Introduction

In this assignment we are tasked with implementing two filters for smoothing ARGB colour images by means of mean and median smoothing. We also want to distinguish the advantage of implementing parallelism in programming efficiency. We then look forward to making this possible by creating a serial and parallel program that performs both the mean and median smoothing algorithm, then comparing the serial to parallel performances of both. We are also interested in finding out which of the two smoothing algorithms is more computationally expensive.

Method

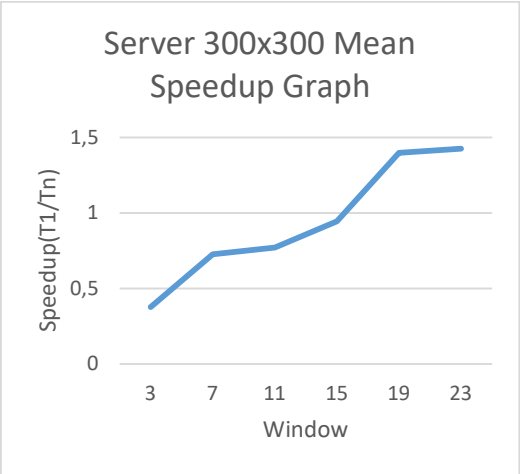
In doing the experiment, we developed 4 programs mainly, MeanFilterSerial.java, MeanFilterParallel.java, MedianFilterSerial.java, and MedianFilterParallel.java. These programs helped in comparing the performance of both the Serial and Parallel programs. There were helper classes such as the ImagePixels.java that is responsible for reading and writing of images. MeanThread.java and MedianThread.java classes which are thread classes for MeanFilterParallel.java and MedianFilterParallel.java classes respectively. The parallel algorithm is implemented using the ForkJoin framework, divide and conquer approach, whereby I recursively check after every divide which is greater between the height or width then split with respect to the greater one into 2 equal halves then send each chunk to a different thread until a sequential cutoff of 1000 pixels is met then sequential perform the Mean and Median filtering on both respectively. The 1000 pixels sequential cutoff became the optimal sequential cutoff after I had explored different cutoff values as it resulted in the greatest speedup.

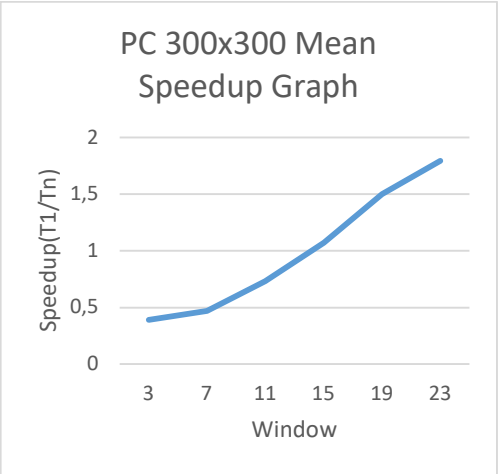
In the main classes I have 2 methods tic() and toc() which I call exactly before and after my filtering processes implementation respectively to obtain the time the filtering processes take regardless of other program processes like obtaining and printing the images. I then ran my programs on the CS Nightmare Servers (8 core) and my PC (i5 8 core), gathered data from both then plotted each of their speedup graphs altering filtering windows and image dimension sizes below.

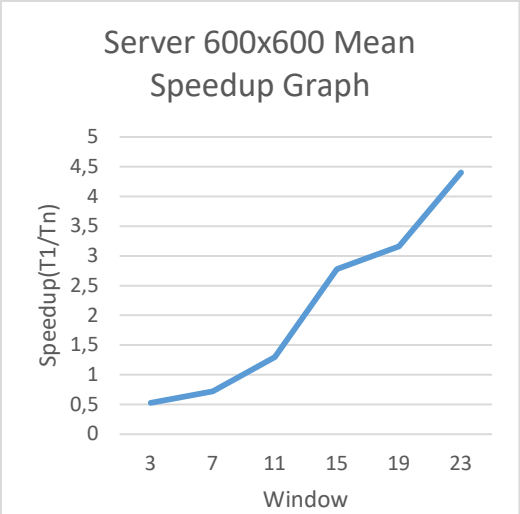
When I was still splitting my image in one way at first, which was only with respect to height only, my program would run forever for some image dimensions resulting in a stack overflow error. This was because for large enough images dividing it down from one dimension will never end you in having pixels less than the value of the width number of pixels. So, to accommodate images with width pixels greater than the sequential cutoff I had to choose an algorithm that splits the images in both height and width.

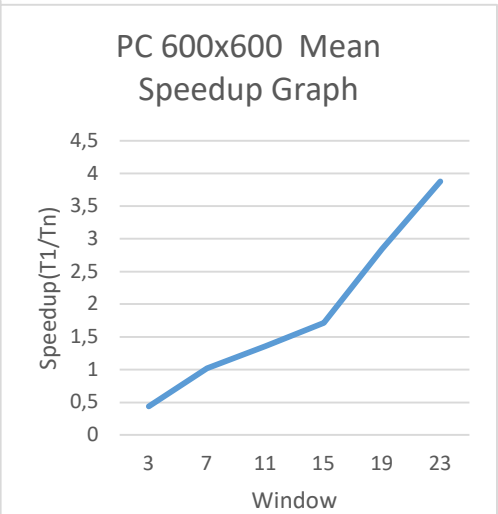
Results

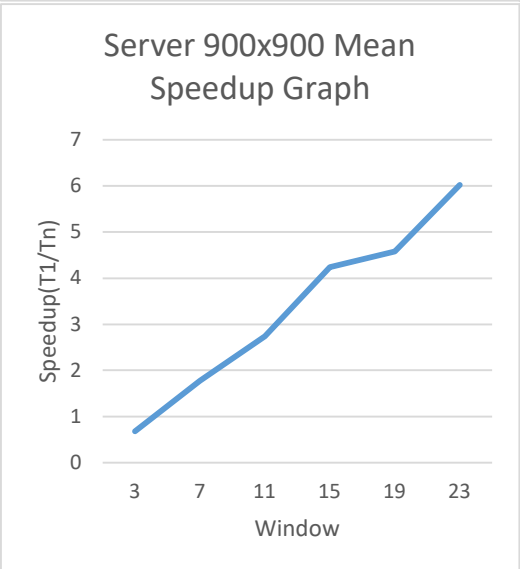
Mean Filters Speedup Graphs

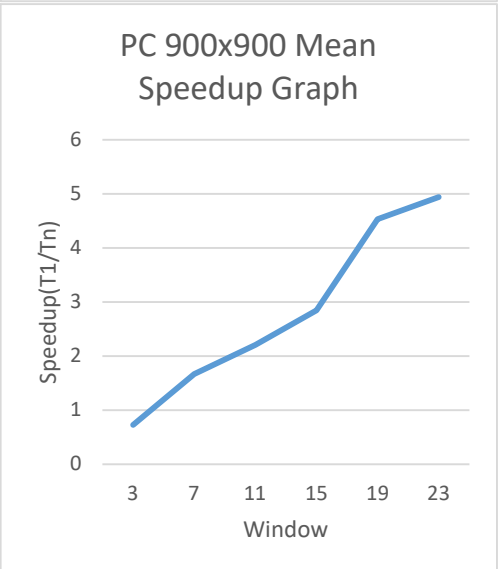

Server 300x300 Mean Speedup Graph
Speedup(T1/Tn)
1,5
1
0,5
0
3
7
11
15
19
23
Window

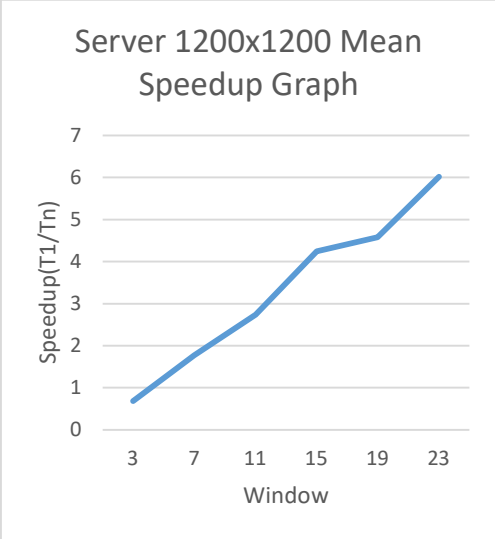

PC 300x300 Mean Speedup Graph
Speedup(T1/Tn)
2
1,5
1
0,5
0
3
7
11
15
19
23
Window

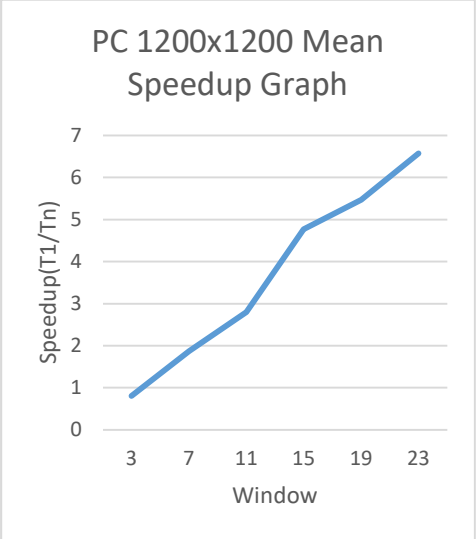

Server 600x600 Mean Speedup Graph
Speedup(T1/Tn)
5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
3
7
11
15
19
23
Window

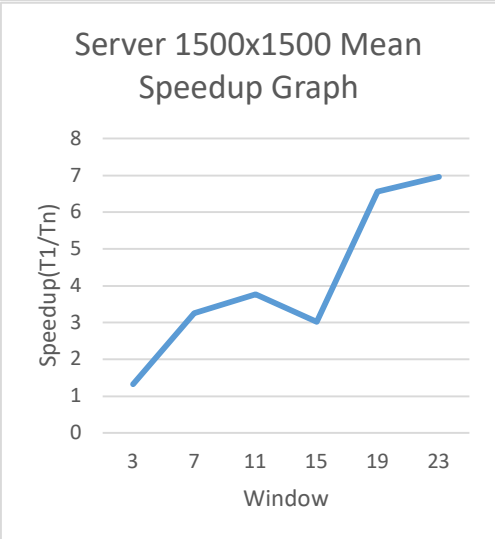

PC 600x600 Mean Speedup Graph
Speedup(T1/Tn)
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
3
7
11
15
19
23
Window

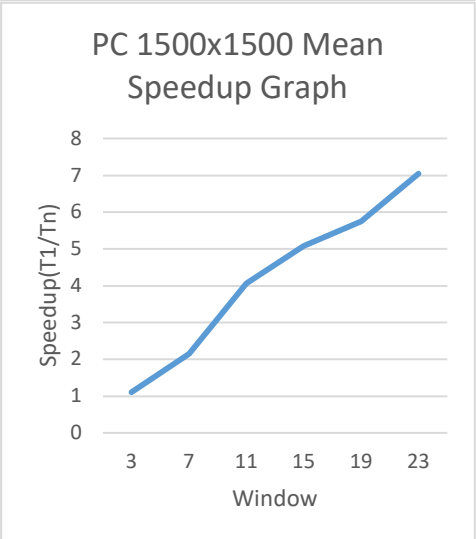

Server 900x900 Mean Speedup Graph
Speedup(T1/Tn)
7
6
5
4
3
2
1
0
3
7
11
15
19
23
Window

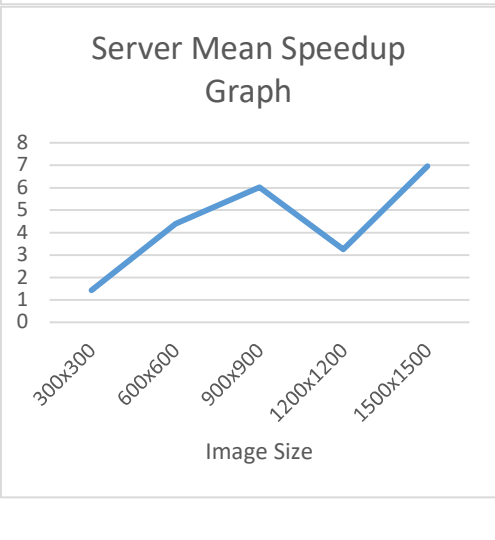

PC 900x900 Mean Speedup Graph
Speedup(T1/Tn)
6
5
4
3
2
1
0
3
7
11
15
19
23
Window

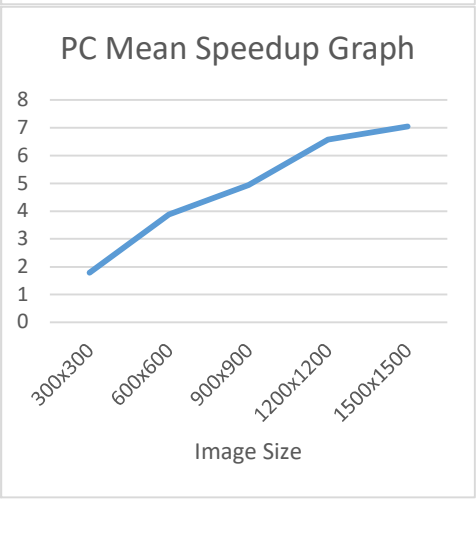

Server 1200x1200 Mean Speedup Graph
Speedup(T1/Tn)
0
1
2
3
4
5
6
7
3
7
11
15
19
23
Window


PC 1200x1200 Mean Speedup Graph
Speedup(T1/Tn)
0
1
2
3
4
5
6
7
3
7
11
15
19
23
Window

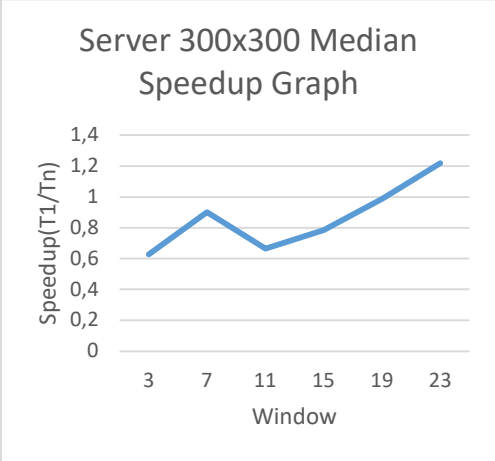

Server 1500x1500 Mean Speedup Graph
Speedup(T1/Tn)
0
1
2
3
4
5
6
7
8
3
7
11
15
19
23
Window

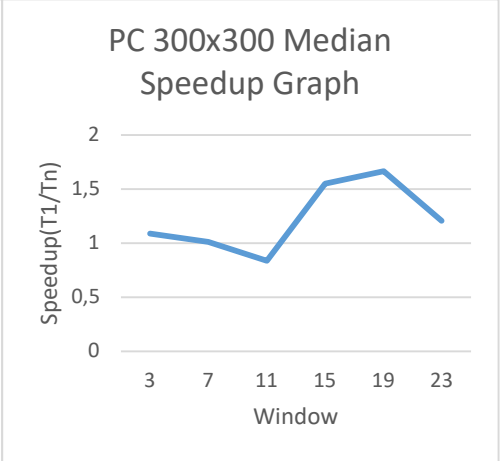

PC 1500x1500 Mean Speedup Graph
Speedup(T1/Tn)
0
1
2
3
4
5
6
7
8
3
7
11
15
19
23
Window

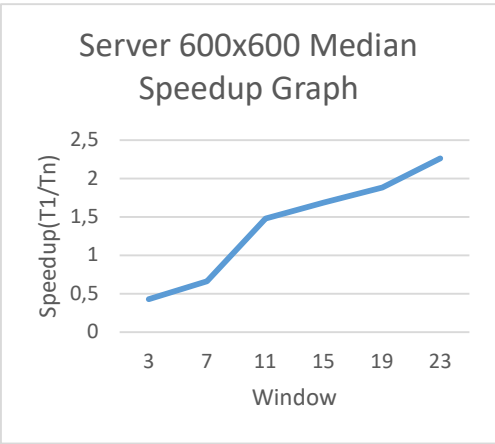

Server Mean Speedup Graph
0
1
2
3
4
5
6
7
8
300x300
600x600
900x900
1200x1200
1500x1500
Image Size

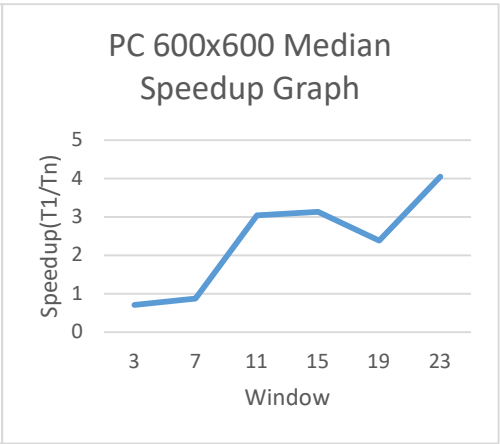

PC Mean Speedup Graph
0
1
2
3
4
5
6
7
8
300x300
600x600
900x900
1200x1200
1500x1500
Image Size

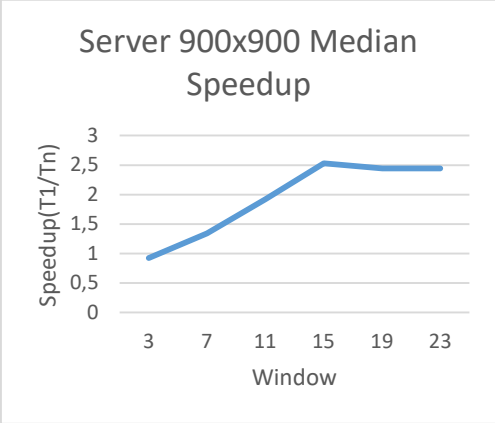
Median Filters Speedup Graphs

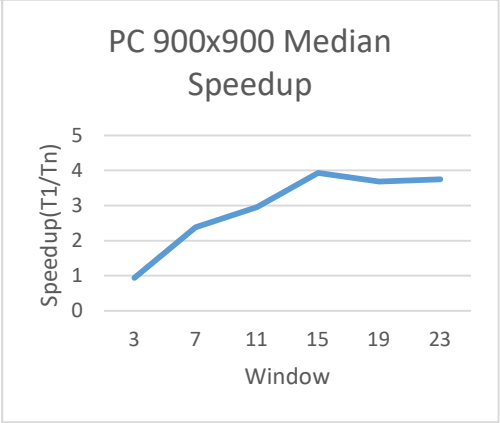


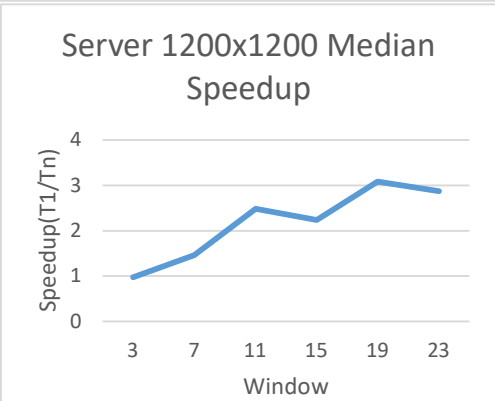


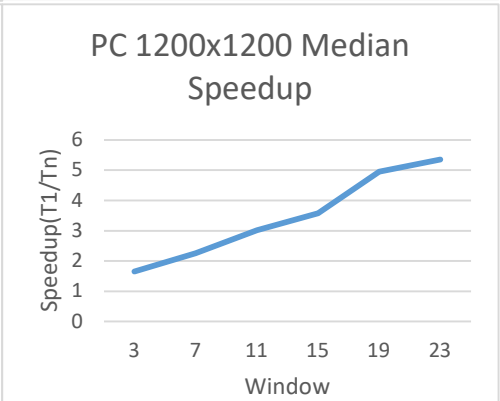


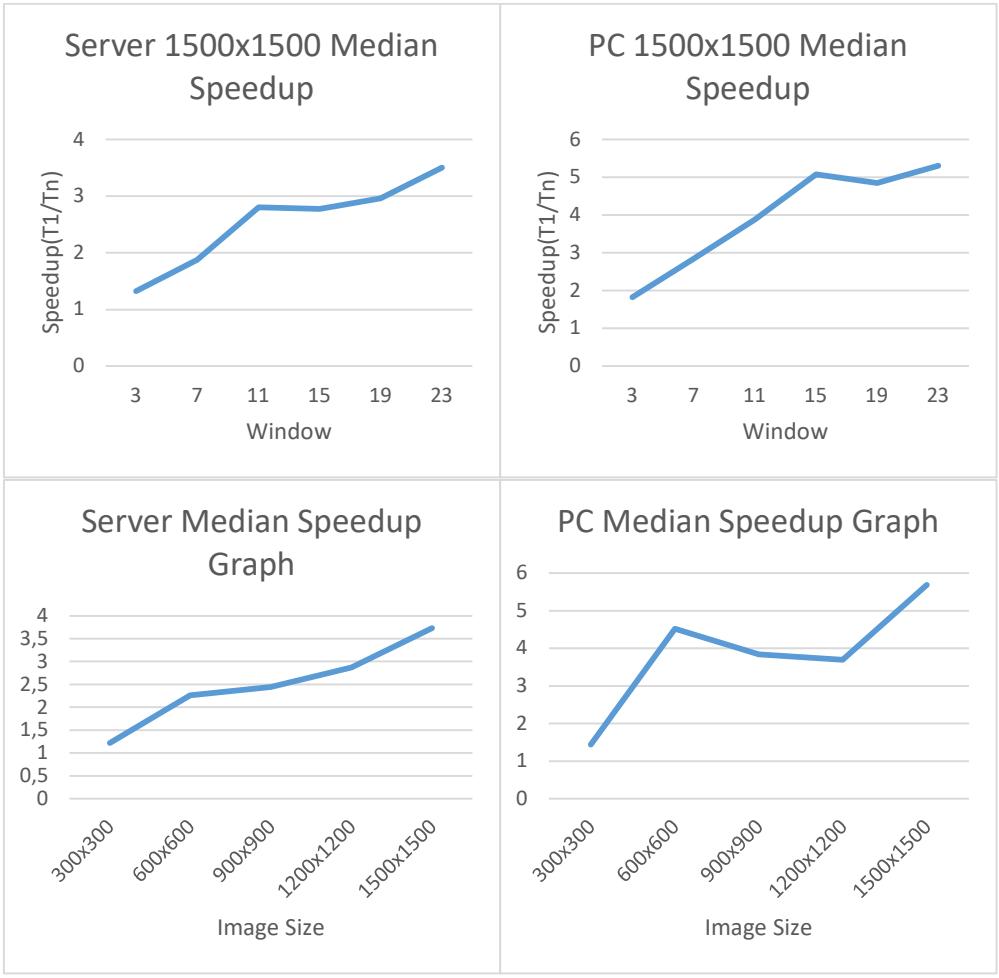



Server 900x900 Median Speedup
Speedup(T1/Tn)
3
2,5
2
1,5
1
0,5
0
3 7 11 15 19 23
Window


PC 900x900 Median Speedup
Speedup(T1/Tn)
5
4
3
2
1
0
3 7 11 15 19 23
Window


Server 1200x1200 Median Speedup
Speedup(T1/Tn)
4
3
2
1
0
3 7 11 15 19 23
Window


PC 1200x1200 Median Speedup
Speedup(T1/Tn)
6
5
4
3
2
1
0
3 7 11 15 19 23
Window



Above are results of the speedups of both the Mean and Median filters for different image sizes at different window sizes. They both best performed at a sequential cutoff of 1000 pixels and hence all the above graphs are results at the sequential cutoff of 1000 pixels.

Median Filtering proved to be more computationally demanding than Mean Filtering as the highest execution time values were observed from its results. On both the Server and PC the speedup proved to on average be directly proportional to the window size and image size. The maximum speedup were values observed at the highest image and window size being 7 for the Mean Filters and 3.5 and 5.3 for the Median Filters on the Server and PC respectively. This however proved to respect the theoretical speedup rule that T1/Tn < or = n with n being equal to 8 in our experiment.

The maximum speedup values for the Mean Filter were greater than those of the Median Filter and that is due to the higher computational expensiveness of the Median Filtering.

I would say my measurements to be accurate and reliable as they only measured the Filtering processing time of both programs only and not other processes in the classes. I also didn't obtain speedup values such that the speedup rule doesn't. There were anomalies in my program as my speedup graphs weren't always consistently increasing as expected and this was because the number of cores to be used by the program alter from time to time and that is why we can't say how many cores will be used.

Conclusion

After having had performed the experiment I conclude that Parallelisation is efficient for obtaining speedup when dealing with significantly large computational problems. The results showed without doubt that we obtain speedup when we increase the problem size whether it's by increasing image or window size. From the experiment we also confirm that Median Filtering is more computationally expensive than Mean Filtering as we obtain at most bigger execution time with Median Filtering than Mean Filtering at constant problem sizes.