Using GPUs for NLP in Informatics

Doing Research in Natural Language Processing
Tom Sherborne
11/13 October 2023
Overview

- **GPUs in Machine Learning**
- **Working with GPUs**
- **What is a cluster and slurm**
- **When to use a cluster**
- **How to access and use it**
- **Walkthrough running experiments**
- **Resources for workflow**

- **Getting help**
- **Demo + Tutorial on the ILCC cluster (Friday!)**
Reading the room

- ✅ I’m comfortable with shells/bash, SSH and remote access
- ✅ I am comfortable writing my own experimental code
- ✅ I know how to use CUDA and run GPU experiments
- ✅ I have used a cluster (any cluster) before
- ✅ I have used a Slurm managed cluster before
GPUs in Machine Learning
Machine Learning demands many calculations

>>> import torch
>>> a = torch.randn((1024,512))
>>> b = torch.randn((2048,1024))
>>> torch.matmul(b,a)
# Approximately 1B operations!

- CPUs have few, high power processing cores
- On a CPU, each product must be calculated sequentially leading to slow processing.
- But each operation is a simple instruction so can this be sped up?
- Can we delegate processing to many smaller processing cores?
What is a GPU?

- GPUs enable **rapid parallel processing** of operations.

- Many small cores working in **parallel** rather than a few large CPU cores.

- ~4300 cores means **less concurrency constraints**!

- Useful for graphical tasks and gaming but now a **must-have tool** for Machine Learning and Scientific Computing.
Working with GPUs

- We use the **NVIDIA CUDA interface** to integrate GPUs into our code.
- All our code today is in **PyTorch** which plugs into CUDA without us writing GPU instructions. Other options exist if desired.
- Write code as normal then move matrices to the GPU for speed up.
- **GPUs have their own memory.**
  - Small models fit entirely on a GPU (not BERTLlama!)
  - Or process data through a GPU model in batches.
  - Need multiple GPUs? We will look at this later...

```python
>>> import torch
data = a = torch.randn((1024,512))
data = b = torch.randn((2048,1024))
data = torch.matmul(b,a)  
7.39s to compute 1000x

>>> a = a.cuda()
data = b = b.cuda()  
data = torch.matmul(b,a)  
2.36s to compute 1000x
```
Workflow of using a GPU in NLP

you write code

your data

input

CPU runs main process for model

model + prediction

output

GPU does heavy lifting

.cpu()
A typical experiment outline

#!/bin/bash

1. Create folders, check data and environment

2. Train **model** using GPU

3. Generate predictions and scores from test set

4. Cleanup from experiment

5. Done!

- A model training experiment can be automated into a **shell-script** to complete setup, training, inference and cleanup.

- This is for **after model development**, when you are confident that your model operates as intended.

- Formatting your experiments as a script makes migrating to clusters **easy!**
What is a cluster?
What is a cluster?

- An arrangement of servers to **execute** computationally intensive work on dedicated high-performance machines in the background.

- You log into the **head node**, format your experiments and then **submit** scripts as “**jobs**”.

- Your jobs are assigned a **compute node** (with a GPU) which runs your script and accesses a **shared or local file system** for data.

- Jobs are assigned, managed and controlled using a **scheduler** program. Informatics uses the **Slurm** scheduler.
Why use a cluster?

Single GPU experiments

- ✅ Debug models during development with direct shell access to model e.g. using PDB
- ❌ GPU also required to run monitor and other processes.
- ❌ One experiment at a time.
- ❌ Computer possibly not usable during experiments.

Cluster experiments

- ❌ No direct access to shell. Hard to debug errors.
- ✅ GPU dedicated to your experiment.
- ✅ Run many parallel experiments.
- ✅ Sharing GPUs maximises usage without grinding your own PC to a halt.
What do we have in Informatics?

- **ILCC cluster:** [ilcc-cluster.inf.ed.ac.uk](ilcc-cluster.inf.ed.ac.uk) / escience6
  - ~80 GPUs for your work across various machines. CDT students have their own partition.
  - A combination of NVIDIA RTX2080 Ti / NVIDIA RTX1080 Ti cards with 11GB memory.
  - One very large storage disk (ostrom) connected by NFS.

- **PGR Cluster:** [mlp.inf.ed.ac.uk](mlp.inf.ed.ac.uk) / uhtred
  - Crannog[01-07] each has **4xA40s** (48GB memory) + 510 GB RAM
  - Damnii[1-12] each has **7/8 RTX 2080s** (11 GB memory) + 190GB RAM
  - 123 GPUs for use between all PGR students.
  - Other partitions (e.g., ‘Teach-Standard’) are used by others and sometimes shared. e.g., ‘landonia’ machines have some A6000s.

- **EIDF** is a new resource to be demo’d in Second Semester (uses a Docker/Kubernetes system)

- Some CDT-NLP students will refer to **CSD3**. This is no longer available through the CDT
Head Node
ilcc-cluster.inf.ed.ac.uk

/home is shared!
/disk/scratch is not!

/home
Disk spaces on the cluster

- Like DICE, you will have a home folder as `/home/${USER}/`
- Move data between machines using `rsync` or `scp`.
- Your user space is on a network disk that all nodes can access.
  - `/home/` is actually `/disk/nfs/ostrom`
  - 168TB disk shared between all users. Keep results and environments here.
  - This disk is large but also slow.
  - There are no backups! Got important work? Copy it out of the cluster.

- Each compute node has a local disk drive at `/disk/scratch/`
  - This is fast to read and write to during an experiment.
  - Save weights and large file here during training.
    - Copy what you need back to your user space at the end.
    - Delete everything else you haven't stored from here at the end.
What is Slurm?

- Slurm is an open source scheduler that controls the allocation and execution of jobs on our cluster.

- You write your experiment script then...
  - You submit your script to the Slurm controller while logged into the head node.
  - Slurm finds an available compute node and assigns resources to execute your script.
  - You can monitor your job output and status using Slurm monitoring commands.
  - No free compute node? Slurm places your jobs in a queue to execute when a GPU is free.
Slurm commands

- **sbatch** - submit a job for hands-off execution on the cluster.
- **srun** - request an interactive shell session on a compute node (for debugging)
- **squeue** - check the execution of your jobs and the queue of waiting jobs
- **sinfo** - check cluster information
- **scontrol** - update job configuration (won’t be covering today)
Use ssh to access the head node

For example:

```bash
ssh ${USER}@ilcc-cluster.inf.ed.ac.uk
ssh ${USER}@mlp.inf.ed.ac.uk
ssh ${USER}@${cluster_name}.inf.ed.ac.uk
```
You SSH onto the head node.

Submit your job using `sbatch`.

Slurm assigns the job to a compute node and then executes the job in the background.
Slurm assigns you an interactive session on the compute node (like SSH)

Useful if your job is going wrong somewhere/debugging.

No automatic processing and job is not a background process.
Comparing `sbatch` and `srun`

**sbatch**
- ✅ Your experiment runs as a background process without direct supervision.
- ✅ Run all your experiments in parallel on compute nodes.
- ✅ The intended use case for cluster computing.
- ✅ Go home and rest. Your work is happening while you sleep!

**srun**
- ✅ Gives you an SSH-like session on a compute node. Useful if something has gone wrong and you need to check your model on the cluster.
- ❌ Hoards GPU resources if used excessively.
- ❌ The cluster becomes less useful and effective.
- ❌ Encourages poor experiment design and babysitting your jobs.
Everything all together...

- Assume that experiments are **bash scripts** that specify all steps of computation.
- We access a cluster by SSHing on to the **head node**.
- **Submit an experiment job using** `sbatch` **to request a** **compute node** **to run the job.**
- **Slurm** manages the allocation, execution and running of jobs.
Cluster Workflow
Anatomy of an `sbatch` script

#SBATCH Args here....

conda activate pt

rsync data /home/ to /disk/scratch/

python train.py

python predict.py

rsync results /disk/scratch/ to /home/

rm -rf /disk/scratch/`${USER}`/exp

You will need...

- **Slurm** configuration
- A **Python** environment
- Training and test **data**
- The **model** to train (`model.py`)
- **Training command** (`train.py`)
- **Prediction command** (`predict.py`)
Conda Environments

- Miniconda provides isolated runtime environments for your Python code. This manages your packages so you can be sure what dependencies you are using in your programme.

- Install a specification of packages to an environment and use it for all your experiments!

- Different experiments have different specifications? Use a new environment!

- We will install tools such as PyTorch in an environment.
Data transfer

Head Node

input data -> compute node scratch

DFS <- results from compute node scratch

Compute Node processing

1

2

3

git + results
miniconda
input data
logs

/output/

results
input data

scratch
Software to use

- Moving data:
  - rsync
  - Git Large File Storage
  - $ wget

- Package management & virtual environments:
  - Anaconda

Writing code

- Version control:
  - git
  - Vim

- GitHub
Experiment checklist

☑ A working model pushed to GitHub (or other VCS) to clone from

☑ Code and data in my /home/$USER/ folder on the shared file system

☑ A conda environment to run my Python code within

☑ I know how much RAM and GPUs I need

☑ Bash script defining the stages of the experiment, config and data transfer.
1. Write your code and get it working with a conda virtual environment.
2. Version control your code with git and put it in a repository online with GitHub.
3. Download your code by cloning repo from GitHub.
4. Create/activate conda environment.
5. Get your input data onto the DFS e.g. `scp` / `rsync`.
6. Test your code on an `srun` interactive session.
7. Last minute code edits on command line editors like `vim` or `emacs`.
8. Run YOUR JOBS with `sbatch`.

---

**Local Computer**

- [Image of Local Computer]

**Head Node**

- [Image of Head Node]

  - `ssh`

**Compute node**

- [Image of Compute node]

  - `slurm`
Now what?
Getting help

- **#computing channel in the CDT in NLP slack**
  - Peer support from other cluster users
  - Also useful if you want to help other people out!
  - (Note: I am not in this slack.)

- **Ask your research group**
  - Most senior-ish PhD students have got the hang of the cluster.
  - Most of us are happy to help share our knowledge.

- **Submit Tickets to Computing Support**
  - Try and be as specific as possible.
  - What do you think is the error?
  - Is it reproducible?
  - What Slurm job # caused this?
Cluster etiquette

- **Be nice!**

- Running a lot of jobs? Consider staggering so many users can use the queue

- Or use Array jobs (not covered today but included in the demo)

- If you see someone misbehaving then consider emailing them (they may be unaware)

- Similarly, another user may notify you if they see a process of yours acting improperly (e.g. running Python on the head node)
The cluster-scripts repository

Repo here: https://github.com/cdt-data-science/cluster-scripts

1. scripts to make your life easier
2. examples for quick learns
3. templates for running experiments fast

We will use this in today’s demonstration!
Common mistakes

- **Conda environment not set up properly to use a GPU.**
  - Check `torch.cuda.is_available()==True` in an interactive session.

- **Training fails due to Out Of Memory errors.**
  - Consider adjusting batch sizes to reduce peak GPU memory.
  - Or reformat your model to use multiple GPUs.

- **Nothing happens when I submit using sbatch?**
  - Check your `sbatch` arguments. `sbatch` will fail **silently** if the arguments contain an error
    - e.g., `--partition=illc-cluster`

- **My job stops after a few seconds**
  - The `/disk/scratch` of a compute node might be full. Identify the node and **submit a ticket!**
Not covered today

- **Using EIDF cluster:**
  - Many GPUs and resources but an entirely different experiment paradigm.
  - To be explained in Semester 2

- **CSD3 Cluster:**
  - No longer supported by CDT. You may get access through your supervisor.

- **Multi GPU jobs:**
  - Used to be much more complex but now tools like HuggingFace Trainer / Mosaic Composer can seamlessly use all available GPUs.

- **Using singularity in a Slurm job:**
  - Computing Support have a help page for this: [https://computing.help.inf.ed.ac.uk/singularity](https://computing.help.inf.ed.ac.uk/singularity)

- **Using Array jobs**
  - An example of this is in the Demo on Friday
Demonstration

- Cluster Scripts
- Setting up a workspace
- Experiment walkthrough