Introduction and Theory

Median filtering is a nonlinear digital filtering technique that is often used to remove noise from a data set. In this method, the median filter slides over the data sequentially and replacing each value with the median of its neighbouring entries (the number of which is specified by the filter size). In this assignment, I am going to investigate the effect of parallelising the algorithm used to filter the noise out of the data set. The basic algorithm to filter the result set is as follows (in pseudo code):

```
## loop through the data set and apply the filter to each
element
for each element in dataset{
   element = calculateMedianOf( element and its neighbours );
   ## the number of neighbours to be used depends on filter size
}
```

The problem with this naïve approach is that, because the size of data set is exceptionally large, going through the data set sequentially will take a very long time since the algorithm's runtime is ***O(n)*** and ***n*** is too large to be ignored in this case.

To improve the speed of this algorithm, I am going to parallelise it using the *Java Fork/Join Framework* to give an ***O(log(n))*** runtime. Parallelising will allow me to process different parts of the data set at the same time so that the whole data set is filtered faster. I am going to compare the performance of the sequential approach and the parallel approach, using different filter sizes, data set sizes and also on different computer architectures in an effort to produce the most reliable results and conclusive conclusion to the research.

My predictions for this research are:

- The parallelised program should be much faster than the sequential one for the same filter size and computer architecture.
- The speed of both approaches with small data sizes should not differ by a big margin since *Big O* analysis is not reliable for small "*n*".
- The program's performance should decrease with increasing filter sizes.
- The program's performance should increase with increasing cut off size

Methods

A lot of factors have to be considered to investigate the performance of Parallel vs Sequential approach. The ultimate goal is to see how the Parallel processing approach behaves compared to the Sequential processing approach but to do this, I have to determine the best conditions and create controls to reach this final goal. The architecture details of the "machines" I used to run tests for my research are shown below:

NB: I found a way to set the number of cores a program can use so for one and three cores, I used my computer to record those experiments. This is also an advantage because the only changing variable in the tests is the number of cores. CPU cache size, RAM, COU Architecture and all other specs stay the same. This gives more reliable results for the experiments and tests.

Table 1: Table showing the specs of the computer used in the research

	HP PAVILION DV6
PROCESSOR	Intel Core i7
CORES	4 cores @ 2.20 GHz each (8 logical processors)
RAM	8GB DDR3
CPU ARCHITECTURE	64-bit

The following lays out the steps I took in my research. For steps 1 to 4, I used only my machine (The HP Pavilion DV6) and then the Ultimate Test (step 5) I used other computers including mine. I only used one machine for steps 1-4 because their purpose and results are not necessarily influenced heavily by machine architecture, i.e. their results will be the same regardless of what machine I use. Before I carried out any of the following steps, I had to verify that my program was producing the correct output and that the output for both the sequential and parallel methods were identical. I Started off with small datasets of size 300 and went through them to verify that they were producing the correct results. I created a Junit test that checks if the two outputs from both methods were identical and this test was successful. Only then did I start carrying out these tests.

1. Determine the best number of runs for the program's "warm up".

As I was creating my program and doing random tests, I realised that the program would run extremely slow for the first couple of runs and then starts giving more consistent value. If this issue was not addressed, the results of my research would be very unreliable. To counter this, I ran a couple of tests to determine the most suitable number of runs my program needs to do for a "warm up". For the rest of the tests, the tests and exercises, a "warm-up" session will be run before anything else to ensure optimum performance and reliable results.

2. Determine the best and worst filter sizes and Sequential Cut off sizes.

To confirm my predictions from the introduction, I created a Filter Test that filters the same data set using filters of different sizes from size 3 to size 21 for both the sequential and the parallel methods. I created a similar test for the sequential cut-off where I started with a cut-off of 100 all the way up to 15,000 while incrementing by 100 for each run (a total of 100 different cut off sizes). The best performance conditions are when both the filter size and the sequential cut off process the data in the quickest time. The worst case conditions are when the data is processed in the longest time. These conditions will be used in later tests.

3. Determine the most suitable input data to use.

The type of input will have an effect on the performance of the program. For this exercise, I will determine the best input to use for the Ultimate Test (4) using the data sets provided in the assignment. I will also filter the data sets given with the assignment to analyse the trend they give.

4. Ultimate Test

For this ultimate test, I used my Custom Dataset of size 2 million lines which I have determined to be the most suitable and use as control for the main speed up tests. I optimised my program to use the best conditions based on the previous tests (which will be stated in the results section). For this section, I created a test that uses the Custom Data Set to create input sizes ranging from 50,000 all the way up to 4,000,000 in increments of 50,000 (That is a total of 80 data sets of different sizes).

For each speed-up test, the program had the usual warm-up run to ensure consistent and reliable data. The program will time how long it take to filter each data set (by filtering the same data set 50 times and finding the average time to filter the data set) using both the sequential and parallel methods. After these runs, the program exports the data to a .csv file that can be analysed in Microsoft Excel. This data will basically consist of the time taken to filter each data set of different sizes from 50,000 lines to 4,000,000 lines for both the sequential and parallel methods.

This Ultimate Test is going to be carried out on computers with different architecture to determine how the number of processors affect the speed up of the program.

5. Additional Test: Using MEAN filter instead of MEDIAN filter

I carried out an additional test at the end. For this Test, I used the same conditions I used in the Ultimate Test but in this case, instead of using a median filter, I used a mean filter. This filter uses the average of surrounding values instead of the median. I do not expect the speed up to be as significant as the speed up you get when using the median filter. This is because less computations are required for finding the average compared to finding the median. This is because arrays have to be sorted first when finding the median for median filtering.

Test Results

1. Determine the best number of runs for the program's "warm up".

For this test, I ran the program 15 times to see how the run times vary with time and I obtained the following results:

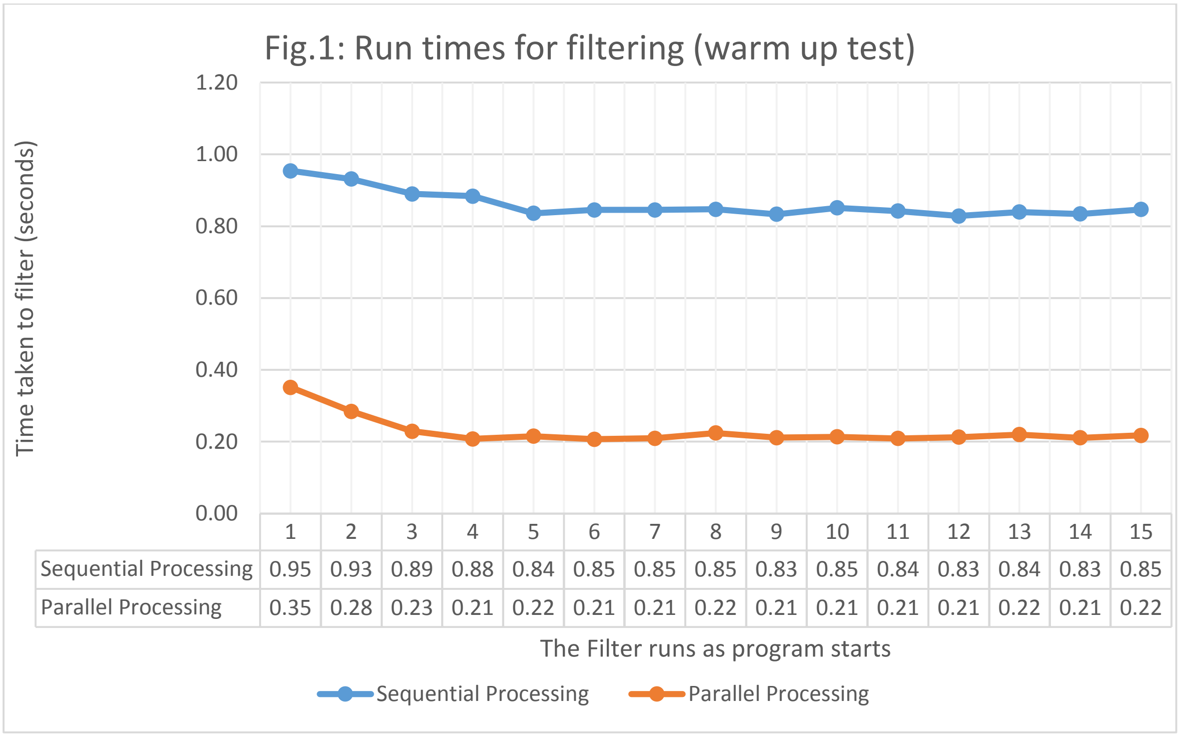


Fig.1: Run times for filtering (warm up test)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequential Processing	0.95	0.93	0.89	0.88	0.84	0.85	0.85	0.85	0.83	0.85	0.84	0.83	0.84	0.83	0.85
Parallel Processing	0.35	0.28	0.23	0.21	0.22	0.21	0.21	0.22	0.21	0.21	0.21	0.21	0.22	0.21	0.22

This table shows the running times of the program as it starts from the first run of the filter to the 15th run. As shown in the program, the runtime at the start is high and not reliable but it eventually levels out by the time it gets to the 6th run therefore for the rest of my tests, I ran the program 8 times as my warm up before carrying out the main test.

The data set I used was ***input4*** given from the assignment, filter size was 3 and the sequential cut off was 1000 and both of these were just arbitrary values I chose. They do affect the output which I was observing (in this case, the number of runs needed for a warm up). The remainder of my tests do a warm up round of 8 runs before carrying out the actual test(s).

2. Determine the best and worst filter sizes and Sequential Cut-Offs.

Filter Size:

For this test, I filtered the data with each filter size from 3 to 21 to assess how the filter size affects the running time of the filtering. For each filter size, I filtered the data set 20 times and calculated the average running time for that specific data size. The results are shown in the graph below.

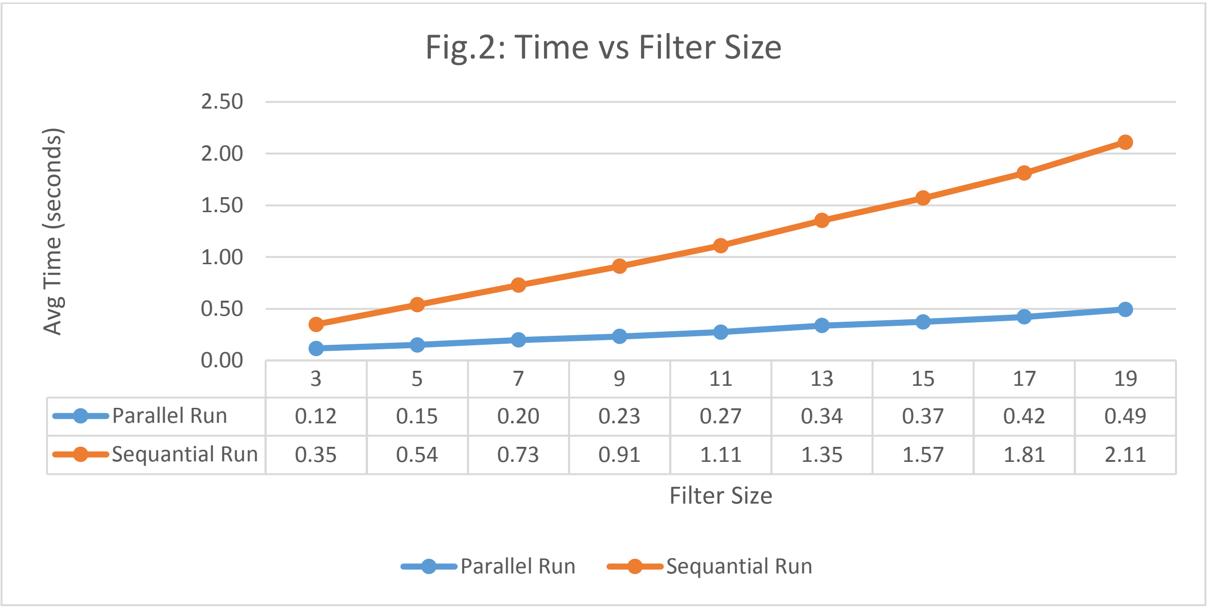


Fig.2: Time vs Filter Size

	3	5	7	9	11	13	15	17	19
Parallel Run	0.12	0.15	0.20	0.23	0.27	0.34	0.37	0.42	0.49
Sequantial Run	0.35	0.54	0.73	0.91	1.11	1.35	1.57	1.81	2.11

The sequential cut off I used for the filter size test was 10 000 for all parallel tests.

Sequential Cut-Off:

For the optimum sequential cut offs, I carried out 2 tests. The first one, I used a filter of size 3 and used sequential cut-offs ranging from that range from 500 lines to 100,000 lines (incrementing by 1,000) each time. For each data set test, the data set was filtered 20 times to obtain an average time. The results for both tests are shown below.

Fig.3

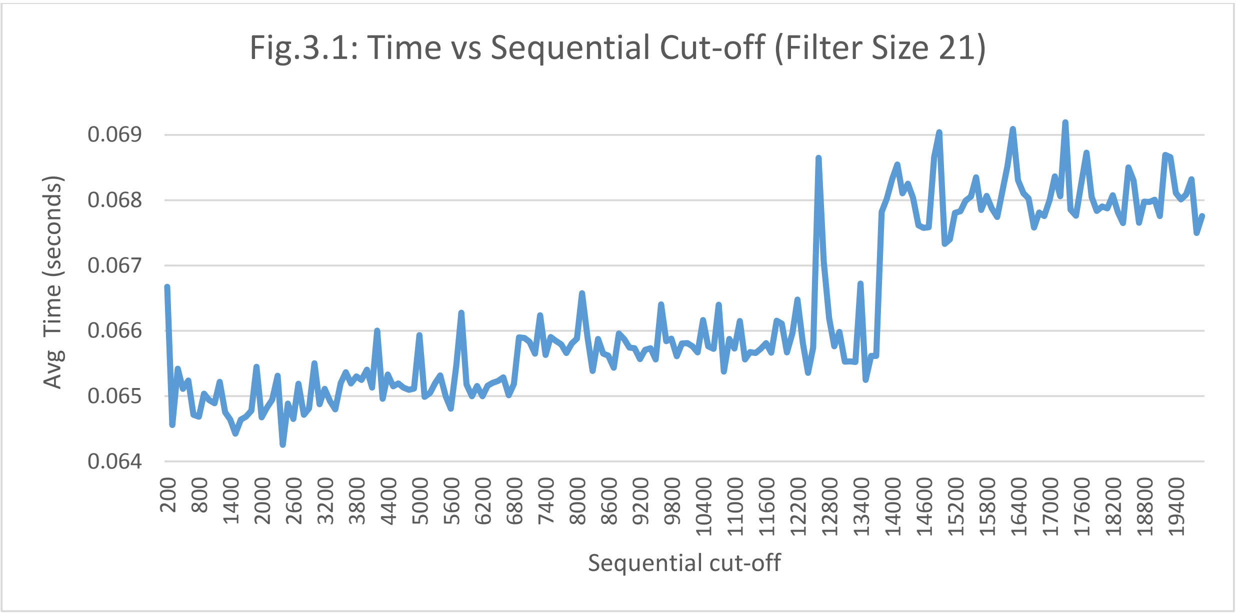
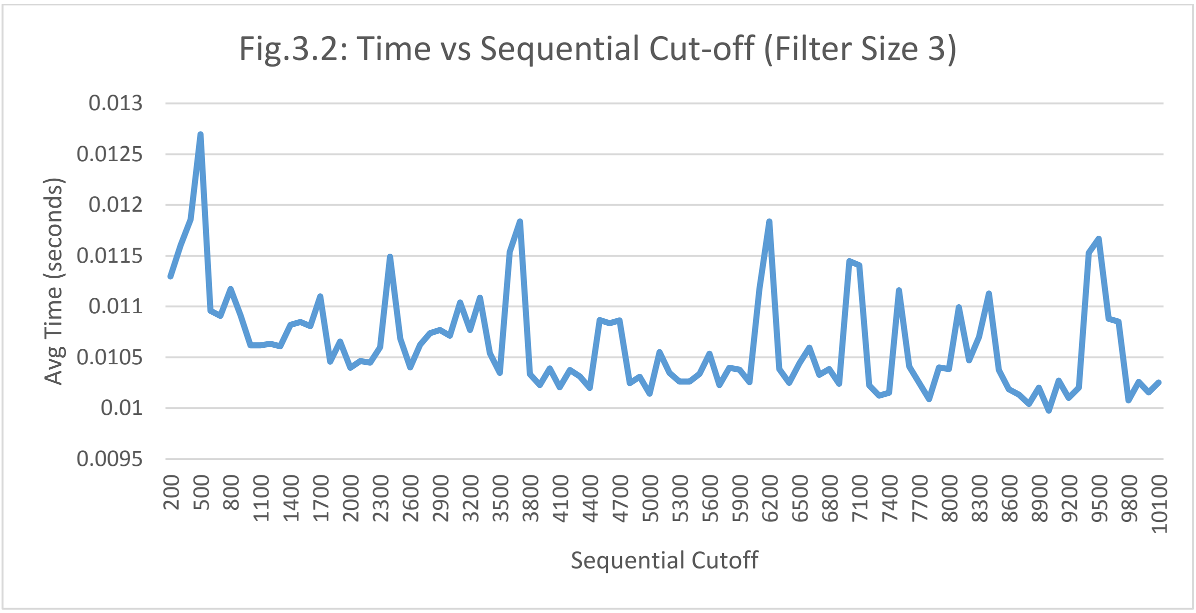


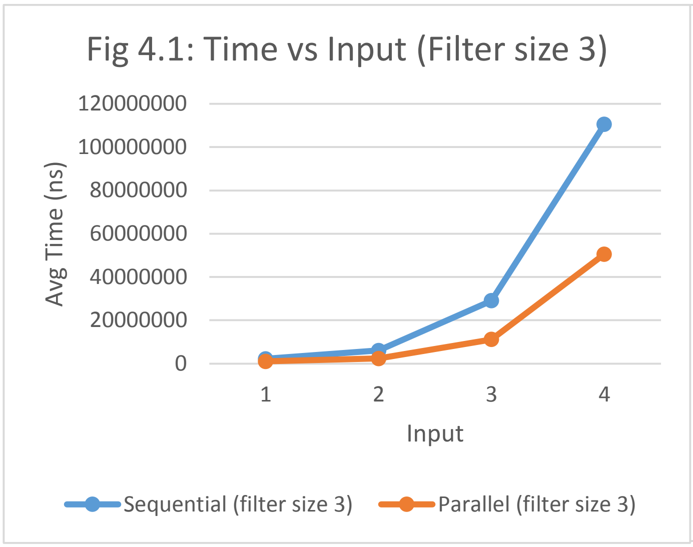
Fig.3.1: Time vs Sequential Cut-off (Filter Size 21)

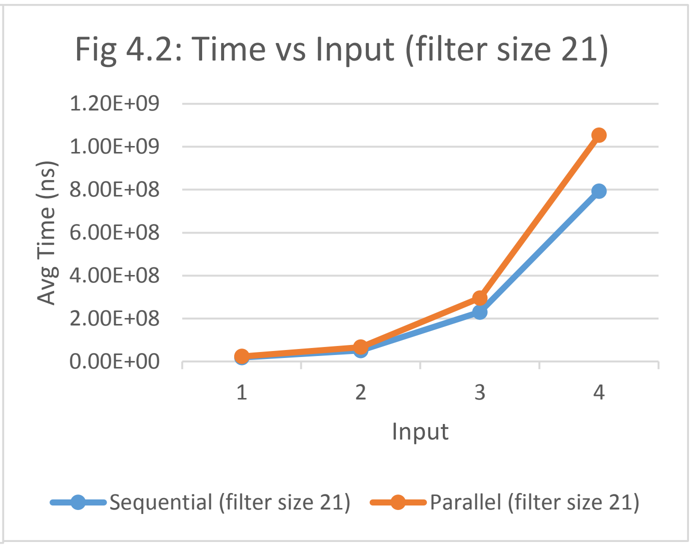


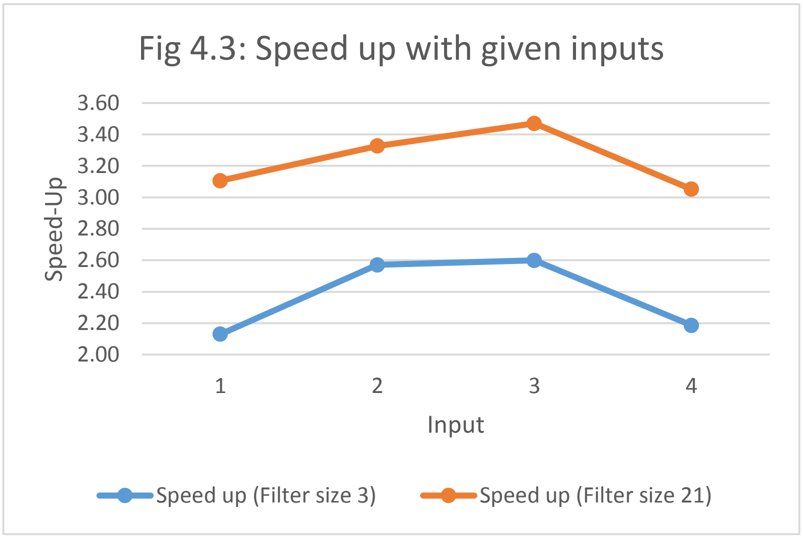
I data set used for these tests was input 3 provided for the assignment. For a filter size of 21, there is a clear trend that as the sequential threshold increases, the filter speed slows down. The best range appears to be between 800 and 2000 but for the following tests, I will use 1500 as my optimum sequential threshold. However, the filter size 3 does not seem to have any good trend (although it generally appears to have an increasing performance as the cut-off increases but more tests would need to be carried out). Partly maybe because the filtering is very fast (more than 5 times faster than filter size 21). So for the rest of the tests I will use my results from filter size 21 as my control.

3. Determine the most suitable input data to use.

I tests my program with the 4 data set provided by the assignment and these are the results I received using my computer, HP Pavilion DV6 (4 cores).







At first glance the data might look odd because (1) you would expect the speed up for input 4 to increase since the data set is bigger than the previous ones and (2) the speed up itself is not x4 as it "should" be on a 4 core CPU.

(1) has turn out to be a result of the type of data inside input 4. The data inside it consists of only negative numbers and all have many decimal places (at least 10 decimal numbers for each value). The other 3 inputs have very few number of decimal place (about 3) and also have a mixture of negative and positive numbers. This inconsistency in the data in the inputs is what has caused the unexpected come out of the experiment.

(2) is a result of the data sizes being used. They are quite small (expect 4) and according to Gustafson's Law, the problem size has too big to have a big speed up. In my case, not only were the problem sizes small, but the sequential threshold was also optimised, giving really good performance.

4. Ultimate Test

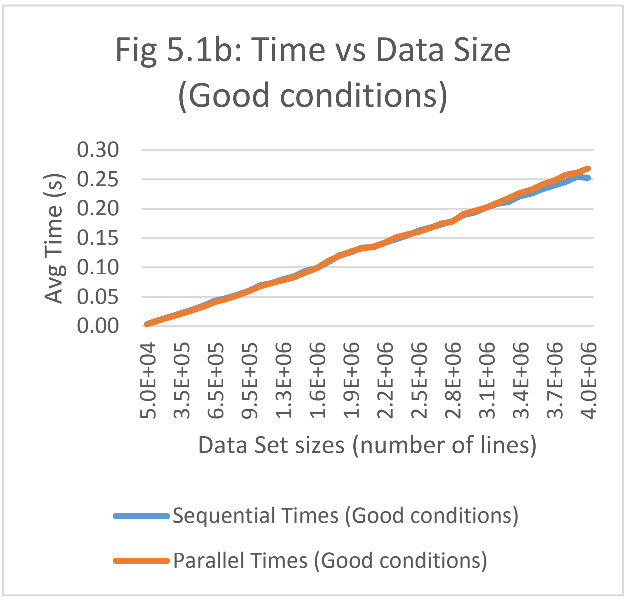
For the ultimate test, it was divided into 2 parts.

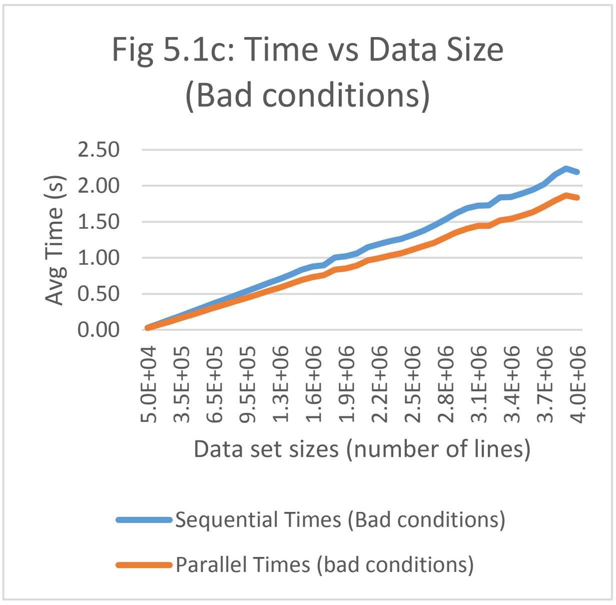
(1) was to use different sized data sets ranging from 15,000 lines to 4,000,000 lines in increments of 50,000 lines (for 3 and 4 cores) and 100,000 lines (for 1 and 2 cores) and filter them with the **BEST** filter (**size 3**) and the **BEST** sequential cut-off (**1,500**). Each data set as usual was filtered 20 times to obtain an average time.

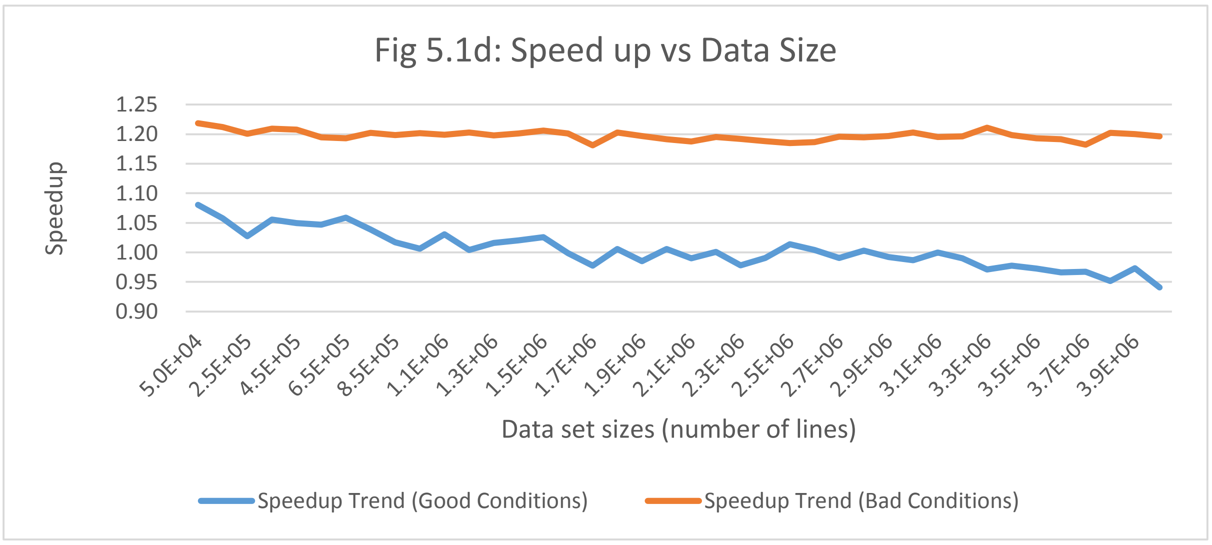
(2) was to use different sized data sets ranging from 15,000 lines to 4,000,000 lines in increments of 50,000 lines and filter them with the **WORST** filter (**size 21**) and the **WORST** sequential cut-off (**100**). Each data set as usual was filtered 20 times to obtain an average time.

I incremented by only 100,000 lines for 1 and 2 core tests because the tests were taking a long time to finish. Both these tests were run on 1, 2, 3 and 4 cores on the same machine for consistency. The results are shown below:

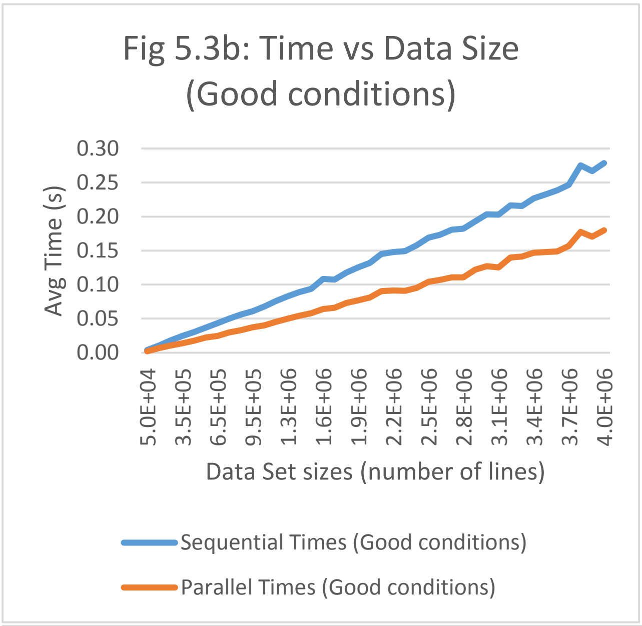
One Core:

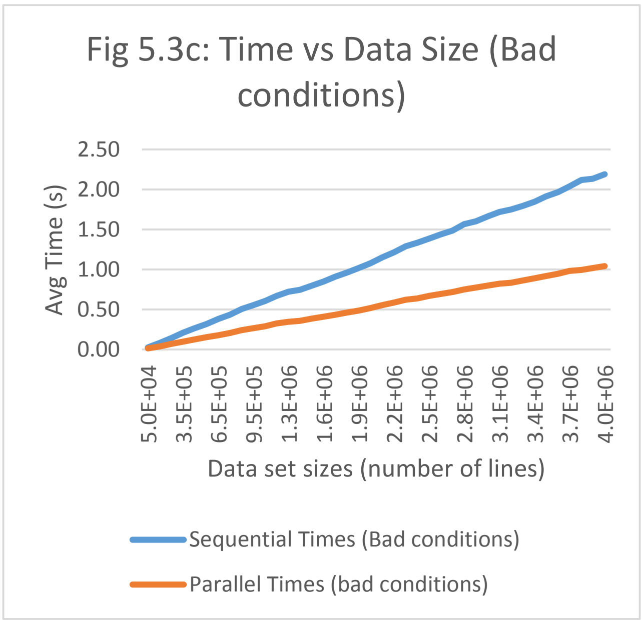


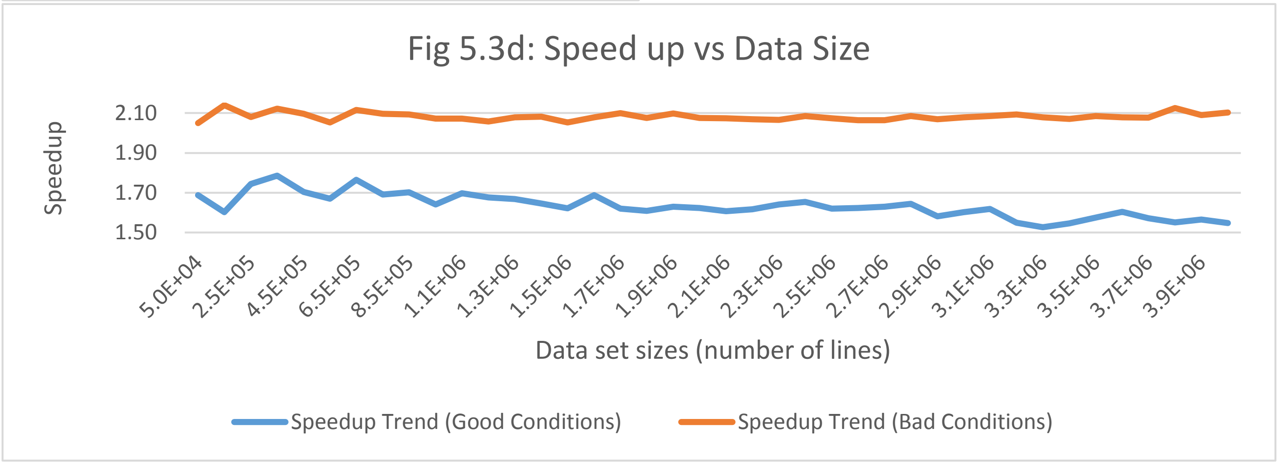




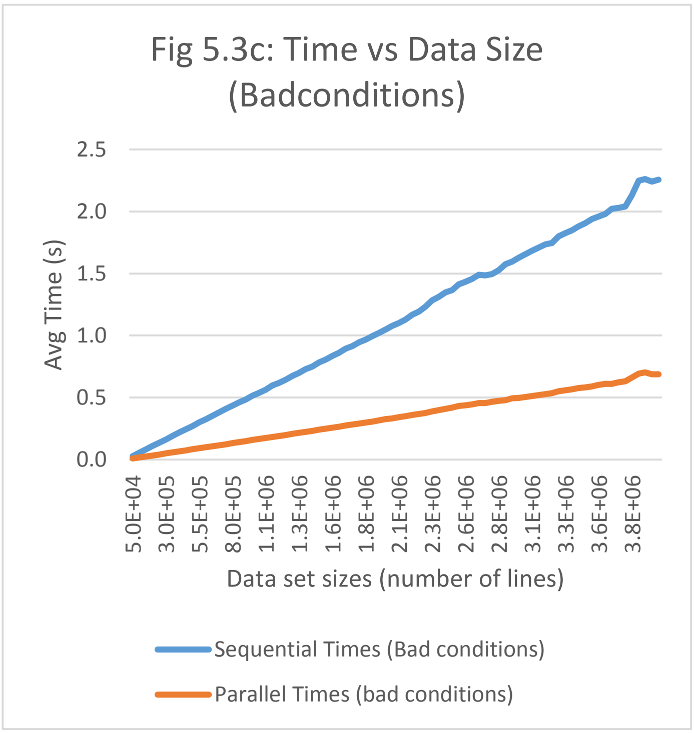
Two Cores:

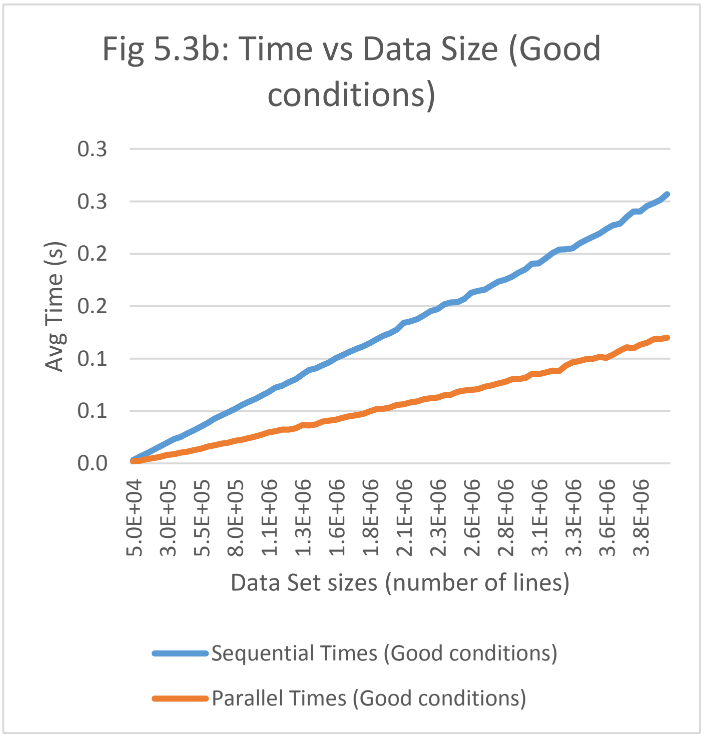






Three Cores:





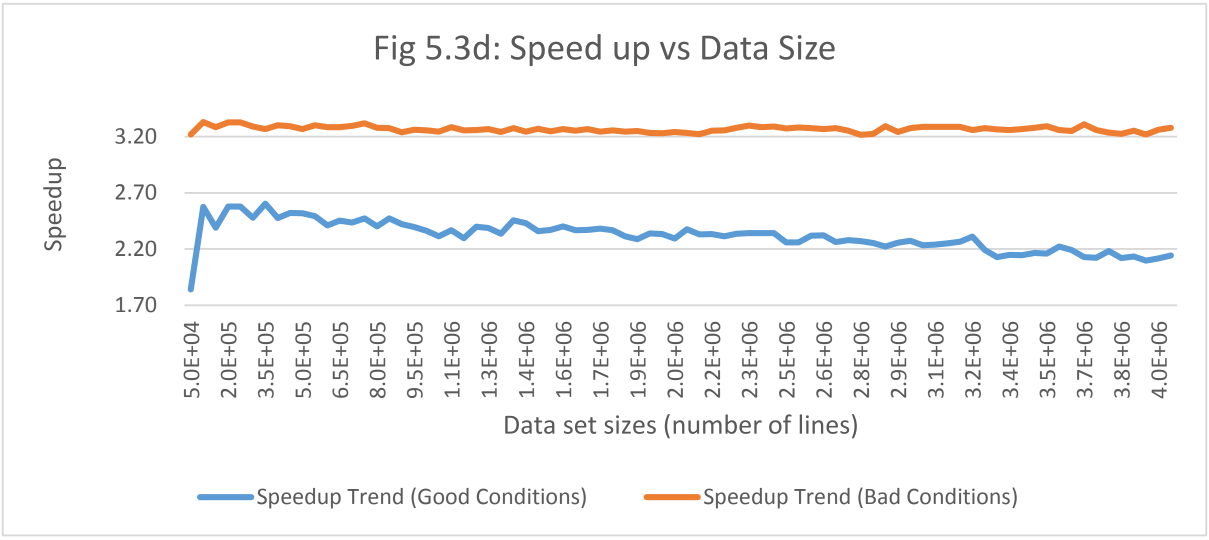


Fig 5.3d: Speed up vs Data Size
Speedup
3.20
2.70
2.20
1.70
5.0E+04
2.0E+05
3.5E+05
5.0E+05
6.5E+05
8.0E+05
9.5E+05
1.1E+06
1.3E+06
1.4E+06
1.6E+06
1.7E+06
1.9E+06
2.0E+06
2.2E+06
2.3E+06
2.5E+06
2.6E+06
2.8E+06
2.9E+06
3.1E+06
3.2E+06
3.4E+06
3.5E+06
3.7E+06
3.8E+06
4.0E+06
Data set sizes (number of lines)
Speedup Trend (Good Conditions)
Speedup Trend (Bad Conditions)

Four Cores:

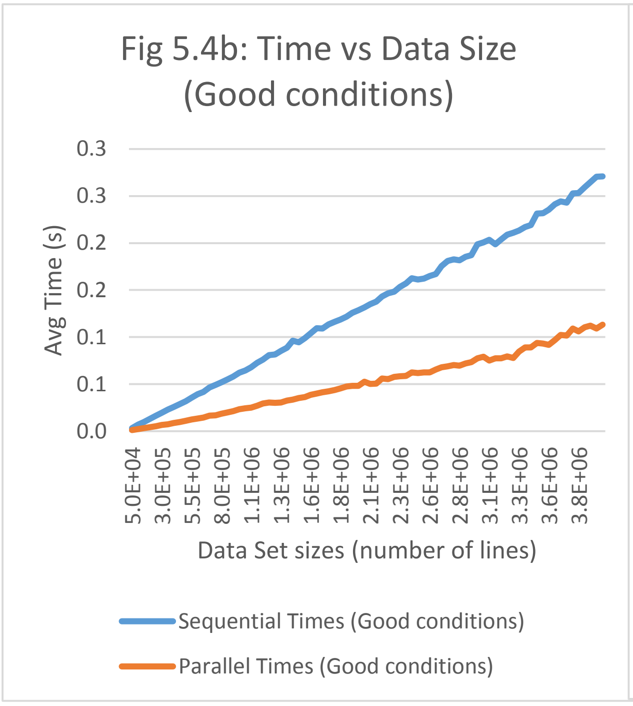


Fig 5.4b: Time vs Data Size (Good conditions)
Avg Time (s)
0.3
0.3
0.2
0.2
0.1
0.1
0.0
5.0E+04
3.0E+05
5.5E+05
8.0E+05
1.1E+06
1.3E+06
1.6E+06
1.8E+06
2.1E+06
2.3E+06
2.6E+06
2.8E+06
3.1E+06
3.3E+06
3.6E+06
3.8E+06
Data Set sizes (number of lines)
Sequential Times (Good conditions)
Parallel Times (Good conditions)

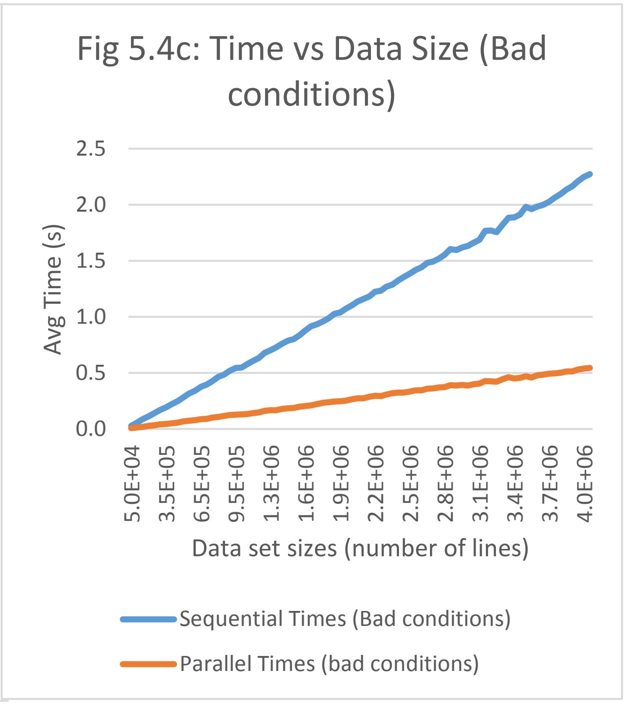


Fig 5.4c: Time vs Data Size (Bad conditions)
Avg Time (s)
2.5
2.0
1.5
1.0
0.5
0.0
5.0E+04
3.5E+05
6.5E+05
9.5E+05
1.3E+06
1.6E+06
1.9E+06
2.2E+06
2.5E+06
2.8E+06
3.1E+06
3.4E+06
3.7E+06
4.0E+06
Data set sizes (number of lines)
Sequential Times (Bad conditions)
Parallel Times (bad conditions)

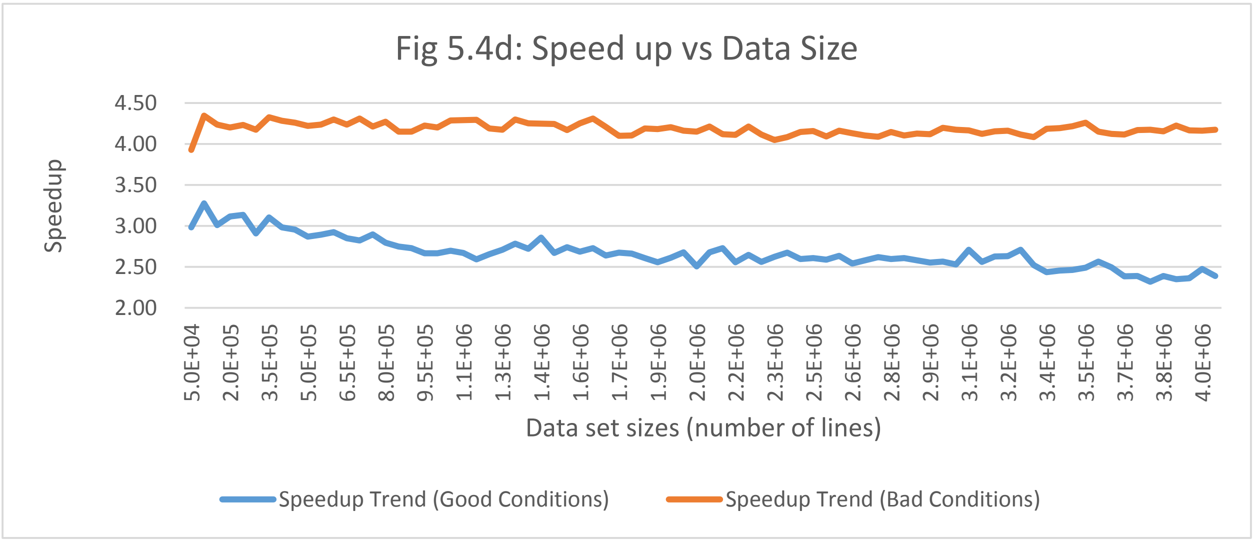
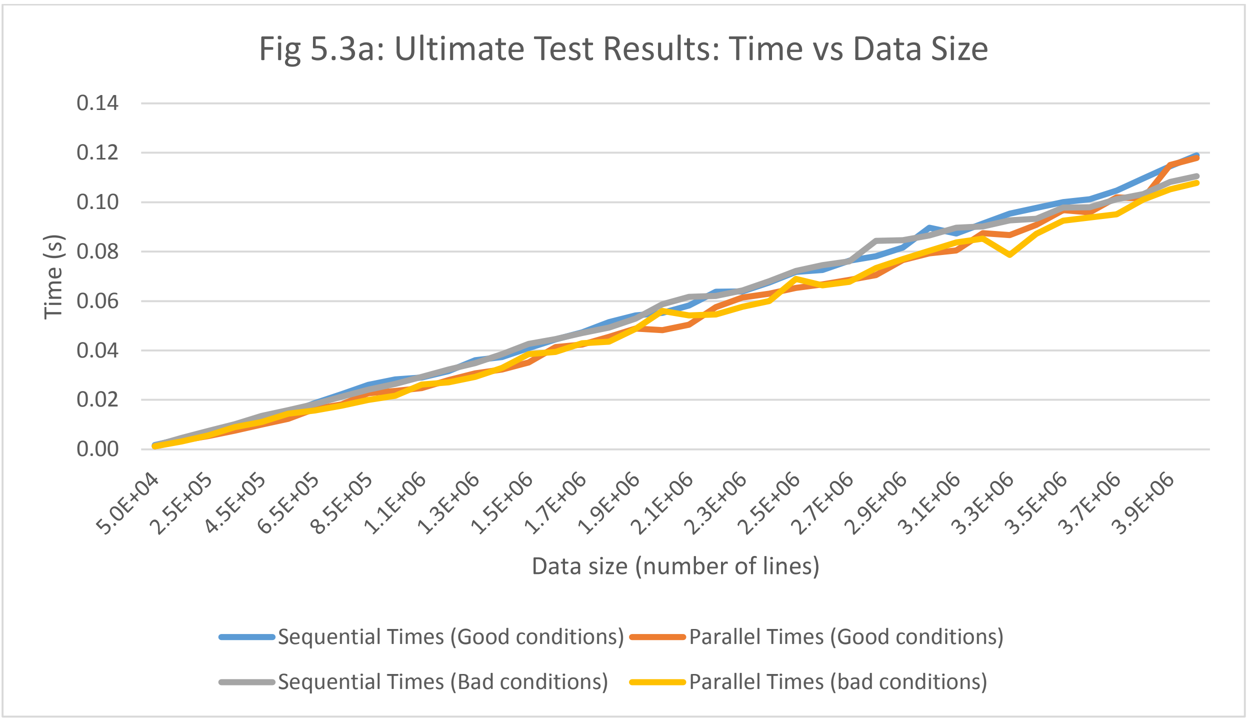
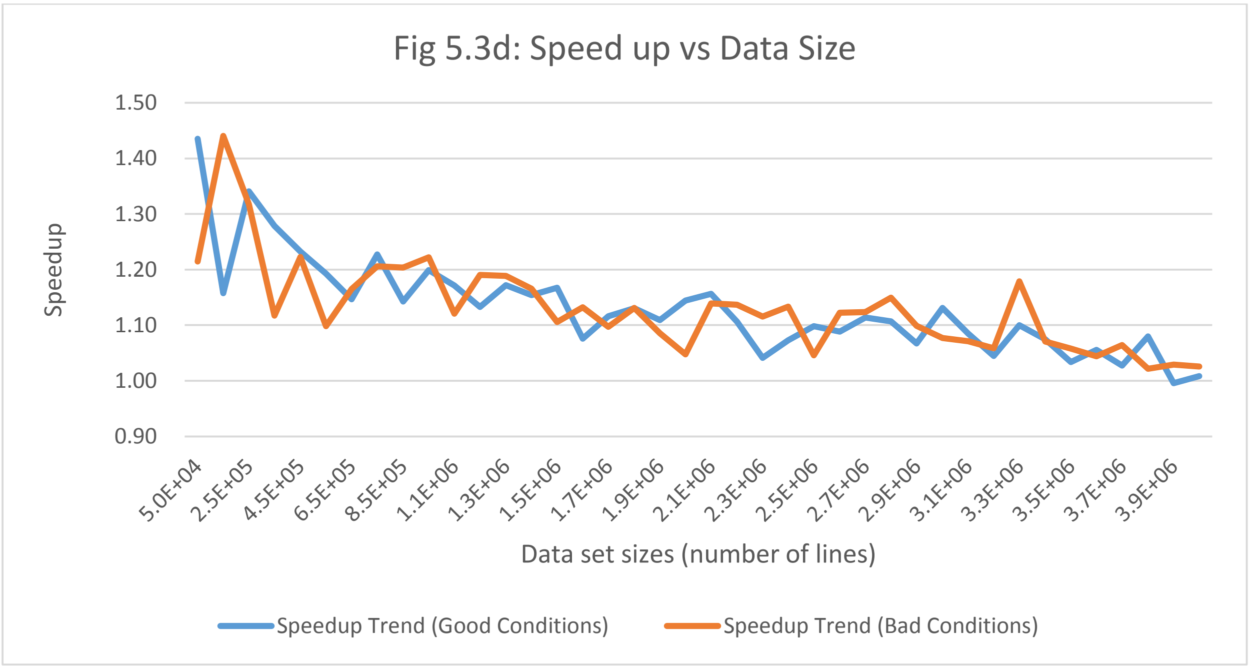


Fig 5.4d: Speed up vs Data Size
Speedup
4.50
4.00
3.50
3.00
2.50
2.00
5.0E+04
2.0E+05
3.5E+05
5.0E+05
6.5E+05
8.0E+05
9.5E+05
1.1E+06
1.3E+06
1.4E+06
1.6E+06
1.7E+06
1.9E+06
2.0E+06
2.2E+06
2.3E+06
2.5E+06
2.6E+06
2.8E+06
2.9E+06
3.1E+06
3.2E+06
3.4E+06
3.5E+06
3.7E+06
3.8E+06
4.0E+06
Data set sizes (number of lines)
Speedup Trend (Good Conditions)
Speedup Trend (Bad Conditions)

5. Additional Test: Using MEAN filter instead of MEDIAN filter

For this additional test, I created a new filter class that filters the data using the mean filter instead tof the median filter. I ran the Ultimate test under the same conditions on 4 cores and the results I obtained are shown below:





Conclusions

Ultimate Tests:

- It is clear that when the smallest filter is being used, with the ideal sequential cut-off, the speed up obtained by using parallel methods isn't as much as the speed up obtained when using a large filter with a bad sequential cut-off. This can be observed from all the tests done in the Ultimate tests on 1 to 4 cores.
- The Size of Filter used does greatly affect the performance of the filter. The smaller the filter size, the fast the program will run but the less speed up you get. The larger the filter, the slower the program runs but the most speed up you get. It may be suitable to use either Filter in certain cases but I would suggest that if a really larger dataset if to be filtered with the smallest filter, it wouldn't be worth it to buy a computer with more than 4 cores because the speed up would not be that much, but it might be worth it to buy a machine with more cores (maybe 8 or 12) because the speed up when using a large filtered is quite significant.
- The Speedup observed when the conditions were "good" i.e, small filter and optimum sequential cut-off, may not be the expected one (for example on four cores the speed up is No more than 3.3 on 4 cores and actually decreases as the data set size increases) but the speed up when conditions were "bad" were as expected (like the x4 speed up on 4 cores). This may be, because the program is executing more instructions with the larger filter (hence a bigger "problem"), the speed up will be much bigger compare to when the program is very efficient (small filter). In a way, this varies both Amdahl's law (speed up isn't much when problem size is small) and Gustafson's Law (speed up would be significant if the problem is big).

Extra Test (Mean Filter):

- The test using a mean filter produced results that were a bit beyond my predictions. I had predicted that the speed up wouldn't be as significant but it turns out there is basically no speed up at all. This may be because of the reasons I mentioned earlier in the methods section. The problem size (in terms of number of calculations) isn't big therefore there wouldn't be much speed up by using parallel methods. In conclusion. I would definitely discourage against investing in a machine with more cores because it would certainly be a waste of money.