Programming Assignment Report

PA3: K Nearest Neighbors

Chad Adams

CS791

Due: February 15th, 2018

1. Introduction

In this project we were tasked with implementing the KNN problem where we fill in missing table values by using the average of the K nearest neighbors. Both sequential and parallel implementations of this project were required. The sequential implementation was fairly straightforward and did not take very long to run on the CPU. The parallel implementation required the use of atomic addition in order to get the neighbor distance values but otherwise functioned similarly to the Sequential method.

2. Methodology

2.1. Sequential

The methodology for the sequential implementation was simple, the program would search through the tree until it found one of the rows that had an unfilled value. It would then use the distance formula and store the result in an array that corresponded to the index of the neighbor. The K nearest neighbors were found by simply assigning the first five results in the table as the nearest neighbor and then comparing any new result with the current largest value of the K nearest neighbors, replacing the largest value if the new value was lower and then finding the largest of the new K nearest neighbors again. Then the missing value would be assigned the simple average of the K values.

2.2. Parallel

The methodology for the parallel implementation was a bit more complex and required the use of atomic addition on the neighbor distance array. Instead of finding the K nearest neighbors during the distance calculations the parallel implementation required them to be found after all the distances had been calculated and the synchronize function had been called. After the K nearest neighbors were found the average of their values in the same column as the missing value and replacing it. The remaining code was kept the same from the sequential implementation as the number of remaining operations were too small to be worth making parallel.

3. Results and Discussion

The results overall had the sequential version clocking in at 3.5ms on average, giving it roughly 26 million operations per second, or 26k per millisecond. The parallel run time was on average 5.84ms with an overall throughput of 15 million per second, or 15k per millisecond. The table below shows the running times over the ten runs of the parallel algorithm along with the average for the sequential algorithm.

Type	Blocks	Threads per Block	Run Time
Average Sequential	-	-	3.5ms
Parallel	256	32	5.86
Parallel	256	32	5.06
Parallel	256	32	5.58
Parallel	256	32	6.12
Parallel	512	32	6.50
Parallel	512	32	6.13
Parallel	512	32	6.17
Parallel	1024	32	6.20
Parallel	1024	32	5.34
Parallel	1024	32	5.40
Parallel Average	-	-	5.84

This result is unusual as the Parallel would normally be faster than the Sequential implementation. This result most likely stems from the nearest neighbor function only calculates neighbor distance in parallel but calls the distance finding function once for each row with a missing value rather than once for the entire table.

4. Conclusion

In this project we implemented a Sequential and Parallel solution to the KNN problem to gain a better grasp of atomic functions. Overall the Sequential implementation achieved superior performance, but this may have been due to the methodology used for calling the KNN function. Through this project the usefulness of atomic functions became apparent.