CSC2002S - Assignment 1 -Report Parallel Programming- 2020
Student Number : LKNTLO001
Assignment: Parallel Programming
Introduction

The aim of this assignment was to experiment the significance of Parallel programming compared to sequential programming. The sequential programming involves using one thread of computation, but parallel programming involves breaking the main thread into small threads which are computed in parallel hence increasing the CPU speed performance. The data input file consists of grids which operations need to be done in order to look for basins. The grid elements need to be computed individually in order to determine whether that element has a basin or not. Hence for a sequential program; the run time is longer as the grid size gets bigger and bigger and a lot of time is taken during computation and hence slower performance. However; the parallel program runs several threads in parallel using the divide and conquer algorithm hence speeding up the performance.
The parallel program is expected to have a shorter run time compared to the sequential program. The assignment aim hence is to proof the fact that parallel programming is faster than sequential program, to determine the good sequential cut off and also to establish the importance of number of cores in a machine.

Methodology and Approach
Three classes were written to carry out the process. The classes are :

1. FindBasin.java which is the class that was used to process the parallelization. The class has a variable called SEQUENTIAL_CUTOFF which was used to vary the sequential cut off. The class also used the Fork-Join framework to implement threading . The RecursiveTask<ArrayList<String>> was used to return the arraylist with indexs of the basins. The approach used a divide-and-conquer algorithm since only one Arraylist was returned after the class was executed. For parallel algorithm the following were done:
   First the data_array was being checked whether its length was greater than the sequential cutOff, if yes, the array was being split into two threads running parallel. If not greater than the sequential cutoff, the array was being processed sequentially. The left thread was **forked into** and the **right thread was computed**. Then, **left.join** was called in order for the left thread to wait for the right thread to finish execution.
   In our experiment , we experimented using different sequential cutoffs so as to get the maximum sequential cutoff where parallel program turns to take long to execute. This is shown in the table below.
2. SequentialFindBasin.java class was used to implement the sequential part of the program. The data was loaded into an array operations were done linearly into the array.
3. Main.java is the main class that was implemented to run the Parallel part of the problem. The FindBasin was used as a helper class to the Main.java where the parallel logic was implemented in the FindBasin class.
4. Timing was done using the method Systemm.nanoTime() to get the runtime in order to perform the measurements of the two algorithms.

5. For better measurements, The average of five execution times was calculated and recorded for drawing graphs and other experiment analysis. This was to improve accuracy of the measurements.
6. The two machines used for testing were: 1. Intel (R) Core (TM) i3-8100 CPU @ 3.60Ghz, 2. Intel(R) Core (TM) i5-3470 CPU @ 3.20GHz.

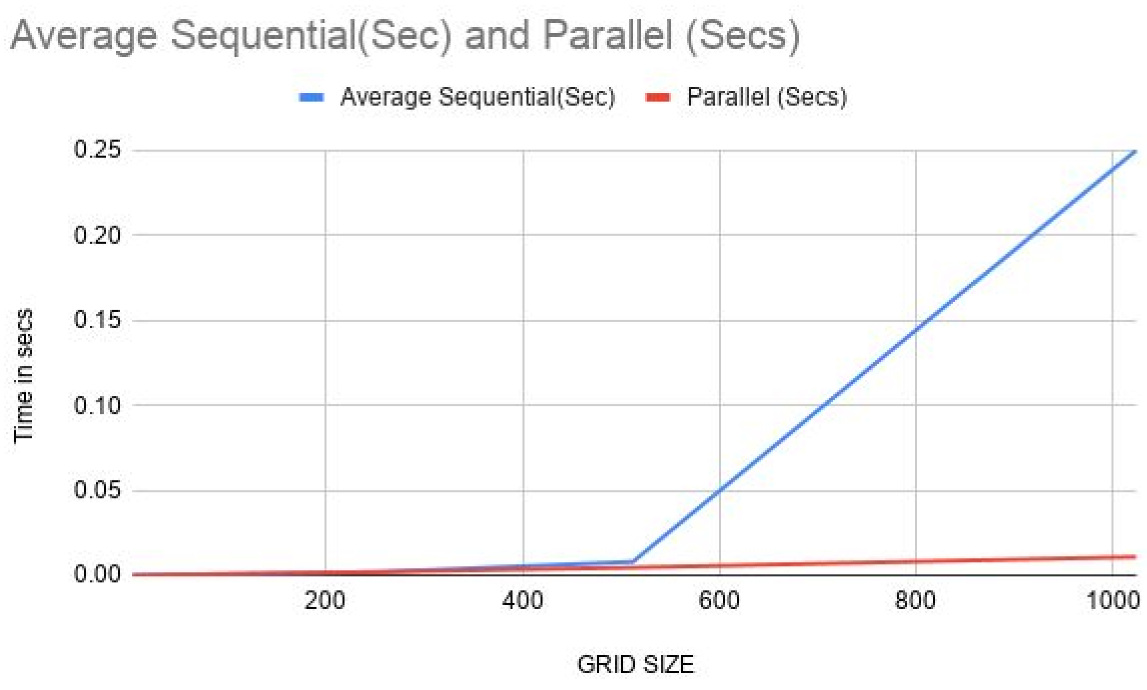
RESULTS AND DISCUSSION

Three experiments were done to test various parameters that affect runtime of code.

1. **Algorithm or programming approach:** Two algorithms were here where the both sequential and parallel algorithms were run to determine which of the two was faster than the other. The table below shows the runtime of the two algorithms over different grid sizes.

GRID SIZE	Average Sequential(Sec)	Parallel (Secs)
4	0.0001	0.00001
4	0.0001	0.00001
256	0.002	0.002049998
512	0.008	0.004599987
1024	0.25	0.01084999

The graph below shows visually the runtime of the two algorithms.

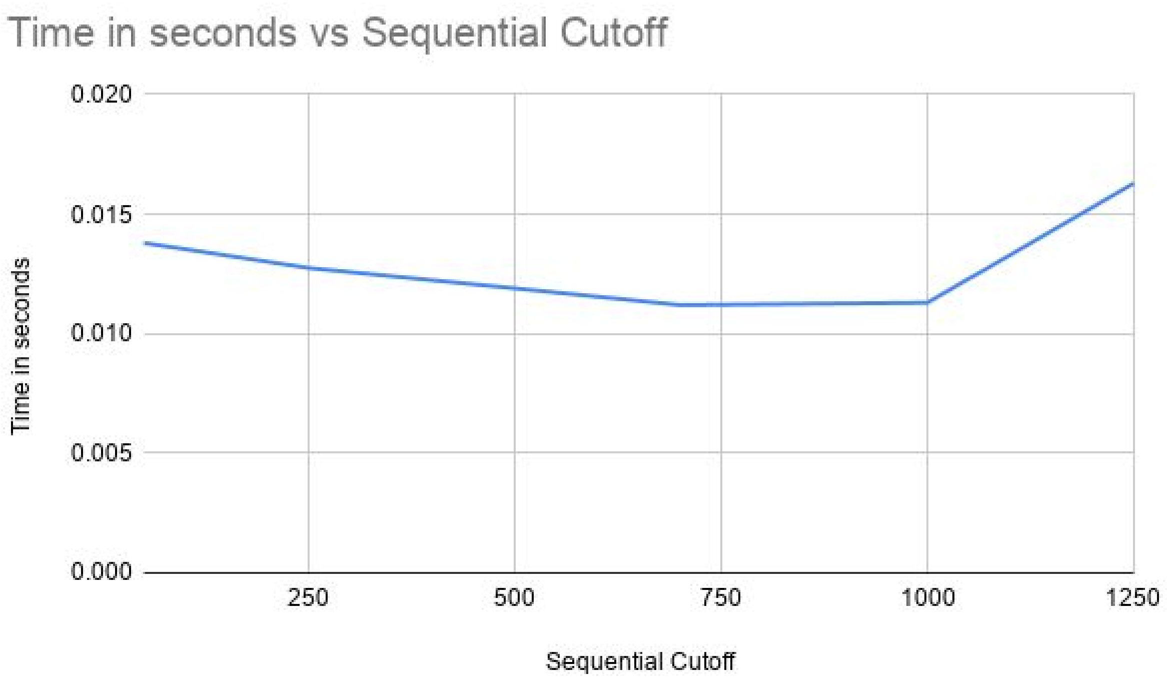


Discussion: It is clear that the parallel algorithm was running faster than the sequential algorithm. The sequential runtime algorithm increases linearly with increase with grid size of the data provided while the runtime in parallel is running logarithmically. Hence, the parallel algorithm makes the computation run faster. The divide and conquer strategy that was implemented contributed much to ensuring that the programming was running faster in parallel.

2. **Effect of sequential cutoff in parallel programming:** The sequential cutoff is used as a threshold to determine whether the remaining task can be implemented sequentially or to parallelize the remaining task. . When the size of the array was greater than the sequential cutoff, the main thread was being parallelized, but if the size was below sequential cutoff, the remaining task was being done sequentially.
   The experiment was tested using the large_in.txt file and the following data was collected. The graph below was also plotted.

Sequential Cutoff	Time in seconds
50	0.01379998
250	0.01275
500	0.011899998
700	0.01120001
1000	0.0113
1250	0.01629998

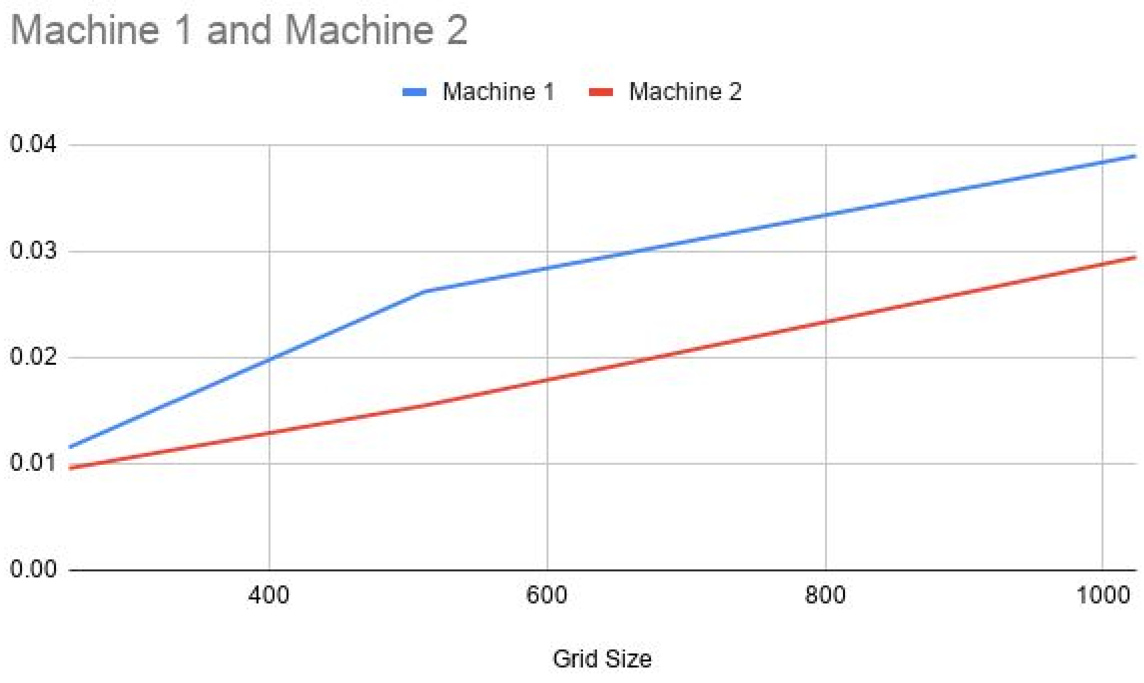
.



Discussion: From the graph above, it is clearly shown that the sequential cut off of our experiment falls in between 750-800. The runtime was decreasing as the sequential cutoff increased, but after the threshold was passed, the experiment started to behave as a sequential program. Hence, it is important to know that even though increasing the sequential cutoff reduces the runtime, increasing the cutoff beyond the threshold, the program behaves as a sequential program.

3. **Effect of number of processors core in parallel programming**
   Two machines were used to test the effect of the number of cores in a machine when doing parallel programming. The first machine which is i3 has 2 cores and the second i5 which has 4 cores. The table below shows the runtime tested against the two machines. A graph was also drawn as shown below.

Machine 1 Specifications : Intel (R) Core (TM) i3-8100 CPU @ 3.60Ghz, Machine 2 Specifications: Intel(R) Core (TM) i5-3470 CPU @ 3.20GHz		
Grid Size	Machine 1	Machine 2
256	0.011534	0.00956
512	0.0262144	0.01548
1024	0.039	0.02945



Discussion: From the graph above, it is clear that machine 2 offered great and faster performance. This was due to the fact that the machine had more cores which were used to run the code in parallel faster. This shows that with an increase in processor cores, the parallel programs can offer great performance.

Conclusion

Parallel programming offers great and effective performance compared to sequential programming moreso in computation of large datasets. However, it is important for the programmer to set the right sequential cutoff to ensure that the program offers the best performance. Currently, many laptops and systems are multi-core and it is important for developers to take advantage of this and run their parallel programs there since many processors cores improve performance of a program.