COURSE PROJECT: COMPUTING RESOURCES (ACADEMIC YEAR 2025 / 2026)

The algorithms at stake in the module 2/2 project "Deep learning in computational imaging" of B31XO (Forward-Backward, ADMM, but also the training and application of DNNs) are computationally expensive and it may rapidly become prohibitive to run them on personal computers when the data and image size of interest increase. Taking advantage of high-performance computing hardware to enable accelerated computing is paramount in this context. GPUs will typically enable a fast implementation of both network training and application, while CPUs will enable to parallelise computations involved in structures such as Forward-Backward or ADMM. For the project, each group will be provided with access to computing resources in the cloud, and install the Python environment for the project. Details on the resources available and instructions to access and use them are provided below.

1 General aspects

1.1 Organisation & budget

All groups will be given access to a GPU cluster. Resources are limited, with a given budget of computing hours for each group. It is advised that you start by using your own computers for the preliminary phase of implementation and debugging, but you are requested to resort to the cloud resources in order to scale up and produce the main and final results. Once you are willing to run your code in the cloud, wrap it into a small script and send it to the cluster. It is also advised that you monitor your budget closely. In order to easily communicate between your local machines and the clusters, there are two possibilities: (i) sending the files to the server once they are ready to run via `scp`; or (ii) use GitHub by pushing on your machine and pulling on the server (note that your code should not be public; remember to use a private repository if you use GitHub).

1.2 Running code on the clusters: generalities

1.2.1 A foreword about scripts

In order to run your code on a cluster, you first need to wrap it into a script version. A script is a self-contained code that can be easily run from command line. For instance, assume that you have written a code displaying `Hello world` and check PyTorch installations in Python. Such code could look like this:

```
import torch

def main():
    print("Hello World")

    # Check PyTorch version
    print("PyTorch version:", torch.__version__)
    # Check if CUDA is available
    cuda_available = torch.cuda.is_available()
    print("Is CUDA available:", cuda_available)
    # If CUDA is available, display CUDA device details
    if cuda_available:
```

```
        print("CUDA version:", torch.version.cuda)
        print("Number of CUDA devices:", torch.cuda.device_count())

if __name__ == "__main__":
    main()
```

and say that your filename is `script_hello.py`. You can run this file as a script by running in your command line:

```
$ conda activate b31xo_cpu
$ python script_hello.py
Hello World
PyTorch version: 2.5.1
Is CUDA available: False
```

Or if you have the `b31xo_cuda` environment installed:

```
$ conda activate b31xo_cuda
$ python script_hello.py
Hello World
PyTorch version: 2.5.1
Is CUDA available: True
CUDA version: 11.8
Number of CUDA devices: 1
```

Try to do so on your laptop. Please check the *Python Environment Installation Guide* if you haven't configured your Python environment for the course yet.

1.2.2 Running scripts on the cluster

In order to run scripts on the cluster, you need to:

1. Send your script to the server;
2. Connect to the server and launch it as a job.

Launching your job on the server will basically tell the computer to run your script in the background. All this will be detailed in the next sections.

1.2.3 Gathering your results

Once your job has run, you will need to gather the results. These can be downloaded to your machine using either `scp` or `rsync`. If you are using GitHub, beware to exclude the data/results files (e.g. datasets, images, and cache files) through the `.gitignore`.

To sum up, here is the pattern you will follow:

1. Write your algorithms on your machine; debug it locally;
2. Once happy with your algorithm, wrap it into a script file;
3. Send the script file to the cluster;
4. Connect to the cluster and send your job;
5. Once the job is complete, download the results on your laptop;
6. Analyse the results and go back to step 1.

Let us now move to how to connect to the clusters and manage your jobs.

2 Accessing and managing the clusters

You will need a username, an IP address of the master node, and a secret key file to access the cluster. The username depends on which group you have enrolled in. It follows the nomenclature **b31xogr{i}**. The IP address and group-specific secret key file will be shared privately with each group. The secret key file is either in `.txt` format or simply an `id_rsa` file. It needs to be stored in a folder of your choice; some may choose `/.ssh/` but any other will do. We will now assume that you have downloaded the key file and put it in `path/to/my/secret_key_file`.

2.1 Logging in for the first time

Before you log in for the first time, you need to set restrictive permissions to the key. The way to do this depends on your operating system.

If you are using MacOS or Linux
Open your terminal, and execute the following commands.

```
$ cd path/to/my/secret_key_file
chmod 600 secret_key_file
```

If you are using Windows
We strongly suggest that you place your secret key file somewhere in your C drive. Doing so will prevent you from running into errors #1 and #2 discussed in Section 2.2.

Whether you are using MacOS, Linux, or Windows, you should now be able to log in to the server. In the terminal, type

```
$ ssh -i path/to/my/secret_key_file -A <username>@<ip address>
```

If you are prompted to continue connecting, confirm by typing `yes`. You should then see “EPS on the cloud" and be on the server. Your command line should show something like that:

```
[username@servename ~]$
```

You can now interact with the server via the command line interface. For example, you can print the current working directory by the command `pwd`.

2.2 If the above steps failed...

We have witnessed the following error messages so far:

Error #1

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: UNPROTECTED PRIVATE KEY FILE! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for 'path/to/my/secret_key_file' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
```

```
Load key path/to/my/secret_key_file: bad permissions
<username>@<ip address>: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

We have witnessed this error in Windows if the secret key file is not placed in the C drive. In this case, please follow the steps below to change the permissions of your key file.

1. Right-click on the key file, then go to Properties
2. Navigate to the Security tab then click Advanced
3. Click Disable inheritance
4. Click Convert inherited permissions into explicit permissions on this object
5. Click Apply then OK
6. Go back to the Properties window and in the Security tab, click Edit
7. Remove all users except the Administrator
8. Make sure that all Allow boxes apart from Special permissions are checked for the Administrator
9. Click Apply then OK

Error #2

```
Load key "path/to/my/secret_key_file": Permission denied
<username>@<ip address>: Permission denied (publickey, gssapi-keyex, gssapi-with-mic).
```

We have witnessed this error in Windows if the secret key file (after modifying its permissions) is not placed in the C drive. The solution is to open the command prompt as an administrator. To do this, click on the Windows icon, search by typing `command prompt`, right-click on Command Prompt then choose Run as administrator.

Error #3

```
Load key path/to/my/secret_key_file: invalid format
```

In this case, try renaming your file to `id_rsa`; the `.txt` extension seems to be problematic. E.g. in Linux or MacOS, do `mv my_key.txt id_rsa` and then re-try to connect with

```
$ ssh -i "/path/to/my/secret_key" -A <username>@<ip address>
```

2.3 Useful commands

The cluster is under Linux. Classic commands are:

- `pwd` to check your current location;
- `ls` to look at what is in the current folder;
- `cd` to move through folders;
- `mkdir` to create a folder;
- `cp` to copy files;
- `mv` to rename files;
- `rm` to remove files;
- `readlink -f <filename>` to check the full path of a file;
- `cat` to parse a file (or `vim`...);

- ... and all the usual Linux tools.

Here are some specific examples that you will certainly need to use.

To login to the cluster:

```
$ ssh -i path/to/my/secret_key_file -A <username>@<ip address>
```

Data can be copied from/to the cluster using `scp`:

```
$ scp -i path/to/my/secret_key_file /path/to/file <username>@<ip
    address>:/destination/folder/.
```

For instance, if you want to upload your dataset on the server, we suggest you zip it (or tar.gz) and then:

```
$ scp -i path/to/my/secret_key_file dataset.zip <username>@<ip address>:/destination/folder/.
```

Similarly, if you want to download some folders (e.g. results) from the server back to your computer, you can:

```
$ scp -i path/to/my/secret_key_file <username>@<ip address>:/pth/to/my/results .
```

`scp` is a standard tool in Linux, Mac, and Windows devices. One can also use `rsync`.

2.4 Running Jobs

The job scheduler is `slurm`. It is responsible for creating the nodes on the fly as well as for terminating them and managing the workload of the cluster. A submission script is required in order to run jobs on the cluster. It has to contain the resources needed as well as the instructions to get the simulation started. An example of a submission script could be a file named `example_job.slurm` containing

```
#!/bin/bash
#
# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=hello_world # name of the job
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --cpus-per-task=1
#SBATCH --time=00:10:00 # maximum duration of the job
#SBATCH --output=log.out # name of the output log file
#SBATCH --error=log.err # name of the error log file

# go to the folder containing the script
cd /your/folder/
# activate python enviroment
export PATH=~/miniconda3/bin:$PATH
eval "$(~/miniconda3/bin/conda shell.bash hook)"
conda activate b31xo_cuda
# launch the python script you wanted to create
python script_hello.py
```

In order to launch that specific job, please follow the instructions in *Python Environment Installation Guide* to install `conda` and configure the environment `b31xo_cuda` on the server. You can then go in the folder containing `example_job.slurm` and type

```
[myid@mgmt ]$ sbatch example_job.slurm
```

Useful commands to interact with the job scheduler:

- `sinfo`: returns the current status of the cluster.
- `squeue`: a list of active jobs along with their names, status and nodes they are running on.
- `list_nodes`: a more comprehensive list of the available nodes and their specs.
- `sbatch <submission_script>`: submits the job to the job scheduler.
- `scancel <job_id>`: cancels the job with the specific ID.

On the cluster, each group can launch up to three jobs: one running and two in queue. Each time a job is over, the next job in the queue is automatically launched.

2.5 Managing your budget

In order to manage the remaining budget, one can use the `mybudget` command.

```
[myid@mgmt ]$ mybudget
+-------------------------------------------------------------------------------+
|  Jobs (total) | Allocation (min) | Usage (min) |  Balance (min)  |
+-------------------------------------------------------------------------------+
|  24           |      27000       |    52.4333  |      26948      |
+-------------------------------------------------------------------------------+
```

We insist that you are entirely responsible for spending your budget reasonably.

2.6 Logging off from the server

Simply use the command `exit` to log off.

```
[myid@mgmt ]$ exit
logout
Connection to <ip address> closed.
```

2.7 Hardware specifications

- GPU: NVIDIA T4 Tensor Core
- GPU Memory: 16GB