Homework 4: MPI

Tuesday April 19th, 2016

Christina Mosnick

Intro to Problem

Goal: Use MPI nodes to solve the L1 nearest_neighbor problem.

Problem: Given an input vector of floats, find the L1 nearest n-neighbors from an input file of same-size float vectors.

Pseudocode / Approach

The data is divided into separate files to parse through. Knowing this, I decided to assign each file to a thread to be parsed. This includes reading, finding each item's distance from the search vector, and sorting and cutting the results.

Master tasks:

1. Navigate directory of data files
2. Send threads each one data file to parse
3. Receive thread's response and statistics messages
4. Add results to global results
5. Add statistics to pre-existing statistics
6. Report results (write stats to file)

Threads' (workers') tasks:

1. Receive a filename to parse
2. Read file lines into memory
3. Find distance for each line
4. Sort and cut results into n-results
5. Measure timing or various operations
6. Send n-results back to master
7. Send statistics back to master

For the sorting of the results, I used std::partial sort.

For MPI communication, I created three custom datatypes:

- cmoz_result_type—holds one singular result in the form of {char[], double} for {fname, distance}
- cmoz_multipleResults_type—holds an array of cmoz_result_type's. The size of this type is dynamically determined based on the number of results desired by the user.
- cmoz_stats_type—holds reporting statistics. {double, double, int} for {fileLoadTime, vectorProcTime, numVectors}

When a thread finished parsing its assigned file, it sends a results object (with tag RESULTS) and a statistics object (with tag STATS) to master.

When master is waiting for a response, it first waits for a message with the RESULTS tag. It then gets the sender of that tag, and receives the STATS message from that same sender.

Testing

I used two approaches when testing on multiple nodes:

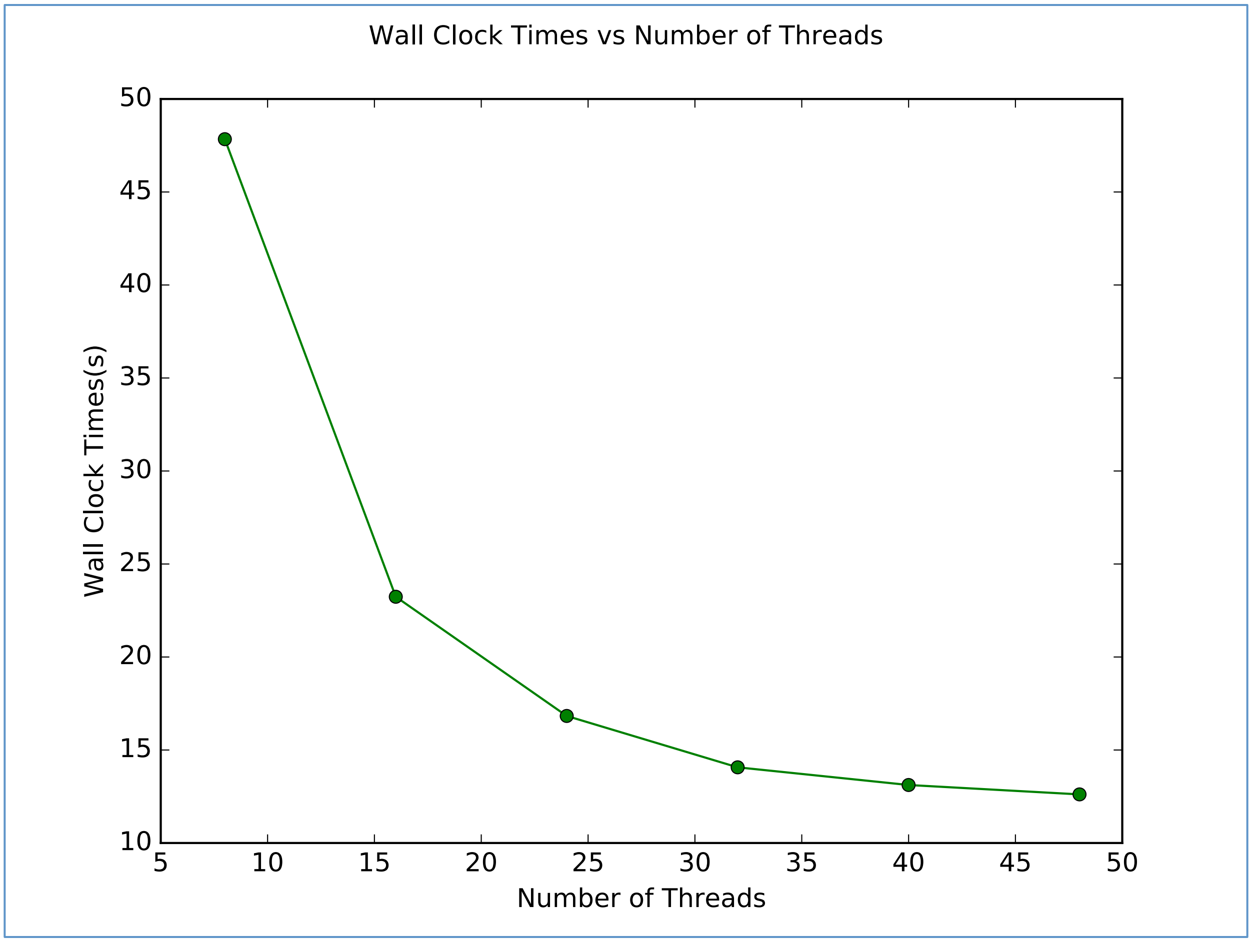
1. Split tasks evenly among all three nodes
   - Example: if testing with 24 threads, give each node 8 threads
2. Use a waterfall approach: use full capacity of first node before using next node
   - Example: if testing using 24 threads, give t02 16 threads(max), t03 8 threads(the rest), and t04 0 threads.

Testing Strategy:

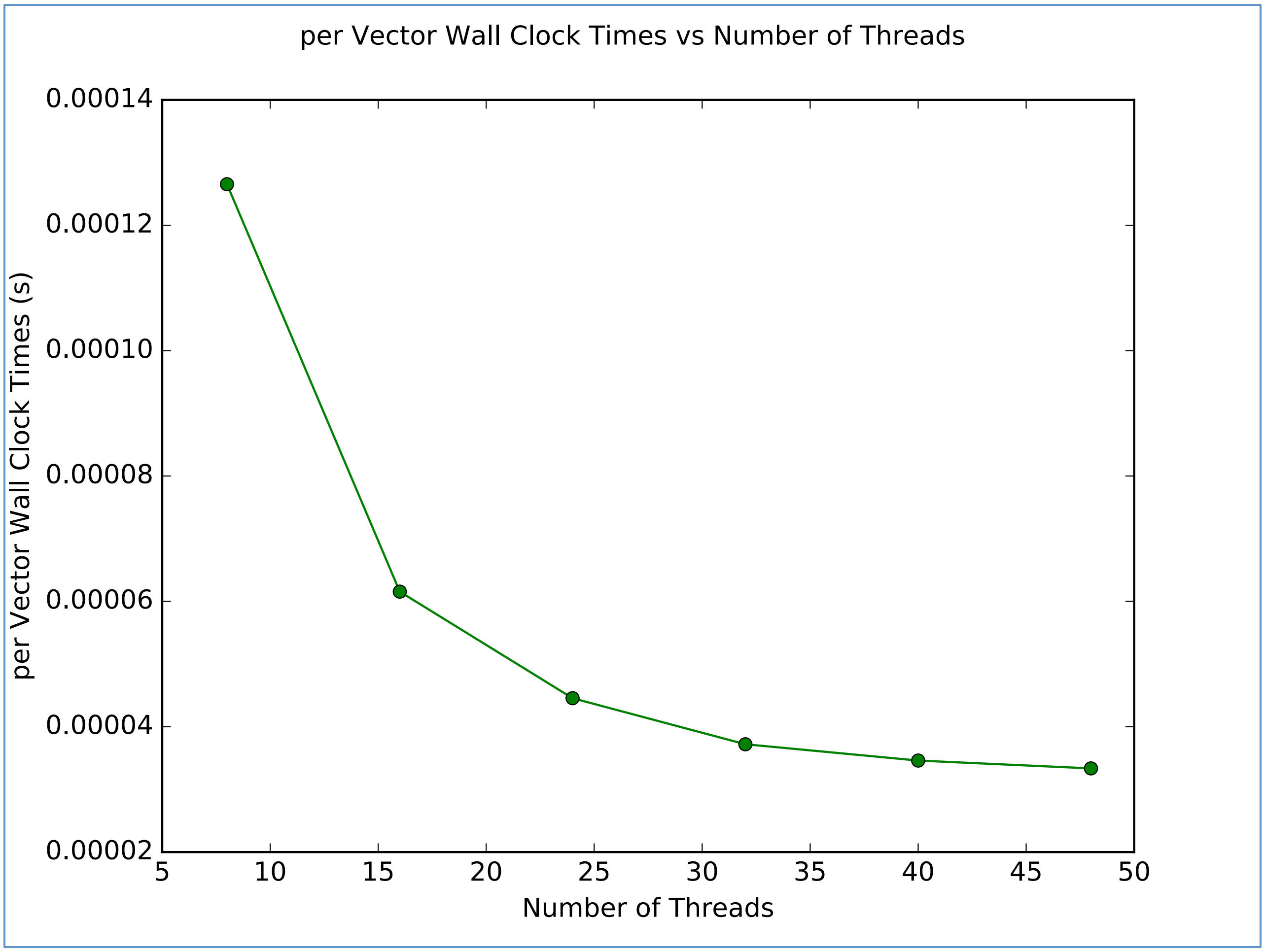
		Number of Threads					
Approach	Host Number	8	16	24	32	40	48
even split	2	2	5	8	10	13	16
	3	3	5	8	11	13	16
	4	3	6	8	11	14	16
waterfall	2	8	16	16	16	16	16
	3	0	0	8	16	16	16
	4	0	0	0	0	8	16

Each test run appends its statistics to a csv file corresponding to the method that was run. When all tests are finished, a python script averages all the data in each file for a number of threads, and plots the data to a pdf. The python script also records the averaged data into a new csv file.

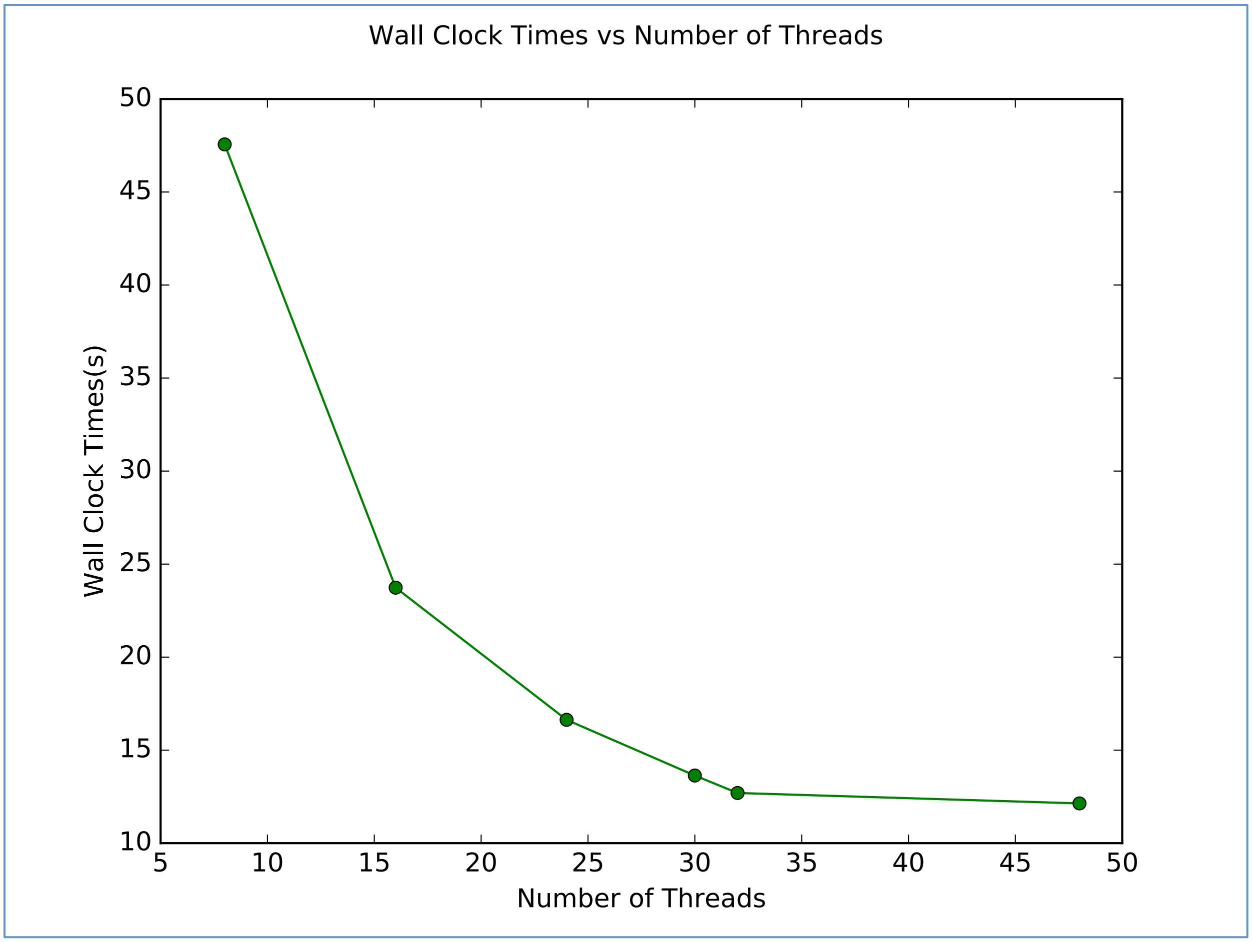
Even Split:



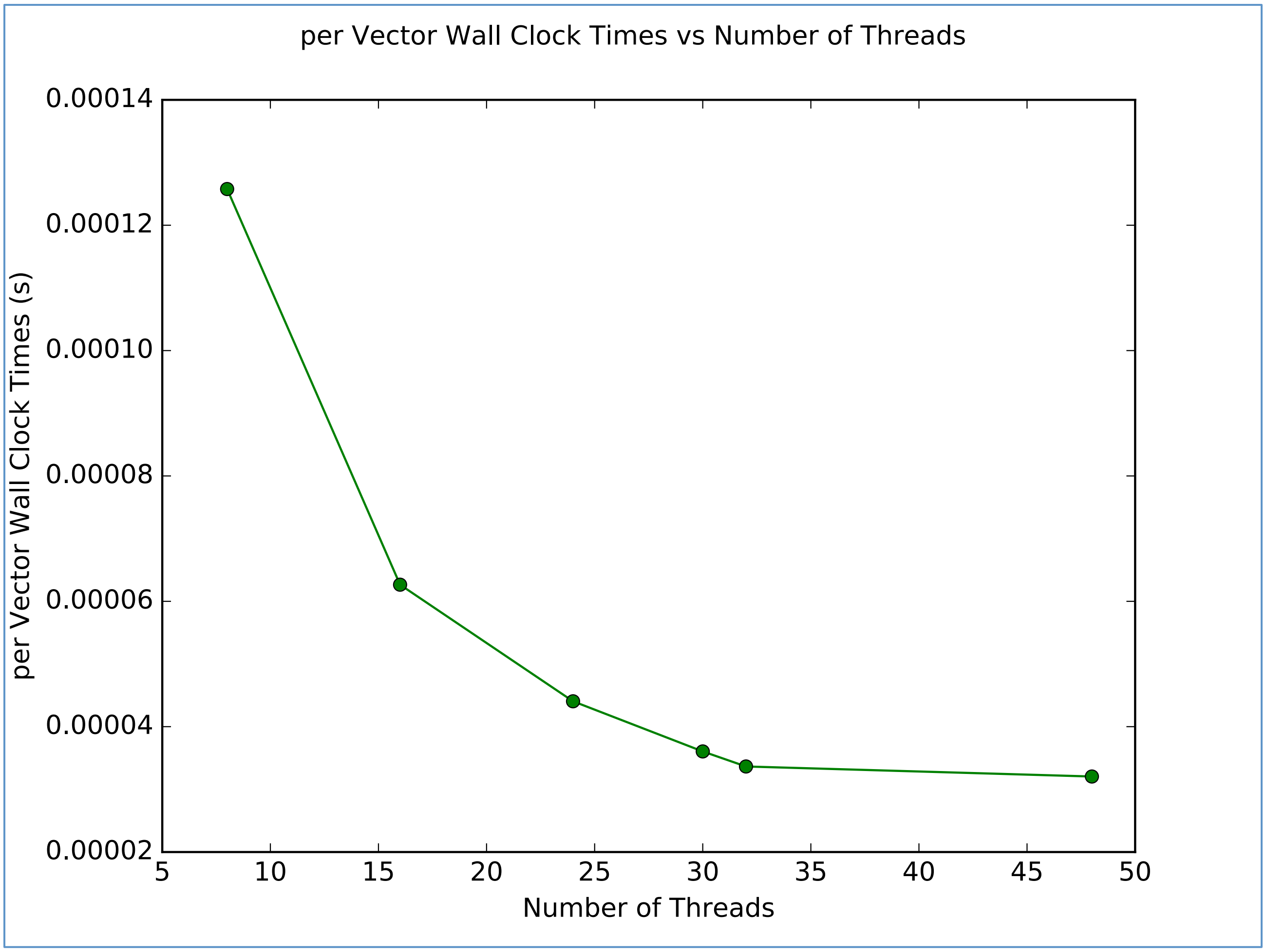
Number of Threads	Average Wall Clock Time	Average per Vector Wall Clock Time
8	47.839693	0.00012655
16	23.2384469	6.16E-05
24	16.8304962	4.45E-05
32	14.06860685	3.72E-05
40	13.1183221	3.46E-05
48	12.61334775	3.33E-05


per Vector Wall Clock Times vs Number of Threads
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
per Vector Wall Clock Times (s)
5
10
15
20
25
30
35
40
45
50
Number of Threads

Waterfall:



Number of Threads	Average Wall Clock Time	Average per Vector Wall Clock Time
8	47.56007486	0.000125773
16	23.7329158	6.27E-05
24	16.62862835	4.41E-05
30	13.6345354	3.60E-05
32	12.69692505	3.37E-05
48	12.1360302	3.21E-05



Analysis:

Both approaches (even split and waterfall) produced very similar times. I tested each approach about 20 times. As you can see, the times decrease significantly as the number of threads increase.

Conclusion:

I would have liked to test the scalability (more files), but with the times my program is outputting now, I simply did not have enough time.

It would also have been good to see how changing the number of results (k) effects performance.

Last, it could be cool to make a 3D dataset and chart plotting number of threads vs. number of results vs. time. This could involve a lot of time in the testing phase to create, but would be very interesting to see interactively with a python 3D plot.