CSCI 5451 Homework 3 Report

Benjamin Chu

November 2023

1 Implementation

For this problem, we are given the following algorithm:

1. Select initial K centroids as the first K data points
2. Assign each data point to the nearest centroid
3. Repeat the following steps until convergence:
    (a) Update each centroid to the average of the data points in its cluster
    (b) Reassign each data point to the nearest centroid

A serial version of this algorithm is implemented in `km_serial.c`.

To parallelize this algorithm, the threads were organized in a 1D fasion, and the thread-blocks were arranged in a 1D grid.

For steps 1 and 2, we assign each thread an initial index of `blockIdx.x*blockDim.x+threadIdx.x`. Then, each thread iterates over the centroids starting from that index with a stride equal to the total number of threads. For step 3b, we do the same, but iterating over the data points instead.

For step 3a, each thread picks a separate range of centroids to work on. Then, each thread iterates over the entire list of data points, but only computes the averages for centroids in the range it is assigned to.

By dividing this way, each thread works on an approximately equal number of centroids/points. However, increasing the total thread count beyond the number of centroids is ineffective for steps 1, 2, and 3a.

2 Timing Results

When running our code on the `large_cpd.txt` dataset with 256 clusters, we get the following timing results:

	2 Blocks	4 Blocks
32 Threads/Block	1573.0017s	887.6370s
64 Threads/Block	954.6752s	563.1324s

When running our code with 512 clusters, we get the following timing results:

	4 Blocks	8 Blocks
32 Threads/Block	1450.5801s	793.9475s
64 Threads/Block	859.5162s	538.3968s

When running our code with 1024 clusters, we get the following timing results:

	4 Blocks	8 Blocks	16 Blocks
32 Threads/Block	3107.1091s	1566.0896s	811.9514s
64 Threads/Block	1727.4386s	891.3885s	512.0679s
128 Threads/Block	1078.4892s	631.5232s	667.6513s

From these results, we can see that the computation time is approximately halved each time the block count or the number of threads per block is doubled. The one exception was when the code was run for 1024 clusters, with 16 blocks and 128 threads per block. This is because our parallelization method does not work well once the total thread count exceeds the number of clusters, as noted earlier. We can also see that it was generally more effective to increase the block count than to increase the number of threads per block.