Assignment 1: Parallel Programming

Aim:

The following assignment aims to compare the speed of parallel programming algorithm and sequential programming algorithm. Using high-precision timing to evaluate the run times of your serial and your parallel method, across a range of input sizes and a range of filter sizes and Experimenting with different parameters to establish the limits at which sequential processing should begin speed tests will all be conducted using data sets that have been produced after multiple tests. the parallel processing algorithm is designed to split the data into threads that will all run at the same time, by making use of the Java Fork/Join framework to parallelize the median filter operation using a divide-and-conquer algorithm, the algorithm is a smooth one-dimensional time series data using a median filter. The filter will not smooth the endpoints.

Methods:

I created two classes, Parallelism and ParaConfig.

1. ***Parallelism:***
   The foregoing class holds the main method used to run the program as well as the timing objects used to save processing times for the various processing methods. In extension to the preceding this class touched the creation and writing of results to a folder called results.

2. ***ParaConfig*** *– contains the methods and constructors needed for the assignment.*

   Fields present in Class include:

```
static int RowVariable;
static int CNT = 0;
static int cutOff;
public int LowPoint;
static float[][] storeData;
static String[] storeResults;
public int upperValue;
static int colmVariable;
```

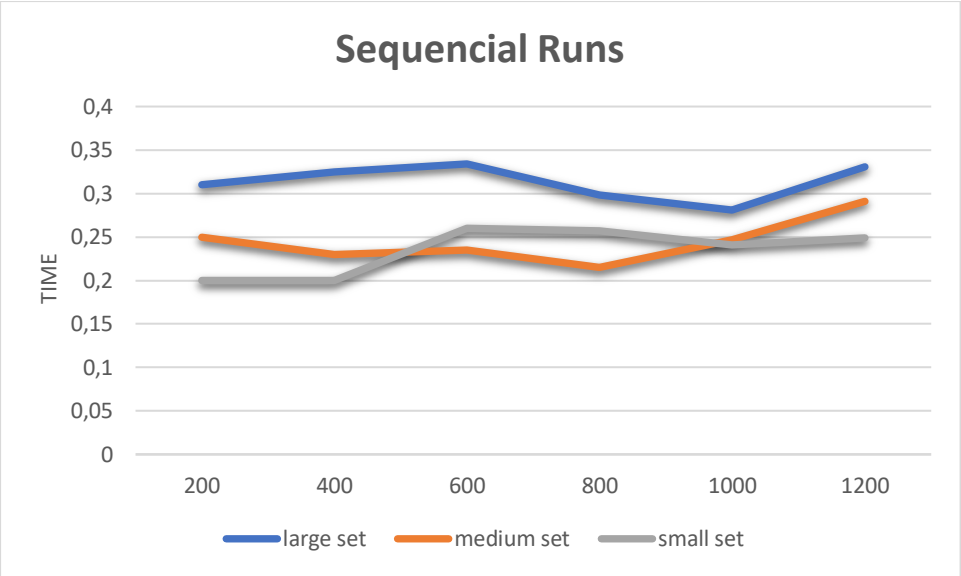
   These methods and constructors make up this class

   - *ParaConfig (String textFile) – it used to call the readFile method which reads data on the text file data and placing it into an array*
   - *Public void readFile (String textFile)- a method used to take a data file and add its information into an array.*
   - *Public void Calculate (int start, int end)- method that calculates and determines basins in each data set.*
   - *Protected void compute () - the method uses the divided and conquer algorithm that we need for our parallel programming.*

As means of verifying the accuracy of my algorithm, I ran various data files and perceived that at the maximum threshold level there was slight difference between sequential and parallel processing times. The processing times were collected by the use of the currentTimeMillis() method, that would only run when a particular processing course is in action. the processing speed data was solicited by running a data sets multiple times at a certain cut-off value before being incrementing. The data set was used to plot graphs on excel to better understand the relationship between sequential and parallel programs.

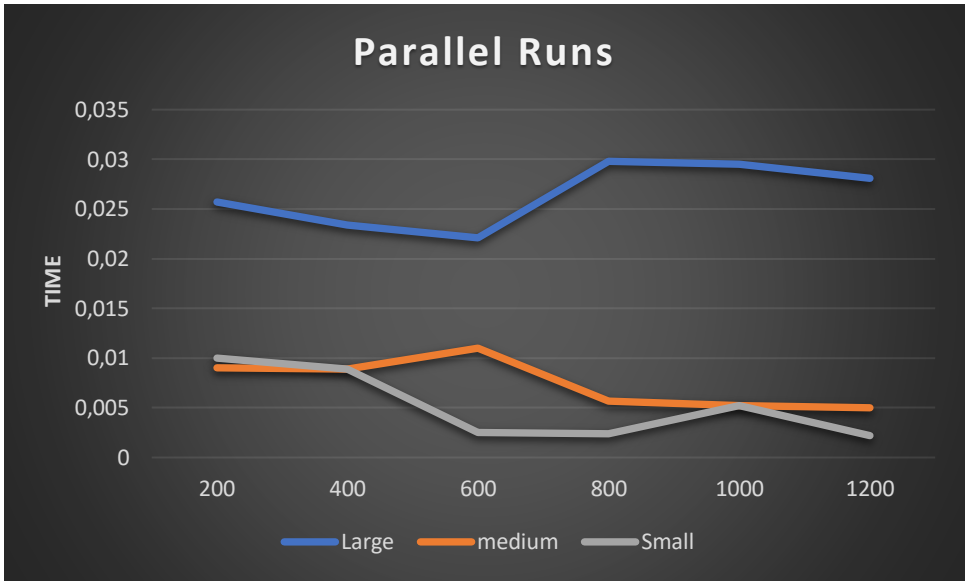
Test results

1. *sequential runs*



Based on the results we've discovered that the sequential processing times increased as the data pool got larger. The results fluctuated frequently owning to the background processes that ran concurrently as the data was being collected, but the time for all the three data pools will only increase when a larger data pool is used

2. *Parallel runs*



Run times for the small data progressively declined continuously a point where the curve just plutoed showing that code was running in the optimal cut-off range. The medium data pool reflects a similar shape to the small data parallel run apart from the curve at 600 showing that it's not an optimal threshold from which the program can run. The run times for the large data pool reach an optimal cut-off at 600 before improving to a less than optimal point at 800. However, the inclination of the line gradually decreases as the cut-off value rises.

3. *The resulting graph will explain the speed up of the code once being executed in parallel. The amounts used underneath were taken by dividing the average sequential times by the identical parallel run times*

Discussion

The use of parallelization for this assignment was quite efficient, calculations and data were prepared at a much quicker rate concerning sequential. Nevertheless, although parallelization is an effective way of dealing with some java programs, if the parallelization methods are not performed within its optimal range, there is going to be a slight difference related to sequential processing. Parallelization was extraordinarily useful for the average and short data pools as speedup taken on those data sets was immense as opposed to the speedup when running the large data pool. The highest speedup obtained was 237 while running the small data pool this was obtained when using an optimal cut-off of 950. Due to the design of my system, its optimal thread count is 3860 anything beyond that would generally result in much slower performance.

Conclusion

Parallelization is a robust form of programming as it appropriates the systems full potential when running a program resulting in more accelerated output times. Though it is a valuable ability it must be applied in a way that will guarantee the best performance from the system on the code execution. Results collected from this assignment.