CS 525 Project 3 Report

Name: Ali Necat Karakuloğlu
ID: 22101543

Implementation Details

i)*Serial*

In this part I used tree generation algorithm given in the project documentation. Similarly, I followed the given algorithm for range search. In tree generation, my code sorts a partitioned array of points according to the k th dimension to find the median point. Then, it recursively crates the tree.

When I checked the results of perf tool, I saw that for large number of points, tree generation dominates the execution. Similarly, when the querries are large, range search becomed dominant. The output of perf tool can be seen in project files.

ii)*Parallel*

For the parallel part, I used “#pragma omp task” directive for parallelization of the tree generation algorithm. The parallel algorithm (or OMP) assigns a thread to each recursion. As far as I have learned, the recursions are stored and when a thread finishes its task, it takes a piece of program to execute. Note that, this operation is regulated by the parameters of recursion and its order.

In the second part of the parallel program, I parallelized the execution of range search. Since it is executed in a for loop with different querries, and due to the irregularities in the range search algorithm (if statements that decide whether there is a recursion or not), I didn’t use task directive. When I tried to use the task directive, the execution time increased significantly. Therefore, I have decided to parallelize only the for loop. For this task, I used “pragma omp parallel for”

Analysis

I expected an inversely proportional reduction in execution time when I used all of the threads available (16 threads). However, The maximum speedup was 10x. For large number of points or querries, tree generation algorithm and range search algorithm becomes dominant. Therefore, for large datasets, it is approximately 97%. We can’t parallelize this 3% because it has to be sequential (3% is mosly input and output overhead). Otherwise the order of inputs and outputs may be erroneous.

Experiments

to test the algorithms, I created 3 different sizes of point array and 4 different querries. Starting from the points, there are 10M (large) , 1M (medium) and 100K (small) elements in the configurations. Similarly, 4 types of querries have 100 (small), 1K (medium), 10K (large) and 100K (very large) range elements. This set of data is generated for 4 dimension and 5 dimension. The speedup is visible when the data gets larger. To generate the speedup results, I used the results from Table 1-4.

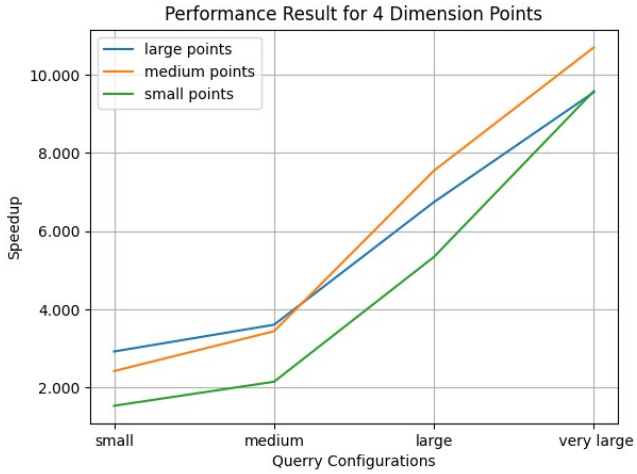

Figure 1: Speedup Graph with Different Data Configurations for 4 Dimensional Data

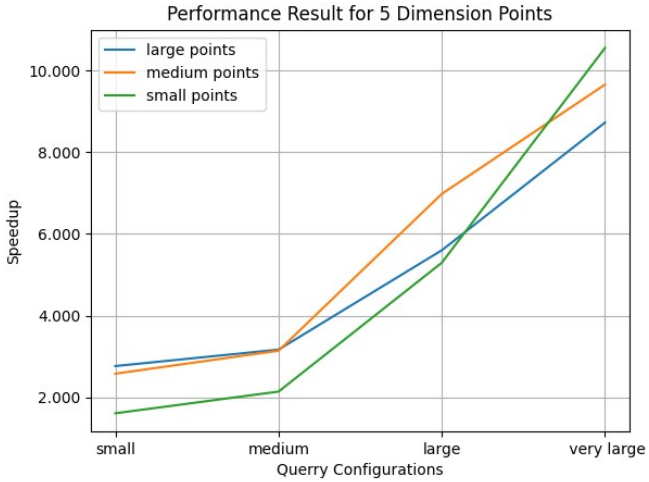

Figure 2: Speedup Graph with Different Data Configurations for 5 Dimensional Data

queries/points dim = 5	large(10M)	medium(1M)	small(100k)
very_large(100k)	1845.87	214.342	13.5732
large(10k)	245.501	25.905	1.5184
medium(1k)	89.7013	6.532	0.426165
Small(100)	72.4992	4.46322	0.310923

Table 1: Results for Serial Execution with 5 Dimensional Data

queries/points dim = 4	large(10M)	medium(1M)	small(100k)
very_large(100k)	2895.99	275.921	13.7919
large(10k)	350.26	31.0301	1.62687
medium(1k)	98.0961	6.88801	0.428309
Small(100)	73.4935	4.26866	0.308513

Table 2: Results for Serial Execution with 4 Dimensional Data

queries/points dim = 5	large(10M)	medium(1M)	small(100k)
very_large(100k)	211.573	22.2038	1.28651
large(10k)	43.8357	3.71163	0.286711
medium(1k)	28.2891	2.07769	0.198749
Small(100)	26.1938	1.72868	0.192825

Table 3: Results for Parallel Execution with 5 Dimensional Data

queries/points dim = 4	large(10M)	medium(1M)	small(100k)
very_large(100k)	303.088	25.7804	1.43876
large(10k)	51.9275	4.11029	0.304508
medium(1k)	27.1935	2.00231	0.199304
Small(100)	25.1466	1.76238	0.20106

Table 4: Results for Parallel Execution with 4 Dimensional Data

Discussion

The parallel program introduces a speedup. However, it is not as much as expected. When the input is large, the performance improves. Also the performance with 5 dimensional data is much bettter than 4 dimensional data.

Platform Specifications:

◦ Operating System: Ubuntu 20.04.3 LTS
◦ CPU: AMD Ryzen 7 3700X 8-Core Processor (16-threads)
◦ Number of Logical Cores: 16
◦ RAM: Corsair Vengeance 8 GB 3200 MHz DDR4, x2
◦ HDD/SSD: SanDisk SSD PLUS 120GB
◦ Compiler: g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0