Implementation of Parallel K-means Clustering Using Elkan's Algorithm

Shriya Sridhar
Northern Arizona University
Flagstaff, Arizona - 86001.
Email: ss3874@nau.edu

***Abstract*—This project aims to implement a parallel K-means clustering algorithm and optimize the number of distance calculations required by means of applying the Elkan's algorithm. Further, the effects of parallelizing the program and utilizing a distributed system to improve performance are studied. The improvement in performance is measured by metrics such as speedup and parallel efficiency.**

I. Introduction

K-means clustering is a machine learning algorithm in which a number of data points are grouped into clusters, and the cluster they are grouped into depends on their position with relation to the centroid or mean of the cluster. This partitions a plane into Voronoi cells or clusters, which are the partitions a plane where each data point of a partition is closest to 'seed' or center of that cell. The centroid in turn represents all the points in the cluster.

The K-means clustering algorithm is useful to find unknown groups that have previously not been identified in the data. Thus we can find groups that have not been explicitly labeled, and this grouping can be used to classify new data available in the future.

The naive Lloyd's algorithm for K-means clustering involves calculating the Euclidean distances from each point to each centroid and then assigning the point to the cluster whose centroid is at the least distance from that point. Then, the centroid is recalculated as the mean of all the points that are now assigned to the cluster. This process is repeated till the points remain in the same cluster and the centroids do not vary in position with each iteration, and is said to reach convergence at this point.

The naive algorithm has a complexity of *O(Nkdi)*, where *N* is the number of points to be classified, *k* is the number of centroids or clusters, *d* is the number of dimensions of the data, and and *i* is the number of iterations needed until the algorithm converges.

Since the naive Lloyd's algorithm measures the distance between each point and each centroid at every iteration, there are *Nk* distance calculations at each iteration, irrespective of whether it is the first or last iteration. However, we might find that most points are obviously closer to some centroids compared to other centroids located at farther distances. Thus, the distance calculations to verify that these centers are far away are quiet redundant.

Further, for the initial iterations, the positions of the centroids may fluctuate heavily, especially depending upon how the initial centroids are chosen. Therefore, most of the distance calculations are necessary due to the wildly different positions of the centroids over different iterations. However, as the algorithm proceeds over successive iterations, the centroids become more stable and do not vary much from their previous position. Therefore, points which were close enough to the centroid will not get reassigned to another centroid despite the position of the centroid changing, since it only varies by a small distance. In these cases, we can reasonably conclude that *Nk* distance calculations are not required and are mostly redundant.

The Elkan's algorithm aims to reduce the number of distance calculations required by eliminating these redundant distance calculations. We determine a lower and upper bound derived from the triangle inequality theorem. Beyond these bounds a point is guaranteed to be within or outside the cluster respectively. Thus we need to calculate the Euclidean distances only for those points which lie within this smaller range, rather than all the points at each iteration.

This project aims to implement the Elkan's algorithm to determine the k-means clustering of an input dataset, parallelized to run on a distributed system using OpenMPI.

II. Theoretical Explanation for Elkan's Algorithm

The Elkan's Algorithm is largely dependent on the triangle inequality theorem to determine the redundancy of the distance calculations. [1] The triangle inequality theorem is stated that,

Theorem 1. *The sum of any two sides of a triangle is always greater than or equal to the measure of the third side, (i.e.) For a triangle with sides a, b, c,*

$$c \leq a + b.$$

From this theorem, two lemmas are derived to help determine the upper and lower bounds. For the first lemma, we apply the triangle inequality to the triangle formed between a point, and two different centroids.

Lemma 1. *For any point x and any two centroids c_1 and c_2, if $d(c_1, c_2) \geq 2d(x, c_1)$, then $d(x, c_2) \geq d(x, c_1)$.*

Proof. From the theorem, we have $d(c_1, c_2) \leq d(x, c_1) + d(x, c_2)$. Thus, $d(c_1, c_2) - d(x, c_1) \leq d(x, c_2)$ or $d(x, c_2) \geq$

$d(c_1, c_2) - d(x, c_1)$. Given the condition $d(c_1, c_2) \geq 2d(x, c_1)$, it follows that $d(x, c_2)$ is also greater than (or equal to) $d(x, c_1)$. □

Thus, the distance to second center is greater than the first one given the condition is satisfied, eliminating it from being a possible centroid for the point to be assigned to.

The lemma can also be used with an upper bound $u(x)$. If this upper bound itself follows the condition $u(x) \leq \frac{1}{2} \min d(c_1, c_2)$, then all the centroids other than c_1 will satisfy the given condition for lemma 1 and none of the distances need to be calculated.

The second lemma helps to reduce distance calculations over successive iterations. Consider that for a point *x*, the current assigned centroid is *c*, and the new mean value computed from all the points in the cluster is *c'*.

Lemma 2. *For any point x and centroid values c and c'. $d(x, c) \geq max\{0, d(x, c) - d(x, c')\}$.*

Proof. We have $d(x, c) \leq d(x, c) + d(c, c')$. Thus, $d(x, c) - d(c, c') \leq d(x, c')$. or $d(x, c') \geq d(x, c) - d(c, c')$. □

This lemma gives us a bound on the variation of the distance from the point *x* to *c'* based on the distance between old and new values of the centroid and which provides an approximation of the new distance $d(x, c')$. $d(x, c')$ is calculated only if absolutely necessary, such as when the centroid has fluctuated greatly, or the point is on the border of the cluster it was assigned to.

The number of distance calculations required depends heavily on the number of centroids *k*, and since the number of centroids chosen is generally chosen such that $k \ll N$, the number of calculations is much lesser than the naive algorithm. For the first lemma, we compute the distance matrix $d(c_1, c_2)$ for all pairs of centroids, resulting and k^2 calculations. For the second lemma, we compute the distance *d(c, c')* between the old and new values of each centroid, resulting in *k* distance calculations. Despite arbitrarily calculating these at each iteration, the total number of distance calculations still reduces dramatically considering that $k \ll N$.

III. Steps for Implementation of Algorithm

A. Initialization

1) Pick initial values of centroids.
2) Set the lower bound $l(x, c) = 0$ for all values of *x* and *c*.
3) Compute the distance matrix $d(c_1, c_2)$ for all pairs of centroids.
4) Determine $\min d(x, c)$ and use lemma 1 to avoid redundant calculations of $d(x, c)$. Assign the corresponding value of c as *c(x)*.
5) If *d(x,c)* is computed, change the value of *l(x,c)* to the value of *d(x,c)* computed.
6) Assign upper bound *u(x)* as $\min d(x, c)$.
7) Set flag *r(x)* as true for all *x*.

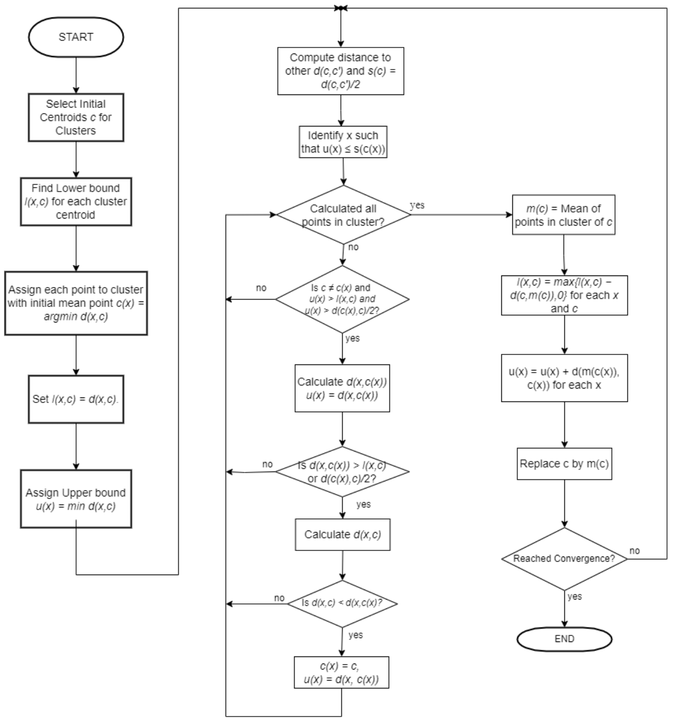

START
Select Initial Centroids c for Clusters
Find Lower bound l(x,c) for each cluster centroid
Assign each point to cluster with initial mean point c(x) = argmin d(x,c)
Set l(x,c) = d(x,c).
Assign Upper bound u(x) = min d(x,c)
Compute distance to other d(c,c') and s(c) = d(c,c')/2
Identify x such that u(x) ≤ s(c(x))
Calculated all points in cluster?
yes
no
Is c ≠ c(x) and u(x) > l(x,c) and u(x) > d(c(x),c)/2?
no
yes
Calculate d(x,c(x)) u(x) = d(x,c(x))
Is d(x,c(x)) > l(x,c) or d(c(x),c)/2?
no
yes
Calculate d(x,c)
Is d(x,c) < d(x,c(x)?
no
yes
c(x) = c, u(x) = d(x, c(x))
m(c) = Mean of points in cluster of c
l(x,c) = max{l(x,c) – d(c,m(c)),0} for each x and c
u(x) = u(x) + d(m(c(x)), c(x)) for each x
Replace c by m(c)
Reached Convergence?
no
yes
END

Fig. 1. Flowchart for Elkan's algorithm

B. Repeat until Convergence

1) For all centers c_1 and c_2, compute $d(c_1, c_2)$.
2) Identify the minimum distance for each c, (i.e.) $\min d(c_1, c_2)$ and set $s(c) = \frac{1}{2} \min d(c_1, c_2)$, where $c_1 \neq c_2$.
3) Identify all points *x* such that $u(x) \leq s(c)$.
4) Among the remaining points, identify points which satisfy the following conditions:
   a) $c \neq c_1$
   b) $u(x) > l(x, c)$
   c) $u(x) > \frac{1}{2} d(c(x), c)$

   If the above conditions are satisfied, proceed with two steps noted below:

   a) If the flag *r(x)* is true, compute *d(x,c(x))* and reassign flag *r(x)* as false. Else, set *d(x,c(x)) = u(x)*.
   b) Reassign *u(x)* to *d(x,c(x))* if it has been calculated.
   c) If $d(x, c(x)) > l(x, c)$ or $d(x, c(x)) > \frac{1}{2} d(c(x), c)$, then compute *d(x,c)*.
   d) Reassign *l(x,c)* to *d(x,c)* if it has been calculated.
   e) If *d(x,c)* ¡ *d(x,c(x))*, then reassign *c(x)* to *c* and flag *r(x)* as true.
5) For each center c, let m(c) be the mean of the points assigned to c .
6) For each point x, calculate new lower bound l(x,c) as max of (l(x,c) - d(c,m(c)), 0)
7) Calculate new upper bound u(x) as u(x) + d(c(x),m(c(x))).
8) Replace each center c by m(c).

The flowchart in Fig.1 gives a diagrammatic representation of the steps for implementing the algorithm.

For the parallelized implementation in this project, the dataset for clustering is divided for the different process ranks. The centroids are for the points are determined separately in these ranks and when the centroids have to be updated, a reduction is performed to obtain the sum of coordinates of the points and number of points assigned to each centroid. This reduction is performed using MPI_Allreduce() to obtain the sum at all ranks. All the ranks compute the new centroids by dividing the global sum of coordinates by the global sum of the number of points for each centroid. If no points are assigned to a particular centroid in an iteration, then the centroid is reassigned to the origin (0, 0).

IV. Experimental Results

The implemented algorithm program is included in the file elkans_act1_ss3874.c. It is run with two datasets with dimensions 2 and 90 (Ionosphere dataset with 2 dimensions and Million Song Dataset with 90 dimensions). The parallelized version implemented using OpenMPI is run on the Monsoon cluster with the number of process ranks np = 1, 4, 8, 12, 16, 20, 24, 28, 32, 36, and 40 (using 1 or 2 dedicated nodes as required).

Though the algorithm is implemented to run till convergence, the version run for performance comparison has only 10 iterations, since it is hard compare different k values which converge at different number of iterations. For example, k = 2 converged at 17 iterations whereas k = 100 converged at 845 iterations for the ionosphere dataset. The convergence checking code is still included but is commented out.

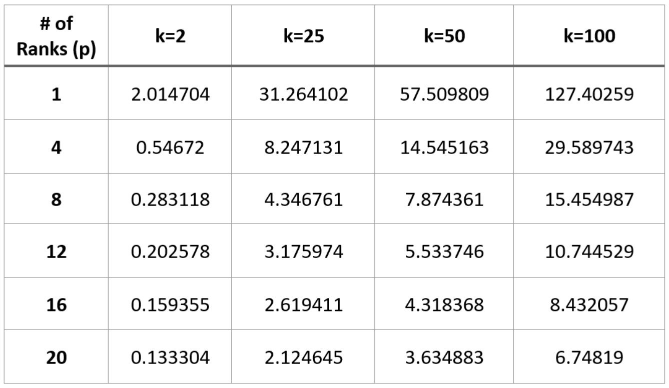
The Elkan's algorithm produces the same centroids and clusters at each iteration as the naive algorithm. It does not does not reduce the variation between the location of the centroid in each iteration, and requires the same number of iterations required to reach convergence as the naive algorithm. Thus it can be used in cases where it has to substitute the naive algorithm without causing changes in different iterations.

The tables 2, 3, and 4 show the time taken for 10 iterations in process ranks *np = 1, 4, 8, 12, 16, 20* run on one node, in total and the distance calculations, and updating the centroids respectively. Table 5 shows the performance metrics such as speedup and parallel efficiency for the same. Tables 6 to 9 summarize the same parameters for *np = 24, 28, 32, 36, 40* run on two nodes.

In the naive Lloyd's algorithm implementation summarized in Table 1, the higher the value of k, the lower was the speedup. This is probably because the number of distance calculations increases by a factor of

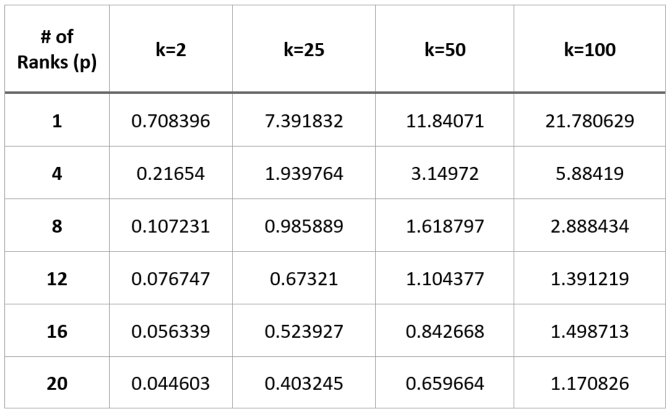
V. Conclusion

Thus the Elkan's algorithm is seen to take much lesser time than the naive algorithm due to less number of distance calculations.



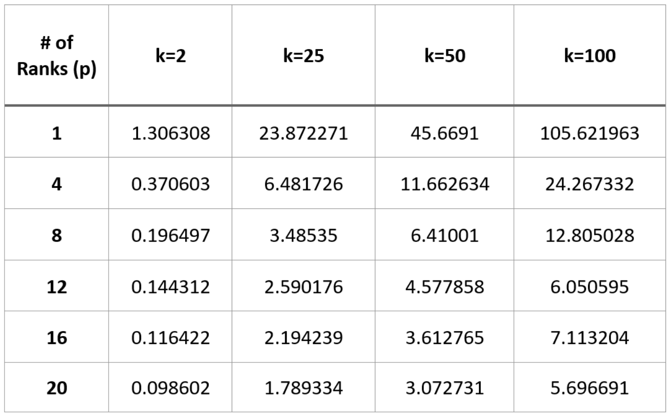
# of Ranks (p)	k=2	k=25	k=50	k=100
1	2.014704	31.264102	57.509809	127.40259
4	0.54672	8.247131	14.545163	29.589743
8	0.283118	4.346761	7.874361	15.454987
12	0.202578	3.175974	5.533746	10.744529
16	0.159355	2.619411	4.318368	8.432057
20	0.133304	2.124645	3.634883	6.74819

Fig. 2. Table 1: Total response time on one node



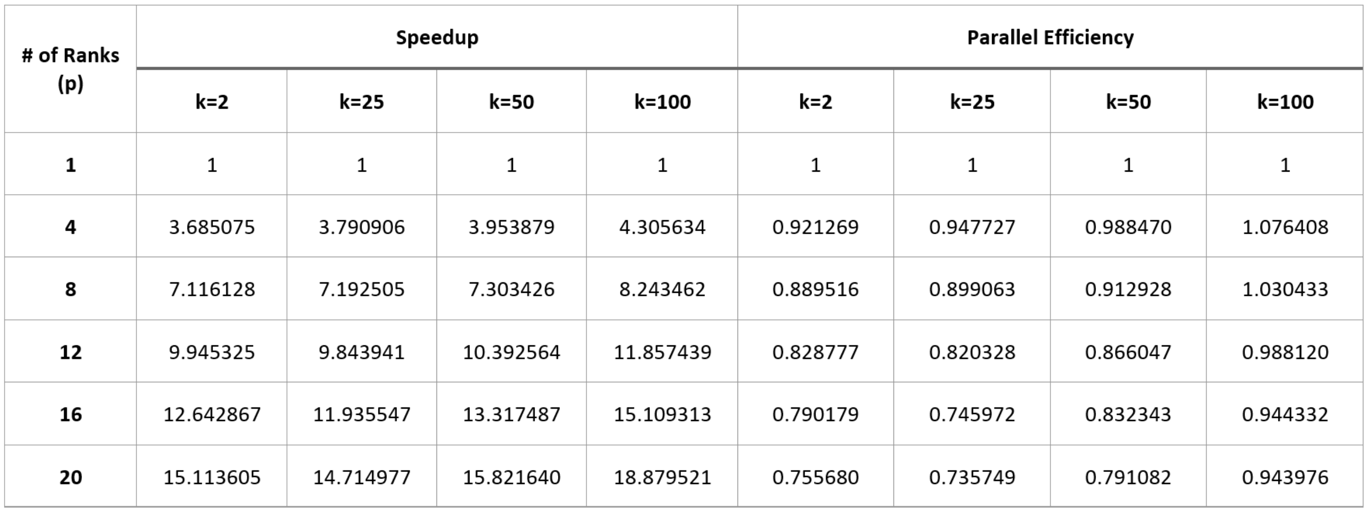
# of Ranks (p)	k=2	k=25	k=50	k=100
1	0.708396	7.391832	11.84071	21.780629
4	0.21654	1.939764	3.14972	5.88419
8	0.107231	0.985889	1.618797	2.888434
12	0.076747	0.67321	1.104377	1.391219
16	0.056339	0.523927	0.842668	1.498713
20	0.044603	0.403245	0.659664	1.170826

Fig. 3. Table 2: Time for distance calculations (for one node)



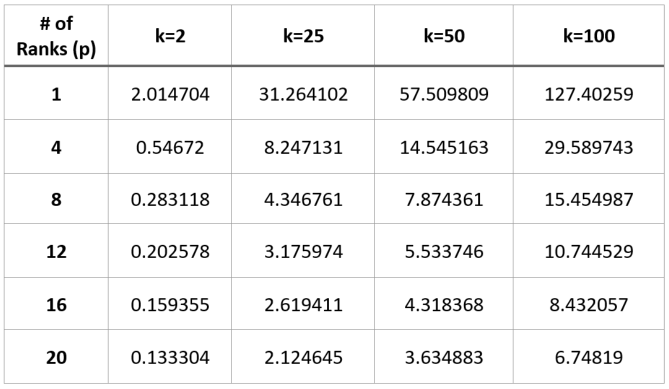
# of Ranks (p)	k=2	k=25	k=50	k=100
1	1.306308	23.872271	45.6691	105.621963
4	0.370603	6.481726	11.662634	24.267332
8	0.196497	3.48535	6.41001	12.805028
12	0.144312	2.590176	4.577858	6.050595
16	0.116422	2.194239	3.612765	7.113204
20	0.098602	1.789334	3.072731	5.696691

Fig. 4. Table 3: Time to update centroids (for one node)



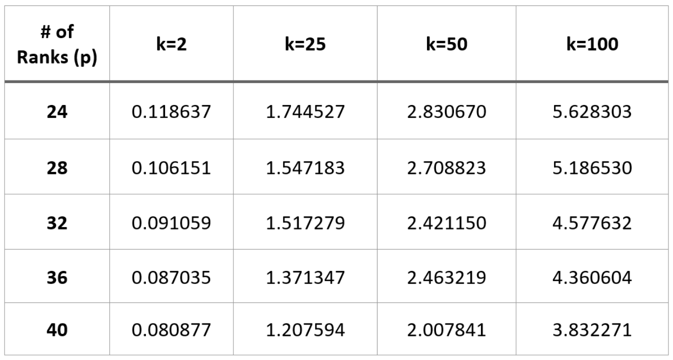
# of Ranks (p)	Speedup				Parallel Efficiency			
	k=2	k=25	k=50	k=100	k=2	k=25	k=50	k=100
1	1	1	1	1	1	1	1	1
4	3.685075	3.790906	3.953879	4.305634	0.921269	0.947727	0.988470	1.076408
8	7.116128	7.192505	7.303426	8.243462	0.889516	0.899063	0.912928	1.030433
12	9.945325	9.843941	10.392564	11.857439	0.828777	0.820328	0.866047	0.988120
16	12.642867	11.935547	13.317487	15.109313	0.790179	0.745972	0.832343	0.944332
20	15.113605	14.714977	15.821640	18.879521	0.755680	0.735749	0.791082	0.943976

Fig. 5. Table 4: Speedup and Parallel Efficiency from Table 1.



# of Ranks (p)	k=2	k=25	k=50	k=100
1	2.014704	31.264102	57.509809	127.40259
4	0.54672	8.247131	14.545163	29.589743
8	0.283118	4.346761	7.874361	15.454987
12	0.202578	3.175974	5.533746	10.744529
16	0.159355	2.619411	4.318368	8.432057
20	0.133304	2.124645	3.634883	6.74819

Fig. 6. Table 1: Total response time on one node



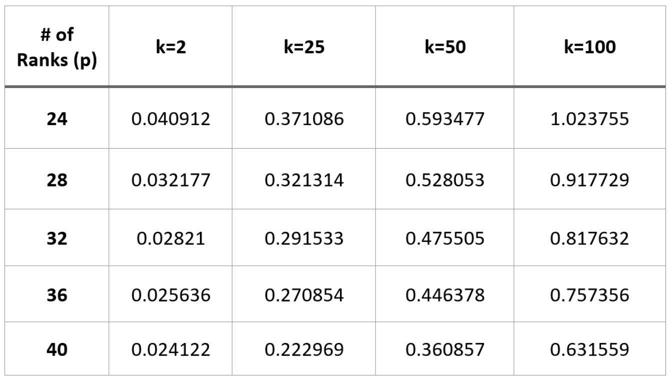
# of Ranks (p)	k=2	k=25	k=50	k=100
24	0.118637	1.744527	2.830670	5.628303
28	0.106151	1.547183	2.708823	5.186530
32	0.091059	1.517279	2.421150	4.577632
36	0.087035	1.371347	2.463219	4.360604
40	0.080877	1.207594	2.007841	3.832271

Fig. 7. Table 5: Total response time on one node

References

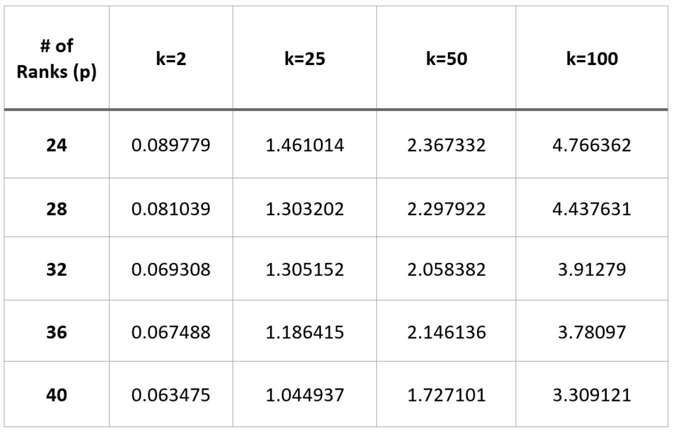
[1] Charles Elkan. 2003. Using the triangle inequality to accelerate k- means. In Proceedings of the 20th international conference on Machine Learning (ICML-03). 147–153.

[2] Wikipedia contributors. "K-Means Clustering." Wikipedia, Wikimedia Foundation, 28 Apr. 2022. https://en.wikipedia.org/wiki/K-means_clustering.



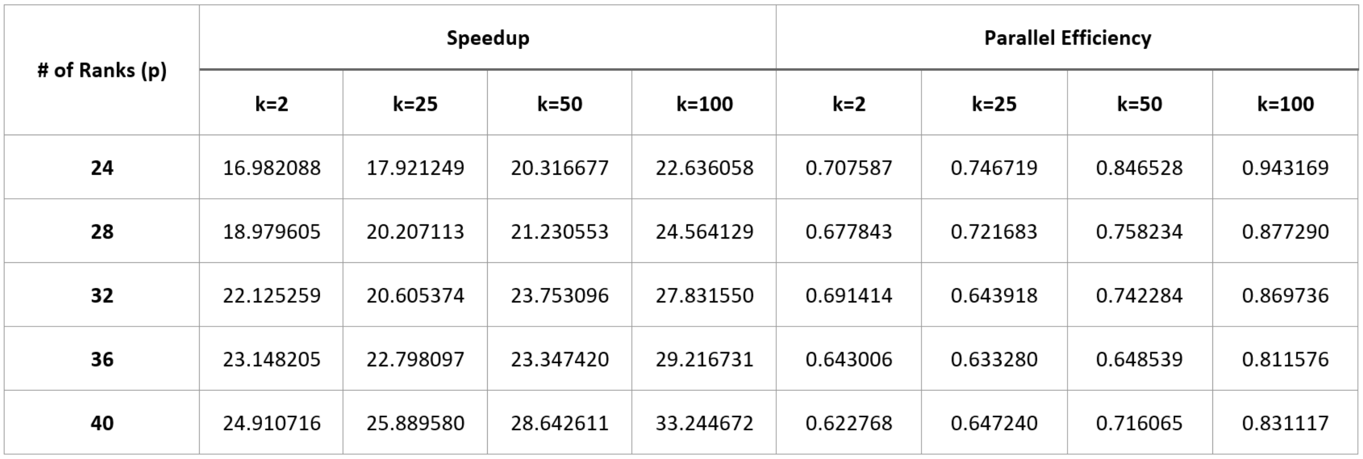
# of Ranks (p)	k=2	k=25	k=50	k=100
24	0.040912	0.371086	0.593477	1.023755
28	0.032177	0.321314	0.528053	0.917729
32	0.02821	0.291533	0.475505	0.817632
36	0.025636	0.270854	0.446378	0.757356
40	0.024122	0.222969	0.360857	0.631559

Fig. 8. Table 6: Time for distance calculations (for two nodes)



# of Ranks (p)	k=2	k=25	k=50	k=100
24	0.089779	1.461014	2.367332	4.766362
28	0.081039	1.303202	2.297922	4.437631
32	0.069308	1.305152	2.058382	3.91279
36	0.067488	1.186415	2.146136	3.78097
40	0.063475	1.044937	1.727101	3.309121

Fig. 9. Table 7: Time to update centroids (for two nodes)



# of Ranks (p)	Speedup				Parallel Efficiency			
	k=2	k=25	k=50	k=100	k=2	k=25	k=50	k=100
24	16.982088	17.921249	20.316677	22.636058	0.707587	0.746719	0.846528	0.943169
28	18.979605	20.207113	21.230553	24.564129	0.677843	0.721683	0.758234	0.877290
32	22.125259	20.605374	23.753096	27.831550	0.691414	0.643918	0.742284	0.869736
36	23.148205	22.798097	23.347420	29.216731	0.643006	0.633280	0.648539	0.811576
40	24.910716	25.889580	28.642611	33.244672	0.622768	0.647240	0.716065	0.831117

Fig. 10. Table 8: Speedup and Parallel Efficiency from Table 5.

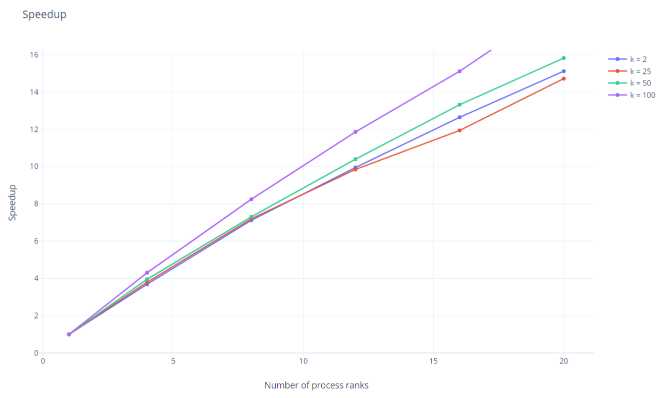


Fig. 11. Speedup plot

[3] M. Gowanlock, "K-means," Pedagogic Modules, 08-Jan-2020. [Online]. Available: https://jan.ucc.nau.edu/mg2745/pedagogic_modules/courses /hpcdataintensive/kmeans_0/. [Accessed: 06-May-2022].