Assignment 2 : Image Processing Parallelized

Aryan Dua
2020CS50475

Divyansh Mittal
2020CS10342

Harihar S
2020CS10878

1 Optimizations

1. We initialise the outputs with bias instead of adding them separately. Thus we modified the convolution operation to add the dot products.

2. For the convolution kernel, each GPU thread separately computes the dot product of two matrices of the same dimension as the kernel. The parallelization is thus across filters, channels and the (x, y) coordinate of the bigger input matrix. This requires the need for atomic updates on the resulting matrix.

3. For the max pooling kernel, each GPU thread separately computes the maximum over the submatrix. The parallelization is thus across channels and the (x, y) coordinates of the matrix.

4. For the ReLU kernel, each GPU thread separately computes the relu function on an independent index of the matrix.

5. For each function call, we have made a separate kernel with constant values. This was because floating operations are expensive on the GPU and could be made faster for specific sizes which were powers of 2. Moreover, this also enables us to unroll the interior loops since they are now of constant size. For example: Suppose `conv`(A, B, C) is being called twice, then we create 2 separate kernels named conv1() and conv2() with the respective values of A, B, C hardcoded inside each kernel.

6. We run through all the samples for each function call, before proceeding to the next function call, like a frontier. Due to this, we obtained quite a high speedup.

7. We store the input matrix in binary format rather than a text file since this would help in a major run time improvement in file reading operations. This was especially observable for a large number of inference samples.

8. Finally, we used streams to pipeline the entire inference process. Firstly, we make sure that all our data transfers from host to device occurs at the start. Here we interleave our kernel computations such that as soon as say $X,$ `conv1` parameters have reached the device, we begin the first layer computation. This way we were able to mask the compute time within the data transfer time.

2 Analysis

Our code gave 99% accuracy on 10k samples. The following table lists the running times of our various implementations, again on 10k samples.

Statistics		
	Sequential	Parallel
IO	0.006	0.163
CONV1	2.403	0.021
POOL1	0.203	0.019
CONV2	13.311	0.020
POOL2	0.058	0.018
FC1	5.934	0.020
RELU	0.018	0.020
FC2	0.670	0.019
SOFTMAX	0.001	0.001
TOTAL	22.626	1.32

Moreover, with streams, the final total time for execution was 0.53s. Thus the initial speedup after implementing kernels was roughly 17.4 and finally 42.69 after utilizing streams and other optimizations.

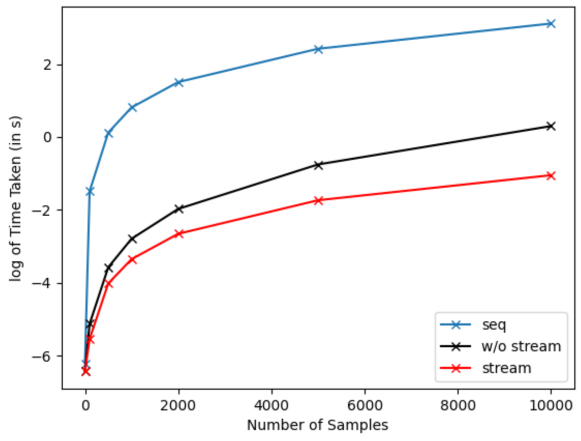

Figure 1: Plot of time taken in the log scale