ECSE 420 - Fall 2020

Parallel Computing

Lab 0 Report

Group 05

Group Members	Student ID
Nicolas Barreyro	260730549
Erion Hysa	260746699

Work Presented To:

Prof. Zeljko Zilic

McGill University

October 1st, 2020

Lab 0 - Simple CUDA processing

Assumptions:

- We assume that the number of threads specified on the lab document is not #threads/block, so we set our number of blocks to 1 and #threads/block to the numbers on the document. Thus we will only have a total of {1,2,4…,256} threads running for the duration of the experiment.

- The GPU on which we ran these experiments is a NVIDIA Geforce GTX 1660 Ti (Laptop version) with 6GB memory.

1. Image rectification

The image rectification consists of transforming the pixels values to center the image by essentially removing negative values from the input picture. The method we determined was simpler to execute is the method of assigning a value of 127 if it is lower than this value, if it is higher we keep the default value of the input image.

To be able to call the rectification we use blocks of size 1, however when rectify is called we use the thread id to know the indexes and we increment this by the number of threads so we know which pixels get rectified by which thread and it is consistent.

Once the image is rectified it is then copied from the GPU memory onto the host memory and saved as a PNG file specified in the arguments and the memory is also freed.

The following images show the differences between the original pictures and the rectification images outputted by the program.

Test_1 original image

Test_1 rectified image

Test_2 original image

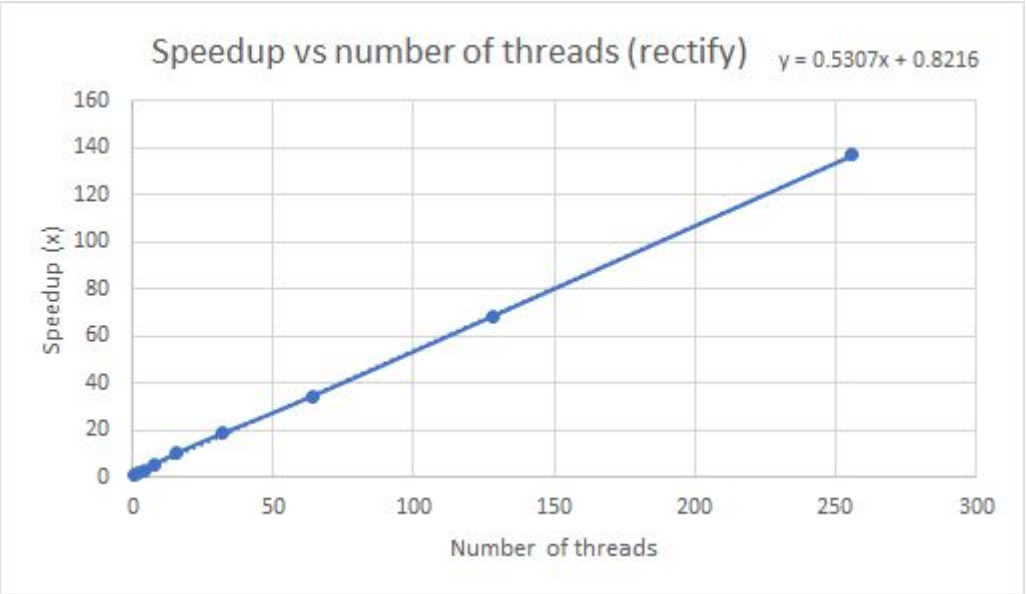
Test_2 rectified image

Test_3 original image

Test_3 rectified image

We can see the images have clearly changed and using MSE we can confirm that the rectification method gives the same output as the image provided.

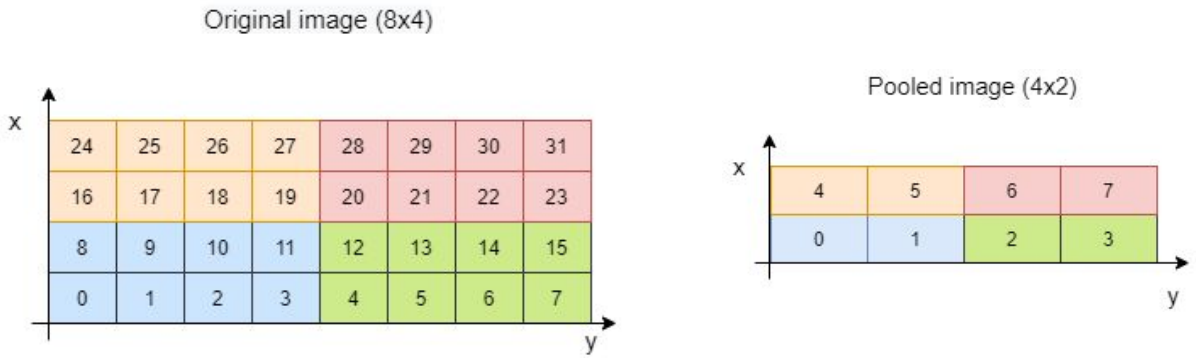
The last element was to check the speedup when the threads were at values 1,2,4,8,16,32,64,128,256. We compare it to the run time of 1 thread to see how the execution speeds up.



The base measure used to calculate the speedup was the time to execute with 1 thread. The measure for the base time is 21.51 seconds. We can see that the speedup behaves in a linear manner as a function of the number of threads.

2. Pooling

As explained in the Lab instructions, pooling takes groups of 4 pixels, finds the maximum values for RGB and alpha among them and assigns them to one pixel in the pooled image. The following figures show how the pixels in the original image are translated into the pooled image. For this example, we assume that the original image has width 8 and height 4 and that there are 4 threads launched by the host.



The colors represent the pixels covered by each thread in the device kernel. Since there are 32 pixels in the original image and 4 threads, each thread will operate on 8 pixels from it and output 2 pixels in the pooled image. For example, the thread will look at the Red value from

pixels 0, 1, 8 and 9, and will assign the max value to the Red position of pixel 0 in the pooled image.

The following images are meant to compare the 3 original images provided to us and their pooled version.

Test_1 original image

Test_1 pooled image

Test_2 original image

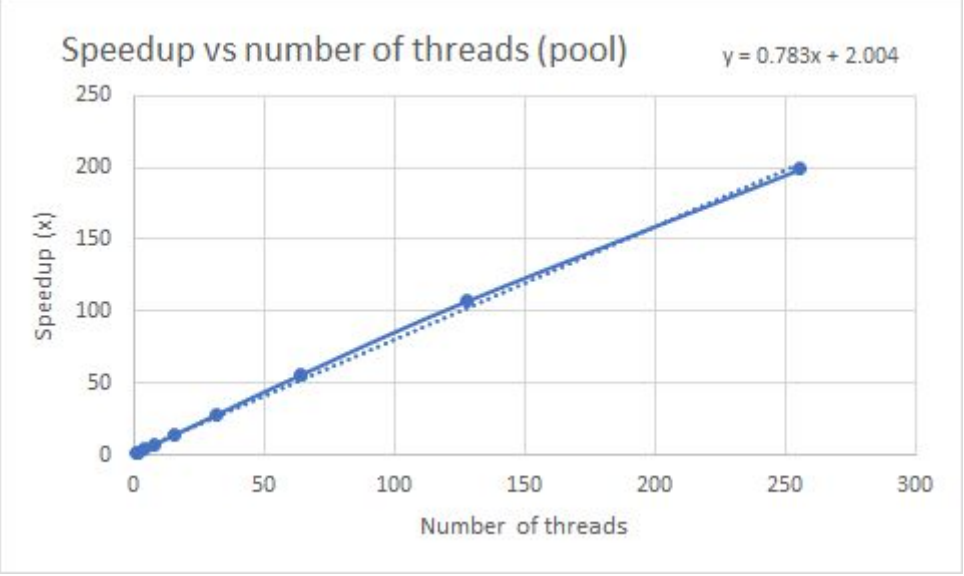
Test_2 pooled image

Test_3 original image

Test_3 pooled image

It might be hard to tell the difference between the images, but the pooled images are a little brighter. It is easier to notice when the images are enlarged. And we found that the effect is more noticeable around the edges in the objects contained in the image. Other than that, the width and height of the pooled version are halved. We also compared our pooled version of the Test_1 image to the one provided to us using the test_equality.cu file and the result was that the images are equal.

In terms of speedup, the number of threads affects the time it takes to process an image significantly. We executed the program with the number of threads from {1,2,4,8,16,32,64,128,256} and the result is plotted in the following graph.



The base time (only one thread) it took to process the Test_1 image was 31.66 seconds. At 256 threads, it took 159 milliseconds (a speedup of 200x). The speedup behaves in a linear manner as a function of the number of threads.