Getting Started with CHPC

This document aims to get you started with the compute environment in the university's Center for High Performance Computing (CHPC). From their "About CHPC" page:

> CHPC also serves as an expert team to broadly support the diverse research computing needs on campus. These needs include support for big data, big data movement, data analytics, security, virtual machines, Windows science application servers, protected environments for data mining and analysis of protected health information, and advanced networking.

Most large models in academia and the industry are built using compute facilities that resemble CHPC, and not in notebook environments. So getting familiar with the cluster and running jobs managed by slurm could be a useful skill on its own. Furthermore, we ask that you use this resource because other resources like Google's Colab do not provide you with *consistent* GPU access.

Preparation: Python engineering fundamentals (CHPC, VSCode, Conda, Git)

We assume no prior knowledge with CHPC, VSCode, Conda or Git, but if you are already using these resources, feel free to skip this part. It is also acceptable that you have a different workflow as long as you are able to queue a python job with Slurm.

Creating an account on CHPC

Make a CHPC account if you don't have one already by following the steps here: Accounts - Center for High Performance Computing - The University of Utah

- ☐ You will need this information:
  - Class: `CS6957`
  - Token: `ebfd6fa04ef7b63a`
  - Semester: Fall2023
  - Expiration Date: 2023-12-10
  - Link: https://www.chpc.utah.edu/role/user/student.php?class=CS6957

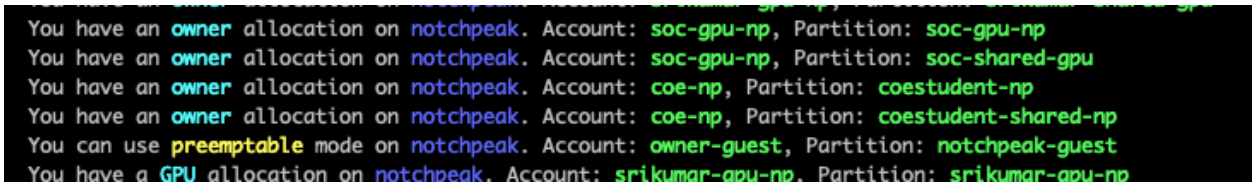
Accessing CHPC

After you did whatever step applies to you above, SSH to `notchpeak2` server by running in a terminal:

```
ssh <your uNID>@notchpeak2.chpc.utah.edu
```

You will be prompted to enter the password associated with your uNID.

Run `myallocation` and check whether you see `soc-gpu-np` (see figure below). You will not have the exact same allocations like me! But you should see `soc-gpu-np`. If not reach out to me or the TA as soon as possible.



If you want to access CHPC from home you need to set up VPN (if you didn't already):

- ☐ KSoC students, our VPN will work: Palo Alto VPN Setup – Kahlert School of Computing – IT Support
- ☐ Other students, see Virtual Private Network (vpn) - Center for High Performance Computing - The University of Utah

Directories

Models and model checkpoints can be huge so we shouldn't save them to our local directories on CHPC. Luckily, CHPC has scratch file systems! Make a directory in one of them: `mkdir /scratch/general/vast/<uNID>` and save your checkpoints and data there. Note that: "On these scratch file system, files that have not been accessed for 60 days are automatically scrubbed."

Editors

We do not require that you use any editor. But many people find VSCode to be very helpful. If you haven't used VSCode before, read about it here: Harvard CS197 Lecture 2 Notes.

- ☐ Download and install VSCode.
- ☐ If needed, familiarize yourself with VSCode Terminal: Integrated Terminal in Visual Studio Code and if not familiar with the shell, scripting, and command-line environment check corresponding lectures here: The Missing Semester of Your CS Education.

Python environment

There are many ways to set up and manage python versions. If you already have one that you know and use, feel free to skip this section, which gets you started with Conda.
If needed read about Conda here: Harvard CS197 Lecture 2 Notes, then download and install Miniconda following: Miniconda — conda documentation.

- [ ] On your browser, right-click on a suitable installer and select “Copy Link Address”, then `wget` that link and installed the downloaded bash file. E.g,. on Linux 64-bit and python 3.8:

```
wget
https://repo.anaconda.com/miniconda/Miniconda3-py38_23.5.2-0-Li
nux-x86_64.sh

bash Miniconda3-py38_23.5.2-0-Linux-x86_64.sh
```

- [ ] Accept the license.
- [ ] Install `miniconda` into the default location
- [ ] Initialize
- [ ] Exit.
- [ ] Ssh again.

Create and activate a test Conda environment with a desired python version, e.g., 3.8:

```
conda create -n <env_name> python=3.8
conda activate <env_name>
```

You can install a package using pip (`pip install`) and see installed packages with `pip freeze`. More about that later.

Getting set up with GPUs

To use NVIDIA’s GPUs, we need cuda drivers. Luckily for us, CHPC folks installed for use we just need to do a few easy steps:

- [ ] You can run `which nvcc` and if cuda drivers are loaded, you'll see something like:
  `/uufs/chpc.utah.edu/sys/spack/linux-rocky8-nehalem/gcc-8.5.0/cuda-11.6.2-hgkn7czv7ciyy3gtpazwk2s72msbw6l2/bin/nvcc`
  But most likely you won’t see this and you will need to load them by running:

```
module spider cuda # which cuda drivers are available through
the module list
module load cuda/11.8.0 #load cuda 11.8
module list # run to see that cuda is loaded
```

Other niceties

I highly recommend using tmux or screen. These programs allows you to get back to your experiments once you close your laptop or SSH connection breaks, and to “babysit” multiple

experiments simulatenously. Here are all commands you need: Tmux Cheat Sheet. Try this "excercise":

```
tmux new -s <session name> # make a new session
tmux a -t <session name> # attach to the sessions
ls # or any command that will output something
```

Press `control+b d` on your keyboard to detach. Now run:

```
tmux ls # see which tmux sessions you have created
tmux a -t <session name> # attach again and you will see the
outputs from before
```

Press `control+b d` again on your keyboard to detach.

We highly recommend using Git. If needed, read about the common workflow here: Harvard CS197 Lecture 2 Notes or if you want to know more check this: Version Control (Git).

Working with slurm

- [ ] Here, I am assuming that you have a python file (or a set of them) that contains the code you wish to run. Suppose the entrypoint to your code is in `main.py`.
  Since it is better to store your models and big files in the scratch space on CHPC (which you created above), make sure that you pass the path to your scratch directory to the program as a command line argument. One way to do this is to use argparse in `main.py` as below:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--output_dir', type=str, help='Directory
where model checkpoints will be saved')
args = parser.parse_args()

output_dir = args.output_dir

# your code which can then use output_dir
```

  I recommend that you do not set any variable values in your code but instead add parser argument for that variable with a default option. This will enable you to run multiple runs of your code with different variables without needing to make multiple copies of your python file.

- [ ] Familiarize yourself with submitting a script to Slurm by reading this: SLURM Scheduler and then make a bash file (e.g. `run.sh`) with these contents and add information that is incomplete:

```
#!/bin/bash
#SBATCH --account soc-gpu-np
#SBATCH --partition soc-gpu-np
#SBATCH --ntasks-per-node=32
#SBATCH --nodes=1
#SBATCH --gres=gpu:3090:1
#SBATCH --time=2:00:00
#SBATCH --mem=24GB
#SBATCH --mail-user=<your email>
#SBATCH --mail-type=FAIL,END
#SBATCH -o assignment_1-%j
#SBATCH --export=ALL

source ~/miniconda3/etc/profile.d/conda.sh
conda activate <your conda environment>

OUT_DIR=/scratch/general/vast/<your uNID>/cs6957/assignment1/models
mkdir -p ${OUT_DIR}
python main.py --output_dir ${OUT_DIR} <other arguments>
```

  In the first three highlighted lines, you are requesting for a GPU node that has a certain amount of memory for a certain amount of time. If you ask for a very heavily used node for a very long period of time, you may end up waiting in the queue. Familiarize yourself with the GPU nodes available in CHPC and how to request for them here.

- [ ] Make a .txt file name `requirements.txt` with the contents below. You might need to revise based on what libraries you actually use:

```
torch --index-url https://download.pytorch.org/whl/cu118
accelerate
tqdm
pandas
```

- [ ] Make a private github repository, add your python, bash, and text files there, commit/push.
- [ ] Ssh to `notchpeak2`.
- [ ] Clone your github repository.
- [ ] Make a conda environment, activate it, `cd` into your repository, and run `pip install -r requirements.txt`

- [ ] You are ready to run your job with `sbatch run.sh`. Consider making a `tmux` session for this.
- [ ] Track the progress by running `tail -f assignment_1-<job id>` after you check the job id by checking the outputs of `squeue -u <your uNID>`
    - [ ] Most likely you will get errors. If so, you will revise your code in VSCode, push changes to your git repo, pull them on the server, and run the job again.
    - [ ] The output of `squeue -u <your uNID>` also shows on which node your job is running. In another tmux session, you can ssh to that node (`ssh <your uNID>@<node>`) and run:

      ```
      ml nvtop
      nvtop
      ```

      To track how much GPU memory are you using. This can be helpful to see how you can maximize the batch size without running into the CUDA memory errors. Tip: Using `gradient_accumulation_steps` with a lower batch size approximates a larger batch size.