portfolio_5

2023-02-23

High Performance Computing (HPC)

In this document we will be going over some of the basics of how to use bristols HPC's, it will serve as a reference guide for me later on. At the end I also provide a job script for running in parallel multiple python jobs which I used to obtain the encoded version of a dataset for a long list of encoders, saving me lots of time!

Logging in to bristol HPC

First lets specify the login details for both the HPC's avaliable at Bristol. For BlueCrystal Phase 4:

- Username: Your UoB username
- Password: Your UoB password
- Hostname: bc4login.acrc.bris.ac.uk

For BluePebble:

- Username: Your UoB username
- Password: Your UoB password
- Hostname: bp1-login.acrc.bris.ac.uk

We use **ssh** to log in. For BlueCrystal Phase 4 we use: **ssh -X @bc4login.acrc.bris.ac.uk** , and for BluePebble: **ssh -X @bp1-login.acrc.bris.ac.uk**.

#TODO: workflow, queing system

Software

To check what software is available on the HPC you can type **module avail** which will list all the software available. If you want to search for a certain piece of software you can use **module spider** . If software you need isn't available you can either try compiling from source, pip installing (or whatever equivalent for the language you are using) or you can contact the HPC team to get them to perform the installation.

Now some commands for managing the software in your environment, to add software: **module add** , to remove software: **module del** , to list all currently added software: **module list**.

If you are using python chances are that not all the packages you need to use will be avaliable on BluePebble, in that case you will need to set up and activate a virtual environment where you can then **pip install** the packages you need. To create a virtual environment (in your current directory) type **python3 -m venv ./mypyenvb**, to then activate it type **source ~//mypyenvb/bin/activate**.

User spaces

There are two user spaces home and work. Home has much smaller amount of memory and number of files you can store than work. To change to home type ___cd /user/home/_, and to change to work type **cd /user/work/**. To get a good view as to the state of your home and work spaces type **~mw16387/quota**. Really should just be in work space, unless you are doing some testing of some kind.

Submitting jobs

To copy your folder to BluePebble use scp eg. **scp -r bp:** (only works if you got the ssh trick set up). To submit a job script use **sbatch** . To monitor your submitted jobs use **sacct -X**. To log into a node and check info type **squeue -j** and from this get the node name (NODELIST column) then **ssh** to log onto the node and **top** to perform live monitoring.

When Your Job is Finished

Check the slurm scripts using cat to check what happened during the job. To copy files back to your computer use scp again **scp -r bp:**

Parallel

There are multiple ways you can carry out running parallel programs and is quite task specific, refer to "https://www.acrc.bris.ac.uk/protected/hpc-docs/training/intro-to-hpc-slurm/run_parallel.html" for more info. We will note that one method is submitting array jobs, this simply submits multiple jobs at once using one job script, good if what your doing is perfectly parrallel, an example of how to do this is included at the end.

Example

I have downloaded a repo from github that can be used to generate encoded datasets of popular computer vision benchmarks. I copied this library to BluePebble, setup a python environment as detailed above, installed all the required packages into my environment and then submitted the below job script. The job script specifies an array of jobs where each job will run the specified dataset (given after the –dataset_name argument where I call python, here its CIFAR100) through an encoder. We specify the list of encoders in the array called "LS" and use the index value of the current job to select an encoder from the list. Running these jobs in parrallel will save alot of time. Another important thing to note is that we specify that we want this job carried out on GPUs as tensor multiplication is carried out much faster on a GPU than a CPU.

```bash
#!/bin/bash
#
#
#SBATCH --partition=gpu
#SBATCH --gres=gpu:2
#SBATCH --job-name=gpujob
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0:05:00
#SBATCH --mem=2G
#SBATCH --array 1-27
#SBATCH --account MATH021322


# Define executable
export EXE=/bin/hostname

# Change into working directory
cd "${SLURM_SUBMIT_DIR}"

# Define list of encoders we want to use
LS=('resnet18' 'resnet34' 'resnet50' 'resnet101' 'resnet152' 'RN50_clip' 'RN101_clip' 'RN50x4_clip' 'RN5

# Execute code
```

```
${EXE}

# Do some stuff
echo JOB ID: ${SLURM_JOBID}
echo SLURM ARRAY ID: ${SLURM_ARRAY_TASK_ID}
echo Working Directory: $(pwd)

echo Start Time: $(date)

# Use this to create virtual env (keeping this here for my reference later on):
# python3 -m venv ./mypyenvb
# We activate the virtual environment
source ~/tmp/mypyenvb/bin/activate
python dataset_encoder.py --pretrained_encoder 1 --regime latent_ER --dataset_name CIFAR100 --dataset_en


echo End Time: $(date)
```

Once the jobs were carried out I checked the slurm output to check the jobs all went smoothly and used scp to copy the results back to my computer. This job script will also serve as a good template for carrying out multiple Deep Learning related jobs later down the line such as carrying out various experiments.