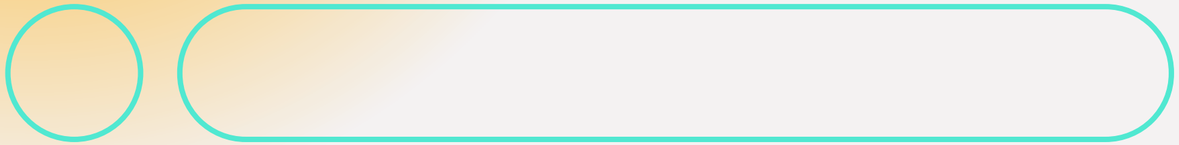
Parallel Implementation of K-means clustering algorithm

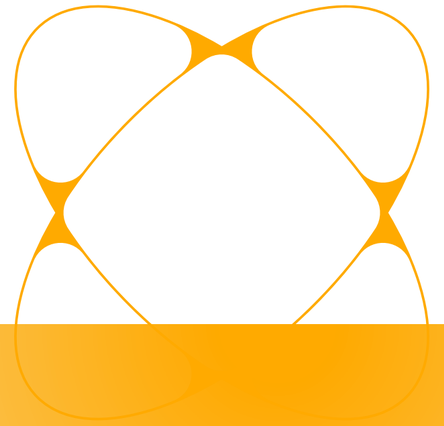
High Performance Computing

Srikiruthika S - 21011101108
Nakshatiraa K N - 21011101080



INTRODUCTION

K-means algorithm is an iterative approach to partition the given dataset into K different subsets or clusters, where each data point belongs to only one cluster. It assigns data points into a cluster such that the sum of squared distances between data points is minimum. The lesser the variation we have inside the cluster, the more the homogenous clusters we get.



PROJECT PIPELINE

APPROACH

Our approach to parallelizing the K-means algorithm focuses on leveraging data-parallelism for efficient clustering while addressing synchronization challenges:

- **Initialization:** The first K data points are chosen as the initial
- **Data-parallelism:** Each thread assigned N/num_threads number of points.
- **Thread function:** Each thread runs a loop (with a max_iter value of 100), in every iteration of which it computes the closest cluster centroid for every point assigned to it, and then assigns the point to that cluster.

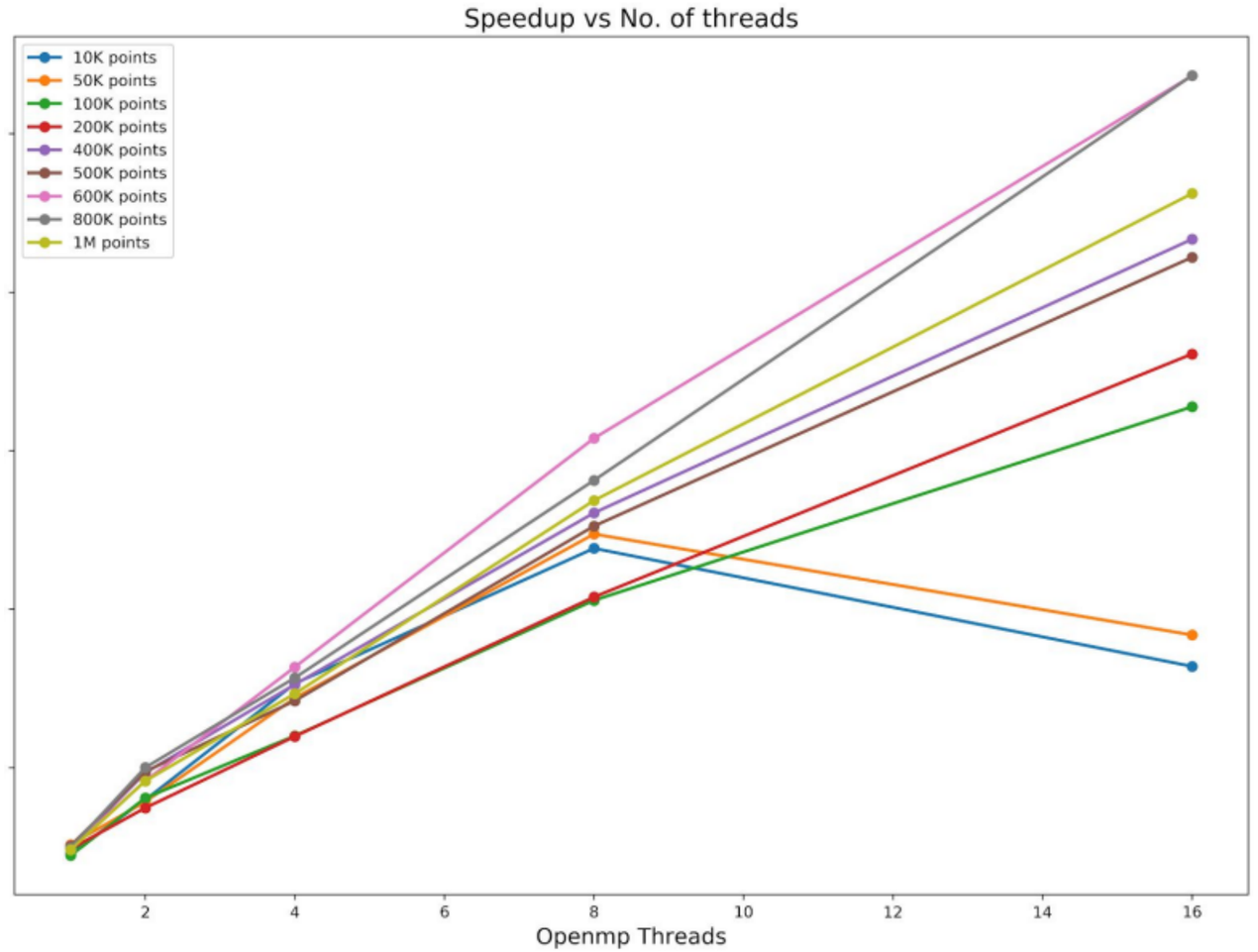
APPROACH

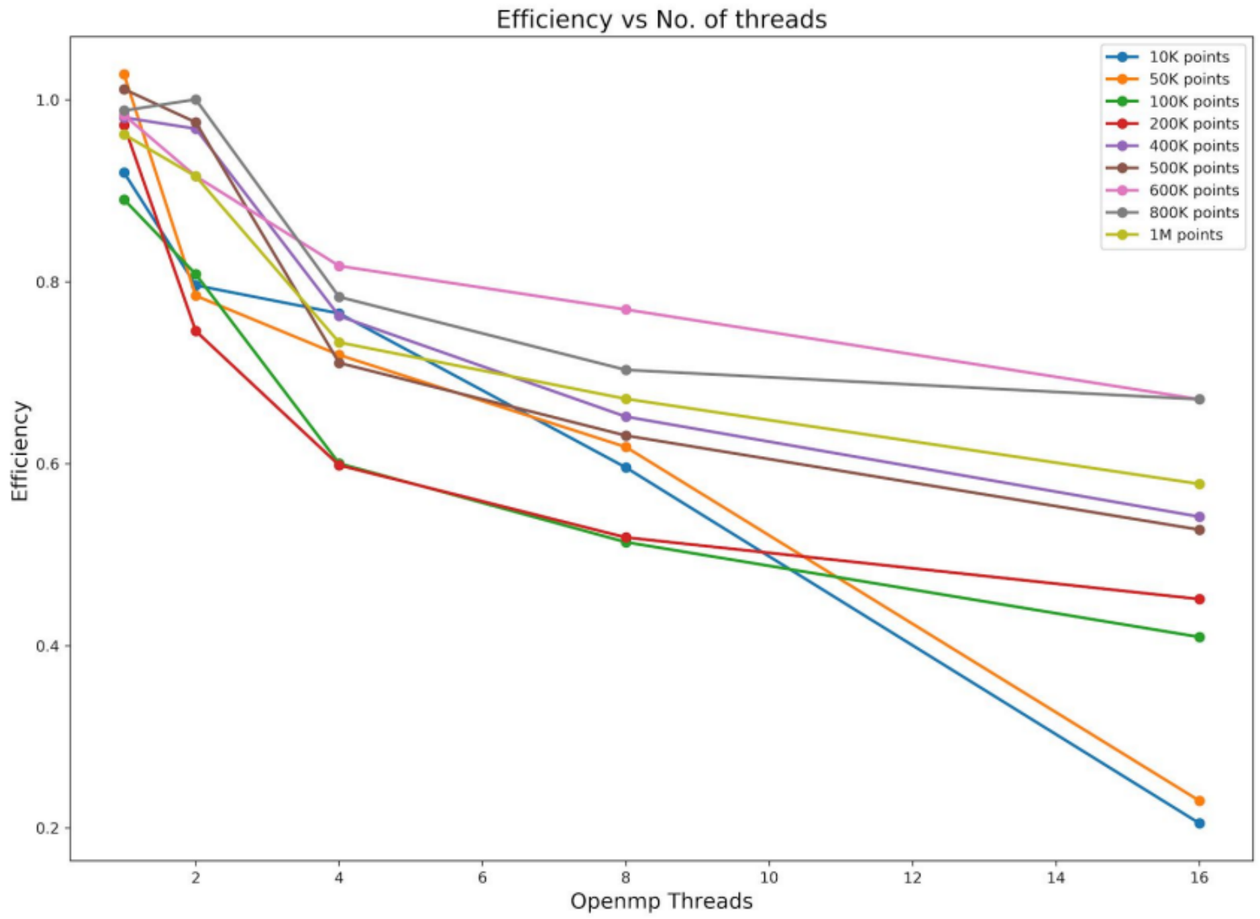
After every point is assigned to a cluster, the global (shared) cluster centroids are updated with the values computed for their coordinates from the points that were assigned to that cluster. This is again an instance of data-parallelism.

- **Stopping-condition:** The L2-norm is computed for every cluster centroid coordinates by comparing against corresponding values in the previous iteration. The norms are then summed and compared against the threshold value, chosen to be 1e-6. All threads break from the iterations loop as soon as the delta value goes below the threshold

SPEED UP vs N

Theoretically, if the problem size is too low, the speedup decreases as the number of cores increases. This is because the cost of overheads will be far more than actual cost of computation, thereby giving low speedup. But for larger problem sizes, the overhead cost is negligible and the work could be distributed among many processors, thus the speedup increases as the number of processors increases. The best speedups are obtained for the larger-sized threads (sizes 800,000 and 1 million). The worst performance is for the dataset-sized 10,000.

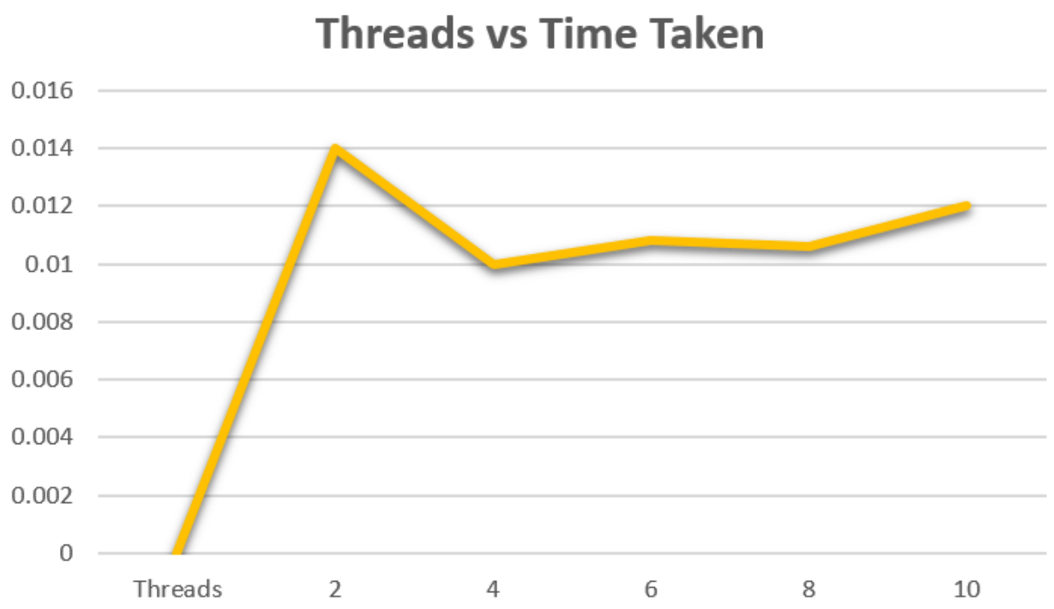


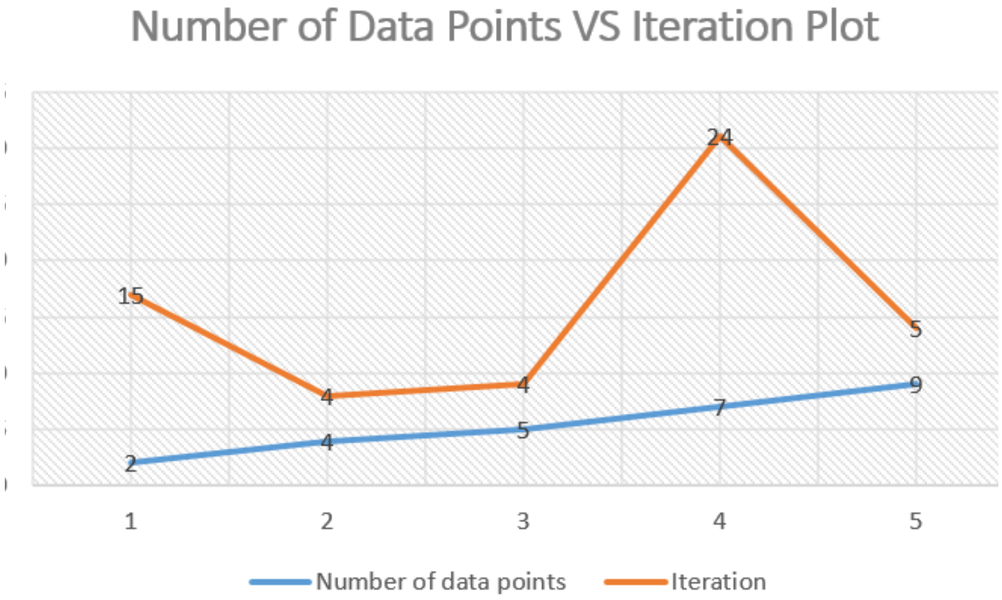


EFFIECIENCY vs N

The efficiency curves decrease for every instance run with an increase in the number of threads. This is again due to the diminishing returns property on including an extra thread - a consequence of the Amdahl’s Law. c

RESULTS


Threads vs Time Taken
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
Threads
2
4
6
8
10


Number of Data Points VS Iteration Plot
15
4
4
24
5
2
4
5
7
9
1
2
3
4
5
Number of data points
Iteration

RESULT

The parallel implementation of the K-means clustering algorithm demonstrated remarkable performance improvements as the number of threads increased, with computation times approaching negligible values for 16 threads.

However, achieving linear speedup proved challenging due to contention for shared resources and communication overheads. While superlinear speedups were observed in some single-threaded runs, efficiency curves exhibited diminishing returns with additional threads, aligning with Amdahl's Law. Notably, efficiency increased with larger problem sizes, indicating better resource utilization. Despite anomalies where efficiency exceeded 1.0 at num_threads = 1, likely due to system latencies, overall efficiency peaked at larger dataset sizes (800,000 and 1 million) and decreased for smaller sizes (10,000).