CSS Assignment One: Parallel Programming with Java Fork/Join Framework

Name: Racquel Dennison

Student Number: Dnnrac003

Date: 14/08/2022

Introduction

The aim of this assignment was to compare the run time of a parallel program against a serial program. The hypothesis of this is that, when using a multi-core computer, a program will run faster when executed using parallelism. The expected outcome of this experiment is that with P processors, the program will run faster as opposed to running a program on one processor.

Methods:

- Parallelisation Algorithm:
  The way that the parallelisation algorithm was executed was using the java fork/Join library. This technique utilized the idea of the divide and conquer algorithm, whereby, at each stage in the algorithm, if a certain condition was met, the image size would be divided into smaller slices for each process to handle. The fork/join method made use of many threads to efficiently distribute the load of slicing the image and processing it.

  The outline of my code was as follows; the constructor had the parameters of the width and the starting point. In the compute method, the base case required that if the width was smaller than a certain number of pixels, then a direct computation needed to take place. If this base case didn't hold, then the program would invoke threads that were split on the width of the image size.

- Experimentation of the sequential value:
  In order to determine the optimal cut off sequential value, multiple experiments were run. I tested values against the height of the program, 10 pixels, 1000 pixels and 100 pixels. The regulars produced showed that the optimal point at which the parallel code ran the most efficiently was at a sequential cut off point of 100 pixels. This can be seen in the results section, where by I compare the results with a sequential value threshold of the height and a value of 100 pixels.

- Validation of the program:

  There were two methods that were used in order to test the validity of this program. First, I was using the human eye. There is an obvious difference between the two pictures. The second method was a test program, this program would first ensure that there was a difference between the original picture and the picture with a filter added to it. The test program would also ensure that both the parallel program and the serial program produced the same results. This program can be found in the src folder. The different images under each filter is shown below.

Mean Filter

Median Filter

Original Image

- Timing of the programs:
  The time measurements were simply taken by using the system.currentTimeMillis() , the purpose of using milliseconds was due to the fact that the program was executing at an extremely fast rate and I wanted to get an accurate measurement. Each measurement was taken in a loop of 10 iterations and was then written to a text file for ease of accessibility. The starting and ending time was taken at the execution of calling the mean/median filter.

- Tested architectures:
  The two machines that the programs were tested on was a Dell laptop with an Intel® Core™ i5-10200H CPU that has 4 cores and 8 logical processors. The other was the departmental servers at UCT. Each program runs at least 10 times in order to take the most optimal time.

- Difficulties faced:
  The biggest challenge faced was understanding how the idea of parallelism worked. It took a substantial amount of time for me to grasp the concept of the fork/join concept and how the program would split the image into chunks that would be managed by each thread. The other problem I faced was finding the optimal sequential value, although quite obvious, I had mistakenly defined it as the height and this was not picked up until a later point in my research. As a result, I was required to restart all my findings.

Results :

The following information came from the tests that were run. The experimental uses were the departmental server, and my laptop. Each was tested at least 10 times in order to find the best speed up time for the execution of the programs.

What can be observed is that 10 pixels is an optimal cut off for the parallel programs. This being said, as larger data sets are used, one can see that the use of parallelism will be optimised as more threads will be created before the sequential cut off is executed.

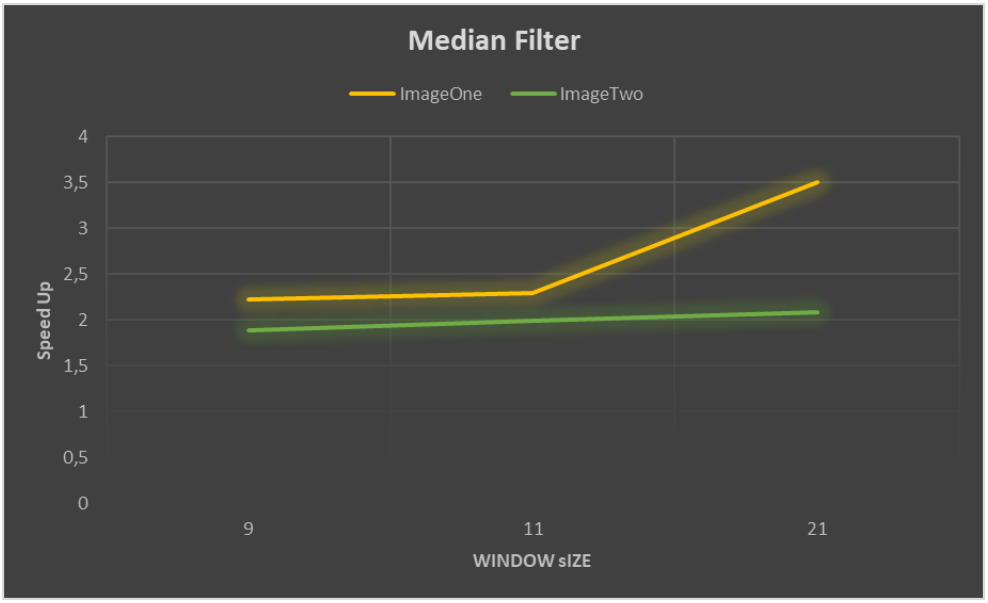
The data sets that were used were images that was a size of 512*512 and an image that had the size of 2185*1622 in sizing. The programs performed really well with both sets of data, however,

it performed particularly well with larger data sets as it could efficiently utilize the idea of calling many threads to distribute the load of the task. This justifies the angle of using parallelism as one can see that it takes a shorter amount of time to do a task as opposed to using a serial program.

Reportings:

Figure One :

This graph shows the speedup time of the programs when using a sequential cut off of the height of the image.



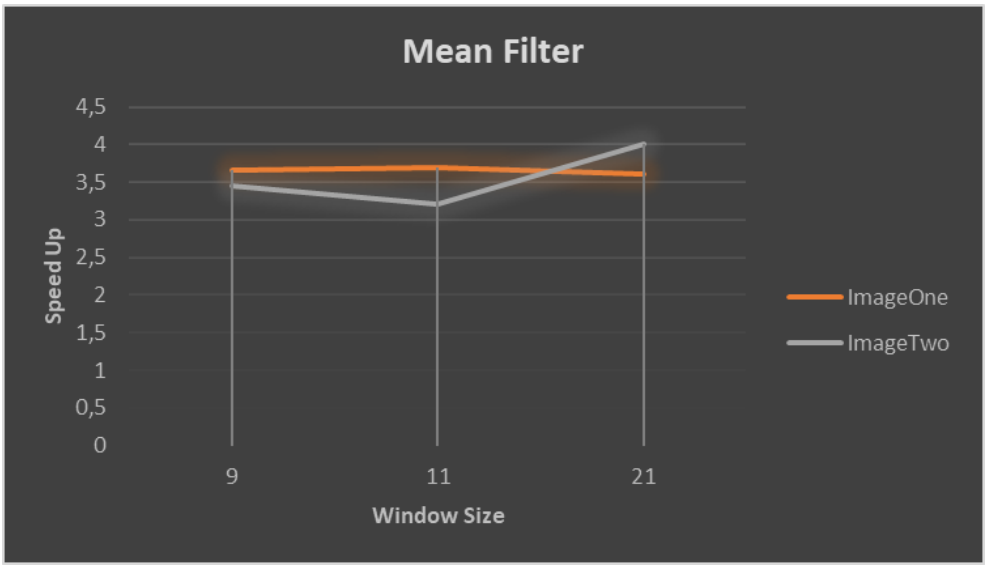
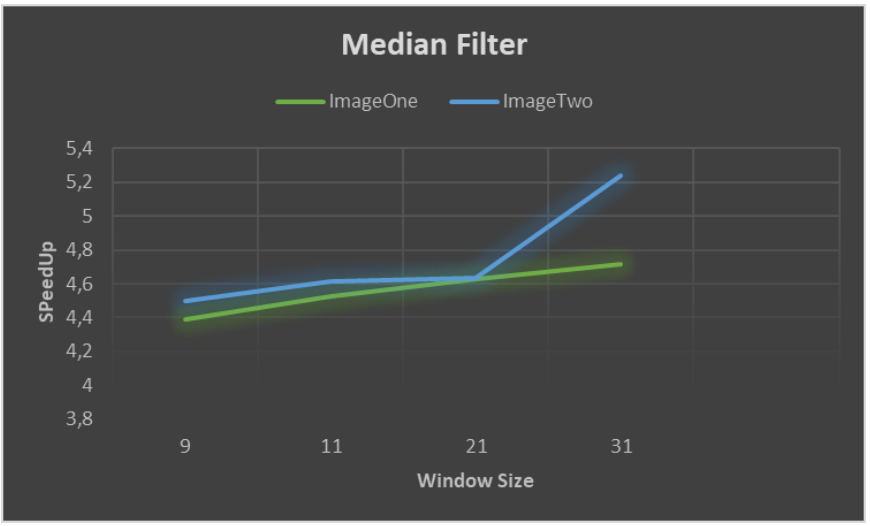


Figure Two:

The following figures are the speedup times of programs that ran with a sequential cut off of width < 100 pixels running on a Dell laptop with an Intel® Core™ i5-10200H CPU

a.



b.

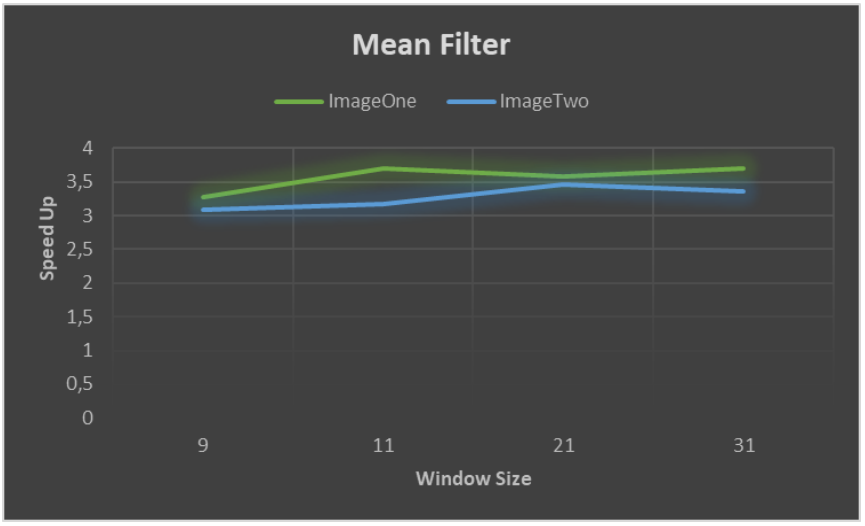
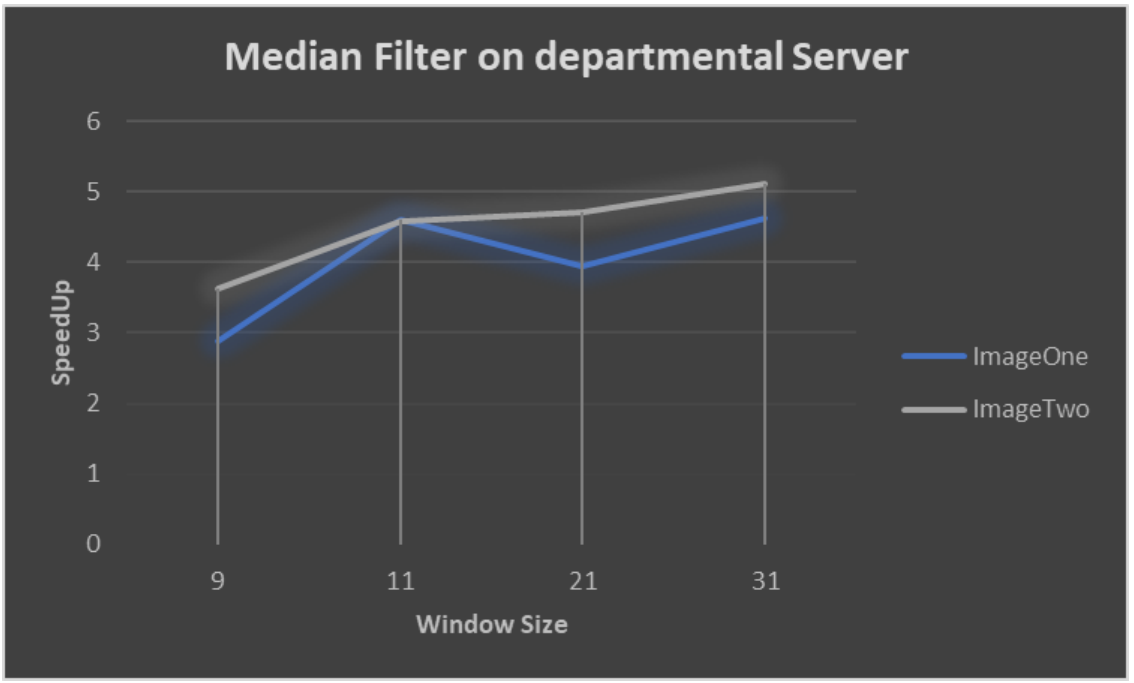


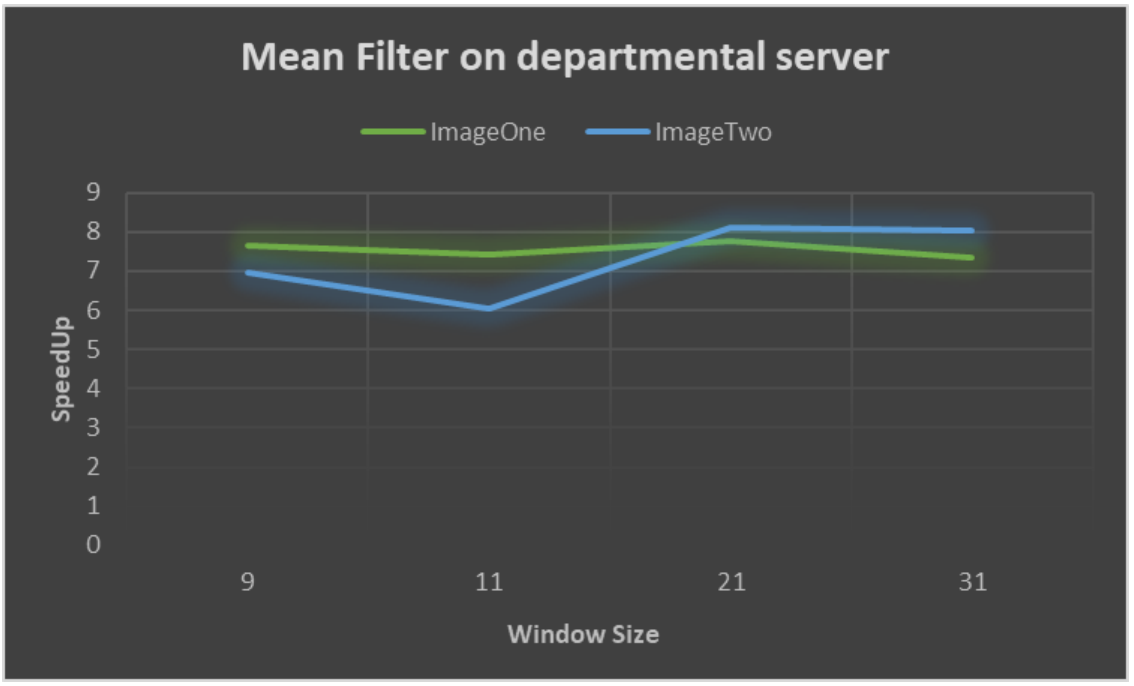
Figure Three:

The following diagrams show the speedup time of both programs when running with a sequential value width < 10 pixels. These tests were runned on both the departmental server and a Dell laptop with an Intel® Core™ i5-10200H CPU.

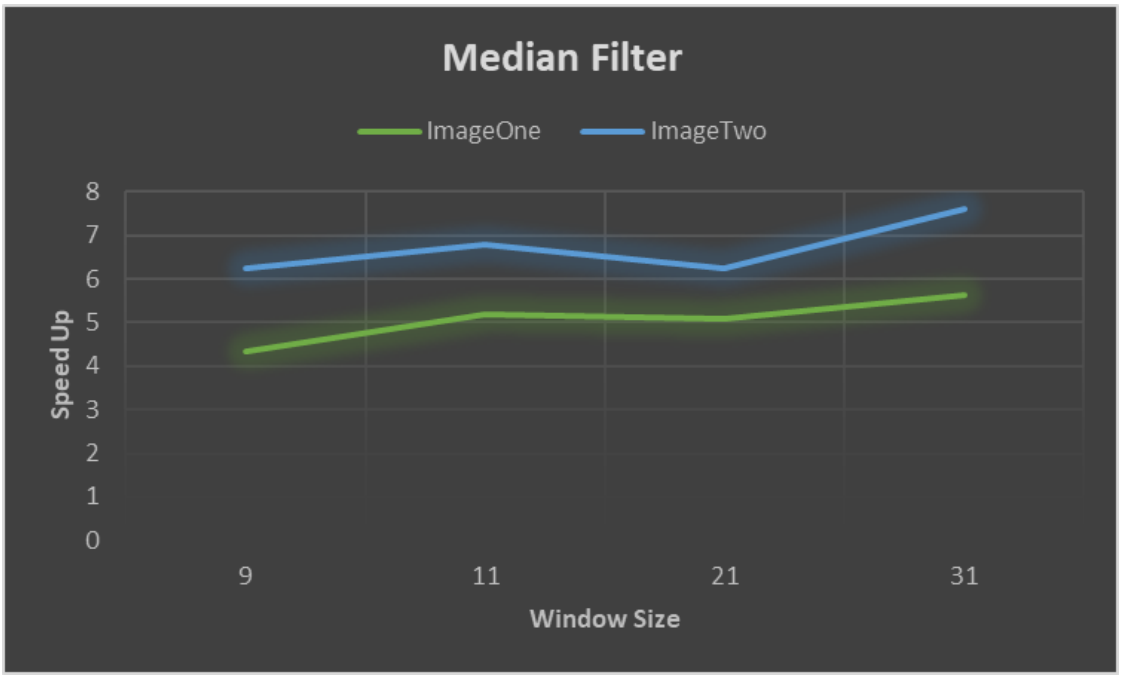
a.


Median Filter on departmental Server
6
5
4
3
2
1
0
SpeedUp
9
11
21
31
Window Size
ImageOne
ImageTwo

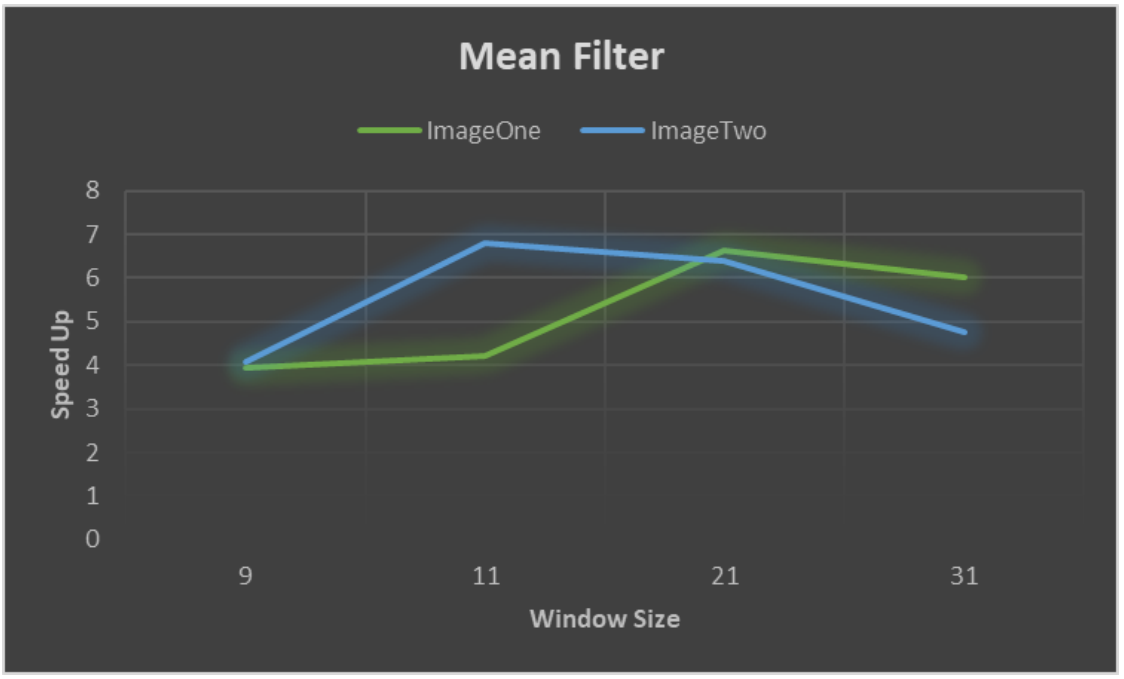
b.


Mean Filter on departmental server
ImageOne
ImageTwo
9
8
7
6
5
4
3
2
1
0
SpeedUp
9
11
21
31
Window Size

c.


Median Filter
ImageOne
ImageTwo
Speed Up
0
1
2
3
4
5
6
7
8
9
11
21
31
Window Size

d.


Mean Filter
ImageOne
ImageTwo
Speed Up
0
1
2
3
4
5
6
7
8
9
11
21
31
Window Size

From the above readings the findings are quite evident to justify the aim of the experiment and the initial hypothesis put forward. The maximum speedup attainable for the median filter was 7.964 and 6.803 for the median filter. The main difference between the speedup time is the fact that the median filter required more computational time that efficiently optimized the idea of parallelism. The maximum speed up is extremely close to the ideal speed up time of 8 as there were 8 processors being used.In accordance with Amdahl's law, we would not expect a speedup time greater than the amount of processors being used.

As seen by the above figures, the sequential cut off time of the height and 100 pixels resulted in lower speedup values. This is attributed to the fact that after a certain value, the program will operate sequentially and that then leads to a longer running time.

The biggest anomalie that was faced is the huge difference between running my programs on my laptop and on the university's server. The server produced higher speedup times. The justification for this is the fact that my computer would have had numerous background processes running that required the distribution of the cores. This would have resulted in my measurements not being as reliable.

Following the experiment and the results observed, the implementation and utilization of parallelism within programs can improve the run time drastically, especially working with a large data set.The use of parallelism in this problem proved itself to be very efficient as evident by the speedup times achieved. That being said, the programmer needs to ensure that the right measurements are implemented. The use of parallelism will only prove itself useful when implemented on a multicore processor. Another important aspect is to ensure that the current processors are not being overloaded with processors. This is extremely important to ensure the utilization of the process is optimized.