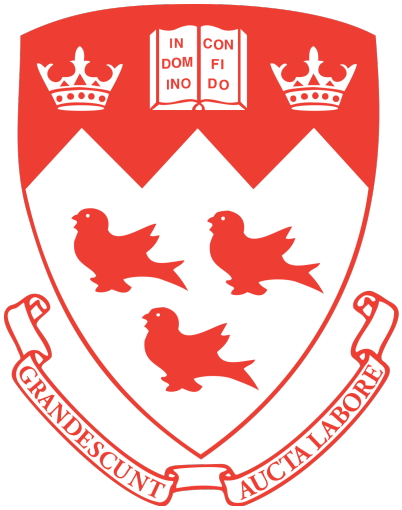
ECSE 420 - Parallel Computing
Fall 2020
Professor Zeljko Zilic

Rebecca Weill - 260804001
Benjamin Szwimer - 260804222
Group 36



Lab 0 - Simple CUDA Processing

October 5th 2020

1. Abstract

The goal of this laboratory was to create two different algorithms that use simple signaling processing to see the effect of parallelism with CUDA. The algorithms include image rectification, and image pooling. Both algorithms were tested with different amounts of threads and the associated runtimes were recorded for analysis. The objective was to conclude if the performance of the algorithms would improve as the number of threads per block increased.

2. Image Rectification Algorithm

The image rectification process comprises scanning every pixel of an imputed image, and changing the RGBA value. In every image, each pixel has a certain value that will determine its brightness. The range that the value could be is from 0 to 255. The 0 and 255 are opposite colors, where the 0 value pixel will output a black pixel and the 255 will output a white pixel and the numbers in between will be all the other colors. The algorithm implemented for image rectification will iterate through all the pixels of the given image and check if said pixel has a value lower than 127. If it does then the algorithm sets its pixel value to 127. Otherwise, if it is greater than 127, it will not modify the value of the pixel.

3. Image Pooling Algorithm

The image pooling algorithm works by taking an image as the input and performing an operation on every 2 by 2 sized matrix of said image. The image-pooling approach was used to implement the pooling algorithm which consists of scanning through a given 2 by 2 matrix of pixels and returning a single pixel of the highest value. It will essentially go through all the pixels of the 2 by 2 matrix and select the highest value for each R, G, B and A value out of all the pixels in the matrix being examined. The algorithm will traverse through every 2 by 2 matrix in the image. After the max-pooling portion is complete, the image that is outputted will be half the size of the original image because if we reduce every 2 by 2 matrix to one pixel then both the width and height will be reduced by half their original sizes respectfully.

4. Parallelization Scheme

Both algorithms had parallelization schemes that were quite similar. They both take in threads as an argument and segment the given image depending on how many threads are used at running time of the algorithm.

The rectify operation will take the width of the image and assign a thread to each particular segment of the imputed image. Depending on the number of threads, rectify will perform the operations on every separate segment of the image. For example, if there are 8 threads, the image will be cut up into 8 different parts and the rectify algorithm will be applied to every segment of the image. The application of rectify on these different partitions of the image all happen in parallel.

The pool operation will calculate the amount of pixels that are in the given image and will distribute the number of pixels evenly to every thread. Each thread will apply the pooling algorithm on their designated pixels. Let us take for example an image that is 4 by 4 pixels and use 2 threads. Since the algorithm works by taking in groups of 2 by 2 pixels, it will find the maximum within those 4 blocks of pixels. Since there are a total 16 pixels, the pooling algorithm compares a grid of 2 by 2 pixels so dividing these 16 pixels by 4, gives a total of 4 different sections composed of 2 by 2 pixels. Each thread will apply the pooling algorithm to 2 sections each which occurs in parallel.

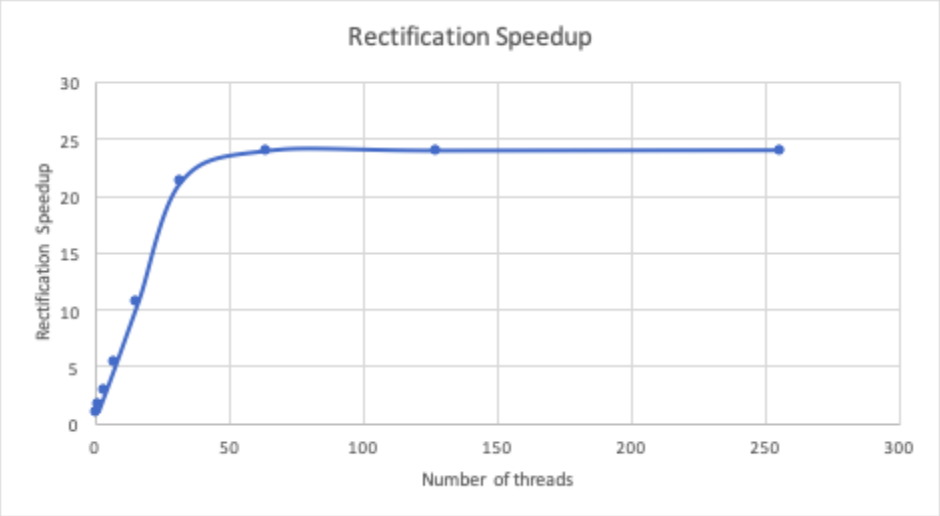
The rectify and pool algorithms both use threads as a way to divide the image in different subsections in order to apply the respective algorithms on smaller chunks of the image, all in parallel, leading to faster results as the number of threads goes up. Hence, there is a correlation between algorithm running time and thread count. As the number of threads go up, the algorithm should execute in a shorter amount of time.

5. Testing

We ran each algorithm, rectify and pool, 3 times for each thread number in the range {1,2,4,8,16,32,64,128,256} and recorded the running time each time. We took the average of these running times and computed the speedup. Then the speedup values were plotted along with the number of threads for both algorithms. The raw data and graphs are located below.

NB Threads	Trial 1	Trial 2	Trial 3	Average Running Time	Speedup
1	308.261902	305.724487	262.306854	292.0977477	1
2	187.872162	158.779388	184.568542	177.073364	1.649586031
4	103.305084	103.692352	93.762848	100.253428	2.913593615
8	53.191681	53.144798	53.094498	53.143659	5.49638006
16	27.090912	27.059296	27.052992	27.06773333	10.79136343
32	13.684832	13.693984	13.681472	13.68676267	21.34162437
64	12.18288	12.153536	12.169248	12.16855467	24.00430911
128	12.157952	12.157984	12.153824	12.15658667	24.02794104
256	12.140064	12.13632	12.14464	12.14034133	24.06009351

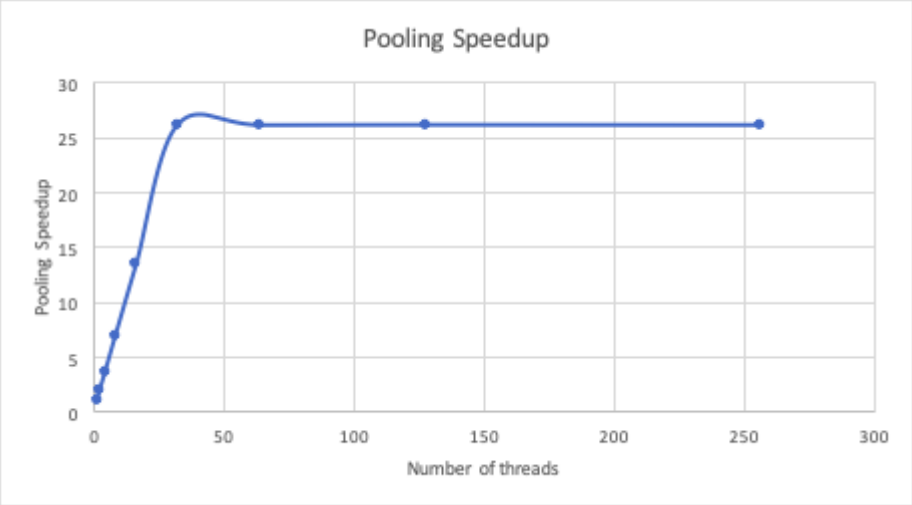
Table 1: Rectification Running Time Table



Graph 1: Rectification Speedup Plot

NB Threads	Trial 1	Trial 2	Trial 3	Average Running Time	Speedup
1	199.0103	193.075134	199.024384	197.036606	1
2	95.907776	105.186172	97.157921	99.41728967	1.981914883
4	55.08691	55.15715	55.085758	55.10993933	3.575337015
8	28.576897	28.550144	28.568577	28.565206	6.897783478
16	14.776704	14.692352	14.706848	14.72530133	13.38081996
32	7.544672	7.544864	7.530496	7.540010667	26.13213889
64	7.53152	7.568736	7.549056	7.549770667	26.09835645
128	7.541504	7.548896	7.539552	7.543317333	26.12068369
256	7.538688	7.523936	7.581696	7.548106667	26.1041099

Table 2: Pooling Running Time Table



Graph 2: Pooling Speedup Plot

6. Image outputs

Figure 1: Test_1.png provided in lab

Figure 2: Test_2.png provided in lab

Figure 3: Test_3.png provided in lab

Figure 4: Result of applying our rectify algorithm on Test_1.png

Figure 5: Result of applying our rectify algorithm on Test_2.png

Figure 6: Result of applying our rectify algorithm on Test_3.png giving

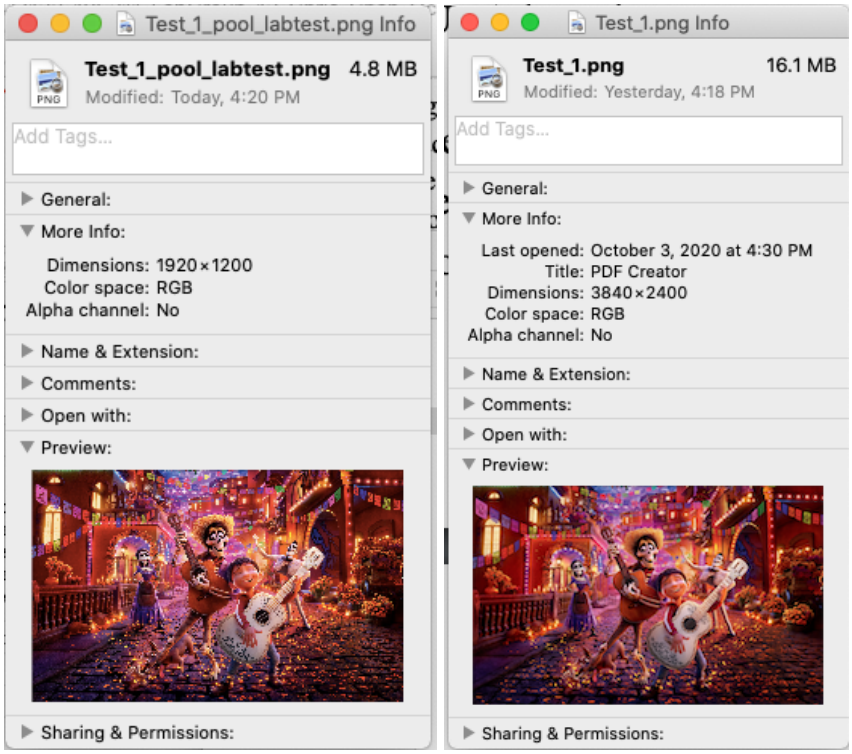


Figure 7: Details of applying our pool algorithm on Test_1.png (left), details of Test_1.png (right)

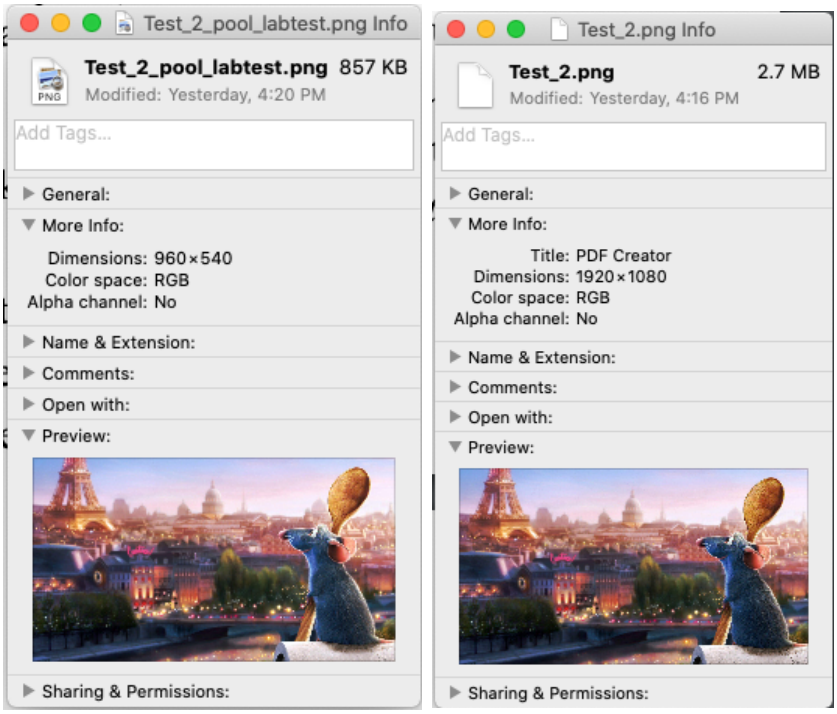


Figure 8: Details of applying our pool algorithm on Test_2.png (left), details of Test_2.png (right)

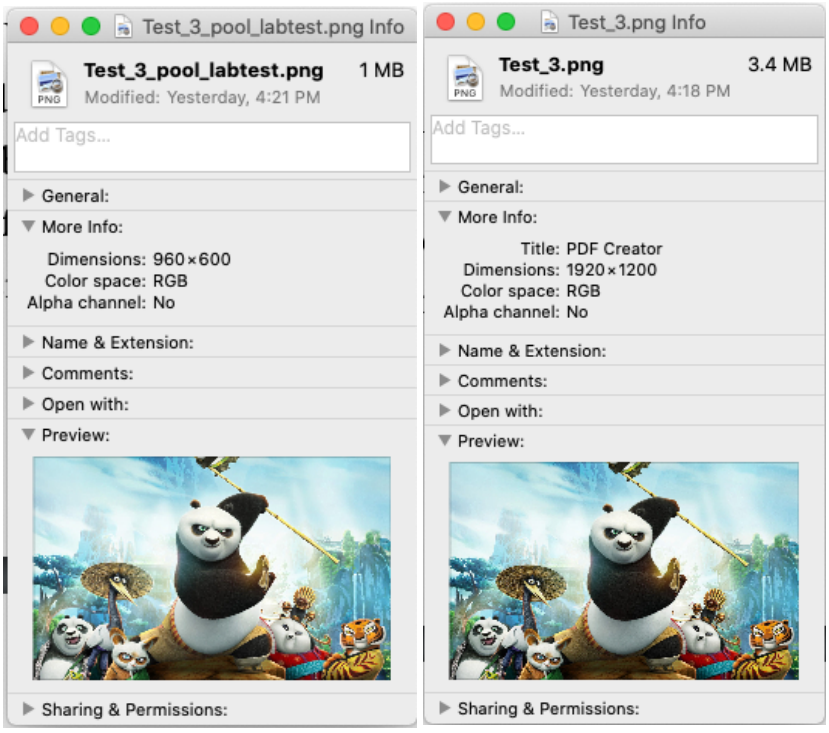


Figure 8: Details of applying our pool algorithm on Test_3.png (left), details of Test_3.png (right)

As shown by the figures above, the algorithms implemented successfully rectify and pool the images provided for the lab.

7. Conclusion

In conclusion, in both tests we can observe that once the number of threads reaches 32, the speedup remains roughly the same, hence it reaches its peak efficiency at 32 threads, any additional threading does not make the algorithm any faster. The number of threads below 32 do not fully maximize the cores of the GPU leading to slower running times. When the number of threads exceed 32, there are too many threads running on each core of the GPU which ultimately leads to a time efficiency plateau which is clearly depicted in the graphs for both algorithms.