Name: Shriya Sridhar

NAU Student ID: 6071166

ss3874@nau.edu

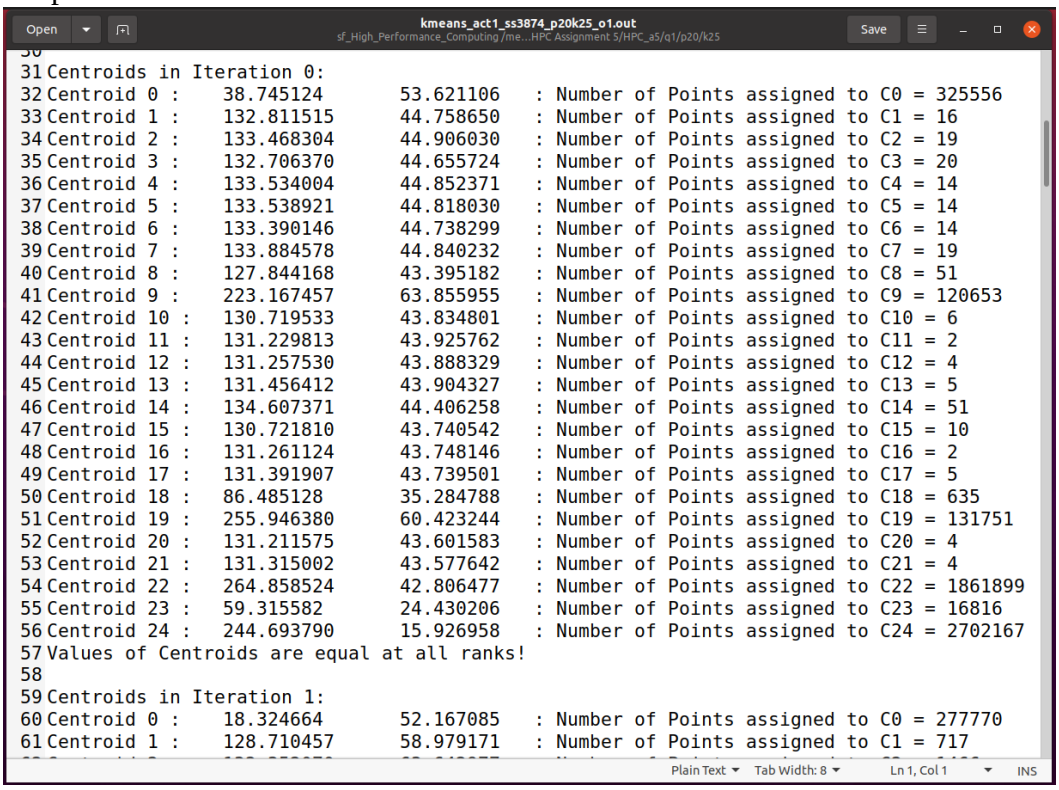
CS 599 –High Performance Computing

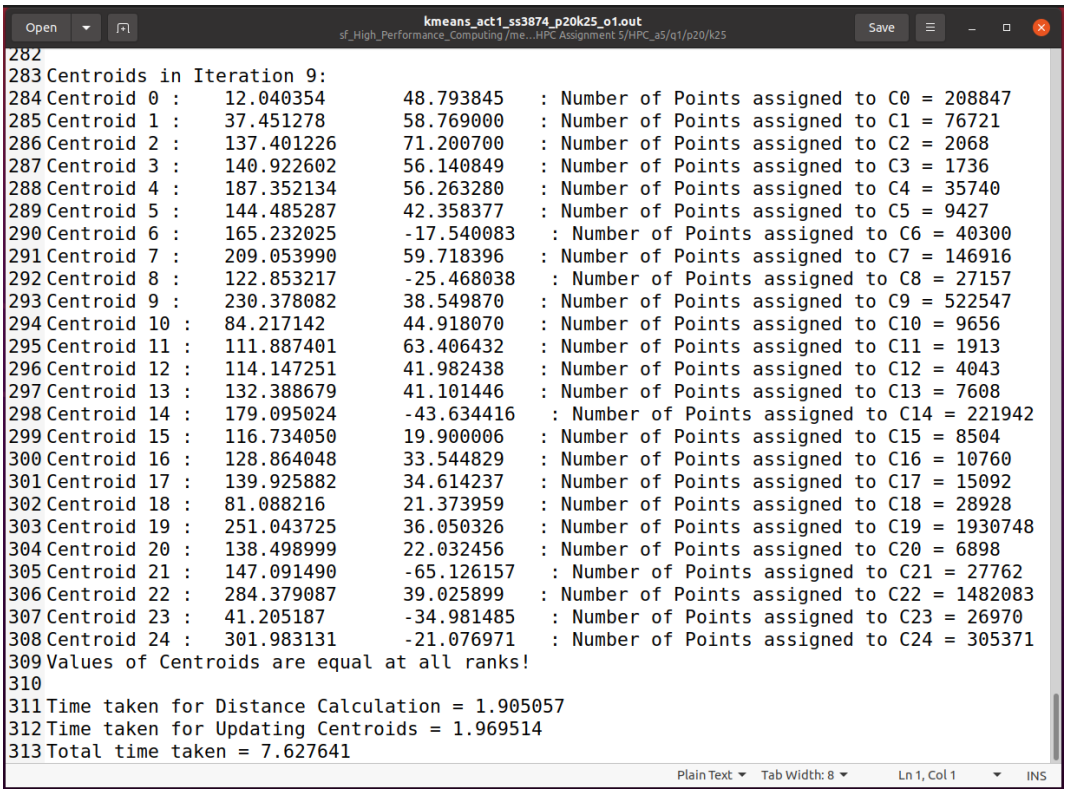
Assignment 5: K-Means

Programming Activity #1

- The program aims to determine the K-means cluster centroids for the given dataset.
- This involves two main steps:
  - To assign a centroid to each point based on which is the closest.
  - Determine a new centroid based on the points that have been assigned to it in the previous step.
- These 2 steps are looped over 10 iterations, and the clusters become progressively more accurate after each iteration.
- The distance calculations and updating the centroids are performed in loops, but the time measured for these steps are not easily measured around the loop, since there is a validation step for each iteration within the loop.
- Therefore, the time is iteratively measured in each loop and added to the time measurement variable.
- Since the distance calculation and assigning and updating of centroids are intermingled in their loops, they are measured around their particular lines. This leads to the total time being considerably larger than the sum of the distance time and centroid updating time.
- However, separating out these loops for easier time measurement would necessitate a distance matrix to store the distances from each point to each centroid, which would otherwise not be needed.
- Validation is done by using the plots attached, and also by determining that the centroids are the same at all ranks. This is done by finding the minimum and maximum of the centroids, and if both are the same, then the centroid value has to be the same on all ranks.
- From the output screenshots of the first and last iteration (for k = 25), we can see that the points are more well distributed for the 10^{th} iteration compared to the first one.
- There are 4 plots attached as .png files corresponding to k = 2, 25, 50 and 100 for validation.

Output Screenshot:





Q1: What component of the algorithm is straightforward to parallelize?

The distance calculations are easily divided among the ranks by simply dividing the points in the dataset computed by each rank. Thus, it is easy to parallelize.

Q2: When parallelizing the algorithm, what component of the algorithm requires communication between process ranks?

The calculation of new centroids requires the communication of the number of points assigned to each centroid and the sum of the points' coordinates for each between all the ranks. This requires more care while parallelizing.

# of Ranks (pp)	**k=2**	**k=25**	**k=50**	**k=100**	**Job Script Name (*.sh)**
1	15.09829433	149.585818	294.943403	586.939971	HPC_a5/q1/p1/k2/kmeans_act1_ss3874_p1k2_t1.sh
4	3.772459	38.928629	73.85086	146.93457	HPC_a5/q1/p4/k2/kmeans_act1_ss3874_p4k2_t1.sh
8	1.897085	18.899914	37.00603	73.966801	HPC_a5/q1/p8/k2/kmeans_act1_ss3874_p8k2_t1.sh
12	1.263627	12.657739	25.174069	49.912126	HPC_a5/q1/p12/k2/kmeans_act1_ss3874_p12k2_t1.sh
16	0.947663	9.58264	18.832987	37.02072	HPC_a5/q1/p16/k2/kmeans_act1_ss3874_p16k2_t1.sh
20	0.755897	7.627641	15.060642	30.156116	HPC_a5/q1/p20/k2/kmeans_act1_ss3874_p20k2_t1.sh

Table 2: Maximum cumulative time spent performing distance calculations.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100	**Job Script Name (*.sh)**
1	3.299902667	37.737731	74.410349	148.768322	HPC_a5/q1/p1/k2/kmeans_act1_ss3874_p1k2_t1.sh
4	0.819563	9.655096	18.720494	37.35036	HPC_a5/q1/p4/k2/kmeans_act1_ss3874_p4k2_t1.sh
8	0.415455	4.759574	9.369248	18.653287	HPC_a5/q1/p8/k2/kmeans_act1_ss3874_p8k2_t1.sh
12	0.277207	3.161319	6.677006	13.364303	HPC_a5/q1/p12/k2/kmeans_act1_ss3874_p12k2_t1.sh
16	0.209087	2.512607	5.021967	9.379296	HPC_a5/q1/p16/k2/kmeans_act1_ss3874_p16k2_t1.sh
20	0.163946	1.905057	4.003464	8.131203	HPC_a5/q1/p20/k2/kmeans_act1_ss3874_p20k2_t1.sh

Table 3: Maximum cumulative time spent updating means, including performing synchronization between ranks.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100	**Job Script Name (*.sh)**
1	4.316099667	36.4065	70.926014	139.987481	HPC_a5/q1/p1/k2/kmeans_act1_ss3874_p1k2_t1.sh
4	1.108379	10.449221	17.828824	35.224849	HPC_a5/q1/p4/k2/kmeans_act1_ss3874_p4k2_t1.sh
8	0.571532	4.757694	8.952169	18.089332	HPC_a5/q1/p8/k2/kmeans_act1_ss3874_p8k2_t1.sh
12	0.382214	3.232702	6.489334	12.692237	HPC_a5/q1/p12/k2/kmeans_act1_ss3874_p12k2_t1.sh
16	0.288081	2.510274	4.813258	9.098083	HPC_a5/q1/p16/k2/kmeans_act1_ss3874_p16k2_t1.sh
20	0.230414	1.969514	3.843603	7.811754	HPC_a5/q1/p20/k2/kmeans_act1_ss3874_p20k2_t1.sh

Table 4: Speedup computed using the data in Table 1.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100
1	1	1	1	1
4	4.002242127	3.842565789	3.993770729	3.994566908
8	7.958680994	7.914629559	7.970144406	7.935181231
12	11.94837902	11.81773601	11.71615931	11.75946645
16	15.93213445	15.61008428	15.66099966	15.85436402
20	19.97401013	19.61101971	19.58372047	19.46338086

Q3: Describe how you implemented your algorithm.

- The algorithm involves two main steps:
    - To assign a centroid to each point based on which is the closest.

    - Determine a new centroid based on the points that have been assigned to it in the previous step.
- These 2 steps are looped over 10 iterations, and the clusters become progressively more accurate after each iteration.
- The initial value of the centroids is assigned by using the first few data points in the data set.
- For assigning a centroid to each point, the distance between each point and each centroid is calculated.
    - If it is the first distance calculation, the centroid is temporarily assigned as the closest centroid, and the distance is stored as minimumDistance.
    - For subsequent distance calculations, the distance is compared with the value stored in minimumDistance. If the newly computed distance is lesser, the minimumDistance is updated, and that centroid is assigned as the closest centroid.
- After determining the closest centroid, we increment the number of points assigned to the centroid, and add the coordinates of the new point to localSumOfCentroidPoints. This will help to calculate the new centroid in the next step.
- The global sum of the coordinates and the number of points assigned to each centroid is computed using MPI_Allreduce(). By using MPI_Allreduce() instead of MPI_Reduce(), all the ranks get to know the global sums (instead of only the root rank).
- 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.
- Thus, this program implements the second idea mentioned in the assignment for synchronization between ranks.
- If no points are assigned to a particular centroid in an iteration, then the centroid is reassigned to the origin (0, 0).

Q4: Explain the (potential) bottlenecks in your algorithm.

The main bottleneck is possibly the communication involved in MPI_Allreduce(). While it is simpler to implement, there is communication from each node to every other node, leading to p^2 messages for one value being shared. We need to share the sum of coordinates for each dimension, and the number of points assigned to each centroid, leading to 3 values for each centroid (since this dataset is 2-dimensional). Perhaps using MPI_Reduce() followed by MPIBcast() from rank 0 would reduce the communication overhead.

Q5: How does the algorithm scale when k=2**? Explain.**

The speedup for k = 2 is nearly ideal for all values up to p = 20, which has a speedup of 19.974 and has a super linear speedup for p = 4 (Speedup = 4.002242). Thus, it scales well for 2 K-means. This is probably because there is less number of values that need to be communicated between ranks during synchronization.

Q6: How does the algorithm scale when k=100**? Explain.**

There is a lower speedup for k = 100 compared to lower values of k, but it still good. The speedup for p = 20 is 19.463380 when k = 100 compared to 19.974 for k = 2. This is because there are more number of values that need to be communicated between ranks during synchronization.

K-Means: Multiple Nodes

Table 5: Total response time.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100	**Job Script Name (*.sh)**
24	0.650113	6.466928	12.762639	26.086781	HPC_a5/q2/p24/k2/kmeans_act1_ss3874_p24k2_t1.sh
28	0.55268	5.524046	10.923365	21.698476	HPC_a5/q2/p28/k2/kmeans_act1_ss3874_p28k2_t1.sh
32	0.488758	4.92884	16.870281	33.61924	HPC_a5/q2/p32/k2/kmeans_act1_ss3874_p32k2_t1.sh
36	0.447281	4.354811	8.48711	31.549816	HPC_a5/q2/p36/k2/kmeans_act1_ss3874_p36k2_t1.sh
40	0.396372	3.887957	7.77329	29.784782	HPC_a5/q2/p40/k2/kmeans_act1_ss3874_p40k2_t1.sh

Table 6: Maximum cumulative time spent performing distance calculations.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100	**Job Script Name (*.sh)**
24	0.136173	1.584126	3.341097	6.237762	HPC_a5/q2/p24/k2/kmeans_act1_ss3874_p24k2_t1.sh
28	0.119723	1.366099	2.833013	5.609581	HPC_a5/q2/p28/k2/kmeans_act1_ss3874_p28k2_t1.sh
32	0.102353	1.193566	2.447631	5.027275	HPC_a5/q2/p32/k2/kmeans_act1_ss3874_p32k2_t1.sh
36	0.091768	1.062233	2.089484	4.35145	HPC_a5/q2/p36/k2/kmeans_act1_ss3874_p36k2_t1.sh
40	0.082099	0.946337	2.008203	3.789953	HPC_a5/q2/p40/k2/kmeans_act1_ss3874_p40k2_t1.sh

Table 7: Maximum cumulative time spent updating means, including performing synchronization between ranks.

# of Ranks (pp**)**	k=2	k=25	k=50	k=100	**Job Script Name (*.sh)**
24	0.207961	1.75128	3.410329	7.462321	HPC_a5/q2/p24/k2/kmeans_act1_ss3874_p24k2_t1.sh
28	0.177032	1.482754	2.905844	5.737492	HPC_a5/q2/p28/k2/kmeans_act1_ss3874_p28k2_t1.sh
32	0.158412	1.391129	9.856454	19.657547	HPC_a5/q2/p32/k2/kmeans_act1_ss3874_p32k2_t1.sh
36	0.152233	1.210027	2.251288	19.134822	HPC_a5/q2/p36/k2/kmeans_act1_ss3874_p36k2_t1.sh
40	0.132698	1.058979	2.164749	18.608413	HPC_a5/q2/p40/k2/kmeans_act1_ss3874_p40k2_t1.sh

Table 8: Speedup computed using the data in Table 5 (When compared with time for p = 1).

# of Ranks (pp**)**	k=2	k=25	k=50	k=100
24	23.22410771	23.13089275	23.10990721	22.49951694
28	27.31832947	27.07903193	27.00114873	27.04982465
32	30.89114517	30.34909188	17.48301661	17.45845447
36	33.75572478	34.34955455	34.75192415	18.60359411
40	38.09122323	38.47414413	37.94318789	19.70603548

- **Q7: On two nodes, how does the algorithm scale when** K=2**? Compare with the single node results.**

  The speedup for k = 2 is not as high as the single node results, but that is also contributed by the increase in p. For p = 24, the speedup is 23.224 and the speedup for p = 40 is

38.091, which is lesser than ideal. Thus, it scales not well enough for k = 2, but considering the high value of p, it is good enough. There is more communication between ranks during synchronization compared to single node.

- **Q8: On two nodes, how does the algorithm scale when** K=100**? Compare with the single node results.**

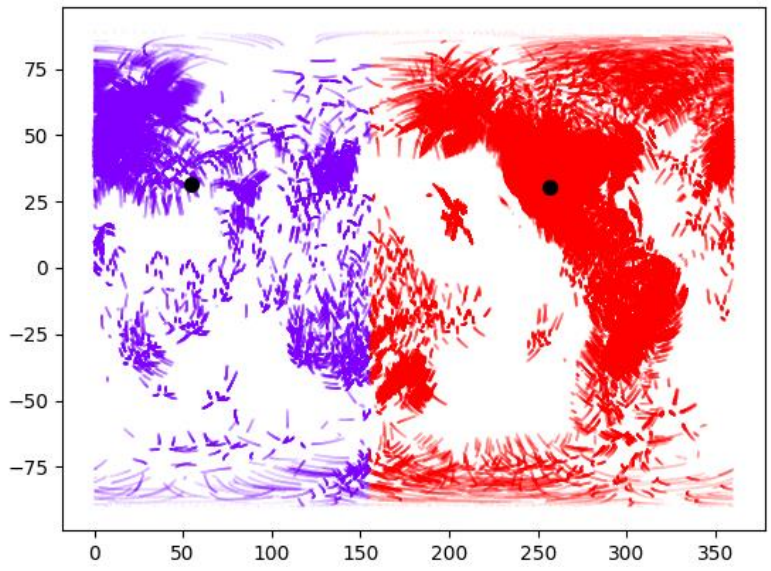
  The speedup for k = 100 is very poor. Compared to the single node results, we get a speedup of 22.499 for p = 24, and the speedup for p = 40 is just 19.706, which is almost the same as the 19.463 speedup for 20 ranks on a single node. Again, there is a lot of communication required between ranks during synchronization.

- **Q9: Under what conditions do you expect the k-means algorithm to perform well on multiple nodes (e.g., two or more)? When preparing your response, consider the following factors: the number of centroids, the size of the dataset, the data dimensionality, and the number of iterations.**

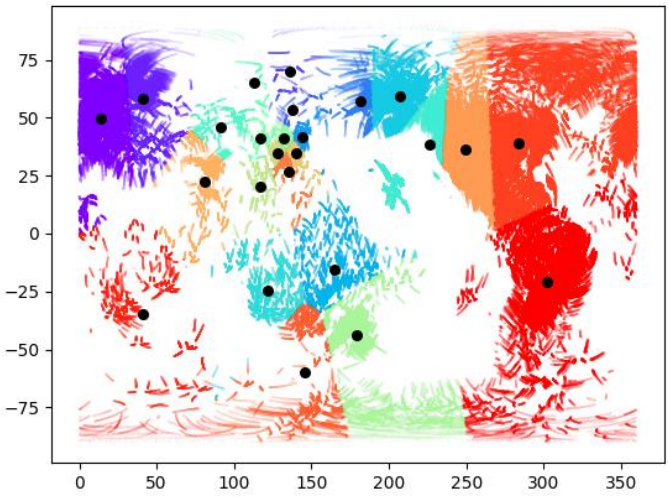
  Lesser number of centroids would require less communication, thus would improve the performance on multiple nodes. The large dataset would not be much of an obstacle for performance, since it is recreated at all ranks, but it increases the memory requirements. The data dimensionality would also tremendously increase the time required. The number of iterations would increase the time required for the program, but it would be high for a single rank too, thus the speedup would still be high.

Plots of the clusters and centroid at the (*l*-1)th iteration:

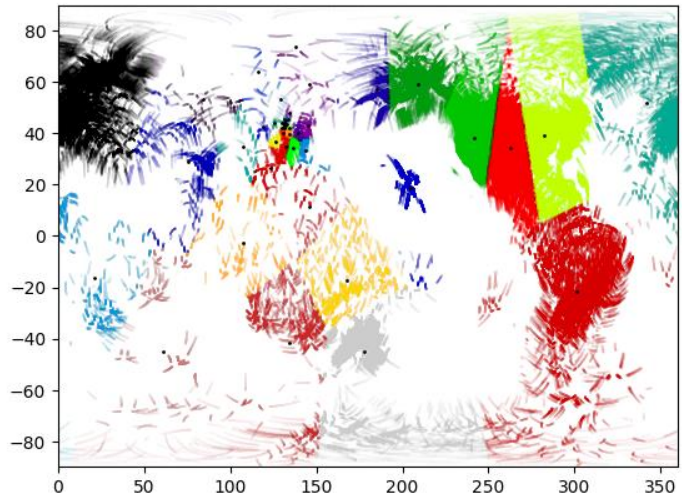
k = 2



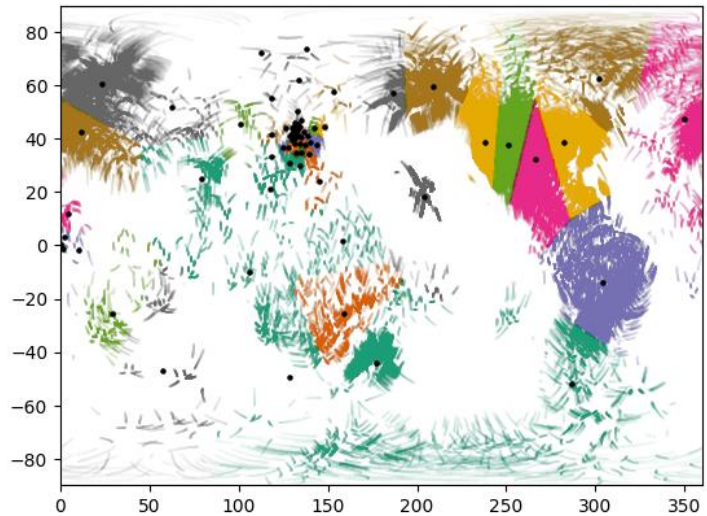
k = 25


75
50
25
0
−25
−50
−75
0
50
100
150
200
250
300
350

k = 50


80
60
40
20
0
−20
−40
−60
−80
0
50
100
150
200
250
300
350

k = 100


80
60
40
20
0
−20
−40
−60
−80
0
50
100
150
200
250
300
350