gpus_computecanada

April 9, 2020

1 An Introduction to Using GPUs on Compute Canada

1.1 1) Accounts: Helios or Beluga? You get to choose!

You can choose to work on Helios or Beluga with GPUs. If you want to use Helios, you'll first have to get an account with Calcul Québec (See step 3 after you have Compute Canada's accounts all set up https://www.calculquebec.ca/en/academic-research-services/procedures/)

You can log onto Helios like this, remembering that your Calcul Quebec password may be different than your Compute Canada one:

1.2 Helios:

ssh @helios.calculquebec.ca

1.3 Beluga:

ssh @beluga.computecanada.ca

1.4 2) job scripts and setup

As you do when submitting jobs using multiple CPUs, you must also submit a job script when running on GPUs.

Compute Canada uses the resource manager/job scheduler, Slurm (https://support.ceci-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html).

1.5 a) Helios: Checking whether CUDA can be accessed

(scripts thanks to Christine!)

You'll want to first find your RAP ID with:

helios-info

With that in hand, You'll be able to request up to 8 GPUs/node for Helios K20 nodes and 16 GPUs/node on Helios K80 nodes.

1.6 test.sh: sample job script to run GPUs on Helios

#!/bin/bash #PBS -N #PBS -A #PBS -l walltime=300 #PBS -l nodes=1:gpus=2 cd "${PBS_O_WORKDIR}" python test.py

1.7 test.py: sample script which checks to see whether CUDA is available

[ ]:
```
import torch as t
CUDA = t.cuda.is_available()
print(CUDA)

### This should error when you run it locally.
```

The cd command into ${PBS_O_WORKDIR} makes sure to execute the file where the job is submitted.

Hopefully, this script will return True after you submit it with:

sbatch test.sh

1.8 b) Beluga: tensorflow-gpu / Keras

Do you want to accelerate your machine learning training to less than half the amount of time it takes on one CPU (or more? I don't really know the scaling, but you get to find out!)? Well this is your section!

After logging onto Beluga, you're going to want to load your favorite python version, e.g. Python 3.6:

module load python/3.6

Then, you're going to want to create a Python virtual environment that you can activate before you start your job. You'll probably want to create your environment in the PROJECT storage area (1TB of space/group)

My project folder path is like this: /project/def-acliu/sabrinab

Create a virtual environment called test-env (do change this name when you make yours!):

virtualenv –no-download test-env

Activate your virtual environment:

source test-env/bin/activate

Install the GPU version of tensorflow:

Be careful here! Are you using tensorflow 2.0 already? Wow, you are way ahead of the game if you do. I'm not. So I have to specify the tensorflow version that I want. Beluga already has a bunch pre-configured and available so try installing the version you want.

pip install –no-index tensorflow_gpu==1.14.1

The no-index flag ensures that you're installing a package that was compiled by Compute Canada people and can potentially help with missing/conflicting dependencies.

If you're using the Keras framework, you'll also have to install that with:

pip install –no-index keras == your version here

1.9 test.sh: sample job script to run a GPU on Beluga

#!/bin/bash #SBATCH –gres=gpu:1 # request GPU "generic resource" #SBATCH –cpus-per-task=6 # maximum CPU cores per GPU request: 6 on Cedar, 16 on Graham. #SBATCH –mem=128G # memory per node (may change depending how much memory your script needs) #SBATCH –time=0-1:00 # time (DD-HH:MM)

source /project/def-acliu/sabrinab/test-env/bin/activate python /project/def-acliu/sabrinab/train.py

Try a sample job script submission like the one above to run train.py on a GPU. Surprisingly, easy! This saved us from having to learn CUDA and directly interact with the GPU. We'll save that for next time… or maybe the next life.

[ ]: