CSCI 5461 Assignment -1

Name - Varun Vijaya Kumar (ID - 5296193)

Data structures:

1. **clusAvg** - data structure holding the centroids.
2. **data** - data structure holding the input data.
3. **cluster** - int array corresponding to number of data points and holding the value which indicates the cluster the data point belongs to.

Code is written to be run serially and in parallel with a makefile option, i.e -DUSE_PTHREADS and -DUSE_OPENMP for pThreads and OpenMP respectively.

Ideology :

Other than the input and output semantics. The core algorithm consists of 2 parallelizable parts:-

1. Assign new centroids - for all data points assign the closest centroid (euclidean distance).
2. If there are any changes in cluster assignments then update the cluster centroid.

I have tried maximum to convert step 1 and step 2 under parallel constructs. Step 1 and step 2 both consist of processing which is are independent of each and run for 0->numDataPoints. Thus for step 1 and step 2 individually, we can exploit parallelism here. There is an implicit barrier between the two steps, as step1 has to be completed to start step 2.

For pthreads - For step 1, all iterations are independent for numDataPoints iterations, thus distributed all the points by the numThreads by pThreads. Divergence is handled by last pthread, which takes last block and the remaining data points.
For step 2, we have to update the centroid(i.e. clusAvg). To access a shared variable by many threads, I have used a mutex lock to safely add the values and average it.

For OpenMP - For step 1, which spans over numDataPoints is easily divided by numThreads using OpenMP. To check if there was no changes in the clusters, I have used a reduction variable to check there was any change in the cluster assignment.
For step 2, I have to use omp atomic to average the cluster values from numDataPoints, also the counter is easily incremented using atomic function __sync_fetch_and_add(), as it is just an int incremental, whereas for centroid average, there is no atomic add for double datatype, thus had to use omp atomic.

Results :

Execution time:

Execution time in **seconds**

pTHREAD	Threads=1	Threads=2	Threads=4	Threads=8	Threads=16
Cluster=256	770.05	389.56	198.52	127.57	71.81
Cluster=512	1891.72	1222.19	672.34	436.01	140.57
Cluster=1024	3023.33	1592.14	867.52	522.9	287.38

Execution time in **seconds**

OPENMP	Threads=1	Threads=2	Threads=4	Threads=8	Threads=16
Cluster=256	811.51	485.38	232.12	138.88	89.66
Cluster=512	1675.88	843.16	431.78	230.66	149.54
Cluster=1024	2993.57	1658.3	796.88	471.45	243.3957

SpeedUp:

pTHREAD	Threads=1	Threads=2	Threads=4	Threads=8	Threads=16
Cluster=256	1	1.97	3.879	6.036	10.723
Cluster=512	1	1.54	2.81	4.34	11.21
Cluster=1024	1	1.89	3.49	5.78	10.52

Execution time in **seconds**

OPENMP	Threads=1	Threads=2	Threads=4	Threads=8	Threads=16
Cluster=256	1	1.67	3.49	5.84	9.05
Cluster=512	1	1.98	3.88	7.27	12.01
Cluster=1024	1	1.78	23.72	6.29	12.18