Minimalist MNIST Inference Engine, Compare CPU vs GPU with CUDA and cuDNN Acceleration

Coursera, CUDA Advanced Libraries course, capstone project:

https://github.com/jquk/CUDA-practice-capstone

INDEX

1. Introduction and Purposes

This project demonstrates the advantages of accelerating a neural network for MNIST digit recognition using a GPU with CUDA and the cuDNN and cuBLAS libraries, as compared to doing it in a CPU.

I also had a secondary purpose, which was learning how to create, train and run a simple neural network from scratch in C++.

2. Code Structure

The code has gone through a major refactor.

It has been split into several library files, which helps understand the architecture, and simplifies the code review and maintenance.

The entry points are `**src/mnist_titest_on_cpu.cpp**` and `**src/mnist_titest_on_gpu.cpp**`, both much smaller now, containing only the `main` function.

Under `**lib/**` you can find files for neural network management, and other helpers, for both the CPU version and GPU version (their names indicate for which).

3. Preparations

Run the following commands in the indicated order:

Clone the repository:

`git clone https://github.com/jquk/CUDA-practice-capstone`

`cd CUDA-practice-capstone`

Download and extract the MNIST digit dataset for training:

`make download-mnist`

`make extract`

The dataset will be placed under `content/`.

4. Compilation and Execution on CPU

Makefile recipe for building the CPU-version:

`make build-for-cpu`

Makefile recipe for running the CPU-version:

`make run-on-cpu`

5. Compilation and Execution on GPU

Makefile recipe for building the GPU-version:

`make build-for-gpu`

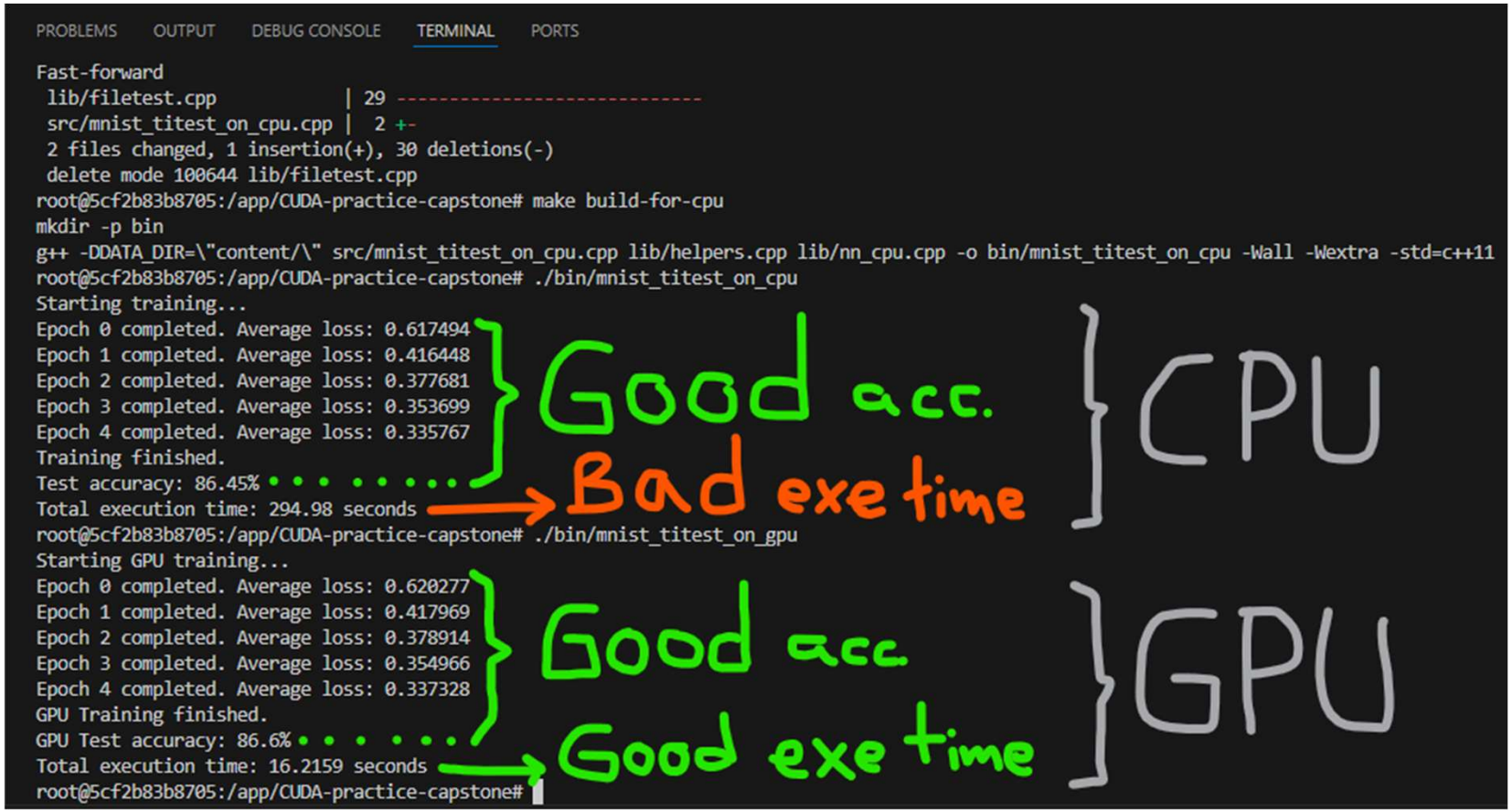
Makefile recipe for running the GPU-version:

`make run-on-gpu`

6. Comparing CPU vs GPU Results

Similar accuracy results.

GPU version is much faster.



7. Further Future Improvements

Initially I got remarkably different average loss values for the CPU-version and GPU-version, and not only that, the CPU-version showed an unconsistent learning, often increasing the average loss on the final training epochs.

The reason was a different **‘learning rate’** value (had 0.1 for the CPU-version and 0.01 for the GPU version).

After setting both to 0.01 they show a similar learning rate over epochs, and accuracy, as expected.

From there I could dig deeper into the following techniques to improve results in both program versions: **‘learning rate decay’,** add **‘bias terms’** in layers, could improve **‘weight initialization’,** and add **‘gradient clipping’.**

Thank You!