**Activity 1**

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

**Ans:** The distance calculation between cluster centroids and data points is straightforward to parallelize when they’re divided among the appropriate number of process ranks.

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

**Ans:** The process ranks need to communicate with each other regarding their local cluster sizes and local clusters’ weighted means to be able to calculate global clusters centroids.

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 1 | 6.522189 | 10.537508 | 14.707599 | 23.373634 | CS599\_HW5\_activity1\_p1k2.sh  CS599\_HW5\_activity1\_p1k25.sh  CS599\_HW5\_activity1\_p1k50.sh  CS599\_HW5\_activity1\_p1k100.sh |
| 4 | 1.622921 | 2.596438 | 3.687901 | 5.861329 | CS599\_HW5\_activity1\_p4k2.sh  CS599\_HW5\_activity1\_p4k25.sh  CS599\_HW5\_activity1\_p4k50.sh  CS599\_HW5\_activity1\_p4k100.sh |
| 8 | 0.814202 | 1.307275 | 1.842972 | 2.927838 | CS599\_HW5\_activity1\_p8k2.sh  CS599\_HW5\_activity1\_p8k25.sh  CS599\_HW5\_activity1\_p8k50.sh  CS599\_HW5\_activity1\_p8k100.sh |
| 12 | 0.549337 | 0.868869 | 1.229402 | 1.952834 | CS599\_HW5\_activity1\_p12k2.sh  CS599\_HW5\_activity1\_p12k25.sh  CS599\_HW5\_activity1\_p12k50.sh  CS599\_HW5\_activity1\_p12k100.sh |
| 16 | 0.407375 | 0.652550 | 0.923828 | 1.471823 | CS599\_HW5\_activity1\_p16k2.sh  CS599\_HW5\_activity1\_p16k25.sh  CS599\_HW5\_activity1\_p16k50.sh  CS599\_HW5\_activity1\_p16k100.sh |
| 20 | 0.327971 | 0.523459 | 0.742087 | 1.175666 | CS599\_HW5\_activity1\_p20k2.sh  CS599\_HW5\_activity1\_p20k25.sh  CS599\_HW5\_activity1\_p20k50.sh  CS599\_HW5\_activity1\_p20k100.sh |

**Table-1: Total response times**

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 1 | 2.018701 | 6.023714 | 10.201934 | 18.885627 | CS599\_HW5\_activity1\_p1k2.sh  CS599\_HW5\_activity1\_p1k25.sh  CS599\_HW5\_activity1\_p1k50.sh  CS599\_HW5\_activity1\_p1k100.sh |
| 4 | 0.493826 | 1.465423 | 2.546278 | 4.720123 | CS599\_HW5\_activity1\_p4k2.sh  CS599\_HW5\_activity1\_p4k25.sh  CS599\_HW5\_activity1\_p4k50.sh  CS599\_HW5\_activity1\_p4k100.sh |
| 8 | 0.247875 | 0.742422 | 1.277662 | 2.354532 | CS599\_HW5\_activity1\_p8k2.sh  CS599\_HW5\_activity1\_p8k25.sh  CS599\_HW5\_activity1\_p8k50.sh  CS599\_HW5\_activity1\_p8k100.sh |
| 12 | 0.170246 | 0.490001 | 0.850784 | 1.576478 | CS599\_HW5\_activity1\_p12k2.sh  CS599\_HW5\_activity1\_p12k25.sh  CS599\_HW5\_activity1\_p12k50.sh  CS599\_HW5\_activity1\_p12k100.sh |
| 16 | 0.123229 | 0.368063 | 0.639248 | 1.181044 | CS599\_HW5\_activity1\_p16k2.sh  CS599\_HW5\_activity1\_p16k25.sh  CS599\_HW5\_activity1\_p16k50.sh  CS599\_HW5\_activity1\_p16k100.sh |
| 20 | 0.099982 | 0.295110 | 0.514728 | 0.942947 | CS599\_HW5\_activity1\_p20k2.sh  CS599\_HW5\_activity1\_p20k25.sh  CS599\_HW5\_activity1\_p20k50.sh  CS599\_HW5\_activity1\_p20k100.sh |

**Table-2: Maximum cumulative time spent performing distance calculations.**

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1.521390 | 1.538519 | 1.527965 | 1.509351 | CS599\_HW5\_activity1\_p1k2.sh  CS599\_HW5\_activity1\_p1k25.sh  CS599\_HW5\_activity1\_p1k50.sh  CS599\_HW5\_activity1\_p1k100.sh |
| 4 | 0.417228 | 0.415454 | 0.425823 | 0.426196 | CS599\_HW5\_activity1\_p4k2.sh  CS599\_HW5\_activity1\_p4k25.sh  CS599\_HW5\_activity1\_p4k50.sh  CS599\_HW5\_activity1\_p4k100.sh |
| 8 | 0.216544 | 0.217269 | 0.210665 | 0.215328 | CS599\_HW5\_activity1\_p8k2.sh  CS599\_HW5\_activity1\_p8k25.sh  CS599\_HW5\_activity1\_p8k50.sh  CS599\_HW5\_activity1\_p8k100.sh |
| 12 | 0.149306 | 0.142322 | 0.142508 | 0.143646 | CS599\_HW5\_activity1\_p12k2.sh  CS599\_HW5\_activity1\_p12k25.sh  CS599\_HW5\_activity1\_p12k50.sh  CS599\_HW5\_activity1\_p12k100.sh |
| 16 | 0.105319 | 0.105529 | 0.107018 | 0.114535 | CS599\_HW5\_activity1\_p16k2.sh  CS599\_HW5\_activity1\_p16k25.sh  CS599\_HW5\_activity1\_p16k50.sh  CS599\_HW5\_activity1\_p16k100.sh |
| 20 | 0.086086 | 0.084968 | 0.088212 | 0.088417 | CS599\_HW5\_activity1\_p20k2.sh  CS599\_HW5\_activity1\_p20k25.sh  CS599\_HW5\_activity1\_p20k50.sh  CS599\_HW5\_activity1\_p20k100.sh |

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

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 |
| 4 | 4.02 | 4.06 | 3.99 | 3.99 |
| 8 | 8.01 | 8.06 | 7.98 | 7.98 |
| 12 | 11.87 | 12.13 | 11.96 | 11.97 |
| 16 | 16.01 | 16.15 | 15.92 | 15.88 |
| 20 | 19.89 | 20.13 | 19.82 | 19.88 |

**Table-4: Speedup computed using the data in Table 1**

**Q3: Describe how you implemented your algorithm.**

**Ans:** In my algorithm, process 0 allocates the number of data rows to be handled to all the other process ranks using MPI\_Bcast(). All processes (including process 0) then take the first KMEANS (number of clusters to be formed) data points from the original dataset to be the initial cluster centroids during the first iteration.

Using the initial cluster centroids, each process rank assigns the data points within their respective allocated rows in the dataset to their closest cluster. Each process then communicates with the other processes about their local cluster size using MPI\_Isend() and receives other remote processes’ local cluster sizes using MPI\_Recv(). Using this information, each process rank then calculates their weighted average or the local cluster centroids. Each process then again communicates about their local cluster centroids/ weighted average of points in the local clusters with other processes using MPI\_Isend() and MPI\_Recv(). After this, combining the available and received information, each cluster then constructs identical global values for the cluster centroids to be used in the next iteration of the K-means algorithm.

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

**Ans:** The communication overhead associated with the sharing of local cluster sizes and local weighted average cluster points can be a potential bottleneck with an increasing number of clusters to be formed. Also, a process cannot move forward to the next iteration until it has received the required information from the other processes to calculate and update the global cluster centroids in the current iteration. If there is any load imbalance with one of the processes performing a significantly higher amount of computation, it might become a performance bottleneck.

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

**Ans:** The algorithm scales significantly well when k = 2. In all the cases, the parallel efficiency of the algorithm remains 100% or very close to 100% (~99%).

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

**Ans:** The algorithm also scales exceptionally well when k = 100. The parallel efficiency ranges from 99.75% to 99.4% even when the number of clusters to be formed is quite high compared to the k = 2 scenario.

**Activity 2**

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 24 | 0.289058 | 1.515194 | 0.63904 | 1.011423 | CS599\_HW5\_activity2\_p24k2.sh  CS599\_HW5\_activity2\_p24k25.sh  CS599\_HW5\_activity2\_p24k50.sh  CS599\_HW5\_activity2\_p24k100.sh |
| 28 | 0.254992 | 0.423546 | 0.563013 | 0.886985 | CS599\_HW5\_activity2\_p28k2.sh  CS599\_HW5\_activity2\_p28k25.sh  CS599\_HW5\_activity2\_p28k50.sh  CS599\_HW5\_activity2\_p28k100.sh |
| 32 | 0.238092 | 0.578787 | 0.517691 | 0.802734 | CS599\_HW5\_activity2\_p32k2.sh  CS599\_HW5\_activity2\_p32k25.sh  CS599\_HW5\_activity2\_p32k50.sh  CS599\_HW5\_activity2\_p32k100.sh |
| 36 | 0.438168 | 0.785979 | 0.479123 | 0.742446 | CS599\_HW5\_activity2\_p36k2.sh  CS599\_HW5\_activity2\_p36k25.sh  CS599\_HW5\_activity2\_p36k50.sh  CS599\_HW5\_activity2\_p36k100.sh |
| 40 | 0.423708 | 2.387303 | 0.670521 | 1.706121 | CS599\_HW5\_activity2\_p40k2.sh  CS599\_HW5\_activity2\_p40k25.sh  CS599\_HW5\_activity2\_p40k50.sh  CS599\_HW5\_activity2\_p40k100.sh |

**Table-5: Total response times**

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 24 | 0.084254 | 0.249411 | 0.428906 | 0.787664 | CS599\_HW5\_activity2\_p24k2.sh  CS599\_HW5\_activity2\_p24k25.sh  CS599\_HW5\_activity2\_p24k50.sh  CS599\_HW5\_activity2\_p24k100.sh |
| 28 | 0.084254 | 0.218067 | 0.364832 | 0.675401 | CS599\_HW5\_activity2\_p28k2.sh  CS599\_HW5\_activity2\_p28k25.sh  CS599\_HW5\_activity2\_p28k50.sh  CS599\_HW5\_activity2\_p28k100.sh |
| 32 | 0.063019 | 0.184419 | 0.321225 | 0.591434 | CS599\_HW5\_activity2\_p32k2.sh  CS599\_HW5\_activity2\_p32k25.sh  CS599\_HW5\_activity2\_p32k50.sh  CS599\_HW5\_activity2\_p32k100.sh |
| 36 | 0.056737 | 0.164531 | 0.289690 | 0.525170 | CS599\_HW5\_activity2\_p36k2.sh  CS599\_HW5\_activity2\_p36k25.sh  CS599\_HW5\_activity2\_p36k50.sh  CS599\_HW5\_activity2\_p36k100.sh |
| 40 | 0.050852 | 0.149226 | 0.256178 | 0.473667 | CS599\_HW5\_activity2\_p40k2.sh  CS599\_HW5\_activity2\_p40k25.sh  CS599\_HW5\_activity2\_p40k50.sh  CS599\_HW5\_activity2\_p40k100.sh |

**Table-6: Maximum cumulative time spent performing distance calculations**

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** | **Job script** |
| --- | --- | --- | --- | --- | --- |
| 24 | 0.082313 | 1.141821 | 0.083913 | 0.083794 | CS599\_HW5\_activity2\_p24k2.sh  CS599\_HW5\_activity2\_p24k25.sh  CS599\_HW5\_activity2\_p24k50.sh  CS599\_HW5\_activity2\_p24k100.sh |
| 28 | 0.082313 | 0.104232 | 0.080896 | 0.084506 | CS599\_HW5\_activity2\_p28k2.sh  CS599\_HW5\_activity2\_p28k25.sh  CS599\_HW5\_activity2\_p28k50.sh  CS599\_HW5\_activity2\_p28k100.sh |
| 32 | 0.081363 | 0.083135 | 0.089798 | 0.085447 | CS599\_HW5\_activity2\_p32k2.sh  CS599\_HW5\_activity2\_p32k25.sh  CS599\_HW5\_activity2\_p32k50.sh  CS599\_HW5\_activity2\_p32k100.sh |
| 36 | 0.085000 | 0.310068 | 0.092094 | 0.088149 | CS599\_HW5\_activity2\_p36k2.sh  CS599\_HW5\_activity2\_p36k25.sh  CS599\_HW5\_activity2\_p36k50.sh  CS599\_HW5\_activity2\_p36k100.sh |
| 40 | 0.088194 | 2.158295 | 0.325632 | 1.106614 | CS599\_HW5\_activity2\_p40k2.sh  CS599\_HW5\_activity2\_p40k25.sh  CS599\_HW5\_activity2\_p40k50.sh  CS599\_HW5\_activity2\_p40k100.sh |

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

| **# process ranks (p)** | **k = 2** | **k = 25** | **k = 50** | **k = 100** |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 |
| 24 | 23.25 | 6.95 | 23.02 | 23.11 |
| 28 | 25.58 | 24.88 | 26.12 | 26.35 |
| 32 | 27.39 | 18.21 | 28.41 | 29.12 |
| 36 | 14.89 | 13.41 | 30.70 | 31.48 |
| 40 | 15.39 | 4.42 | 21.93 | 13.7 |

**Table-8: Speedup computed using the data in Table 5**

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

**Ans:** On two nodes, the algorithm still maintains good parallel efficiency for a smaller number of process ranks (85.59% to 96.86%). Compared to single node results, these parallel efficiencies are lower. For a higher number of processes ranks (p = 36, 40), we see degradation in parallel efficiencies for two nodes compared to the single node results. However, this can possibly happen either the communication overhead associated with the algorithm or inconsistent time trials. I think it might be an inconsistent time trial as for k = 50 and 100 on two nodes, we see much improvement in speedups and parallel efficiencies following closely single node results.

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

**Ans:** The algorithm maintains good parallel efficiencies mostly in the range of 87% to 96.29%. However, we observe a parallel efficiency of only 34% for p = 40. In my opinion, this can be due to inconsistencies in time trials. Compared to single node results, the scalability here follows closely that of the single node.

**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.**

**Ans:** In my opinion, any increase in the number of centroids (number of clusters) and data dimensionality will negatively influence the performance of the algorithm by causing increasing communication overhead (more data to communicate taking longer times in cross node communication). Except for these scenarios, I think the algorithm will perform well on multiple nodes. That is, with large dataset with a small number of centroids and lower data dimensionality, the algorithm is expected to perform better.

**Validation plots of cluster centroids:**

