Operating Systems – 2: CS3523

Programming Assignment 1 – Report

Suraj Telugu – CS20BTECH11050

➢ Design of Program:

1) This program gives nearest point to the source from given set of points and time taken to find the nearest points using multithreading.

2) Initially define a struct for coordinates. All the global variables that will be used in main thread and worker threads are defined. File pointer is also defined as global variable. Define a function for Euclidean distance which uses the given formula and finds distance between two given coordinates.

3) In main function **getinputs_infile** function takes the input as per given order and format. **getpoint_infile** function is defined which reads point in given format (x, y) as string and converts it into coordn.

4) The points are taken in the form of an array of type coordn(struct). Two global variables points per thread and rempts (remaining points) are defined and values are assigned in main. These values are used in Thread function.

5) Global variable of type coordn* are defined and dynamic memory allocated for Nearpts array(stores nearest points in each worker thread). Array of threads of type pthread_t of given number of threads is created using **pthread_create**. The number of points is divided in each thread using pts_per_thread, the last thread takes the remaining points (rempts). The threads are then joined (executed parallely).

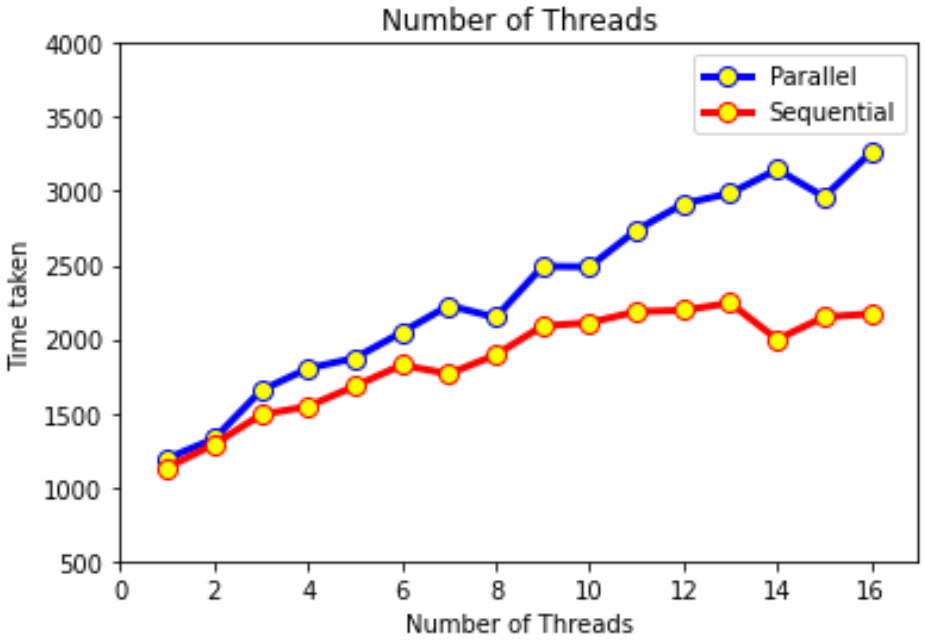
6) In thread function it takes the address of first point of its set as input. The minimum distance is first taken float max and then it is updated by each point using Euclidean distance function by which we find the point with minimum distance of the set in each thread and it is stored in Nearpts array.

7) In Main thread the point with minimum distance is calculated from the points in Nearpts array and stored in Nearestpt coordn variable using same process as above.

8) Clock starts before pthread_create function and ends after finding Nearest point. The time (in micro seconds) and nearest points are then printed in the given format.

➢ Comparison of the performance of Sequential and Parallel Execution:

- Graph 1:



Note: The above graph is obtained for set of coordinates range (1,3000)
Number of points = 5000 and source = (1500,2500)

Analysis:

1) Though in parallel execution multiple threads run at same time for large number of points each thread takes long time as they share data, registers, stacks. In parallel execution for calculating nearest point from 5000 points it takes a large amount data manipulation and due to multithreading the data available for each thread is less.

2) Due to parallelism the processor time is shared among threads and time taken might increase due to interrupts. Since Nearpts is a global variable for large number of points there might be data and may result wrong output

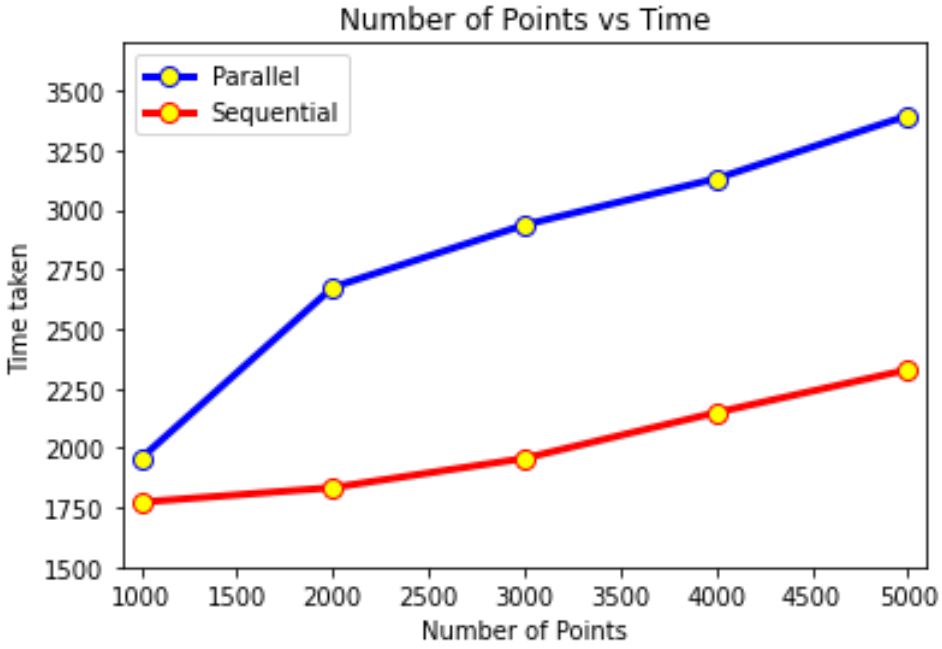
3) In sequential execution the next thread only starts when the before thread terminates. Therefore, there would not be any data insufficiency.

4) After thread1 completes execution and nearest point in thread1 is obtained thread2 starts executing so there would not be any data loss or interrupts.

Conclusion:

In above graph we can see the behaviour, initially parallel and sequential take nearly same time for large number of threads and points sequential execution is better than parallel execution. For small number of points the parallel execution might be better because each thread executes at same time and no thread has to for the other thread to complete, whereas in sequential threads wait resulting in long time.

- Graph 2:



Note: The above graph is obtained for set of coordinates range (1,3000)

Number of threads = 16 source = (1500,2500)

Analysis:

1) As the number of points increases number of computations increases so in both parallel and sequential execution time taken to execute increases.

2) Since the number of threads is 16 in parallel execution due to resource sharing and interrupts time taken is more compared to sequential execution.

3) In sequential execution as each thread executes only after the previous thread is terminated there is no resource sharing and hence less time.

Conclusion:

We can observe that as number of points increased difference between time taken for sequential and parallel execution is increased. We can see that in parallel execution the work load on each thread is increased as number of points increases resulting in more time. The difference is minimum for 1000 points hence we can say for small number of points parallel execution can work better than sequential execution.