Report Lab 1 COL380

Shashwat Shivam
2016CS10328

The number of centroids in dataset =10.

Parallel Algorithm:-

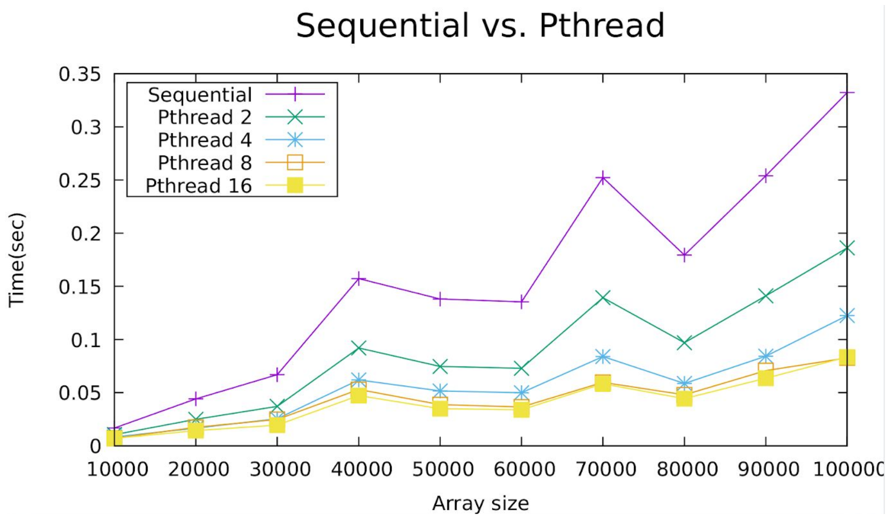
1. A structure was defined which stores x, y, z co-ordinates along with the number of points counted into that structure.
2. Using this structure a vector of vector of pointers to instances of this structure was defined. The size of this vector of vectors was **Num-threads x num of centroids**.
3. The first step of copying the data points to data points with cluster was parallelized by assigning each thread with its own range of values to modify.
4. After this seeded random initialization of the associated cluster of each point was done. (Sequentially to keep consistent with the sequential algorithm).
5. Now each thread is assigned its own range of data points using which the thread sums the values into respective centroid position into its own thread_aggregates vector of vectors where the first index is the tid and second index is the cluster number. (the thread aggregates also store number of points counted into each cluster so that it is easier to average afterwards).
6. After this, each thread is assigned a centroid number for which it collects each threads aggregates and averages out the value.
7. This average is written into the centroids array sequentially.
8. Now the data is again divided among the threads to assign closes cluster.
9. Steps 5 to 8 are repeated until number-of-changes in the iteration is less than 1 per cent of the data points or a certain max number of iterations is reached.

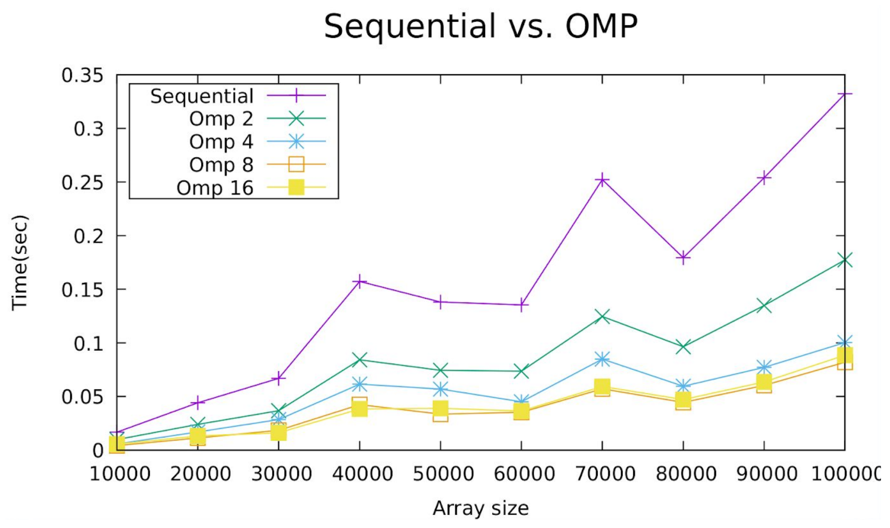
Expected Performance:-

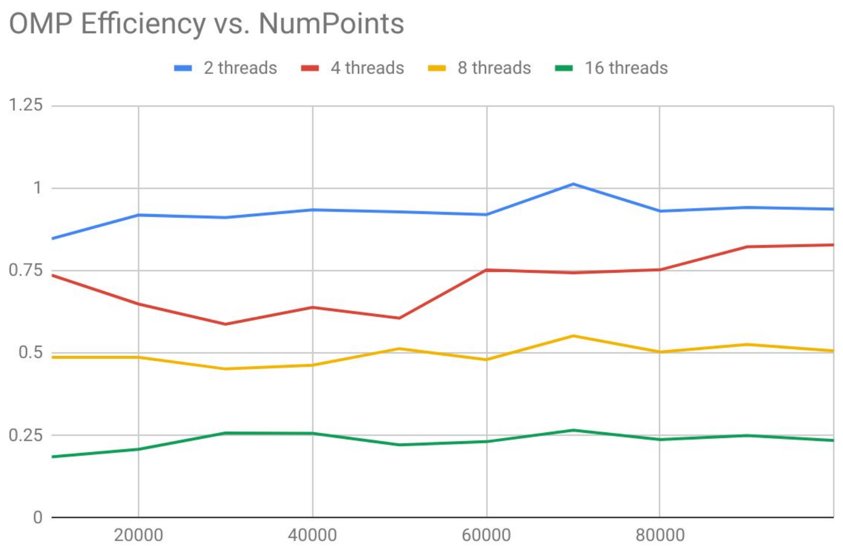
- Since all the steps are parallelized in the iterations to move the centroids, the expected speedup is high.
- The speedup is only reduced by the sequential parts of the writing of centroids into the array of centroids and initialization steps.
- Apart from these, the reduction in speedup can be attributed to recreating threads in pthread code (which will be slightly less in openmp code due to single creation of thread for do-while loop).
- Also, memory overheads are another overhead which reduces speedup.
- Assuming 15% of the code to be serial, according to Amdahl's the speedup achieved should be as follows:-
    - 2 threads:- 1.74
    - 4 threads:- 2.76
    - 8 threads:- 3.90
    - 16 threads:- 4.92
- Predicted Efficiencies:-
    - 2 threads:- 0.87
    - 4 threads:- 0.69
    - 8 threads:- 0.4875

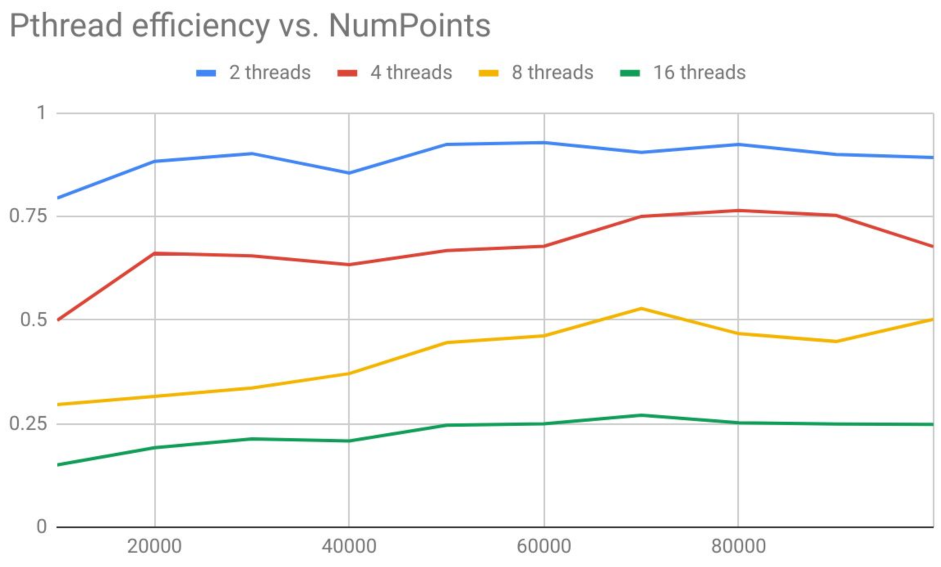
- 16 threads:- 0.3075

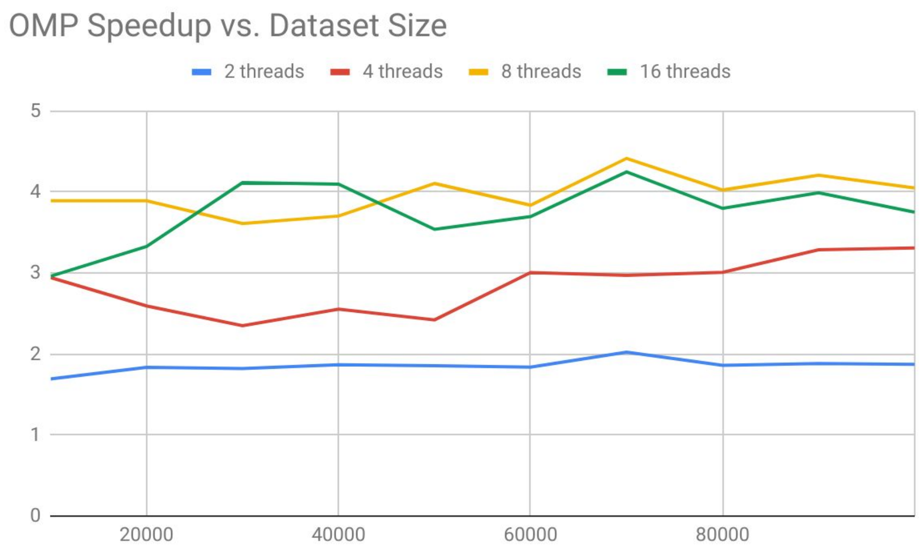
Actual Results (Graphs):-

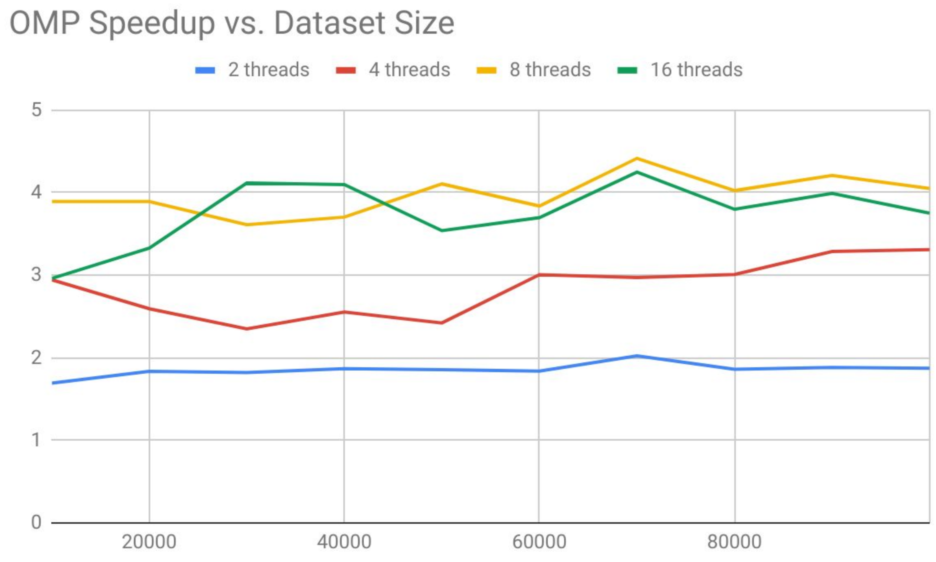


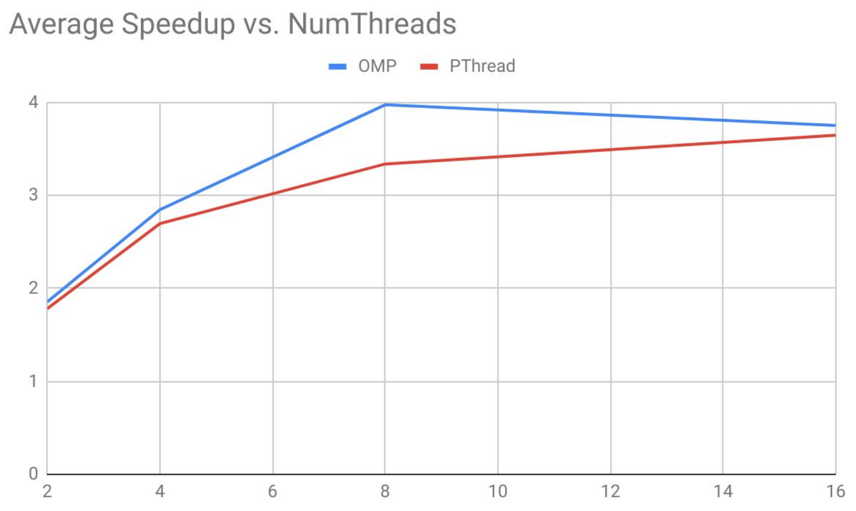





Pthread efficiency vs. NumPoints
2 threads
4 threads
8 threads
16 threads
1
0.75
0.5
0.25
0
20000
40000
60000
80000


OMP Speedup vs. Dataset Size
2 threads
4 threads
8 threads
16 threads
5
4
3
2
1
0
20000
40000
60000
80000


OMP Speedup vs. Dataset Size
2 threads
4 threads
8 threads
16 threads
5
4
3
2
1
0
20000
40000
60000
80000



Actual Results(Discussion):-

- The average speedups for OMP attained were as follows:-
    - 2 threads:- 1.854879724
    - 4 threads:- 2.844766551
    - 8 threads:- 3.973963162
    - 16 threads:- 3.752498189
- And for pthreads were as follows:-
    - 2 threads:- 1.781775458
    - 4 threads:- 2.696257705
    - 8 threads:- 3.337520411
    - 16 threads:- 3.647544049
- The speedups show a slight linear increase as the dataset size is increased.
- The speedup of 16 and 8 thread are very close due to the inability to run 16 threads simultaneously.
- We can see that the results are similar to those as predicted above. The deviations can be due to the following facts:-
    - 16 threads cannot achieve high speedups because of thread switching overheads on 8 thread device.
    - Pthread code recreates threads in every loop and therefore incurs high overheads as the number of threads increases.
- For efficiency, we can see that efficiency increases as the number of points increases and efficiencies are lower for a higher number of threads. The predicted values are also close to real values.