1.2 Report: You must submit an assignment report in pdf format. Your clear and concise report should contain the following:

* An Introduction, comprising a short description of the algorithm and why it is worth considering parallelization.
* A Methods section, giving a description of your approach to the solution, with details on the parallelization. This section must explain how you validated your algorithm (showed that it was correct), as well as how you timed your algorithms with different input, how you measured speedup, the machine architectures you tested the code on and interesting problems/difficulties you encountered.
* A Results and Discussion section, demonstrating the effect of data sizes, filter sizes and sequential limits on parallel speedup. This section should include speedup graphs and a discussion. In this section, we expect you to should answer the following questions:
* Is it worth using parallelization (multithreading) to tackle this problem in Java?
* For what range of data set sizes and filter sizes does your parallel program perform well?
* What is the maximum speedup obtainable with your parallel approach? How close is this speedup to the ideal expected?
* What is an optimal sequential cut-off for this problem? (Note that the optimal sequential cut-off can vary on dataset size and filter size.)
* A Conclusions section (note the plural) listing the conclusions that you have drawn from this project. What do your results tell you and how significant or reliable are they?

# 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:

[[[Table showing architectures.]]]

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.

**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.

Test Results

**1. Determine the best number of runs for the program's “warm up”.**

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

**3. Determine the most suitable input data to use.**

**4. Ultimate Test**

**Conclusions**