ECE 408 Final Project Milestone 2

Team 7: Peiming Wu, Xiuyi Qin, Evan Chang, Rio Martinez

Initial Timing Results

Training iterations	5 Iterations	10 Iterations	20 Iterations
Training time (s)	119.51	237.96	476.62
Test accuracy (%)	96.11	96.54	96.86

Potential Performance Optimizations

Texture Memory

Since the input images are fixed, the first thing we came up with is probably to put input images into constant memory to speed up the reading process. However, due to the relatively small size of the constant memory, our 4,000 input images may run out of constant memory. Hence, we may want to use texture memory, which is also a type of read-only memory that could be used for general computing like constant memory even though its primary purpose is for graphic applications.[1]

Shared Memory and Tiling

Each thread in blocks read input data from global memory. This reading operation from global memory has huge overhead in space and time.

For matrix multiply: Fully connected layers are matrix multiplications. We can use a tiled algorithm and shared memory to reduce access to global memory.

For convolution: All threads use the same convolution masks, so we can put it in shared memory to reduce the global reading in each block.

Unrolling for loops

In the sequential implementation of character recognition, multiple nested for-loops are needed to individually process each pixel of the input image. On a CPU this is undesirable because each iteration is executed serially. By using a CUDA-enabled device to perform our image processing, we can effectively “unroll” these nested for-loops by assigning each iteration to a single CUDA thread. These CUDA threads can be processed in parallel, which dramatically reduces the time needed to process large datasets such as MNIST.

Atomic Operations

Kernels that implement dot products have few outputs and many inputs. With one thread per output element, we will have very few threads and poor utilization of computing units. This can be resolved by having each thread compute one input element and summing at the end. We can use atomic operations to prevent race conditions when computing the final sum.

Reference:

[1] Texture Memory in CUDA: What is Texture Memory in CUDA programming. (n.d.). Retrieved from http://cuda-programming.blogspot.com/2013/02/texture-memory-in-cuda-what-is-texture.html

[2] Kirk, D., & Hwu, W.-mei W. (2017). *Programming massively parallel processors: a hands-on approach*. Amsterdam; Boston; Heidelberg: Elsevier.

[3] Rajas Chavadekar, rvc-mnist-cnn-gpu, Retrieved from https://github.com/rvcgeeks/rvc-mnist-cnn-gpu

[4] LeNet-5 - A Classic CNN Architecture, Muhammad Rizwan. (2018) Retrived from: https://engmrk.com/lenet-5-a-classic-cnn-architecture/