# LAB 1 REPORT

The Lab consists of three implementations of the K-Means Algorithm :-
1. Sequential
2. Parallel using Pthreads
3. Parallel using OpenMP

**Structure of Sequential Code :**
1. K-means functions is called.
2. Random centroids are generated.
3. Distance of each point is calculated from all the centroids and each point is assigned a cluster(closest centroid) based on this.
4. The number of data points for which the new cluster assignment is different from previous one are counted(**changed_points**).
5. New Centroids are computed for each cluster.
6. If the **changed_points==0** we are done else we loopback to step 2.
7. Write the clusters obtained in data_point_cluster.

The parallel codes are directly adapt from the sequential one adding locks in the critical sections and parallelising the components that can be done parallely.

Of the above 3,4 and 7 are done in parallel in the Pthread and OpenMP versions.

## PARALLELISATION STRATEGY

**In steps 3 and 4 the following tasks are done.**
a) For each point check to which cluster it belongs w.r.t. the current centroids.
b) Maintain (i)Sum of coordinates of points. and (ii) Count of points. ; in the new cluster to obtain the new centroids.

To Do the above each thread maintains three new **private variables**:
i) private_changed_points ii) private_new_cluster_sum iii) private_new_cluster_count

**Computation for a data-point**
For each point we have to (The code is also well documented to Understand)
i) Compute its distance from all K current centroids.
ii) Check which is minimum.
iii) Check if the new cluster is different from the previous cluster assignment.
iv) Update the private variables accordingly.

**Equal Load Distribution**
Since computation done for each datapoint is same the load balancing between the threads came inherently. And each thread was give same size iteration space of data-points.

**Thread Computation**

i) For each point in its iteration space a thread does the "Computation for a data-point" and update it's private variables.
ii) Acquire a lock and update the three global variables.
    i)changed_points ii) new_cluster_sum iii) new_cluster_count
iii) Frees the lock.

**Why Private Variables?** By maintaining a private variable we escape from

1.**Data Race** :- Each thread acquires the lock before updating the global(shared) section.
2.**False-sharing** :- Had we maintained a shared array (with divided space to work upon) the also due to Cache-coherency this would have led to weak performance.
3.**Updating the global variable every time** :- Had we updated the global variables each time then due to too much of lock acquiring and realising it would have weaken our performance. Hence, we are only updating the global variables once a thread has done all its computation in it's iteration space.

**In step 7 :**

We only have to the final cluster assigned to all the data-points in the data_points_cluster variable which has to begin as output.

As we only have to write and write of each datapoint is independent of the other.
1) We just divide the the iteration space of N points into t sections where t is the number of threads.
2) Each thread writes the cluster of the points in its tiertion space to the data_points_cluster variable.
3) We wait until all the threads have written their part.

Hence the K-Means is computed

**PLOTS ATTACHED BELOW**

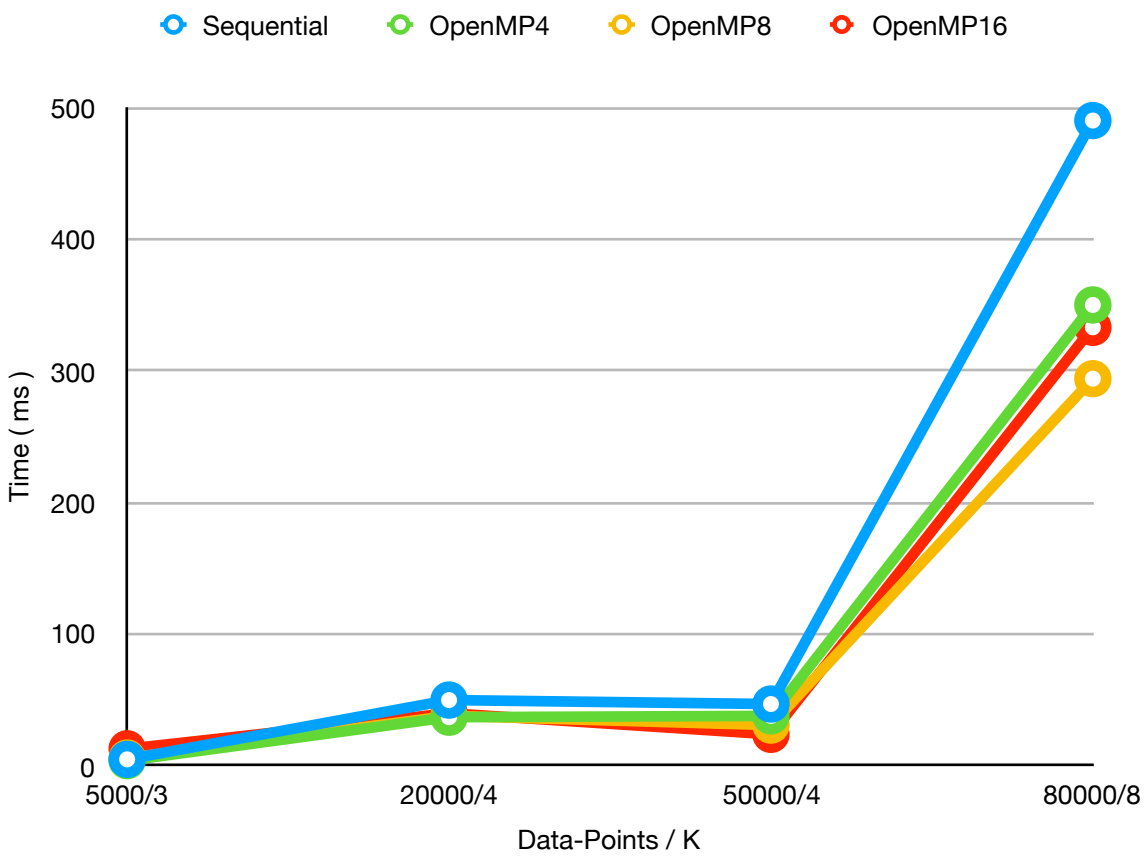
**Conclusion**

1. OpenMP provides easy user interface. We only have to write a few lines and we have parallelised the code, however it is actually the complex Pthreads syntax which is behind it. This explains the higher wall clock time taken by OpenMP code than Pthreads code.

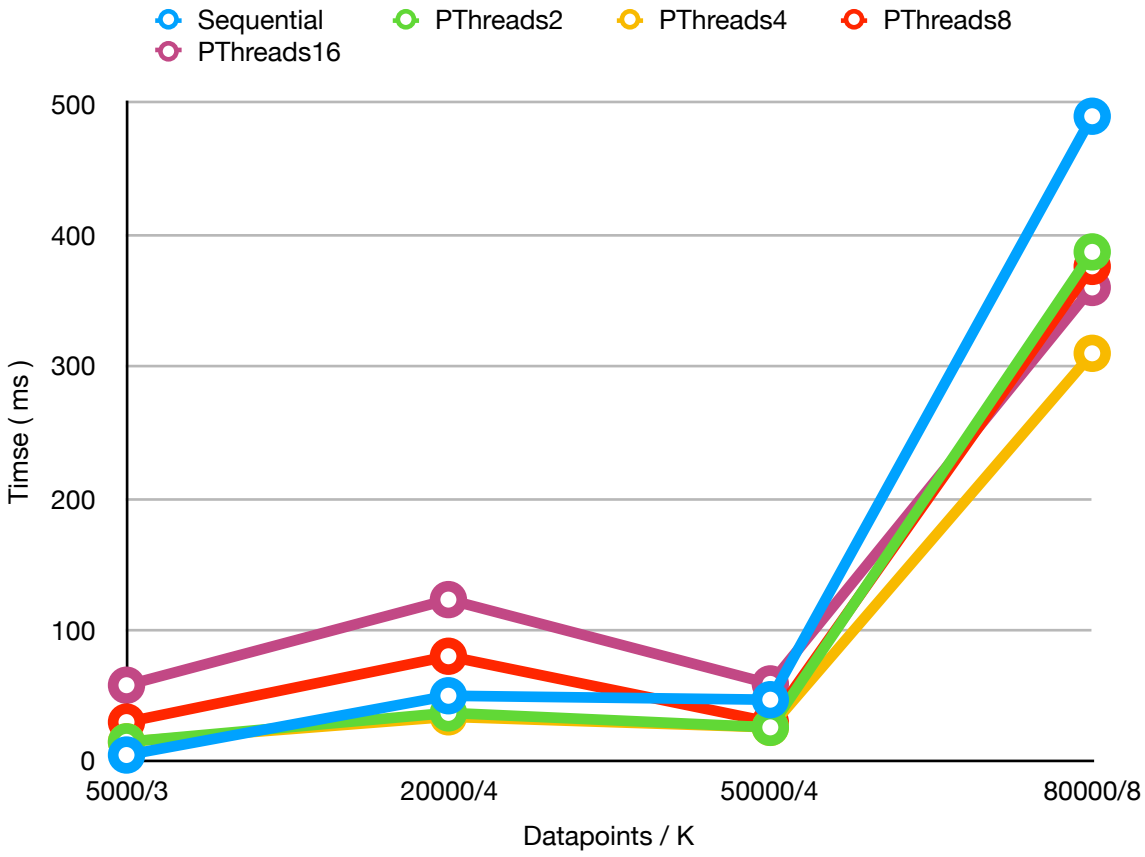
2. Higher number of threads used for smaller number of tasks leads to overhead du to initialising, extra computation required by threads and synchronisation. Hence explains the large time taken 16 threads for smaller inputs.

3. As the number of cluster increases the time taken explodes. This can be explained as most of the serial component highly depends on K.

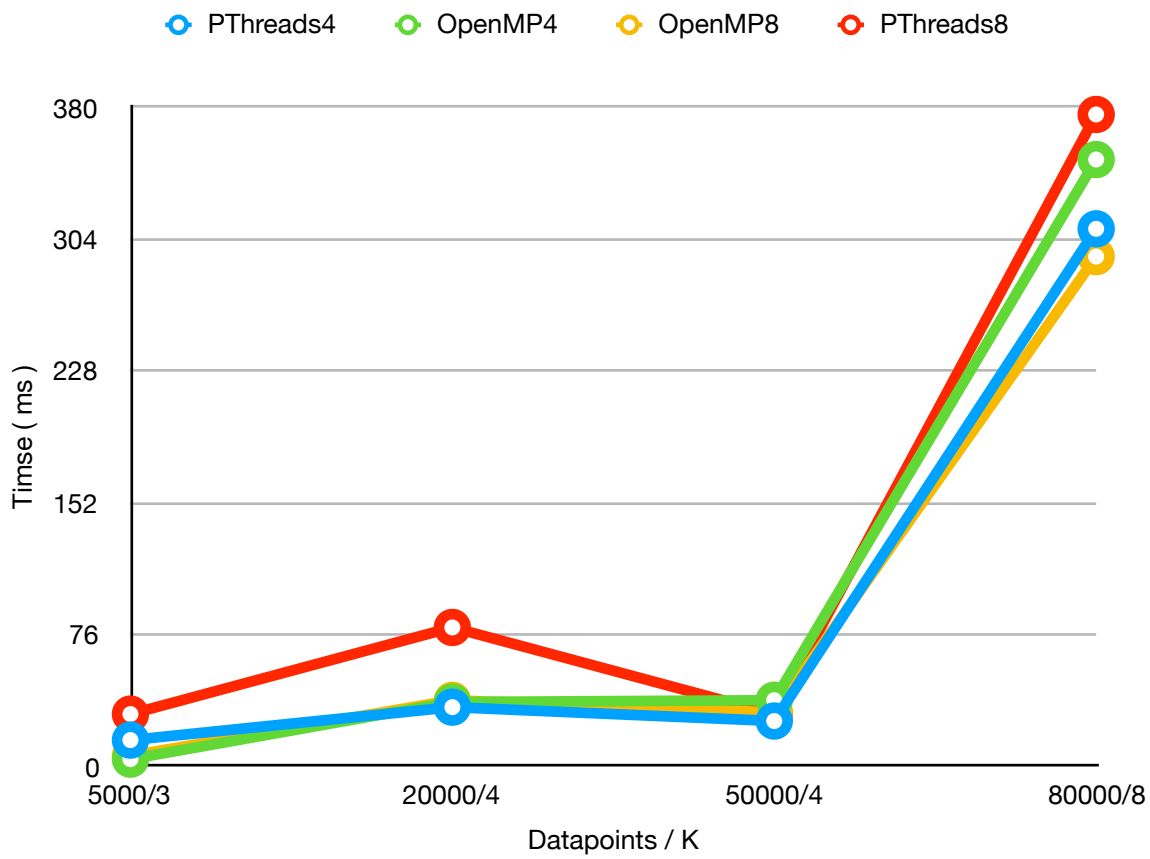
## Sequential Vs OpenMP




## Sequential Vs PThreads



OpenMP Vs PThreads




Sequential Vs PThreads Vs OpenMP for Constant K