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 .

# 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. 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 2ms on average, giving it roughly 45 million operations per second, or 45k per millisecond. The average distance between the estimated values and the replaced values was <insert value here>. The parallel run time was on average <value> with an overall through put of <value> per second, or <value> 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 | - | - | 2ms |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |
| Parallel | | | |

## 4. Conclusion

.