CSC2000S Assignment1

By: Lindani Khuzwayo

Khzlin012

INTRODUCTION

This report will focus on comparison of parallelism and sequential execution of the same algorithm. Firstly, one need to understand parallelism in computer terms. A parallel algorithm is one that executes multiple instructions at the same time on different processing devices then combines the results to produce one output. While a sequential algorithm is one that executes instructions one after the other in the order of the actual code from top to bottom. This project aims to investigate and compare the time taken by both the sequential and parallel algorithm hypothetically the parallel one is faster for the same amount of work done due to the splitting of the work load.

METHODS

The aim of the project as defined above is to compare the time taken to execute the same instructions by the parallel and sequential algorithm. Given a set of input data of different sizes. One has to apply a Median filter on the data set and measure the time it takes to processes the data with a parallel and sequential algorithm. The approach used was to filter raw data given the filter size and the size of the array one needs for storing input data. The actual filtering process is to implement a median filter which slides over the data, item by item, generating a new data set, where each item is replaced by the median of the neighbouring entries. Thus, the size of the filter which is already specified, determines the number of neighbours considered for the median. The filter size can range between 3 and 21.

Creating two separate programs is the simplest way for one to be able to compare the performance of each of the programs. Therefore, a thread class namely Median.java and Main.java is created for the parallel implementation Main.java is the driver class and Median.java implements parallelization using the Java Fork/Join framework to parallelize the median filter using a divide and conquer algorithm. On the other hand, MainSequentail.java was the sequential implementation of the median filter with no threads to help it run faster.

Tests were run and data was collected. For testing the run time for each different “SEQUENTIAL_THRESHOLD” in intervals of 1000, 10000, 100000 and so on was smaller for the parallel algorithm Main.java than time taken to run the sequential algorithm MainSequentail.java. this proves the hypothesis mentioned above to be true. For each “SEQUENTIAL_THRESHOLD” different sample sizes for input data set were used for example, for SEQUENTIAL_THRESHOLD=1000 running the two algorithms for sample size n=100,1000,10000and so on and recording the time taken for each run by each algorithm. The same procedure was done for SEQUENTIAL_THRESHOLD=10000 and so on. Results are show and discussed under RESULTS AND DISCUSSION

RESULTS AND DISCUSSION

Input data size	Sequential time	Parallel time
n=100	Run took 0.0044 milliseconds	Run took 0.0025 milliseconds
n=1000	Run took 0.0294 milliseconds	Run took 0.0093 milliseconds
n=10000	Run took 0.4543 milliseconds	Run took 0.0672 milliseconds
n=100000	Run took 15.1607 milliseconds	Run took 2.9736 milliseconds
n=1000000	Run took 23.9122 milliseconds	Run took 8203.865 milliseconds

SEQUENTIAL_THRESHOLD=1000 filter size is 3

Input data size	Sequential time	Parallel time
n=100	Run took 0.0035 milliseconds	Run took 0.0011 milliseconds
n=1000	Run took 0.0313 milliseconds	Second Run took 0.0086 milliseconds
n=10000	Run took 0.5453 milliseconds	Run took 0.0834 milliseconds
n=100000	Run took 15.5887 milliseconds	Run took 2.9596 milliseconds
n=1000000	Run took 23.9019 milliseconds	Run took 8598.366 milliseconds
n=10000000		

SEQUENTIAL_THRESHOLD=5000 filter size is 3

Input data size	Sequential time	Parallel time
n=100	Run took 0.005 milliseconds	Run took 0.0026 milliseconds
n=1000	Run took 0.0338 milliseconds	Run took 0.017 milliseconds
n=10000	Run took 0.4348 milliseconds	Run took 0.0675 milliseconds
n=100000	Run took 13.814 milliseconds	Run took 3.1445 milliseconds
n=1000000	Run took 23.9453 milliseconds	Run took 8598.366 milliseconds
n=10000000		

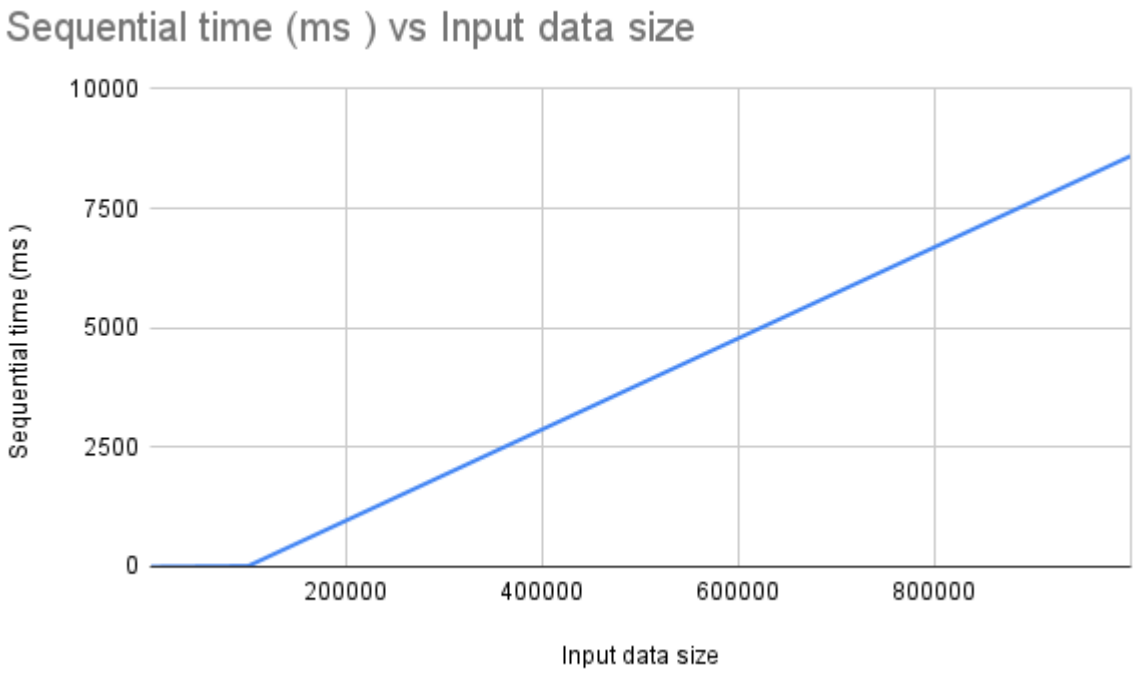
SEQUENTIAL_THRESHOLD=10000 filter size is 3(above)

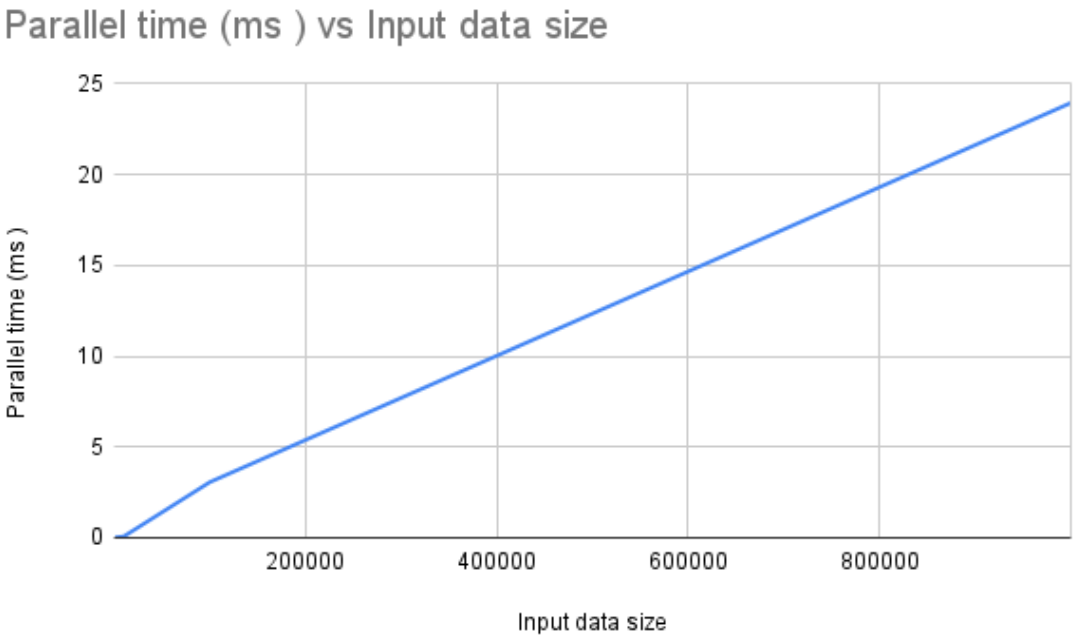
Input data size	Sequential time	Parallel time
n=100	Run took 0.0042 milliseconds	Run took 0.001 milliseconds
n=1000	Run took 0.031 milliseconds	Run took 0.0071 milliseconds
n=10000	Run took 0.5328 milliseconds	Run took 0.1116 milliseconds
n=100000	Run took 13.2126 milliseconds	Run took 3.0405 milliseconds
n=1000000	Run took 8711.798 milliseconds	Run took 26.6183 milliseconds

SEQUENTIAL_THRESHOLD=100000 filter size is 3

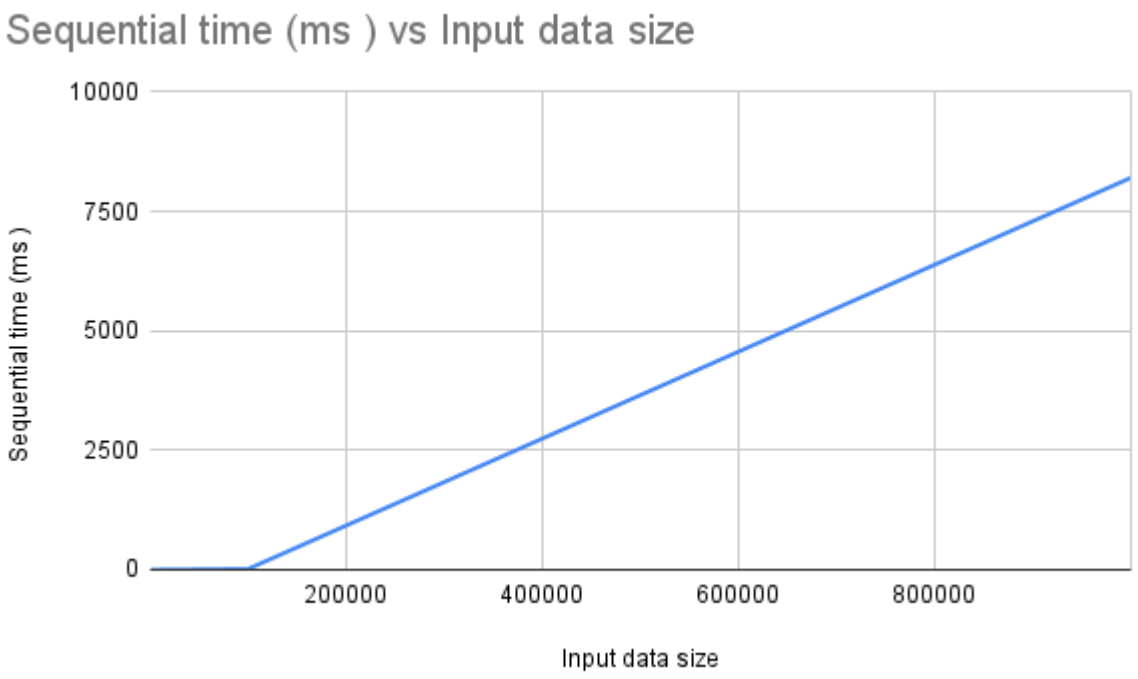
Graphs

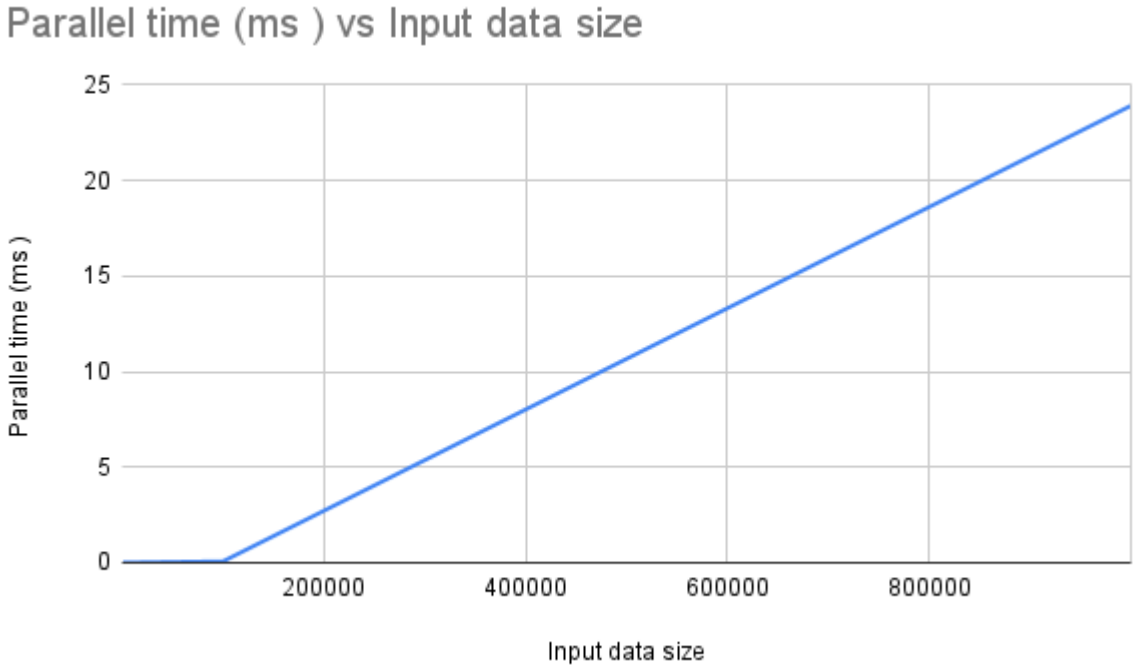
SEQUENTIAL_THRESHOLD=10000 filter size is 3





SEQUENTIAL_THRESHOLD=1000 filter size is 3





Git usage

```
    add small comments to code
:...skipping...
commit 3ab6127d7aca83f6c77d26a2230537717f115dca (HEAD -> master)
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:19:32 2021 +0000

    clean up code

commit ce5c65344e484515c985f1b7be51a42540a51ea3
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:16:02 2021 +0000

    add small comments to code

commit 6655d09d91e1d87a0310f2912b737df33dcdcbba
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:12:35 2021 +0000

:...skipping...
commit 3ab6127d7aca83f6c77d26a2230537717f115dca (HEAD -> master)
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:19:32 2021 +0000

    clean up code

commit ce5c65344e484515c985f1b7be51a42540a51ea3
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:16:02 2021 +0000

    add small comments to code

commit 6655d09d91e1d87a0310f2912b737df33dcdcbba
Author: Ubuntu <ubuntu@ip-172-31-14-41.eu-west-2.compute.internal>
Date:   Tue Aug 24 19:12:35 2021 +0000

    commit my java fils
```

According to the results it is clear that using multithreading for this problem is more efficient than the using a normal sequential program. Because looking at time taken to run the parallel algorithm is always faster than take it takes for the sequential to do the same takes which means that the sequential algorithm is expensive compared to the parallel. For this current program the range of data size where the filter works well is 100 to 100000 from 1M to 10M it sometimes takes longer thus this is due to the sequential cut-off which is not a perfect on because they are different for every data set. The maximum observed speedup obtainable is 0.0011 Ms which is impressive. 1000 seems to be the optimal sequential cut-off for this problem due to the fact that it produced an overall time better than the other sequential cut-

offs. The optimal number of threads is given by (number of takes/number of cores) for example for data set of size 100 threads per core are 100/4 equals 25.

CONCLUCIONS

The set of results obtained on this investigation allows one to conclude that parallel algorithms are more efficient when compare to sequential algorithms. Few obversions worth noticing is that the higher the sequential cut-off the faster the algorithm is, thus there's a direct proportion between time and sequential cut-off. As that is the case parallel algorithms can be slower when the perfect sequential cut-off is not provided and that optimal value will differ for different input data sizer. These results are concrete and valid, what's noticeable is the collection process these results were obtained in a repeated manner these processes were allowed to run 1000 times each time they executed which also helps in warming up the processor.