Lab4 Report

UID: 004270644 Name: Yuhong Wang

1. Stategies

All the data was collected from the AWS server.

I wrote four kernels for this lab. The first kernel was to load the bias into the C buffer, the second kernel was the main convolution computations, the third one was to eliminate all negative elements, and the last kernel was max pooling.

The first, third and fourth kernel, only have three nested loops, and these three kernels only take a little time. So I just used a three dimensional work group to compute them. The convolution kernel is the one that take more than 90 percent of the total compute time, so I focused optimizing this kernel.

I firstly set up a three dimensional work group, with local group size 1*1*32, and every group dimension corresponds to one dimension of the C buffer. Since I didn't use any local buffer yet. 1*1*32 local work group size gives the best performance of 65 GFlops.

Then I used a local shared buffer to store the weight and cin. I tested this optimization method with several local work group size, for example, 1*1*32, 1*8*8, 1*16*16, 1*8*32 and 1*16*32. All of them had a performance between 100Gflops and 120 Gflops. The results showed the local group size of 1*8*32 gave the best performance of 120 Gflops.

2. Comparison

This lab has a much better performance compared to the lab3. My lab3 has 70 Gflops at best. This lab has about 120 Gflops.

This might because GPU has more cores than CPU, which mean matrix multiplication can be better parallelized for GPU. More importantly GPU are designed for fast context switch. For large matrix multiplication, like the convolutional neural network in this lab, GPU saves more time on data fetching.

3. Memory Usage

For the submitted version, I used the global memory to store all five matrixes, Cout, Cin, Weight, Bias, and a temp buffer C. Cout requires NUM*OUTIMROW*OUTIMROW*sizeof(float), Cin requires NUM*INIMROW*INIMROW*sizeof(float), Weight requires NUM*NUM*KERNEL*KERNEL*sizeof(float), Bias requires NUM*sizeof(float) and C requires NUM*IMROW*IMROW*sizeof(float).

For each work group, I used a local buffer to store a portion of weight, and another local buffer to store a portion of cin that are needed for the current j loop.

For each work item, I used one private float to store the temp C[i][h][w] and store it back after completed all the computations.

4. Challenges

One challenge that I met during this lab is where to set the local memory barrier. I firstly only set the barrier at the end of the loop that load the data into the local shared memory. But I always get an error. Then I figured it out that when one thread finished the computations in current iterations, it will update the shared memory. Other threads will used the updated elements for the previous iterations, which caused error.