K-Means clustering

Pradyumna Meena - 2016CS10375

Default terms

- N = number of points in the dataset
- K = number of clusters
- Num_threads = number of threads to be used in openmp and pthread implementation
- Maximum number of iterations allowed = 200

Key aspects of the implemented algorithm

These are few key aspects which had a significant impact on the performance of the program compared to others

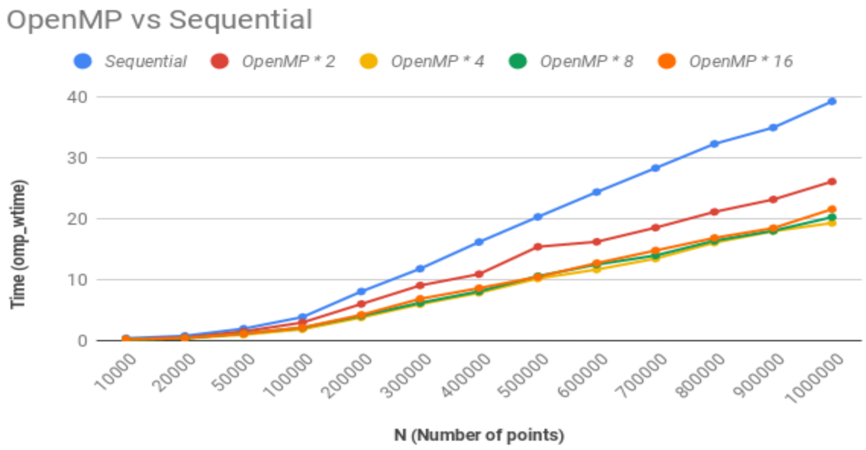
- No use of data structures like vector, arrays to avoid risk of refetching in case of false sharing. Pointers enable us to access same memory location from different threads without the risk of false sharing though data races are still there
- Avoiding additional function for small tasks like distance between two points. Functions doing such small tasks when called repetitively have significant effect on performance

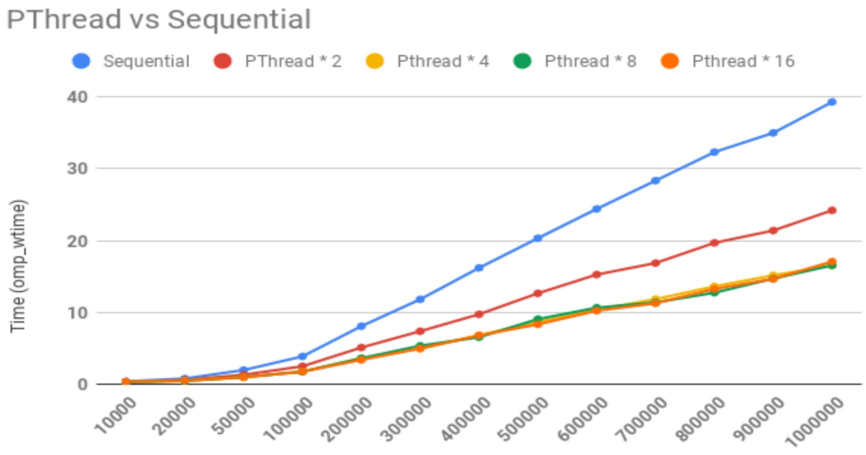
Underlying risks of false sharing and data races

As said above with the use of vectors or arrays comes the risk of false sharing since these had to be shared between the threads and if not taken care properly can lead to refetching of data again and again. Data races come into picture when centroids are updated. In a parallel implementation normal strategy would be to divide the initially given points into N/K parts and each thread updates the centroid depending on the cluster it was assigned to. This may lead to two threads updating the same location and hence inconsistency arises. But these are benign data races since at the end we want the sum to be correct irrespective of the order in which they do it. Even if one or two points slip away it's not really that much important at the scale of points with which we are dealing in the problem.

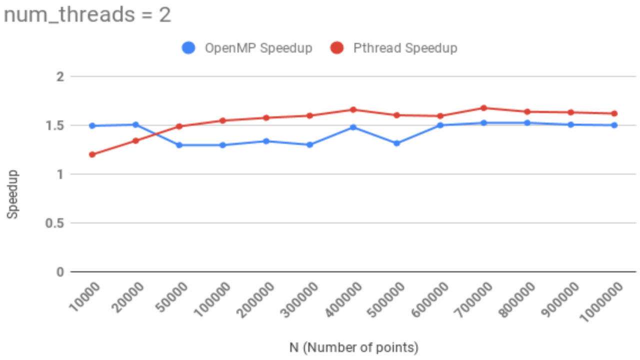
Performance Plots

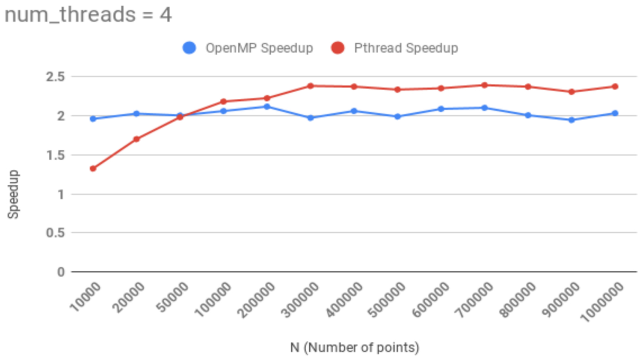
- Number of threads and number of points were varied to observe the performance
- **Convergence criteria:** Number of iterations = 200
- Plot Details
  - First plot shows the time taken by various variations of the program
  - Second plot is for the speedup for openmp and pthread across various values of num_threads.
  - Third plot compares openmp and pthread speedup values.
  - Fourth plot tells about the efficiency of various variations.

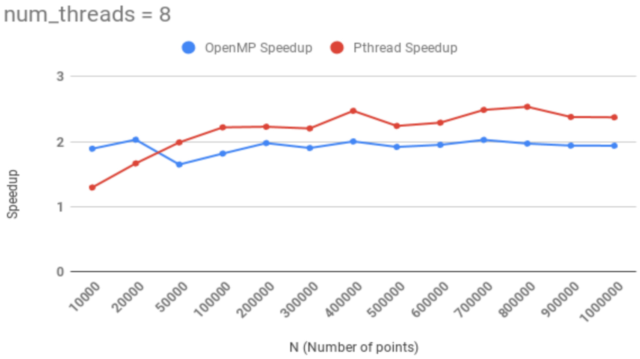


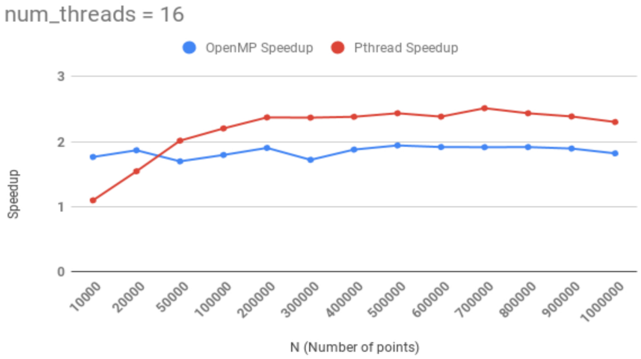


Speedup plot

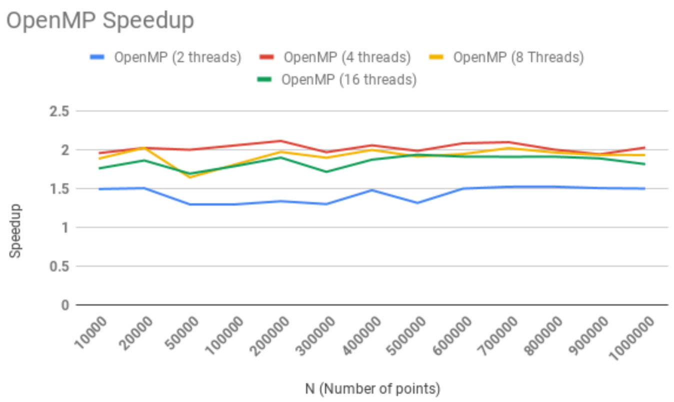


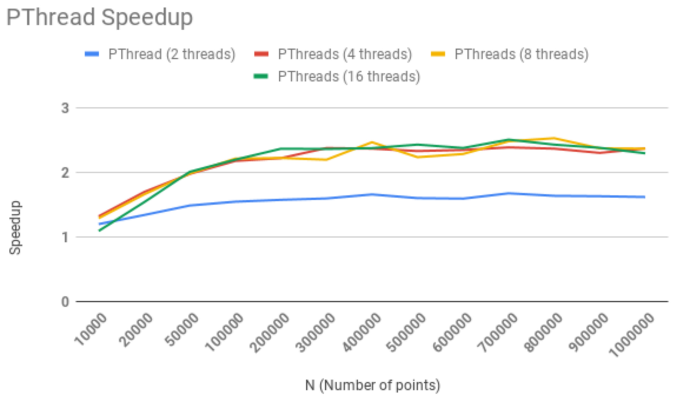




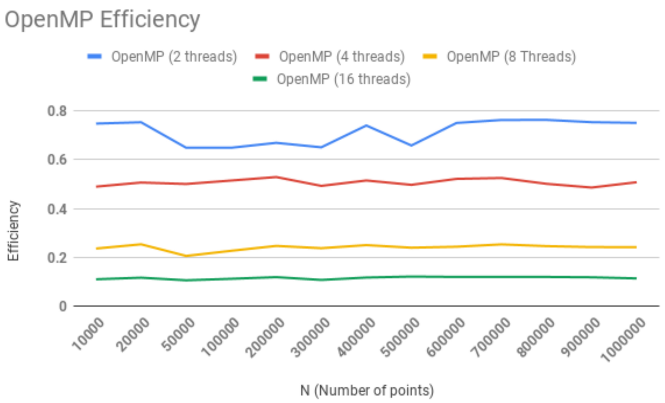


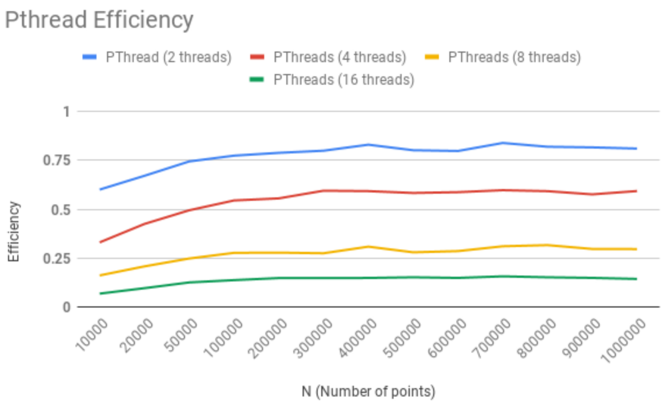
Speedup Profile





Efficiency





Observations

- While varying num_threads choosing 4 threads gives best time. The explanation to it is the fact that my machine has 4 cores. Hence if more threads are simulated then context switching happens and therefore more time is used because of storing state of thread and loading it again since at a given time at max 4 threads can run on machine.
- Speedup for pthread saturates after a particular data-set while for openmp it varies a bit but lies within a range.
- Efficiency for pthread increases and then becomes constant for a particular number of threads.