ECSE 420

Parallel Computing

Lab_0 Report

Group 14

Elsa Emilien 260735998

Anssam Ghezala 260720743

1. Rectification

Test_1.png 3840x2400=9 216 000

Number of threads	Time/ ms	Speedup
1	27313.599609	1
2	14924.447266	1.830125
4	8089.168945	3.376564
8	4338.302734	6.295918
16	2325.626953	11.74462
32	1167.478760	23.39537
64	658.714722	41.46499
128	391.152618	69.8285
256	176.727966	154.5517

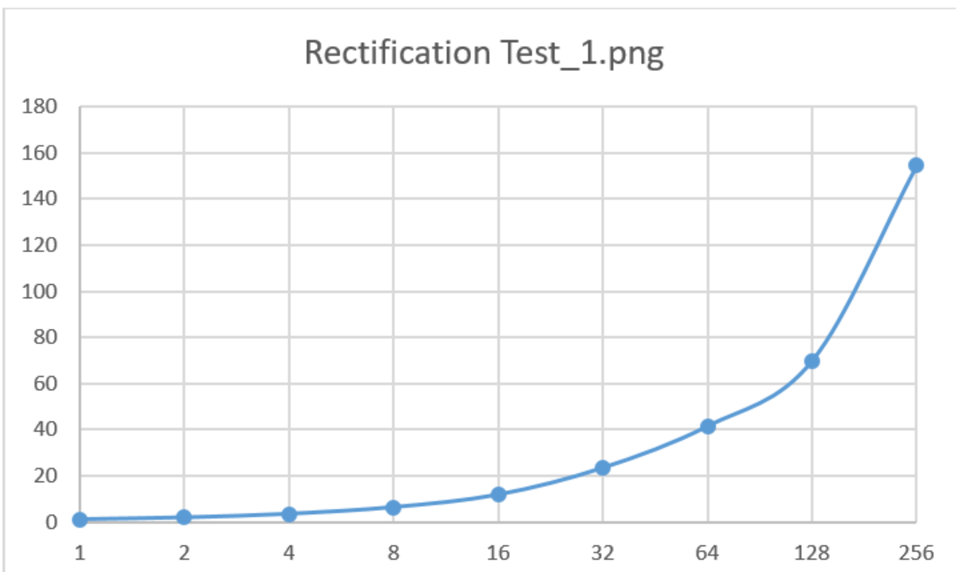

Figure 1 Speedup vs number of threads

Test_2.png 1920x1080 = 2 073 600

Number of threads	Time/ ms	Speedup
1	6578.202637	1
2	3942.781982	1.668417
4	2207.191650	2.98035
8	1190.354004	5.526257

16	540.488831	12.17084
32	367.919830	17.87945
64	157.805573	41.68549
128	97.551491	67.43313
256	49.500160	132.8926

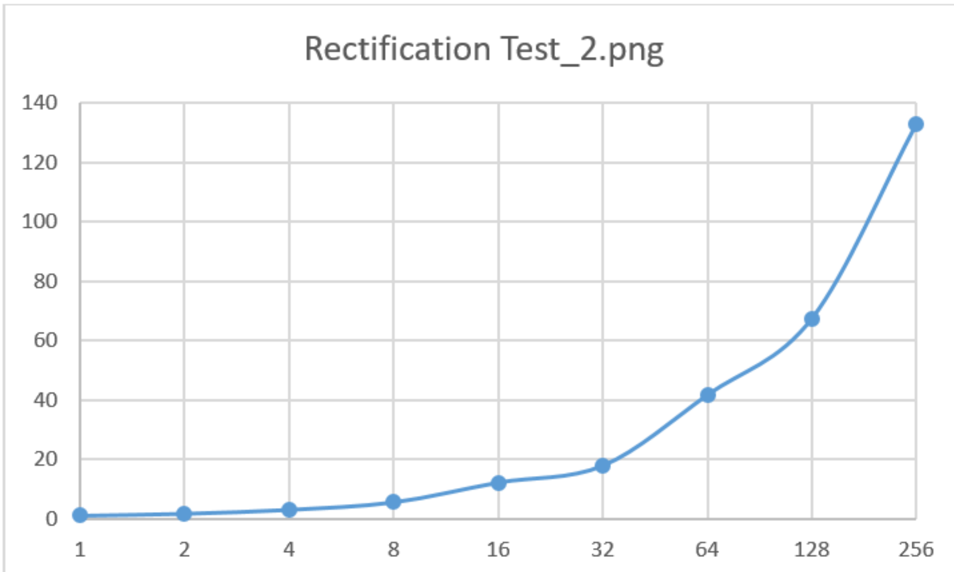

Figure 2 Speedup vs number of threads

Test_3.png 1920x1200 = 2 304 000

Number of threads	Time/ ms	Speedup
1	7775.810547	1
2	4501.922363	1.72722
4	2439.766357	3.187113
8	1291.928467	6.018762
16	607.599426	12.79759
32	408.645447	19.02826
64	201.602142	38.57008
128	86.485153	89.9092
256	54.844257	141.7799

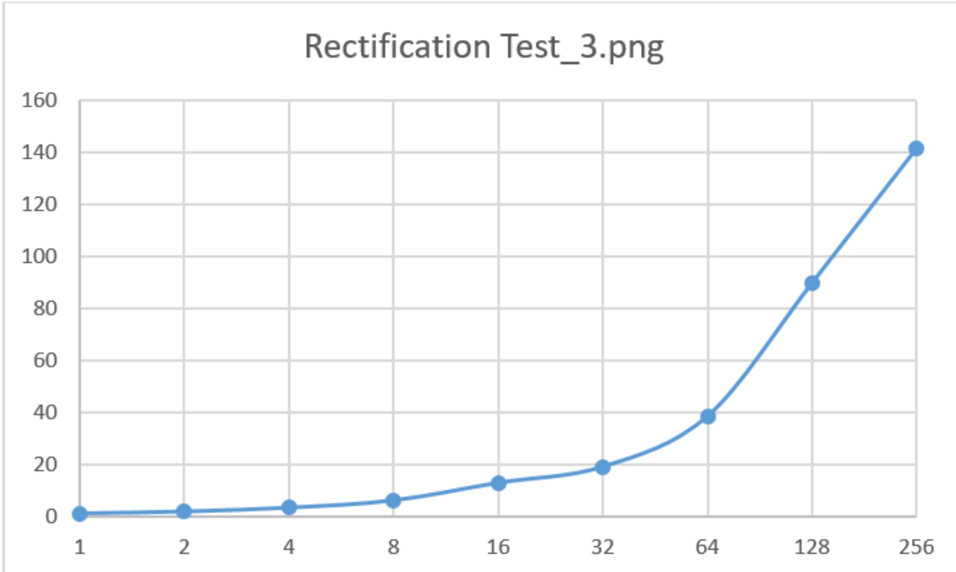

Figure 3 Speedup vs number of threads

Parallelization scheme for Rectification:

For rectification, our parallelization scheme was to divided the 1D array of the lode_png_decoded image into sections, such that the first thread, thread_0 would rectify the first x pixels, where x is the size of the image (width*height) divided by the number of threads. The next thread, thread_1 would then take on the next x pixels and so forth. For each pixel, the threads have to apply the rectification operation on each of the RGBA channels.

We can observe from the graphs a general trend of exponentially increasing speed up with increasing number of threads. This can be explained by the fact that the higher the number of threads, the lesser the number of rectification operation per thread.

We can also observe for each of the Test images that as the number of pixels in an image is bigger, the speed is lower and for images with a lower number of pixels we observe a faster speed. Test_1.png: *9 216 000pixel > Test_3.png: 2 304 000pixel > Test_2.png: 2 073 600pixel in terms of speed we can see the same trend* Test_*1*.png: *27313ms* > Test_*3*png: *7775ms* > Test_*2*.png: *6578ms*

2. Pooling

Test_1.png 3840x2400

Number of threads	Time/ ms	Speedup
1	49595.148438	1
2	25593.787109	1.937781
4	13254.191406	3.741846
8	6827.990723	7.263506
16	3497.781738	14.17903
32	1733.526001	28.60941
64	958.623901	51.73577
128	553.176880	89.65514
256	293.126068	169.1939

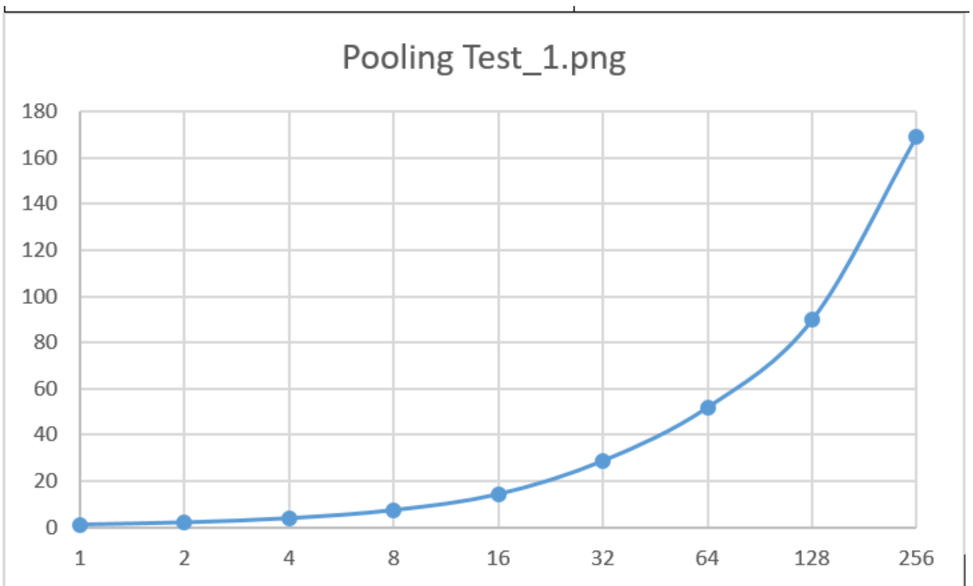

Figure 4 Speedup vs number of threads

Test_2.png 1920x1080

Number of threads	Time/ ms	Speedup
1	11220.334961	1
2	5808.153320	1.931825
4	3034.723877	3.697317
8	1600.941772	7.008584
16	764.713501	14.6726
32	487.029724	23.0383
64	199.066330	56.36481
128	138.407654	81.0673

256	69.804161	160.7402

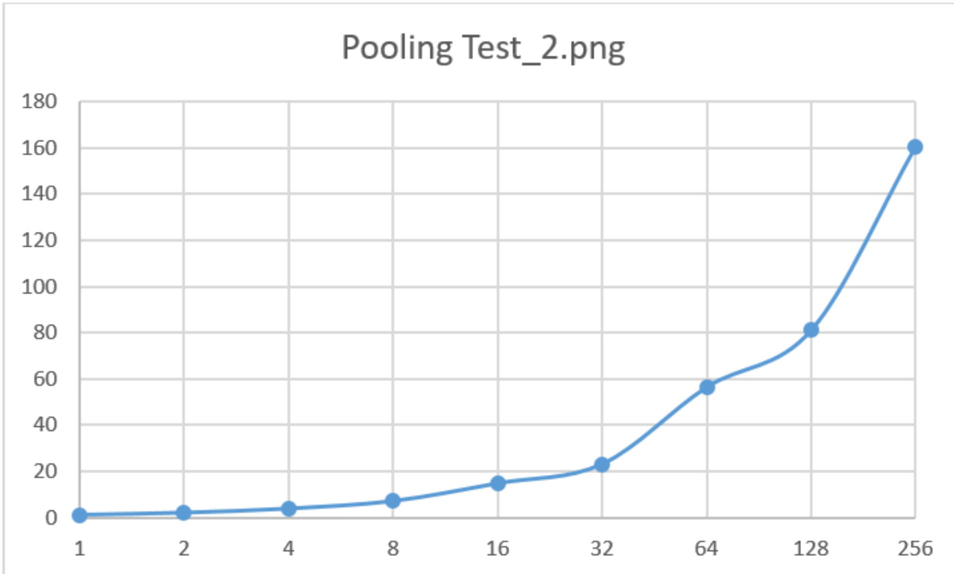

Figure 5 Speedup vs number of threads

Test_3.png 1920x1200

Number of threads	Time/ ms	Speedup
1	12561.204102	1
2	6510.448242	1.929392
4	3381.626465	3.714545
8	1688.385132	7.439774
16	986.074158	12.7386
32	431.726959	29.09525
64	285.558777	43.98816
128	123.543427	101.6744
256	77.114174	162.891

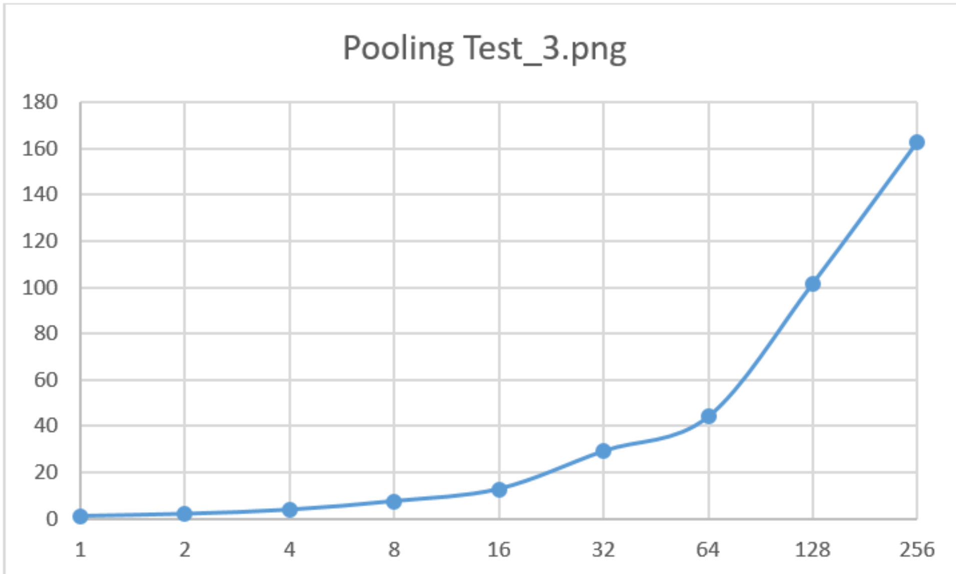

Figure 6 Speedup vs number of threads

Parallelization scheme for Pooling:

For pooling, our parallelization scheme was to give x number of windows to each thread. Here we defined a window as the four pixels constituting the pooling window as defined in the Lab0 question. We also defined x (number of windows per thread) as the size of the image (width*height) divided by (4 * number of threads). We divide by 4 because the size of a pooling window is 2x2. Once we determine x which is the number of windows per thread, we know how many pooling operations will be carried out by each thread. Hence the first thread, thread_0 will do the pooling for the first x windows and the next tread, thread_1 will do the next x windows. For each window, each thread will do the pooling operation on the RGBA channels that constitute each pixel. So thread_0 for instance will do pooling of all the R channels in the window and then proceed to do the pooling for all the G channels in the same window and so on.

We can observe from the graphs a general trend of exponentially increasing speed up with increasing number of threads. This can be explained by the fact that the higher the number of threads, the lesser the number of pooling operations per window hence per thread.

We can also observe for each of the Test images that as the number of pixels in an image is bigger, the speed is lower and for images with a lower number of pixels we observe a faster speed. Test_1.png: *9 216 000pixel > Test_3.png: 2 304 000pixel >*

Test_2.png: 2 073 600pixel in terms of speed we can see the same trend Test_*1*.png: *25593ms* > Test_*3*png: *12561ms* > Test_*2*.png: *11220ms*