Assignment PCP1

Course Code:	CSC2002S
Author:	Bradley Carthew
Student No.:	CRTBRA002
Date:	9 August 2022

1. Methods

1.1. Parallelisation Algorithms

MeanFilterParallel.java

The parallel algorithm for the mean filter is described briefly, below:

- Extract pixel values from image
- Arrange the pixel values in a particular order and populate array
    - This was an important step so that the new pixel values are returned to their original position in the image
- The fork/join framework, was used in tandem with the MeanArray.java class to implement the mean filter in parallel
    - Using a divide and conquer approach the original array (containing pixel values) was broken up recursively until meeting the sequential threshold
    - Thereafter, the resulting array was used to implement the mean filter in series
    - The red, green, and blue values for each pixel are extracted, and used to calculate a cumulative sum within one window
    - The cumulative sum is divided by the window size to determine the mean red, green and blue values
    - These values are used to determine a new “mean” pixel value, which is then stored in an array
    - This process is repeated until each pixel in the array has received a new value
    - Once each sequential compute is finished, the arrays from the left- and right-hand splits are concatenated to form one array with the new pixel values
- The new “mean” pixel values are written to a new image file

MedianFilterParallel.java

The parallel filter for the median filter is described briefly, below:

- Extract pixel values from image
- Arrange the pixel values in a particular order and populate array
    - This was an important step so that the new pixel values are returned to their original position in the image
- The fork/join framework, was used in tandem with the MedianArray.java class to implement the median filter in parallel
    - Using a divide and conquer approach the original array (containing pixel values) was broken up recursively until meeting the sequential threshold
    - Thereafter, the resulting arrays were used to implement the median filter in series

    - The red, green, and blue values for each pixel are extracted, and sorted in ascending order
    - Using the median method, the median value from each window is extracted and stored in an array
    - This process is repeated until each pixel in the array has received a new value
    - Once each sequential compute is finished, the arrays from the left- and right-hand splits are concatenated to form one array with the new pixel values
- The new “median” pixel values are written to a new image file

1.2. Validation

To test the validity of the experiments and make sure that the results from the parallel and serial algorithms were identical, the resulting arrays containing the new pixel values (median/mean values) were compared.

Validation criteria, when comparing experiment results:

- The image being tested must be the same for both the serial and parallel algorithms
- The window size being used must be the same for both the serial and parallel algorithms
- The computer architechture being used must be the same for both the serial and parallel algorithms

Validation process:
- A visual test, which compares the resulting image generated using the serial solution and the resulting image generated using the parallel solution
- An accuracy test, which compares the pixel values generated using the serial solution and the pixel values generated using the parallel solution

1.3. Timing

For the timing of the experiments, each experiment generates a runtime.

The tic() and toc() methods were used to time the overhead and runtime components. The code can be seen below.

```
private static void tic(){
                startTime = System.currentTimeMillis();
}

private static void toc(){
                runTime = (System.currentTimeMillis() - startTime);
}
```

Figure 1: A snippet of code containing the tic() and toc() methods used for timing.

1.4. Optimal Serial Threshold

Determining the optimal serial threshold required some trial and error. The idea was to try and get the serial threshold as small as possible, to reduce the amount of serial computation in the parallel solution.

This process involved:

- Determining the size of the image being processed
- Investigating the number of cores available on the computer at a given time, using the line “Runtime.getRuntime().availableProcessors()” to receive an integer value

1.5. Machine Architechtures

The table below shows the two machine architechtures that the experiments were run on.

Laptop	Processor
Dell G3 15	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 2304 MHz, 4 Core(s), 8 Logical Processor(s)

Table 1: Laptop model and processor for each machine architechture tested on.

Unfortunately, I was unable to test on another machine architechture.

1.6. Problems and Difficulties

The major problems surfaced when trying to develop the parallel versions of the mean and median filter.

The problems included:

- Storing the extracted pixel values in an order that resulted in the new pixel values not changing their position in the image
- Determining the sequential threshold, which was tedious and time consuming
- Learning and experimenting with the fork/join framework to establish a recursive algorithm

2. Results

2.1. Graphs

The mean and median filter solutions (parallel and serial) are tested on two machine architechtures, as described in the “Machine Architechtures” section above.

Each solution is being tested on three images with varying image size:

- Balloons.png (640x480)
- Bird.jpg (1920x1080)
- Girl.jpg (1000x667)

Furthermore, the window size will be varied based on five different values:

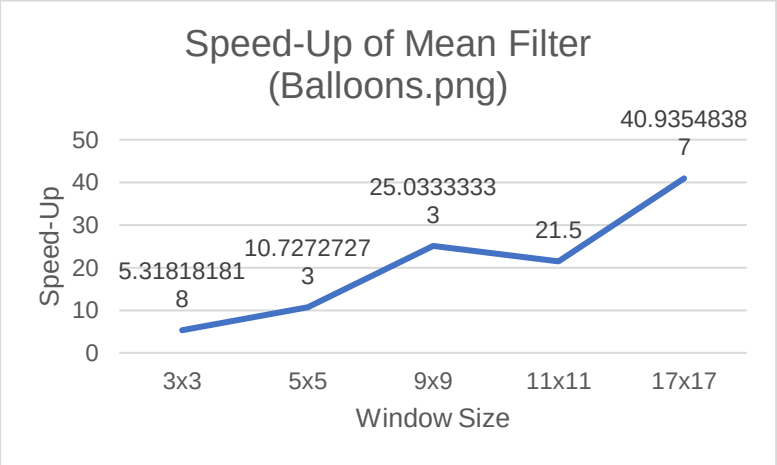
- 3x3
- 5x5
- 7x7
- 11x11
- 17x17

The mean filter and median filter sections below, contain the graphs.

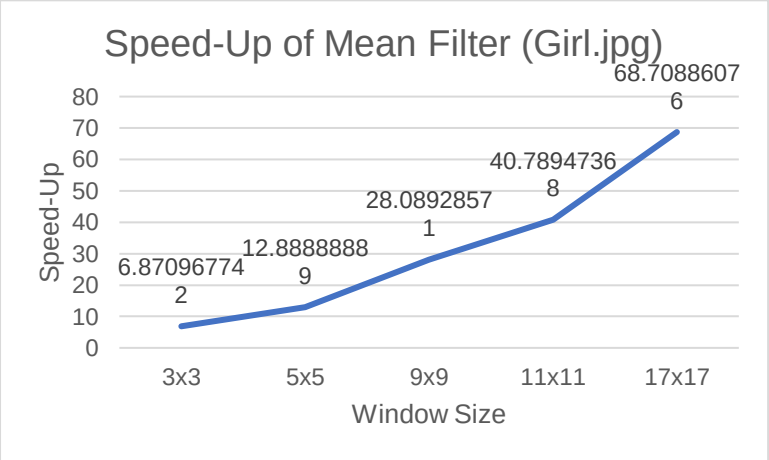
Mean Filter

Machine Architechture: Dell G3 15

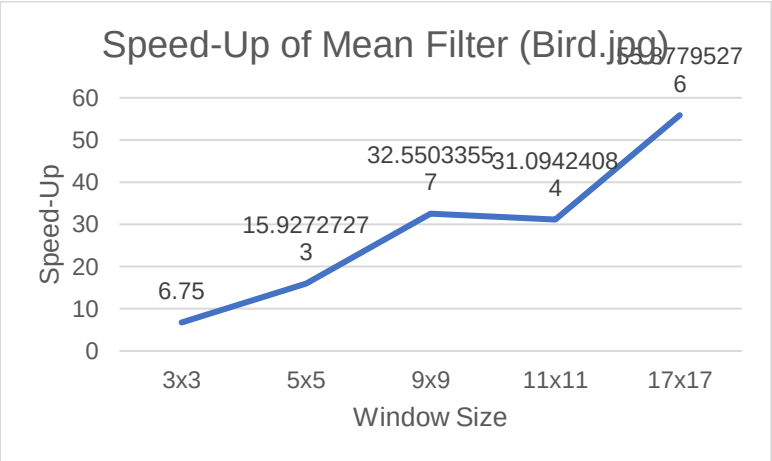
1. *Balloons.png*



2. *Girl.jpg*



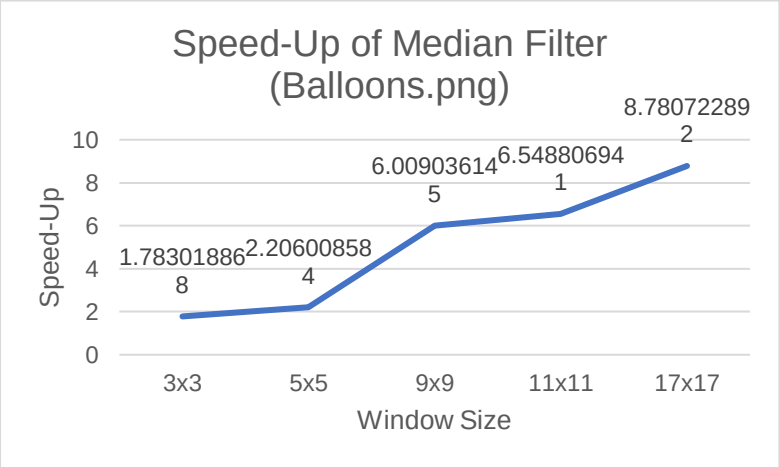
3. *Bird.jpg*



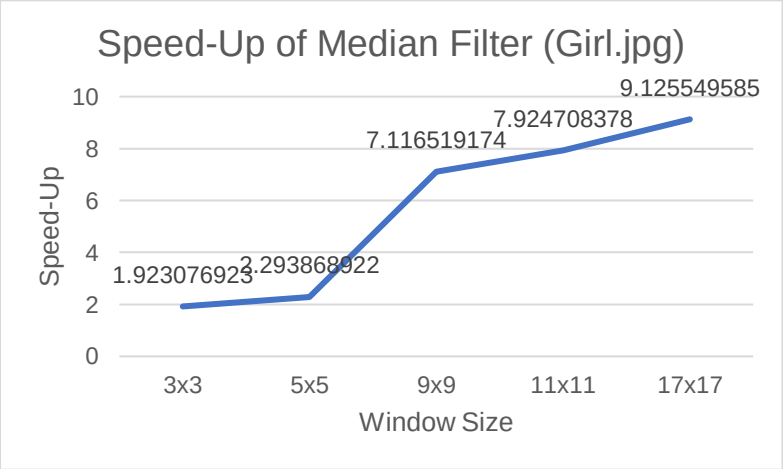
Median Filter

Machine Architechture: Dell G3 15

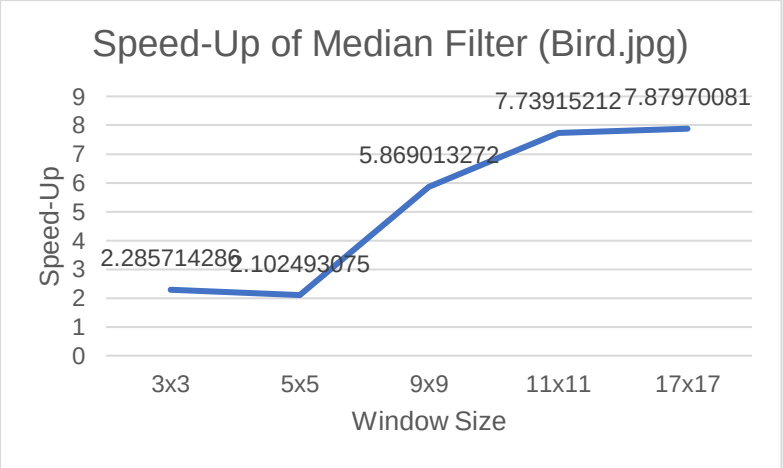
1. *Balloons.png*



2. *Girl.jpg*



3. *Bird.jpg*



2.2. Discussion

2.2.1. What is an optimal sequential cut-off for both parallel algorithms? (Note that the optimal sequential cut-off can vary based on dataset size.)

The optimal sequential cut-off is based on the size of the image. So, for example, if the image is 1920x1080 then the optimal sequential cut-off will be somewhere around this area. If the sequential cut-off is any lower than the image size, then the program will crash.

2.2.2. For what range of data set sizes/ filter sizes do your parallel programs perform well?

Based on the speed-up obtained above, the parallel programs performed excellently within the range of 3 – 21.

2.2.3. What is the maximum speedup obtainable with each parallel algorithm? How do they differ and why? How close is the speedup to the ideal expected?

The maximum speed-up obtained was 68.7 for the mean filter, using the Girl.jpg and a window size of 17.

The maximum speed-up obtained was 9.13 for the median filter, using the Girl.jpg and a window size of 17.

The program seems to perform well on this image and window size for both filter.

2.2.4. How reliable are your measurements? Are they any anomalies and can you explain why they occur?

It is difficult to determine where exactly to start and end timing calls, so there could be a slight inaccuracy in timing between the serial and parallel solutions. Furthermore, the programs that run in the background of the computer can hinder the performance of the algorithm, resulting in anomalies.

3. Conclusions

For both the median and mean filters, there was a speed-up when comparing the serial and parallel solutions. Therefore, for the purpose of mean and median filtering in Java, it is worth designing a parallel solution.