Assignment 1: Threading and Message Passing Interface with a K Nearest Neighbors Model

Gabriel Salmon
Computer Science
Virginia Commonwealth University
Richmond, VA
gtsalmon@gmail.com

Alberto Cano
Computer Science
Virginia Commonwealth University
Richmond, VA
acano@vcu.edu

***Abstract*—We measure the runtimes of serial vs. parallelized K Nearest Neighbors models. The same model will have three main forms: fully serial, parallelized with pthreads, and parallelized with MPI.**

***Index Terms*—KNN, pthreads, mpi, parallelization**

I. Introduction

Many machine learning models require running a large number of simple and identical operations. These operations use and modify separate pieces of data, meaning they can be run in parallel. We find how much two simple methods of parallelizing the K Nearest Neighbors (KNN) model will improve runtime over the serial implementation.

II. Methodology

A. Serial

This version simply compares each test instance to every train instance. The nearest K train instances to each test instance by Euclidean distance are kept, and the mode of their values, or classes, is assigned as the class prediction for that test instance. Our data for this and the other implementations will only use 3 neighbors since we are unconcerned with accuracy.

B. Pthreads

The pthread implementation uses POSIX threads by evenly dividing the set of test instances between them. Each thread will then predict the classes of its assigned instances. This implementation, like the Message Passing Interface (MPI) implementation, will be run once with 1, 2, 4, 8, 16, 32, 64, and 128 threads.

C. MPI

The MPI implementation divides the number of test instances into the number of processes designated (1, 2, 4, 8, ...) with MPI_Scatter(), predicts the classes, and uses MPI_Gather() to assemble the predictions. The ”oversubscribe” option is used for 32, 64, and 128 processes, as the computer this was tested on only has 28 cores.

III. Results

A. Runtimes

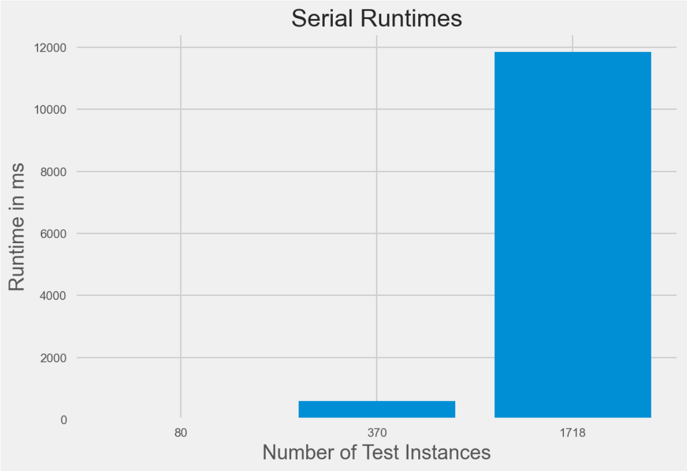
Following are runtimes for each version on a 28 core remote Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz

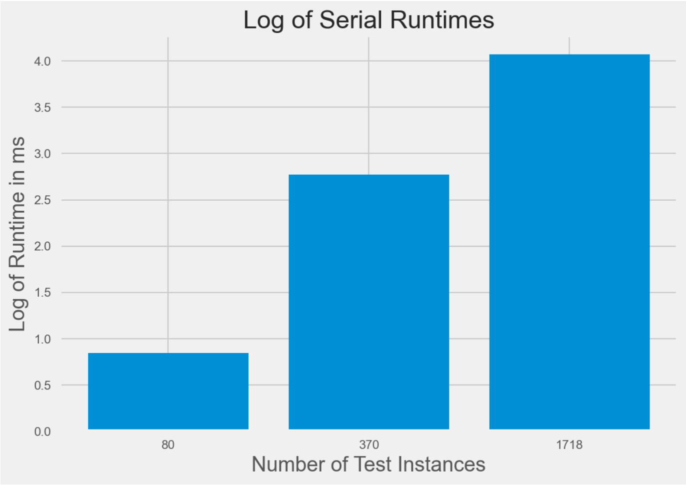
Number of threads	**Number of Test Instances**		
	80	***370***	***1718***
Serial			
1	7	593	11854
Pthreads			
1	7	631	12323
2	4	317	6414
4	2	172	3326
8	1	87	1704
16	1	74	871
32	1	38	624
64	1	35	514
128	3	36	474
MPI			
1	240	635	8242
2	238	442	5042
4	247	360	2646
8	280	348	1824
16	290	345	1380
32	420	581	1458
64	620	657	1383
128	1629	1470	2037

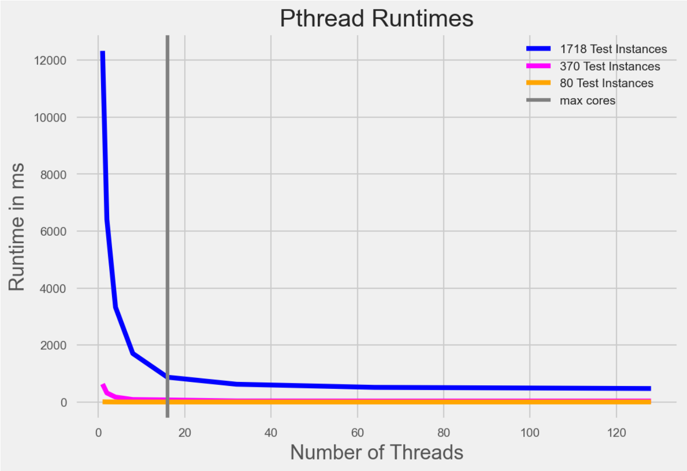
B. Speedups

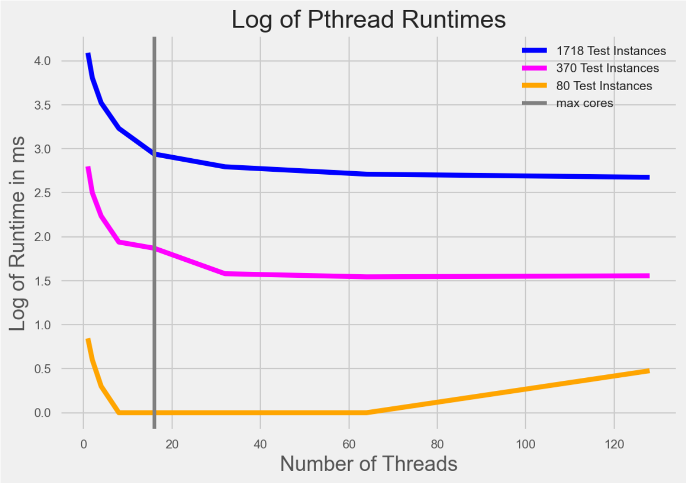
Following are speedups for the Pthread and MPI versions over the Serial version.

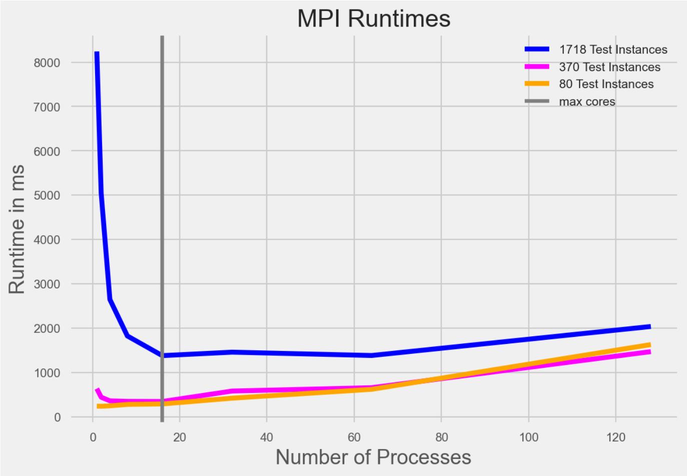
Number of threads	**Number of Test Instances**		
	80	***370***	***1718***
Pthreads			
1	1.0	0.94	0.962
2	1.75	1.871	1.848
4	3.5	3.448	3.564
8	7.0	6.816	6.957
16	7.0	8.014	13.61
32	7.0	15.605	18.997
64	7.0	16.943	23.062
128	2.333	16.472	25.008
MPI			
1	0.029	0.934	1.438
2	0.029	1.342	2.351
4	0.028	1.647	4.48
8	0.025	1.704	6.499
16	0.024	1.719	8.59
32	0.017	1.021	8.13
64	0.011	0.903	8.571
128	0.004	0.403	5.819

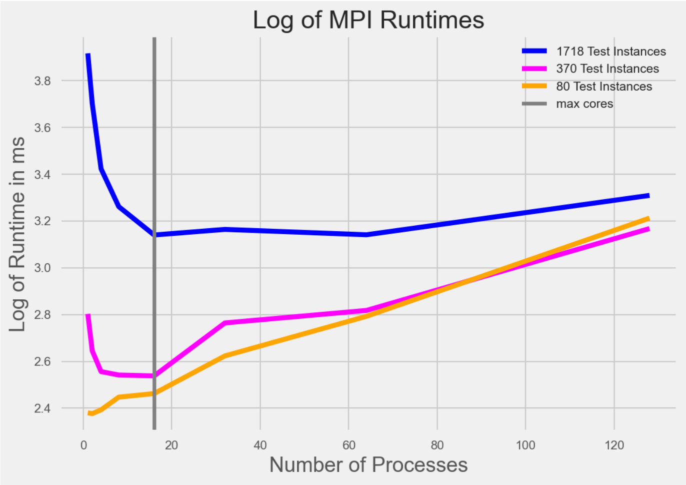

Serial Runtimes
Runtime in ms
0
2000
4000
6000
8000
10000
12000
80
370
1718
Number of Test Instances

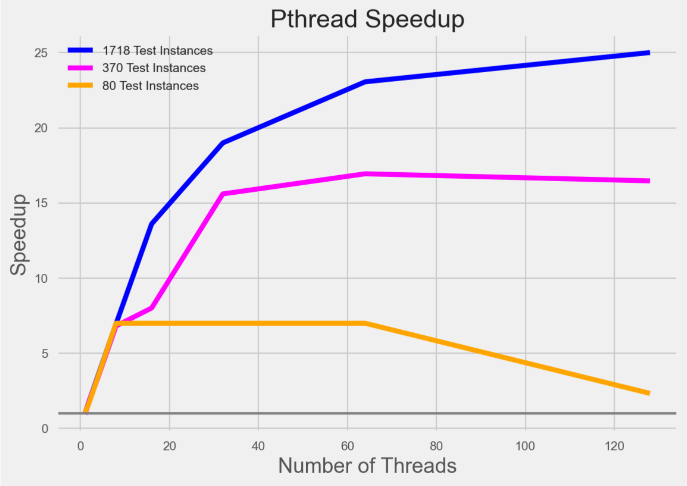

Log of Serial Runtimes
Log of Runtime in ms
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
80
370
1718
Number of Test Instances

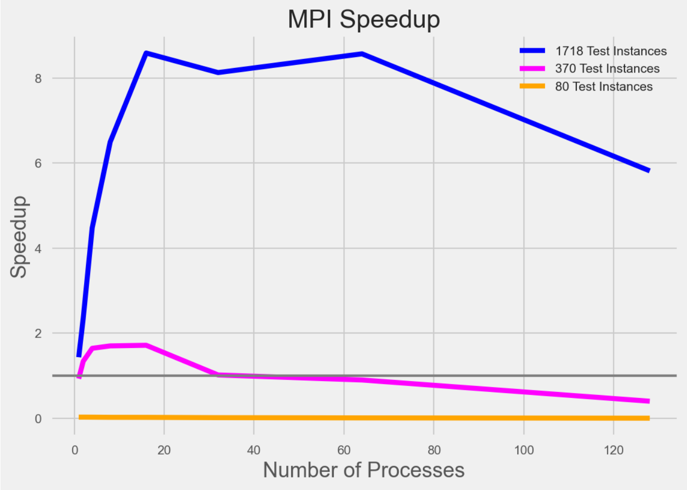

Pthread Runtimes
Runtime in ms
0
2000
4000
6000
8000
10000
12000
0
20
40
60
80
100
120
Number of Threads
1718 Test Instances
370 Test Instances
80 Test Instances
max cores


Log of Pthread Runtimes
Log of Runtime in ms
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100
120
Number of Threads
1718 Test Instances
370 Test Instances
80 Test Instances
max cores


MPI Runtimes
Runtime in ms
0
1000
2000
3000
4000
5000
6000
7000
8000
0
20
40
60
80
100
120
Number of Processes
1718 Test Instances
370 Test Instances
80 Test Instances
max cores


Log of MPI Runtimes
Log of Runtime in ms
2.4
2.6
2.8
3.0
3.2
3.4
3.6
3.8
0
20
40
60
80
100
120
Number of Processes
1718 Test Instances
370 Test Instances
80 Test Instances
max cores





C. Scalability

Anything above the dark grey horizontal line in the two images above represents an improvement in runtime for that number of threads. We can see that the pthread version was universally faster than the serial version. The speedup grew with the number of test instances and number of threads until 64 threads. Interestingly, despite the number of cores on the test machine numbering only 28, a greater number of threads does not slow the Pthread version much. It does, however, slow the already lethargic MPI implementation. Here, the runs with processes ¿16 were almost all slower than the immediately prior runs. In both implementations, we would very likely see continued improvement in performance as the size of the dataset grows.

IV. Conclusions

The benefits in runtime are clearly quite great with either implementation, especially as the amount of data increases. This improvement is only the beginning, as these implementations only parallelize the outer logic of the prediction of each test instance's class. In addition to this, we could parallelize the calculation of the distance to each neighbor and the mode of the neighbors' classes. It is worth noting that there is almost certainly a bug or two in the MPI version, as accuracy decreases with the number of processes by up to 3.5%. While the amount of setup prior to running the KNN algorithm involved in this version is significantly greater than the Pthread or Serial version, I had expected to see some improvement outside of the largest test set. More work will have to be done here.