SJK012 - High Performance Machine Learning - Assignment3

Pablo Muñoz Alcaide

At the moment I have not been able to run it on Patan due to the execution freezes and doesn't show any result or any error. However, in the local machine the implementation does work, obtaining the following results for the processes available in my WSL subsystem. In addition, although it has not been possible to implement it in Patan, this document explains how the deployment would be launched.

The python file where the parallel implementation is developed is called:

neural_network_module_parallel.py

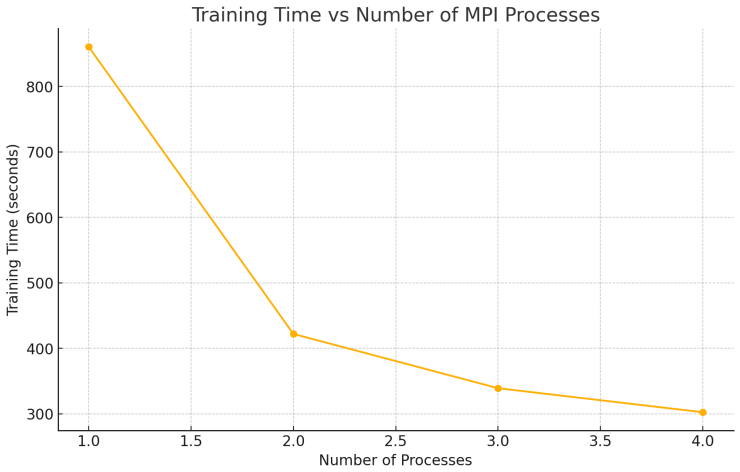
Results & Discussion

Because I had to run on my local machine (WSL) I could only run with 1,2,3, and 4 processes, as it has no more available.

The numeric results obtained are the following:

Processes	Training Time (s)	Training Accuracy	Validation Accuracy
1	860.59	0.49	0.48
2	421.88	0.59	0.55
3	338.96	0.61	0.58
4	302.23	0.64	0.62

Which we can see more clearly in the next graphics:



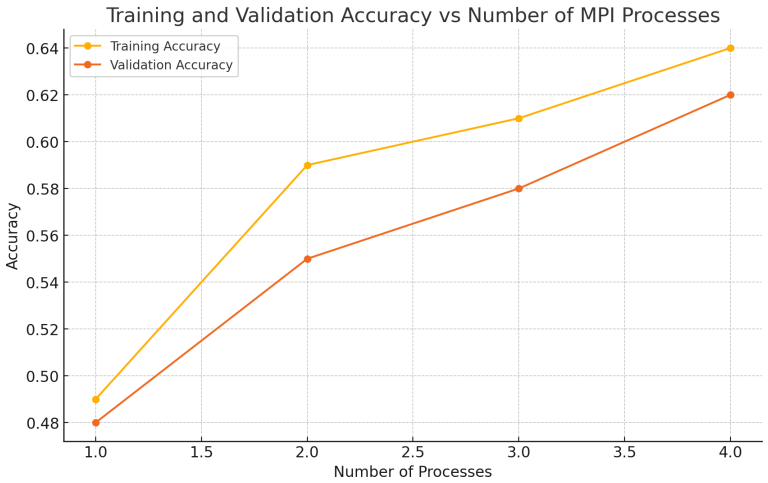
We can see that as the number of MPI increases the training time decreases significantly, this shows that parallelizing effectively reduces the overall training time, which is expected as more processes share the computational load.

Moreover we can calculate the speedup and the efficiency:

Processes	Speedup	Efficiency
2	2,04	1,02
3	2,54	0,85
4	2,85	0,71

Seeing that as the number of processes increases, the speedup also increases, indicating that the parallelization is effective. However, the efficiency decreases as the number of processes increases .Although, this is a common phenomenon due to the overhead associated with managing more processes and communication between them.

We can see the accuracies in the next graphic:



Where both training and validation accuracies improve as the number of MPI processes increases. This could be because of the batch_size handling of the neural network:

```python
num_train = solver.X_train.shape[0]
iterations_per_epoch = max(num_train // solver.batch_size, 1)
num_iterations = solver.num_epochs * iterations_per_epoch
```

Therefore, the batch size determines how many samples are processed before the model's internal parameters are updated. So, when the number of MPI processes increases, the workload is distributed among more processes, this effectively allows for larger batch sizes per iteration, which can lead to more stable gradient estimates and faster convergence, resulting in improved training efficiency and potentially better performance, as is the case.

Patan Implementation

To carry out the implementation in Patan, first is needed to upload the ***neural_network_module_parallel.py*** with the assignment2 files in the same folder as the notebooks.

Then, we have to modify bash:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
PATH="/state/partition1/soft/cuda-10.0/bin:$PATH"
# Update PATH and LD_LIBRARY_PATH
export PATH=/state/partition1/soft/gnu/gcc-9.2.0/bin/:$PATH
export
LD_LIBRARY_PATH=~/optb/lib64:~/optb/lib:/state/partition1/soft/gnu/gcc-9.2.0/lib64/:/state/partition1/soft/gnu/gcc-
9.2.0/lib/:/state/partition1/soft/gnu/libffi-3.2.1/lib64/:${LD_LIBRARY_PATH}

# Set CC to use the gcc-9.2.0 version
export CC=/state/partition1/soft/gnu/gcc-9.2.0/bin/gcc
# Activate python virtual environment
source ~/optb/pyvenv311/bin/activate
```

This way we ensure that we can read the precompiled OpenBLAS and BLIS libraries and activate the python environment, on which we will work, automatically.

Once this is done, we have to configure our ***out1.pbs*** file, which will compile and send it to a queue system where our ***.py*** program will be executed. In this file we indicate the number of nodes and the number of processes per node by means of the following instruction:

#PBS -l nodes=1:ppn=1

And we also add the following lines that ensure that our ***.py*** is executable and that it runs through the virtual environment created.

```
Unset

# Activate the Python virtual environment
source ~/optb/pyvenv311/bin/activate

# Ensure the script is executable
chmod +x ./neural_network_module_parallel.py

# Run the parallel MPI job
$MPIRUN -np $NPROCS $MPIOPT ./neural_network_module_parallel.py
```

Also to ensure that the environment can read the different modules we add to the ***.py*** file the following command in the first line:

#!/home/al437004/optb/pyvenv311/bin/python

This ensures that the script uses the correct python interpreter.

Finally, the pbs file is sent to the queuing system. By means of the instruction:
qsub out1.pbs

When submit the job using the ***.pbs*** file, the system generates two files for logging:

.oXXXXX: This file contains the standard output (stdout) of the job.
.eXXXXXXX: This file contains the standard error (stderr) of the job.

These files are useful for debugging and verifying that your job executed correctly. You can find them in the directory from which you submitted the job.