Umit Yigit Basaran - 21704103

System Specifications

Operating System: Ubuntu 20.04 LTS

CPU: Intel i7 8th Gen

The Number of Logical Cores: 8

RAM: 8 GB

HDD/SSD: 256 GB

Compiler: gcc

Implementation Details

I used the given utility codes to read the files, create the required arrays and write the results to a file. For kd_construct_tree, I used the given algorithm and I implemented a linear time median finding algorithm for median selection. Because the median selection operation is an in-place I used starting index and ending index for constructing each node. I implemented the given algorithm for the range search operation.

To parallelize the serial code by using the OpenMP library, I specifically used “omp task” clauses because of the recursive structure of the code. Also to adjust the scope of the threads, I used “private”, “shared”, “firstprivate” clauses. For the kd_tree construct operation, I used “taskwait” (to combine the results of the recursive functions) and for the range search operation, I used the “critical” clause (to avoid shared data problem).

Analysis

I expected a massive amount of improvement based on the perf command output however the implemented parallel code has very small improvements for recursive operations. In an ideal implementation, the code can speed up to approximately 8 times because in the perf command outputs it can be seen that most of the overhead occurs in recursive functions kd_tree construction and range search.

Experiments

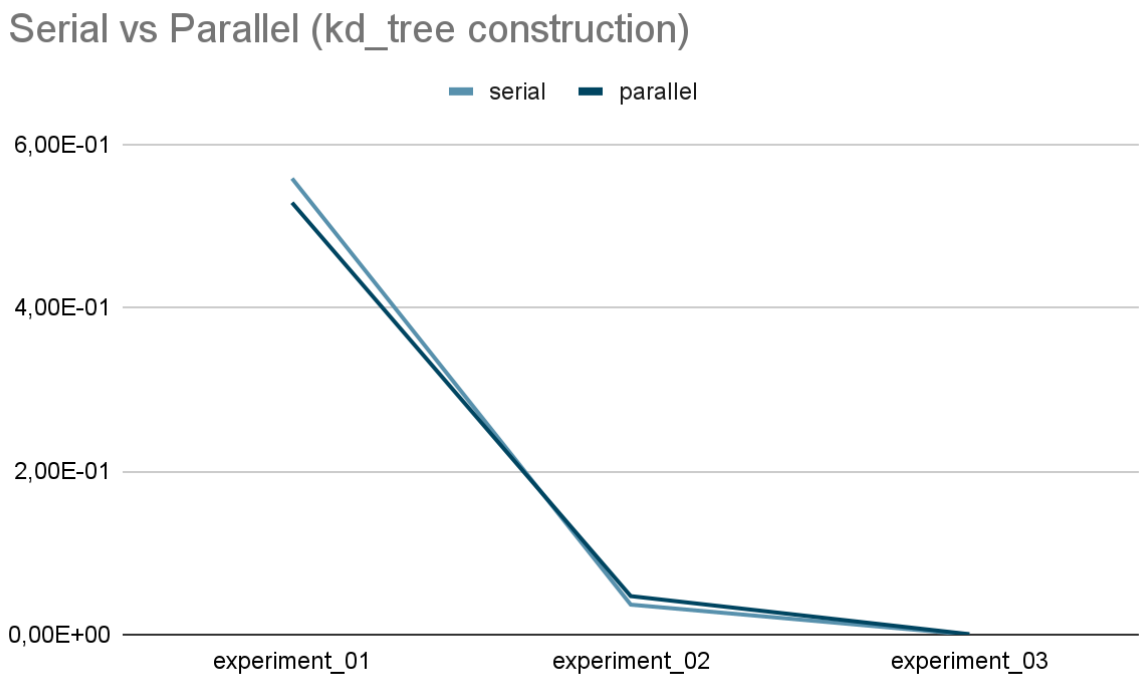

Figure 1 - Serial vs parallel implementation comparison for kd_tree construction

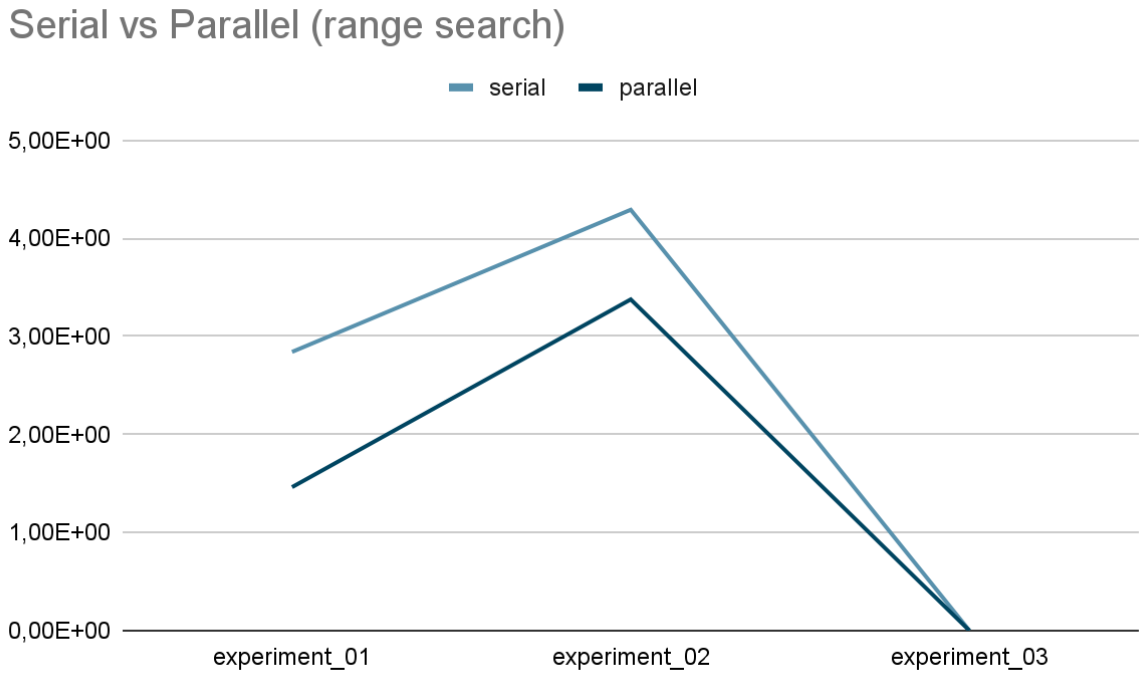

Figure 2 - Serial vs parallel implementation comparison for range search

Discussion

In my implementation parallel implementation does not improve the performance of the serial code. It can cause because of the following reasons. First, the thread creation and thread join operations can overhead a lot and it can cause some extra performance reduction. Second, the implementation of the pragma commands can be optimized so that the parallelization can be seen in the code. I tried to print the threads that execute the available tasks to see whether a thread always executes the tasks or not. However, I saw that different threads execute different tasks. Although the tasks are appointed to the responsible threads, my parallel implementation does not cause a performance improvement.