is searching for that library: $ mv libcublas.so.10.0 libcublas.so.10
3. Then, create a softlink with the location of your home directory, since libcublas.so.10 is present here locally: $ export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/*SJSUID. You can add this one to the end of bashrc file: vim ~/.bashrc*
4. You can now test tensorflow: $ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

2020-08-19 11:03:02.996945: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:48] Successfully opened dynamic library libcudart.so.10.1

2.3.0

Created TensorFlow device (/device:GPU:0 with 11113 MB memory) -> physical GPU (device: 0, name: Tesla P100-PCIE-12GB, pci bus id: 0000:03:00.0, compute capability: 6.0)

GPU Available: True

1. To exit, enter 'deactivate' to exit the python virtual environment, 'exit' to exit the GPU node. If you need to open the Jupyter notebook, you need to keep the current terminal open, otherwise, the GPU node will lost.

## Enable the Jupyter notebook via ssh tunnel

1. The GPU node does not have Internet access, to enable the Jupyter notebook access from your local computer, we need to setup ssh tunnel to the HPC head node, then create a second tunnel from the head node to the GPU node.
2. Open a new terminal in your local machine. (keep another terminal to the GPU node open, otherwise the GPU node will lost) ref: <http://coe-hpc-web.sjsu.edu>
3. Create a ssh tunnel to the HPC host machine, ‘10001’ is your own defined port number, you can select any unused port from 10000 and 63999

$ ssh -L 10001:localhost:10001 [SJSUID]@coe-hpc.sjsu.edu

1. Create a ssh tunnel from the HPC host machine to the GPU node, where ‘10001’ is your previous defined port number, ‘@g1’ is your GPU node id (you can get it from your previous terminal)

[@coe-hpc1 ~]$ ssh -L 10001:localhost:10001 [SJSUID]@g1

1. After you logged into the GPU node, you should load the python3 and cuda9 modules

[@g1 ~]$ module load python3/3.7.0 cuda/10.1

[@g1 ~]$ python -V

Python 3.7.0

1. Activate the python virtual environment:

[@g1 ~]$ source ./venvpy37cu10/bin/activate

1. You can test the tensorflow:

(venvpy37cu10) [sjsuid@g1 ~]$ python -c 'import tensorflow as tf; print(tf.\_\_version\_\_); print("GPU Available: ", tf.test.is\_gpu\_available())'

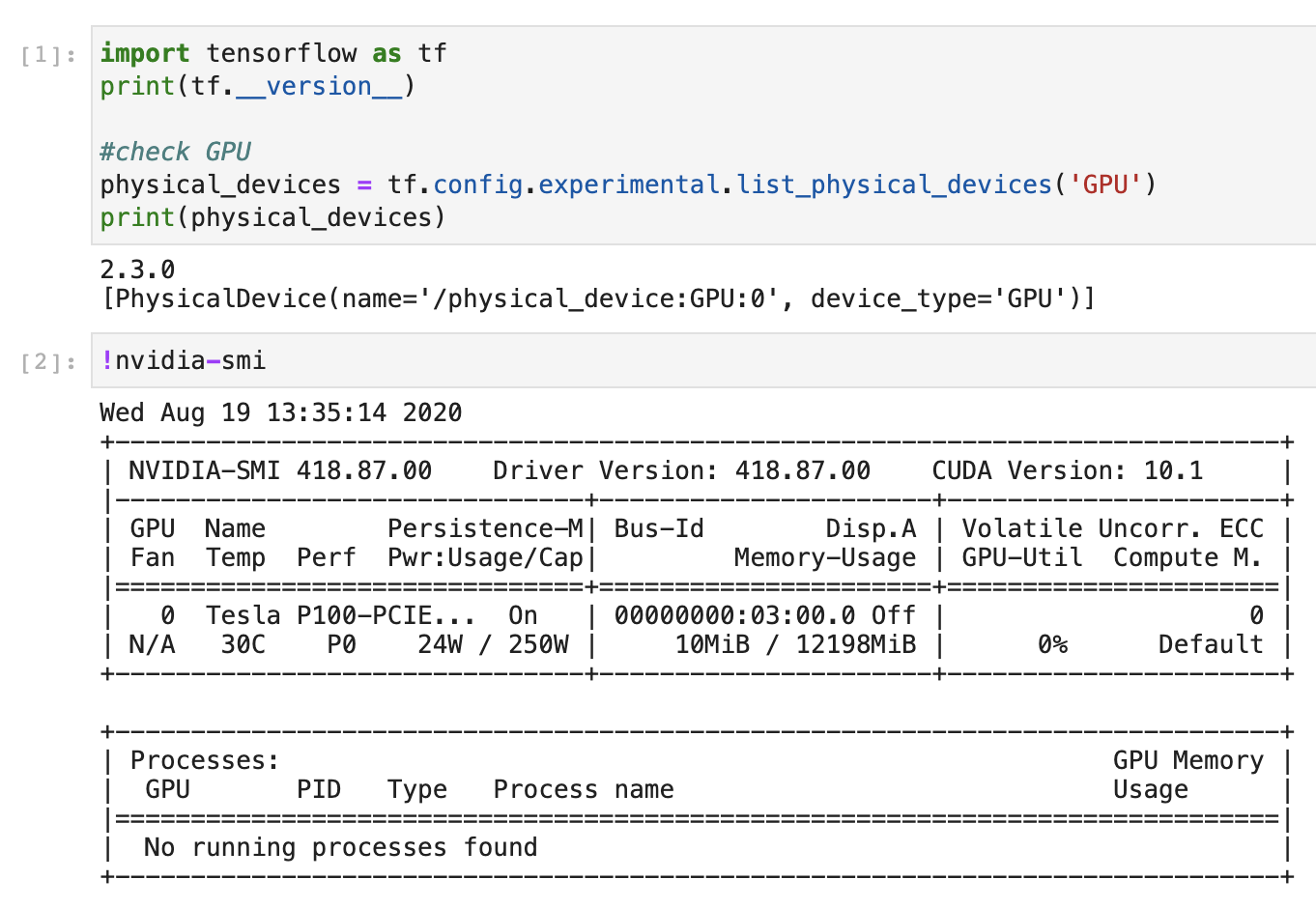
1. Start the jupyter notebook server

(venvpy37cu10) [@g1 ~]$ jupyter lab --no-browser --port=10001

to login with a token:

http://localhost:10001/?token=0beaf6bd5ab2eb0f1012e03d53ba2c17a2d36fc42e5de13a

1. You can test your GPU access in JupyterLab:



If you have your head node IP address, you also can use this command (in your host machine): To exit, enter Control+C to terminate the Jupyter notebook in the GPU node, 'deactivate' to exit the python virtual environment, 'exit' to exit the GPU node.

## Install Tensorflow 2.x Object Detection API

1. Install protoc in HPC (ref the Problems section)
2. Clone the Tensorflow models github: git clone --depth 1 https://github.com/tensorflow/models
3. Go to the models/research folder: cd /home/[SJSUID]/models/research/
4. Run protoc: protoc object\_detection/protos/\*.proto --python\_out=.
5. Perform python package setup:

cp object\_detection/packages/tf2/setup.py .

python -m pip install .

1. Do installation test: python ./object\_detection/builders/model\_builder\_tf2\_test.py
2. Perform Object Detection training:

(venvpy37cu10) [0107xxxxx@g1 object\_detection]$ python model\_main\_tf2\_lkk.py \

> --pipeline\_config\_path='/home/0107xxxxx/TF2Work/pipeline\_file.config' \

> --model\_dir='/home/0107xxxxx/TF2Work/ModelZoo/training/' \

> --alsologtostderr \

> --num\_train\_steps=60000 \

> --sample\_1\_of\_n\_eval\_examples=1 \

> --num\_eval\_steps=500

## Install Pytorch1.6

1. Ref the Pytorch website to install pytorch for cuda10.1: https://pytorch.org/
2. Install Pytorch 1.6 with cuda10.1 (do not upgrade the pytorch version to 1.7 or newer): pip install torch==1.6.0+cu101 torchvision==0.7.0+cu101 -f https://download.pytorch.org/whl/torch\_stable.html
3. Check torch version: pip list | grep torch

torch 1.6.0+cu101

torchvision 0.7.0+cu101

1. Check the cuda access of Pytorch:

import torch

print(torch.cuda.is\_available())

print(torch.cuda.device\_count())`

print(torch.cuda.get\_device\_name())

## Install Detectron2

Following the instructions in <https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md>, choose the right version of pytorch and cuda.

1. Upgrade pip: pip install --upgrade pip
2. Install pyyaml in the head node: (venvpy37cu10) [@coe-hpc1 ~]$ pip install pyyaml==5.1
3. (venvpy37cu10) [@coe-hpc1 ~]$ pip install pycocotools>=2.0.1
4. Install detectron2 for cuda10.1 and torch1.6:

(venvpy37cu10) [@coe-hpc1 ~]$ python -m pip install detectron2 -f \

https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.6/index.html

1. Test import: import detectron2

# Problems

## Python TK problems

* Tensorflow Object Detection API need the python-tk module, otherwise you will have error: ModuleNotFoundError: No module named '\_tkinter'
* If you have root access, you can simply solve this problem by sudo apt-get install python-tk, however, it is not possible in the HPC as we do not have root access
* The module that requires the tk is matplotlib, as it use the tk backend for drawing. If we did not install tk, we can change a different backend
* For any python file or the jupyter notebook that imports the matplotlib, change the backend of matplotlib to use the agg backend (this step should be done at the beginning of the code).

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

This will solve the problem of No module named '\_tkinter'

## Install “protoc” for Tensorflow Object Detection API

You can update the protoc version. The following example is for Tensorflow 1.x Object Detection. The protoc installation method is the same to Tensorflow 2.x Object Detection.

1. Activate the python virtual environment: [sjsuid@coe-hpc1 ~]$ source ./venvpy37cu10/bin/activate
2. (venvpy37cu10) [sjsuid@coe-hpc1 ~]$ git clone <https://github.com/tensorflow/models/>
3. (venvpy37cu10) [sjsuid@coe-hpc1 ~]$ wget <https://github.com/protocolbuffers/protobuf/releases/download/v3.9.2/protoc-3.9.2-linux-x86_64.zip> (TODO: update to the latest version 3.13)
4. (venvpy37cu10) [sjsuid@coe-hpc1 protoc]$ unzip protoc-3.9.2-linux-x86\_64.zip
5. /home/sjsuid/protoc/protoc
6. (venvpy37cu10) [sjsuid@coe-hpc1 bin]$ ./protoc --version

libprotoc 3.9.2

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$ ./protoc/bin/protoc --version

libprotoc 3.9.2

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$ export PATH=$PATH:/home/[sjsuid]/protoc/bin/

(venvpy37cu10) [sjsuid@coe-hpc1 ~]$ protoc --version

libprotoc 3.9.2

1. (venvpy37cu10) [sjsuid@coe-hpc1 ~]$ git clone <https://github.com/cocodataset/cocoapi.git>

$ cd cocoapi/PythonAPI

(venvpy37cu10) [sjsuid@coe-hpc1 PythonAPI]$ make

(venvpy37cu10) [sjsuid@coe-hpc1 PythonAPI]$ cp -r pycocotools /home/[SJSUID]/models/research/

(venvpy37cu10) [sjsuid@coe-hpc1 research]$ pwd

/home/[SJSUID]/models/research

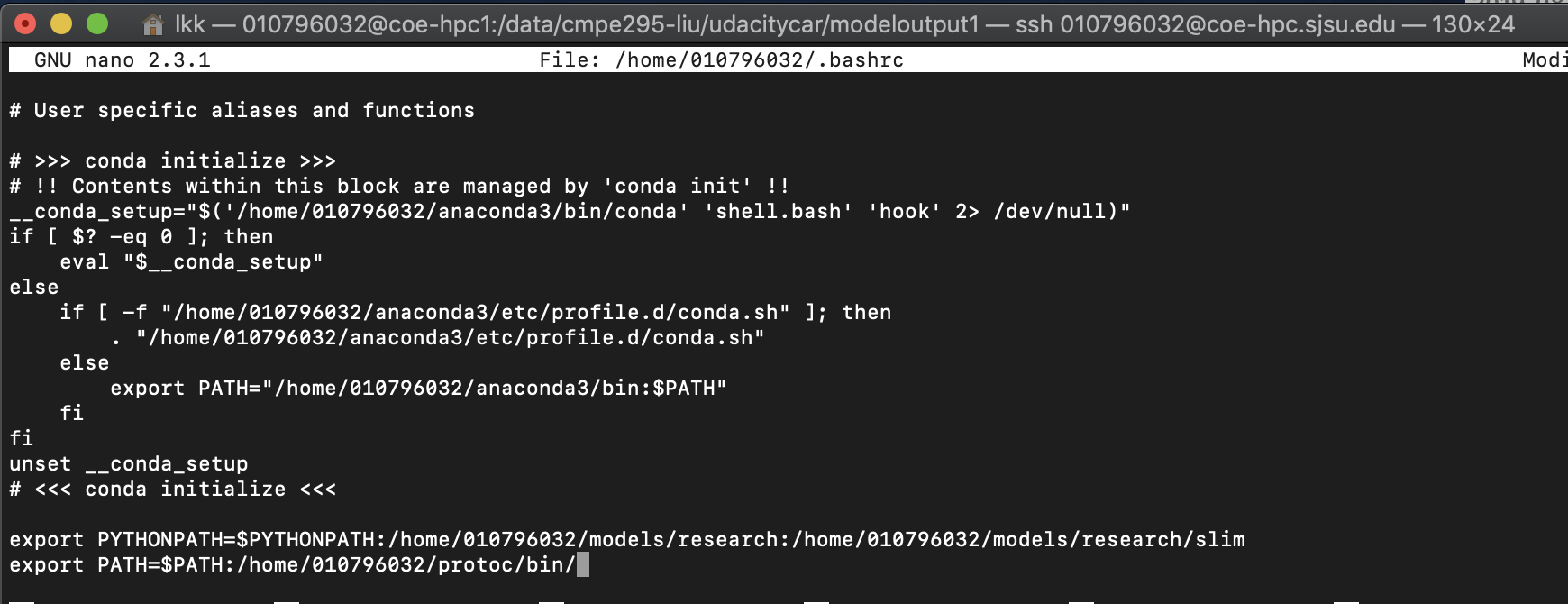
(venvpy37cu10) [sjsuid@coe-hpc1 research]$ protoc object\_detection/protos/\*.proto --python\_out=.

(venvpy37cu10) [sjsuid@coe-hpc1 research]$ export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

(venvpy37cu10) [sjsuid@coe-hpc1 research]$ echo $PYTHONPATH

:/home/sjsuid/models/research:/home/[SJSUID]/models/research/slim

1. We can add the protoc path and PYTHONPATH into bashrc



$ source ~/.bashrc

1. We can run the Object Detection model builder test

(venvpy37cu10) [SJSUID@coe-hpc1 research]$ python setup.py build

(venvpy37cu10) [SJSUID@coe-hpc1 research]$ python setup.py install

(venvpy37cu10) [SJSUID@coe-hpc1 research]$ python object\_detection/builders/model\_builder\_test.py

Ran 16 tests in 0.126s

OK (skipped=1)

MY Commands:

Node access:

ssh 015216357@coe-hpc1.sjsu.edu

module load python3/3.7.0 cuda/10.1

source ./venvpy37cu10/bin/activate

srun -p gpu --gres=gpu --pty /bin/bash

Jupyter access:

ssh -L 10001:localhost:10001 015216357@coe-hpc.sjsu.edu

ssh -L 10001:localhost:10001 015216357@g1

module load python3/3.7.0 cuda/10.1

source ./venvpy37cu10/bin/activate

jupyter lab --no-browser --port=10001