CS426 Project 3

Barış Can 21501886

1 Parallelization Strategy

I first read all the txt files and write them into three arrays as rows, columns and values. Then, using these arrays, I converted the data format which is not the same as mentioned in the assignment into the desired format and store them in row_ptr, col_ind and values arrays. For this purpose, I iterated the whole array, looked for row numbers and sorted the columns and values. I also changed created row_ptr as the column of the first seen number in a row. I then printed the arrays if it wanted and went into the parallel part. I did not include the number of iterations part into the parallel for loop because the next iteration is dependent on the previous one. I also make the required variables like x share since all threads write into it, and iteration_number, the result of the multiplication and adding private since all threads have its own memory and they calculated these variables alone. I used two parallelizations, one for multiplying the matrix and vector and the other for transferring the calculations into again x vector. I used a barrier at the middle since, for the transfer, all threads need to join, and printed the resulting vector for only the master thread. Therefore, I used data parallelism for two places separately but I have one pragma omp parallel section.

2 Discussion

As can be seen from below, I conducted three experiments with the three data given in the assignment with 25 number of iteration. First one is time vs thread count, second is speedup vs thread count and the third one is efficiency vs thread count. Be careful that there are 5 numbers in the time vs threads graph and there are 4 numbers in the speed up vs threads graph. I considered increasing the number of iterations any further, but the results of experiment two was almost big like 2000, so I did not want to increase the iteration number any more to not get bad results, but I also wanted to get high times for the graphs, so I used 25 iterations. For the first and second experiment, while switching from serial to two-thread parallelism, the parallel experiment was two times faster than the serial experiment. In the four-thread parallelism, the experiment again was faster two times than the two-thread parallel experiment which adds up to four-times faster experiment compared to serial experiment. Then, the speed-up starts to reduce in eight-thread parallelism, but they were still faster than two-thread parallelism. For the first two experiments, while the speed up was between 2 and 4, the efficiency always reduced since we are increasing the number of threads as the squares of two, but the results are not increased as much. Finally, the efficiency always reduced but speed up was still more than two times of the serial experiment.

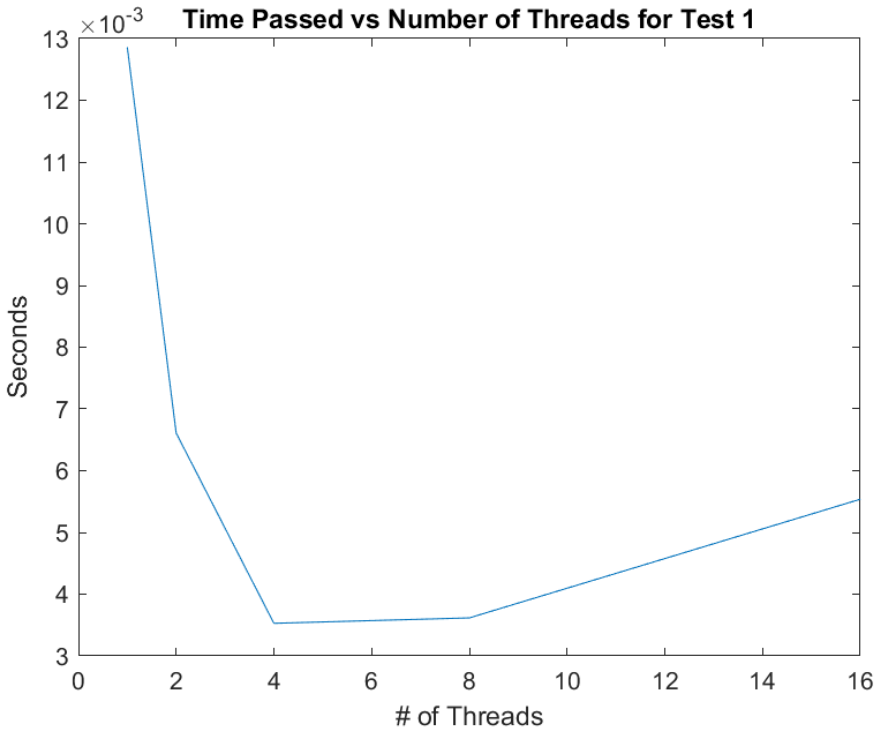

Figure 1: Time Passed vs Number of Threads for Test 1

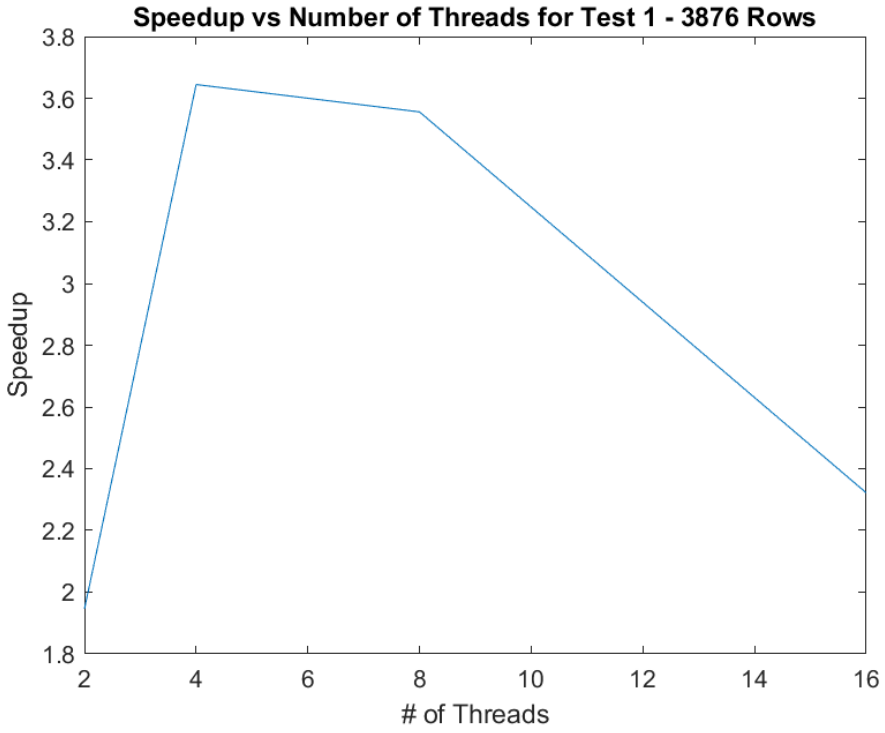

Figure 2: Speedup vs Number of Threads for Test 1

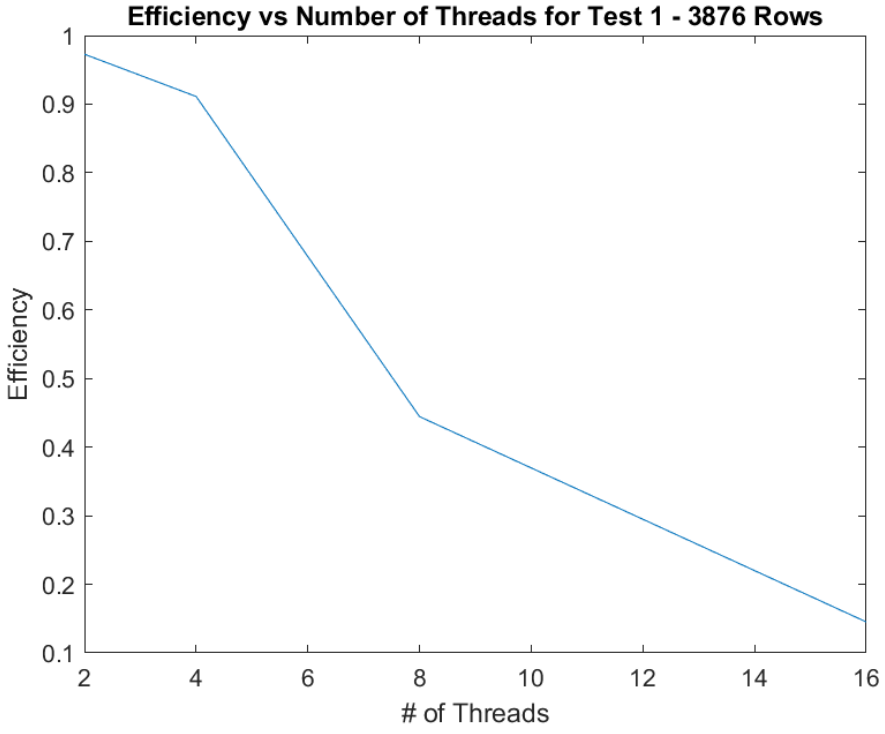

Figure 3: Efficiency vs Number of Threads for Test 1

The size of the matrix in the second experiment was bigger than the first one, so it took more seconds to complete especially in the reading and data conversion phase since the arrays were very big. However, the matrix multiplication did not take that much and completed under 0.08 seconds. Still, it was 6 times slower than the first experiment. Since we are interested in ratios while calculating the speed up and efficiency, results were very similar to the first experiment. Moreover, since the computation is bigger in this experiment, the result of using parallelism is more clear and ideal in this experiment. After the two-thread experiment, the speed up increases to 4 and then reduces to 3.4 which was 2.3 in the first experiment. The efficiency is also better for all thread counts than the other experiments. I think the reason for speed up to start decreasing is the bottleneck of the increasing number of communication between threads. This bottleneck does not show any significance since it is small and the speed-up is high, but after increasing the thread count more, it starts showing itself and decreases the speed up and efficiency. To sum up, this is the best experiment for the parallelism considering the speed up and efficiency values.

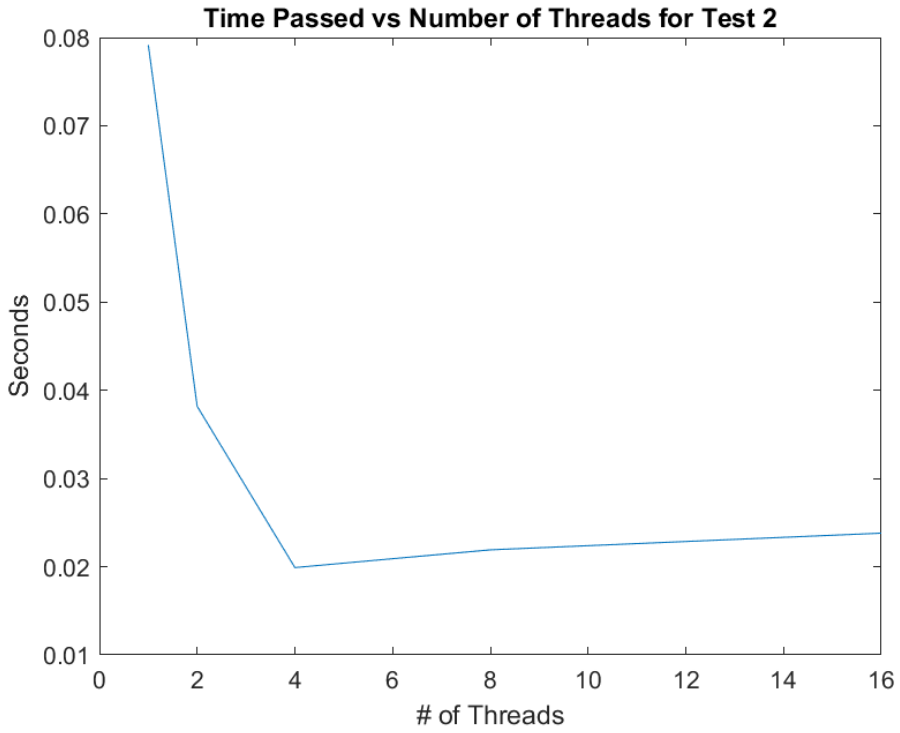

Figure 4: Time Passed vs Number of Threads for Test 2

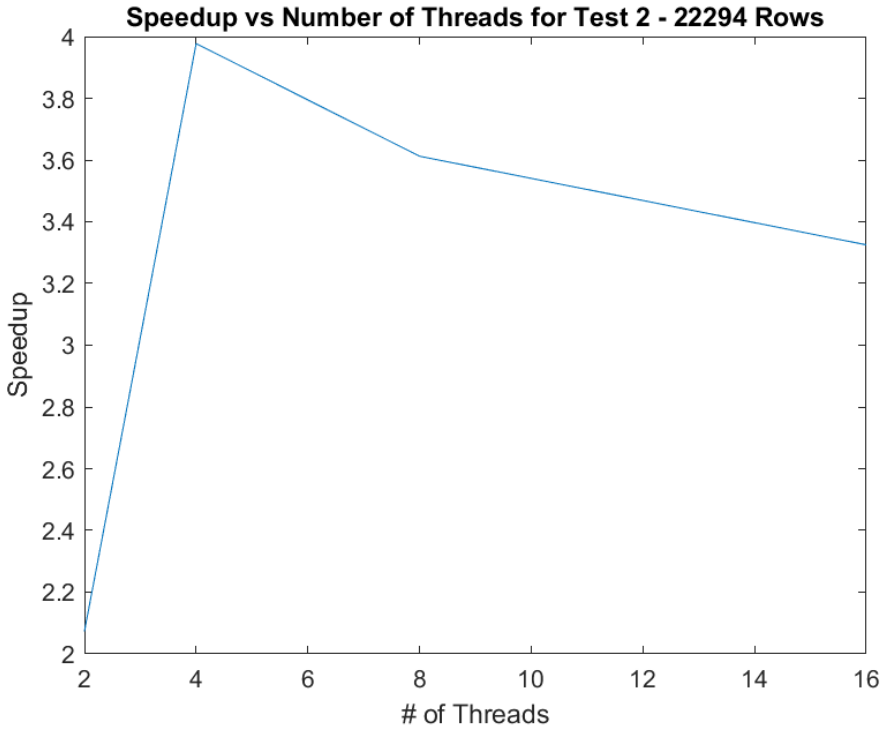

Figure 5: Speedup vs Number of Threads for Test 2

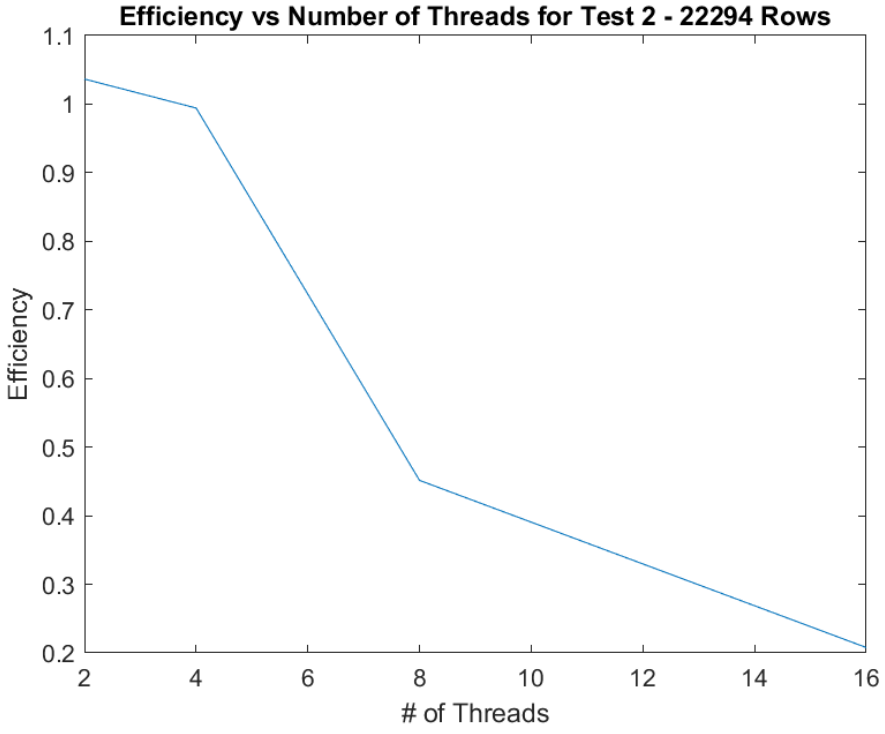

Figure 6: Efficiency vs Number of Threads for Test 2

For the last experiment, the size of the matrix is was the smallest, so using parallelism was not as efficient as the previous experiments. Even if the time passed for the completion of the experiment was smaller than the previous experiments thanks to the size of the matrix, speed up for the threads were very smaller compared to them. The speed-up started two as the previous experiments, but then it increases to 2.2 which was 3.6 and 4 for the previous experiments, and after four-thread parallelism, it reduced drastically and the sixteen-thread parallelism was slower than the serial experiment. Moreover, as can be seen from Figure 9, the efficiency was also worse than other experiments. The reason for that as I mentioned above is since the matrix is very small, it is computed very fast and parallelism cannot decrease the time as much since there are other limitation in the computation and it cannot decrease after a while, so there is no need to parallelism for this size. If I would increase the number of iteration to 1000 and divide the number of iteration by hand to the number of threads so that there would be no dependency between the data of the thread, then I would gain some advantage using parallelism on this array, but as not using parallelism with higher than 2 threads for this experiment was not show any significant difference.

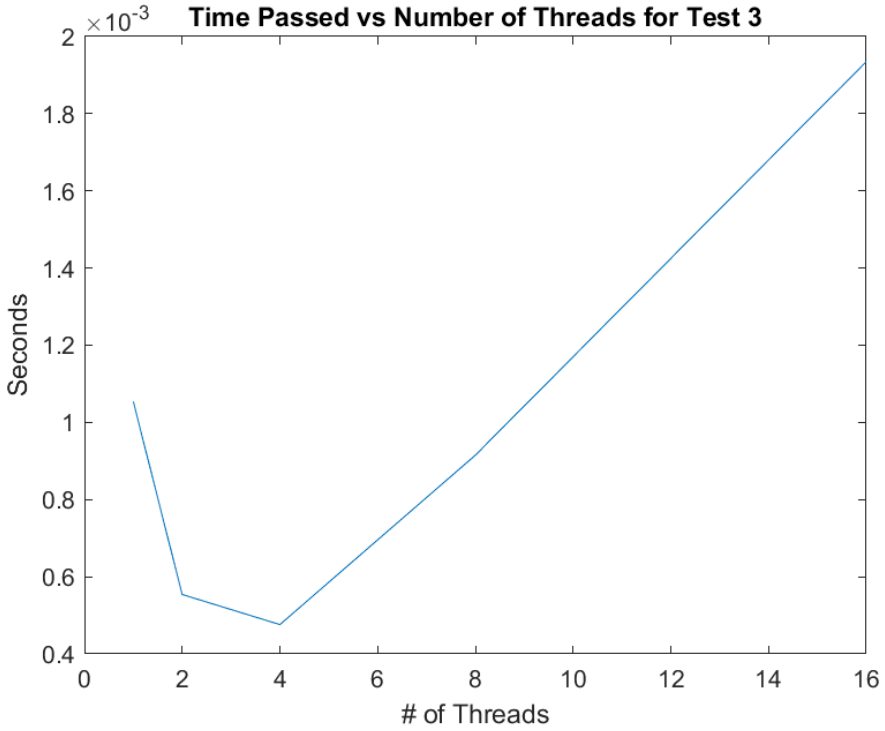

Figure 7: Time Passed vs Number of Threads for Test 3

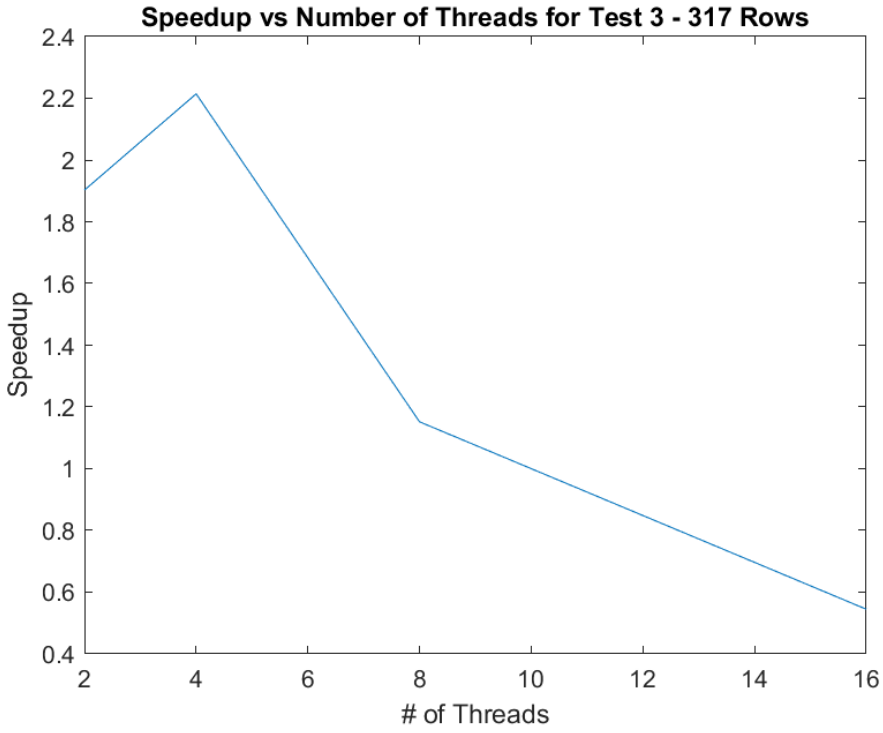

Figure 8: Speedup vs Number of Threads for Test 3

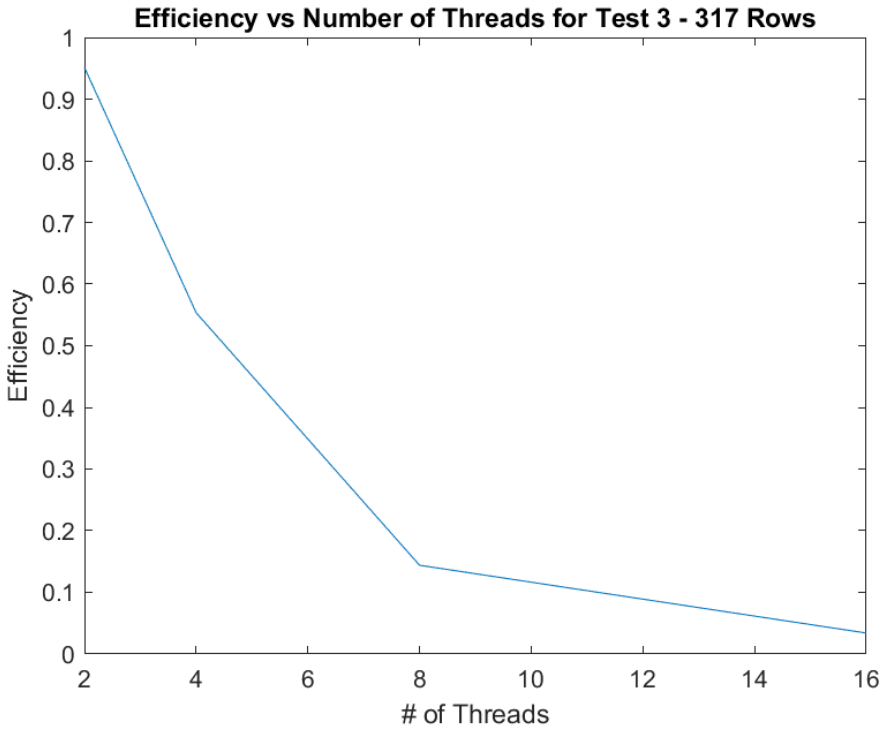

Figure 9: Efficiency vs Number of Threads for Test 3