# Assignment 3 report

## Introduction

The aim of this project is to determine the difference in running time between a sequential and parallel program across various data sizes.

Both programs will read in a file containing sunlight hours of 1 million trees across a 3000 X 3000 metre square landscape.

The size of the dataset will be varied and the runtime will be measured for both programs.

For the parallel program, the sequential cutoff will also be varied.

The parallel program will use a fork-join algorithm, therefore increasing the number of threads and getting  a faster runtime.

Using the data, a graph of sequential cutoff against time and speedup against size will be drawn.

The programs will run on 2 different machine, the first one is a core i7-77HQ(8 cores) and the other is a core i5-3317U(4 cores).

The goal of this project is to find the speedup across the various dataset size to determine how parallelism affects the runtime.

Methods

1. Sequential program

For the sequential program, the input file is read and the sunlight values are inserted in a 2d array. This 2d array will represent the canopy. The coordinates and span of each tree is then read and their respective total sunlight hours in calculated and inserted in an array.

Each element of the array is added to get a total and this total is then divided by the total number of trees to obtain the average sunlight values for all trees. The values required are then printed into a file. The summation and division is what is being measured. It was measured using the java nanoTime to obtain an accurate value.

The size of the dataset was varied and the average calculated. To vary the dataset, new arrays were created, containing 2 times, 4 times, 8 times and 16 times the amount of data from the original. This was done using a method that duplicates the values in an array. . Using this method is very convenient because since the same values are copied into the arrays, the average when divided by the number of elements will still be the same. This was a way to determine if the method was working and it was successful. The program was then run and the runtime measured.

To determine if the program was correct, the average sunlight value calculated was compared to a given value. Both values were the same indicating that the program was correct.

2. Parallel program

The parallel program is similar to the sequential program and only differs in the summation part. For the summation, the parallel program will create multithreads, the number of which will depend on the sequential cutoff. The various threads will perform a part of the calculations and the result will be added to obtain a total. This total will be divided by the number of trees to obtain the average. The parallelized summation is what is being measured in this case, still using nanoTime.

This time also new arrays containing 2 times, 4 times, 8 times and 16 times the amount of data from the original were created and the runtime to calculate the average was measured in each case.

When tested, the parallel program gave a similar result for the average as the given value indicating that the program was correct.

With the runtime values obtained, the speedup was then calculated using the formula:

Time for sequential program using data size x/Time for parallel program using data size x

The program was executed using the 2 different processors and each speedup graph was plotted.

**Results and discussions**

The detailed results are in the core-i7 and the core-i5 files.

The result when running the sequential program was as such. The heading is the dataset size. The average and time is also shown.

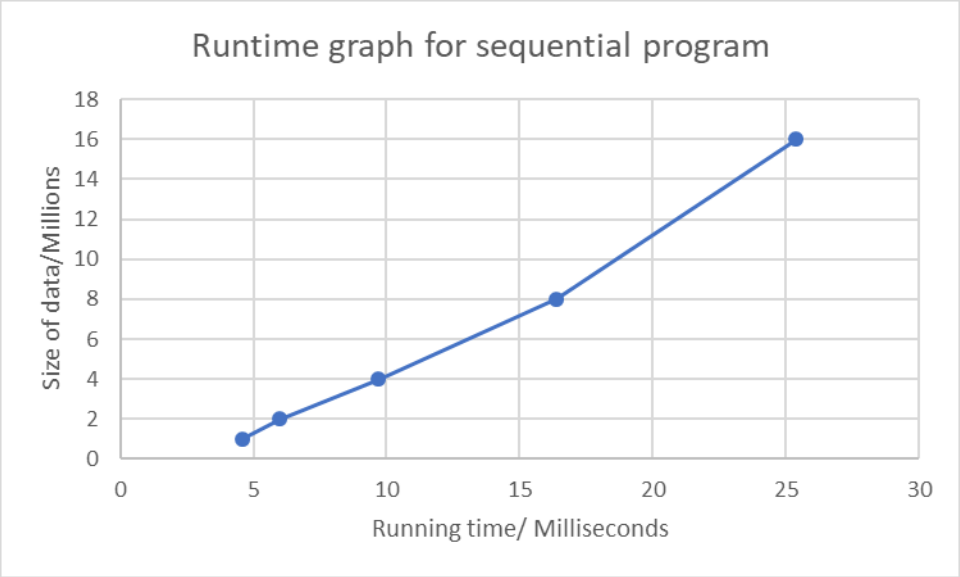
```
1 million
Sequential run
    Average sunlight
    133.0873195
    4.625509 Milliseconds

2 million
Sequential run
    Average sunlight
    133.0873195
    5.962034 Milliseconds
4 million
Sequential run
    Average sunlight
    133.0873195
    9.705764 Milliseconds
8 million
Sequential run
    Average sunlight
    133.0873195
    16.421211 Milliseconds

16 million
Sequential run
    Average sunlight
```

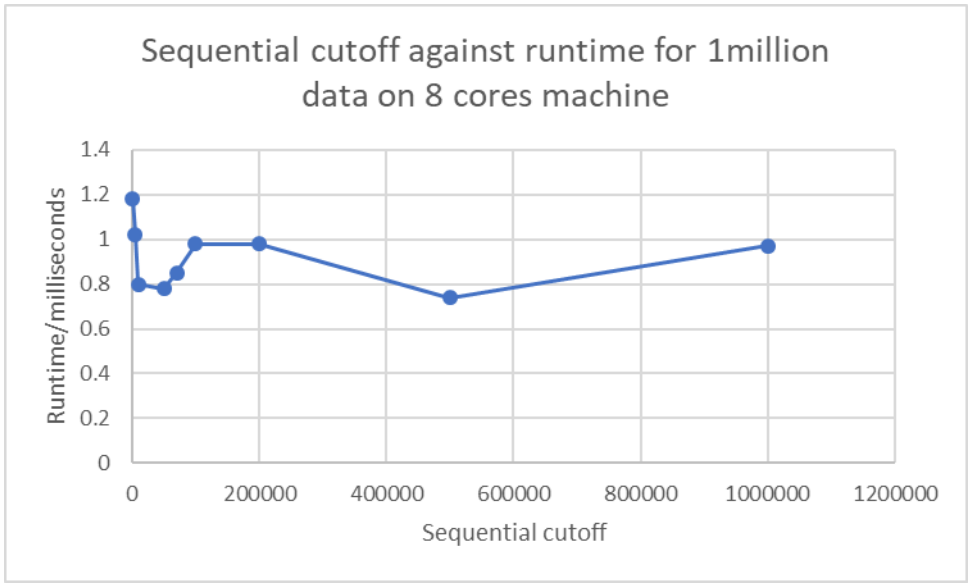
```
133.0873195
25.395077 Milliseconds
```

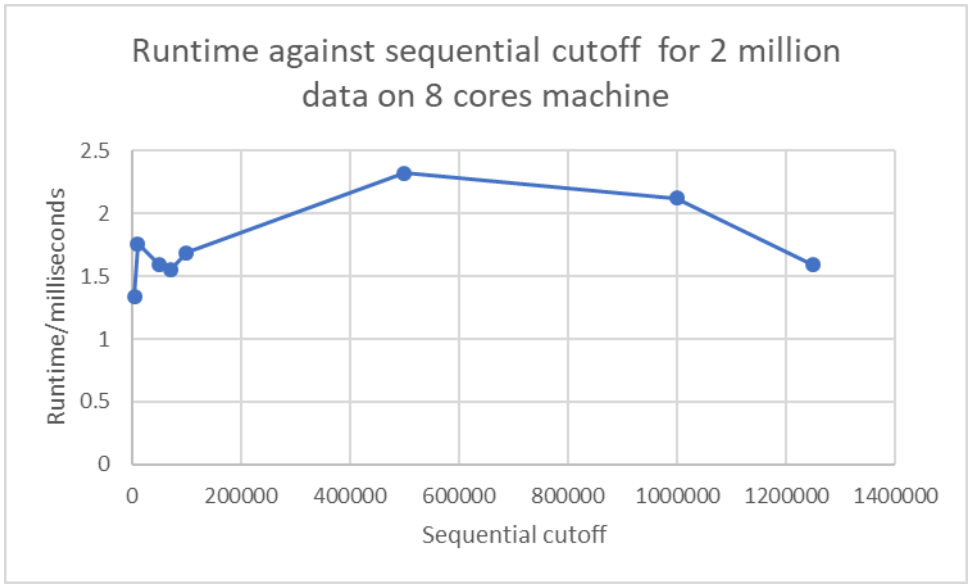
The following graph was obtained for the sequential program.

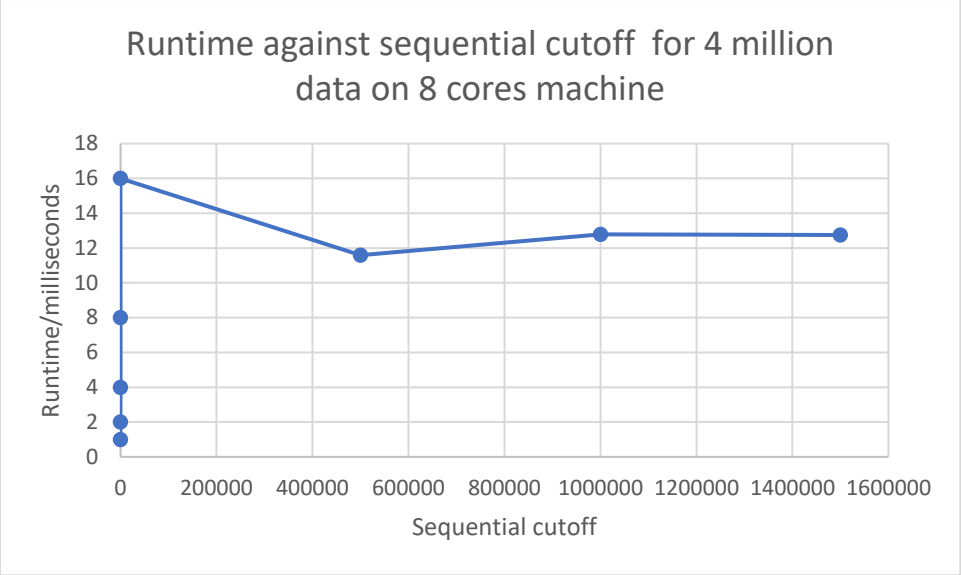


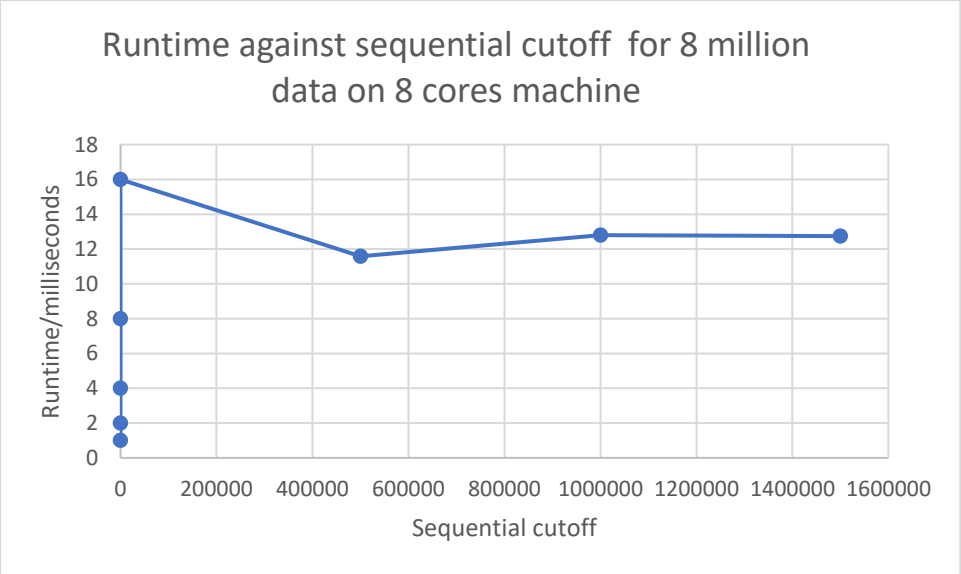

We can see that the time increases almost linearly with the size of the dataset with the runtime almost doubling when the dataset in doubled. This is because the program is being ran on a single core. The single core has to do all the work which is why the time increases.

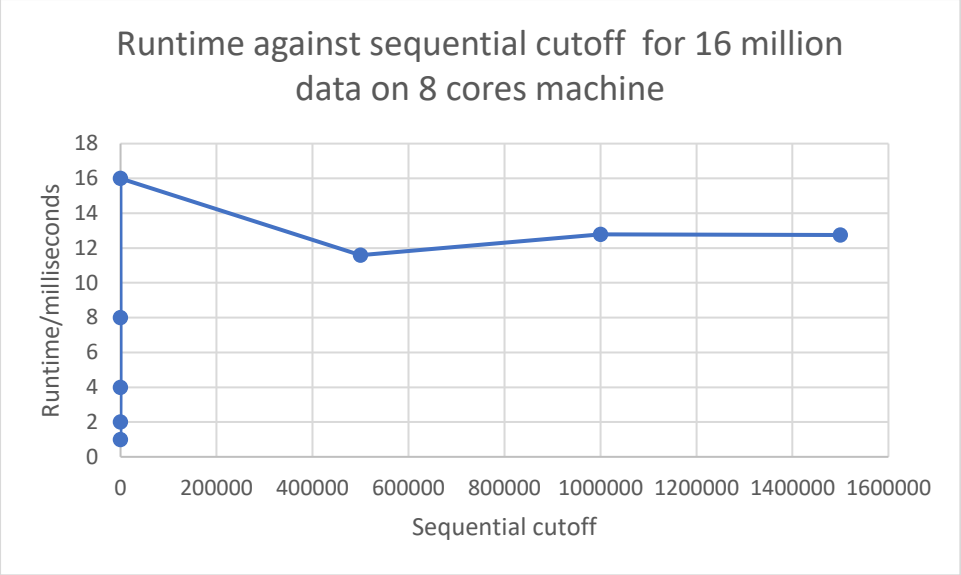
The parallel program was ran on the 8 cores(i7-77HQ) and the graphs of sequential cutoff to runtime was obtained.


Sequential cutoff against runtime for 1million data on 8 cores machine
Runtime/milliseconds
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
200000
400000
600000
800000
1000000
1200000
Sequential cutoff


Runtime against sequential cutoff for 2 million data on 8 cores machine
Runtime/milliseconds
2.5
2
1.5
1
0.5
0
0
200000
400000
600000
800000
1000000
1200000
1400000
Sequential cutoff

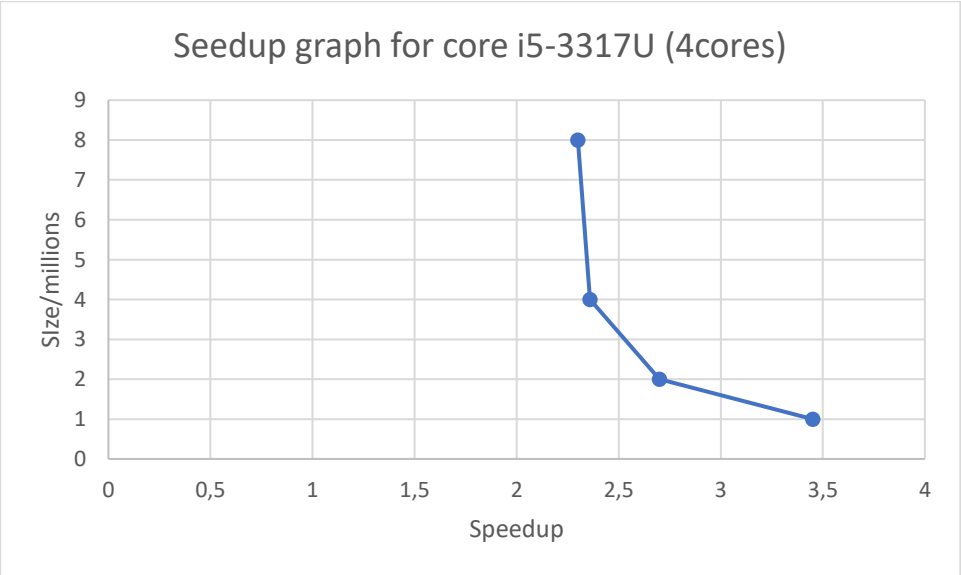

Runtime against sequential cutoff for 4 million data on 8 cores machine
Runtime/milliseconds
18
16
14
12
10
8
6
4
2
0
0
200000
400000
600000
800000
1000000
1200000
1400000
1600000
Sequential cutoff


Runtime against sequential cutoff for 8 million data on 8 cores machine
Runtime/milliseconds
18
16
14
12
10
8
6
4
2
0
0
200000
400000
600000
800000
1000000
1200000
1400000
1600000
Sequential cutoff

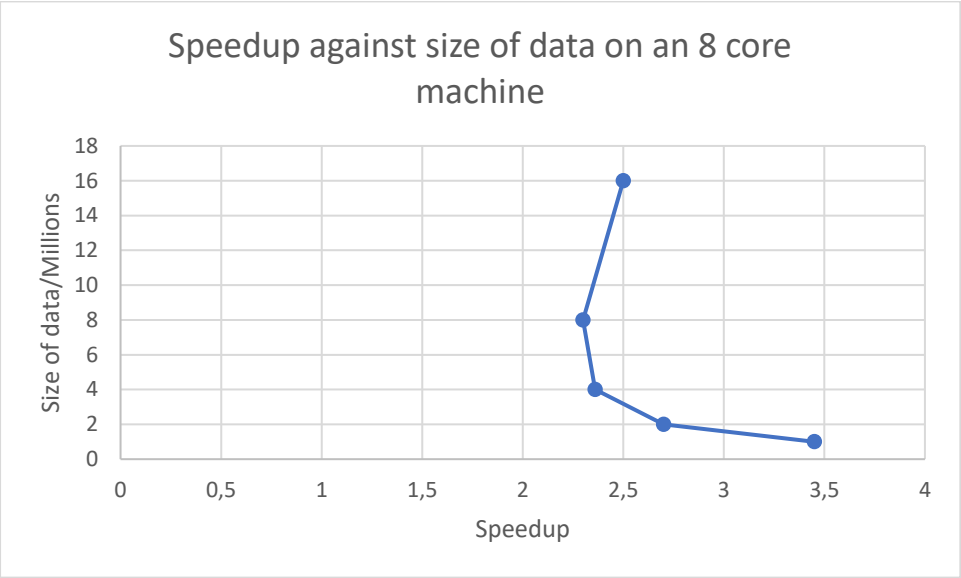



The program was run 5 times and the median value was plotted against the sequential cutoff and the graphs above were obtained . From the graphs we can see that the sequential cutoff is almost always in the 0-200000 range for the best runtime value with the closest value being 70000. This is because this values produces the ideal amount of threads for the 8 cores. This optimal workload for each cores allows the program to run the fastest.

Now we will see the speedup graphs.



The graph above this the speedup graph for the 4 cores. We can see that the speedup is greater when the size of the dataset is smaller.




The graph above is the speedup graph for the 8 core machine. The curve follows a similar trend for both machines. The 8 core machine does produce better results than the 4 core machine.

The graph shows a speedup of at least 2 for both machines, this is because the multithreading allows the other cores to work and therefore obtaining the total faster.

The optimal number of threads for the 4 core machines in 8 and for the 8 core machine, 16. This provides a good division of the total work and is not too many. If too many threads are used, the time taken to load each thread will increase the runtime.

This problem does not really need a parallel program as we can see from the data that the maximum runtime for the maximum amount of data for the sequential program  is only 25 milliseconds and the median runtime for the maximum amount of data for the parallel

program is 10 milliseconds. This provides a speedup of 2.5 which is good but since the runtime is short, it does not really matter.

**Conclusions**

From this experiment we can conclude that using parallelism does decrease the runtime and that is true for every data size.

We can also see that an 8 core machine will produce better speedup compared to a 4 core machine.