

ECE408 - Applied Parallel Programming

Final Report

Neil Singh (ngsingh2)
Patrick McMahon (pfmcmah2)
Matthew Krikorian (krikorn2)

November 17, 2017

Milestone 1

Milestone 1 of this project was mostly setting up our test environment and preparing it to start developing the later parts of this project.

Running Time of m1.1.py

For m1.1.py, we had an accuracy of 0.8673, and an elapsed running time of 9.76 seconds.

Running Time of m1.2.py

For m1.2.py, we had an accuracy of 0.8673, and an elapsed running time of 3.19 seconds.

Most Time Consuming Kernels

Some of the most time consuming kernels when running the nvprof profile were the *implicit_convolve_sgemm* kernel, taking up 36.96% of the execution time, the *activation_fw_4d_kernel* kernel, taking up 14.33% of the execution time, and the *pooling_fw_4d_kernel* kernel, which took up 10.69% of the programs execution time.

Milestone 2

In this milestone of the project, we were tasked with writing sequential convolution code. Using the skeleton from our textbook, and the corresponding data accesses, we successfully implemented the function below.

```
{
    const int B = x.shape_[0];
    const int M = y.shape_[1];
    const int C = x.shape_[1];
    const int H = x.shape_[2];
    const int W = x.shape_[3];
    const int K = k.shape_[3];

    int H_out = H - K + 1;
    int W_out = W - K + 1;

    for (int b = 0; b < B; ++b)
    {
        for (int m = 0; m < M; m++)
        {
            for (int h = 0; h < H_out; h++)
            {
                for (int w = 0; w < W_out; w++)
                {
                    y[b][m][h][w] = 0;

                    for (int c = 0; c < C; c++)
                    {
                        for (int p = 0; p < K; p++)
                        {
                            for (int q = 0; q < K; q++)
                            {
                                y[b][m][h][w] += x[b][c][h + p][w + q] * k[m][c][p][q];
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Figure 1: Our CPU implementation of the convolutional layer in MxNet

Our Results For Respective Models

```
* Running nvprof python m2.1.py ece408-high 10000
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 8.975875
Correctness: 0.8562 Model: ece408-high
```

Figure 2: Accuracy and elapsed time for high correctness and sample size 10000

```
* Running python m2.1.py ece408-low 10000
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 9.003076
Correctness: 0.629 Model: ece408-low
```

Figure 3: Accuracy and elapsed time for low correctness and sample size 10000

Team Contributions

We all met up together and finished the sequential code for the convolutional layer in person. We also crafted this report together for submission.