Assignment 1 Report

K-means Clustering

Rahul Bansal

2016CS10344

Design Decisions-

- Firstly I created a C++ class which will form an object for each point and each of my centroids is an object of this class. This simplifies the code and makes it easy to manipulate the values of centroids in each iteration of the k-means algorithms.
- Implemented another function closest which will tell the closest centroid to a data-point. This was done to make the code parallelizable.
- The algorithm converges when after an iteration of the while loop the centroid number of each of the data-point does not change.

Parallelization Strategy-

- The loop where the closest centroid is located for each of the data-points is parallelised and this loop calls one function in each iteration I.e. the closest function.
- In that function the value of centroid is only read so it can be parallelised for multiple data-points and corresponding to each data-point the value of the closest centroid is updated in the data-point indexed array.
- This loop is executed by multiple threads where each thread performs the closest function for some data-point and updates the value in the c_number array.

Load Balancing Strategy-

- In the Pthreads implementation of the k-means algorithm we have divided the entire loop for N points into loop of N/P points for each of the threads where P - number of threads.
- So each of the thread will have N/P data-points.
- This means the amount of computation for each of the threads is similar so the load is balanced in all the threads.

Program timings for Parallel and sequential implinemtation

n(Number of Points)	k(Number of clusters)	Sequential	pthreads	OpenMP	Num_threads
50000	4	0.040	0.031	0.026	2
			0.027	0.028	4
			0.029	0.027	8
100000	4	0.321	0.225	0.204	2
			0.200	0.224	4
			0.202	0.193	8
200000	5	0.452	0.302	0.293	2
			0.282	0.307	4
			0.275	0.273	8
300000	6	1.154	0.736	0.714	2
			0.670	0.733	4
			0.671	0.664	8
400000	4	0.169	0.142	0.123	2
			0.122	0.136	4
			0.119	0.119	8
500000	5	0.384	0.269	0.249	2
			0.236	0.246	4
			0.236	0.236	8
100000	10	0.670	0.399	0.383	2
			0.365	0.392	4
			0.368	0.357	8
300000	12	2.419	1.403	1.385	2
			1.283	1.383	4
			1.345	1.289	8

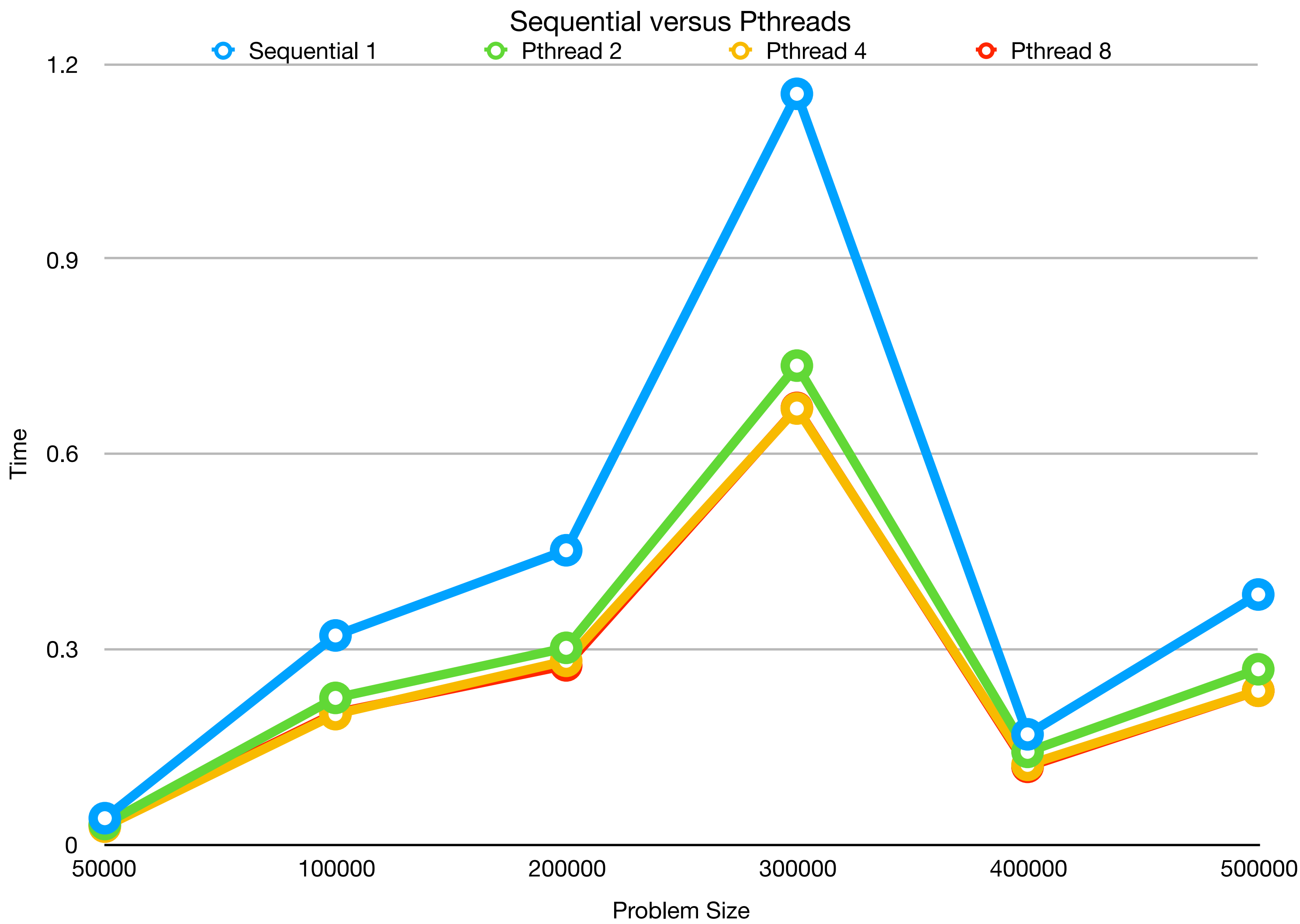
Speed-up of the Program (S) = Sequential time / Parallel time

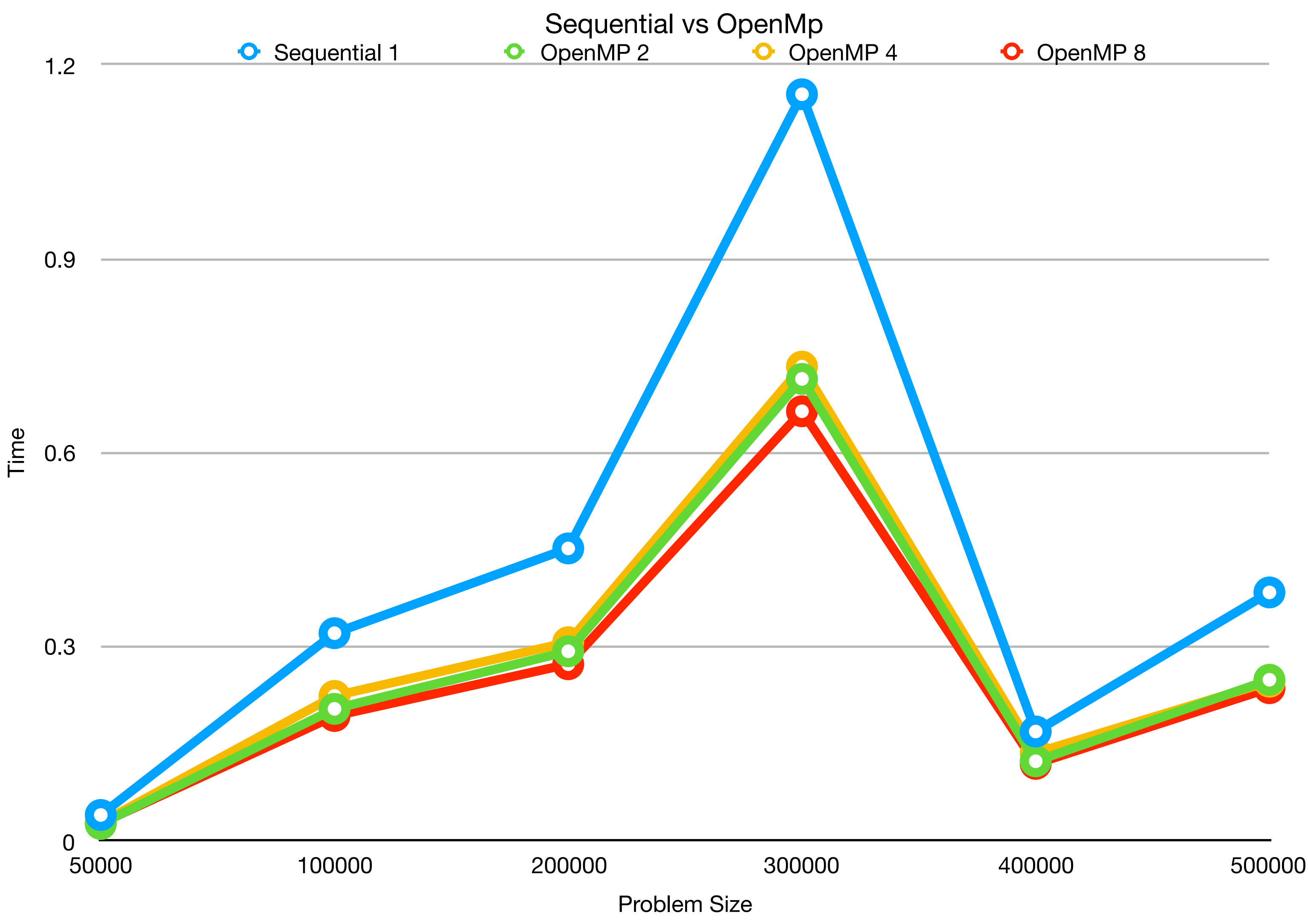
Efficiency (E) = Speed-up(S)/number of threads

<u>For problem size of 500000 </u>-

OpenMP	2-thread	4-thread	8-thread
Speed-up	1.54	1.56	1.62
Efficiency	0.77	0.39	0.20

Pthreads	2-thread	4-thread	8-thread
Speed-up	1.42	1.62	1.62
Efficiency	0.71	0.40	0.20


Sequential versus Pthreads
Sequential 1
Pthread 2
Pthread 4
Pthread 8
1.2
0.9
0.6
0.3
0
Time
50000
100000
200000
300000
400000
500000
Problem Size


Sequential vs OpenMp
Sequential 1
OpenMP 2
OpenMP 4
OpenMP 8
1.2
0.9
0.6
0.3
0
Time
50000
100000
200000
300000
400000
500000
Problem Size

Plots for Speed-up and Efficiency using a problem size of 500000-