Parallel Algorithms
K-means Clustering

Instructor: Dr. Russ Miller

Name: Mrunal Inge

UB Person Number: 50337040

University at Buffalo The State University of New York

OVERVIEW:

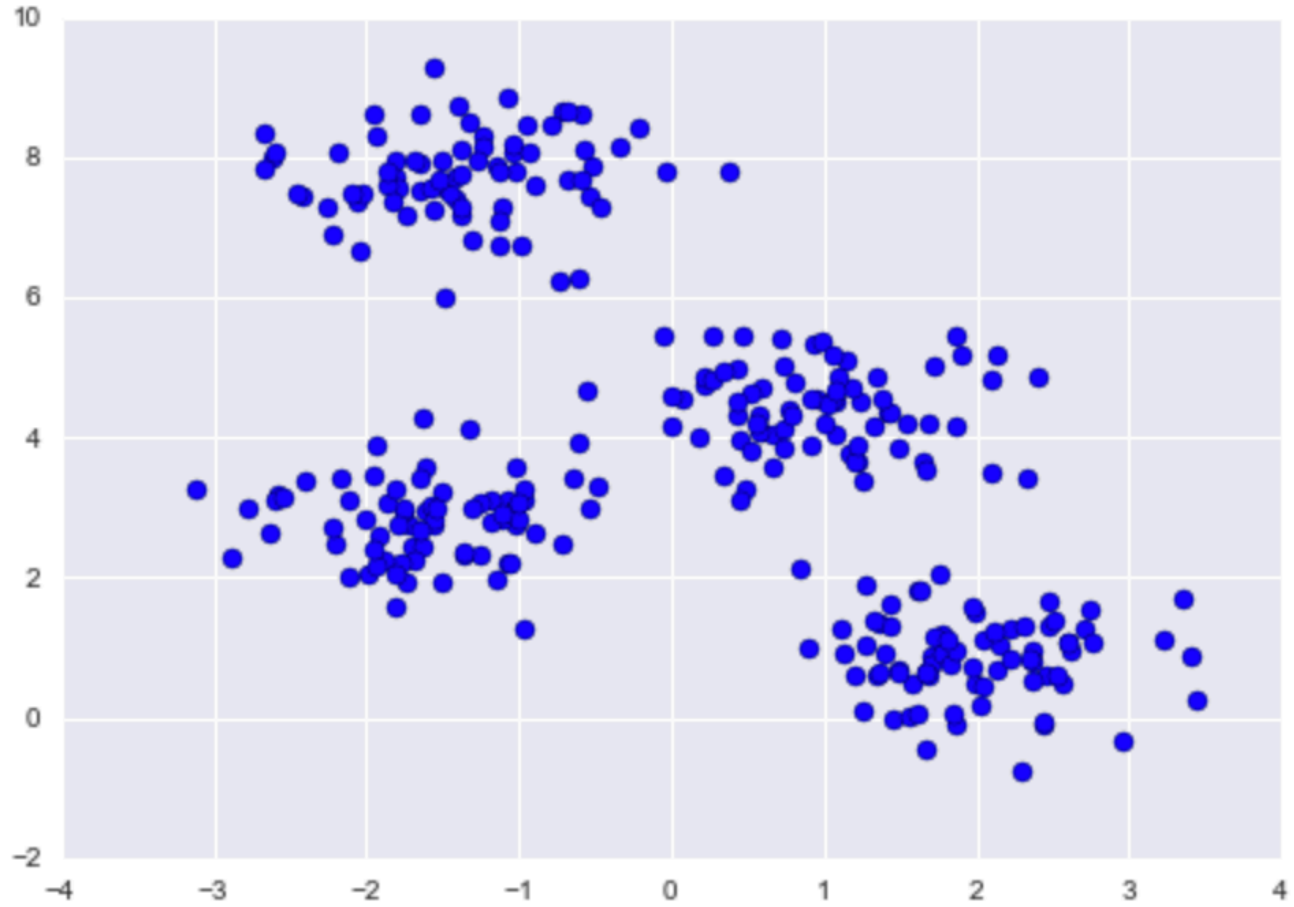
- Introduction to K-means
- Sequential Algorithm
- Parallel Approach
- Sample Readings
- Graphs
- Conclusion

Introduction:

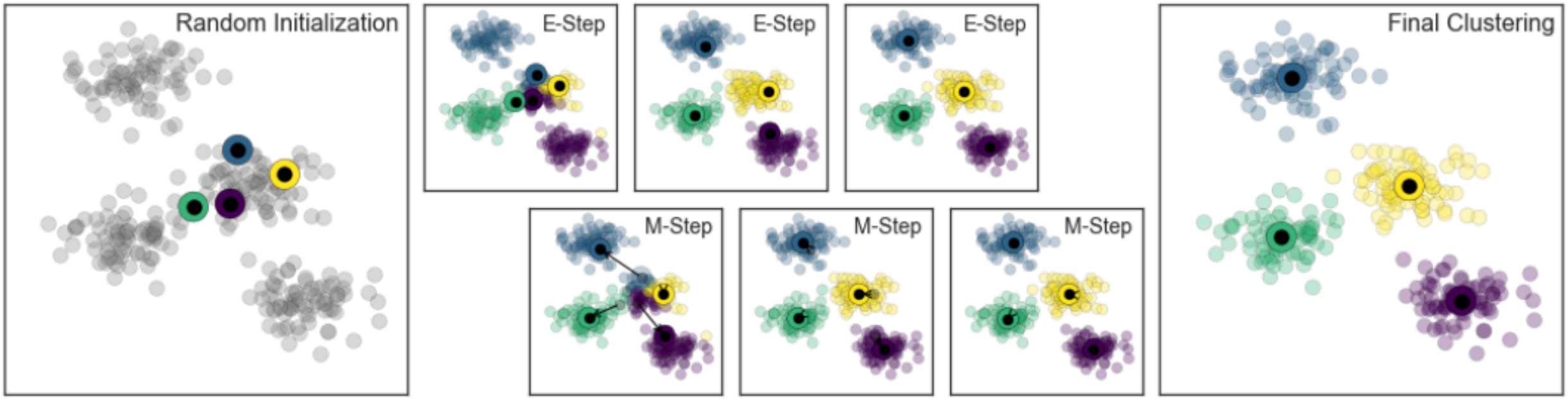
- Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data.
- The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset.
- The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centers, one for each cluster.
- Conventional k-means requires only a few steps. The first step is to randomly select k centroids, where k is equal to the number of clusters you choose. Centroids are data points representing the center of a cluster.

Sequential Algorithm

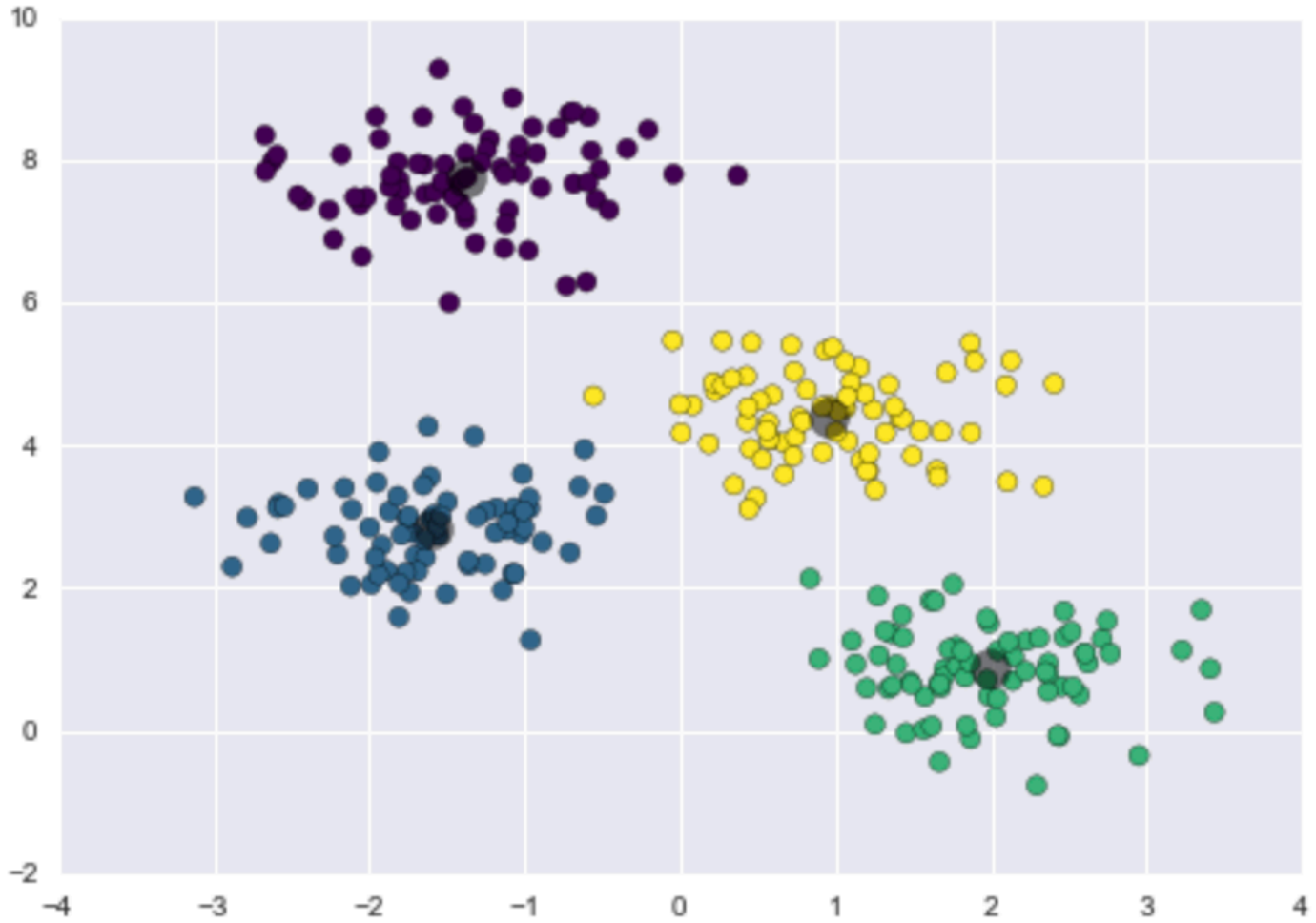
- Specify the number ‘k’ of the clusters to be assigned.
- Randomly initialize ‘k’ centroids.
- Assign each point to its nearest centroid by calculating euclidian distance from the point to the centroid.
- Compute the new centroid by calculating the mean of all points in the cluster.
- Repeat steps 3 and 4 till there is no change in the centroid positions.



Initial Data points


Random Initialization
E-Step
E-Step
E-Step
M-Step
M-Step
M-Step
Final Clustering

Steps



After K-Means clustering

Example:

- Suppose we take a few data points [1,3,5,12,13,14}. Initially lets assume cluster centers to be 1 and 5
- Clusters c1:[1,3] c2:[5,12,13,14]. We recalculate centroids by calculating mean of the cluster: c1=[1,3 c2=[5,12,13,14]= 11
- Now we calculate Euclidean distance of each cluster point w.r.t new centroids and update the clusters c1=[1,3,5] c2=[12,13,14]
- Again we calculate mean c1= mean(1,3,5) =3 c2= mean(12,13,14)= 13
- Again calculate euclidean distance and calculate points and rearrange clusters c1= [1,3,5] c2 =[12,13
- No change in cluster. This will be our stopping point.

Parallel Approach:

- Data is divided among each processor equally.
- The processor with rank 0 initializes k random centroids and broadcasts it to all other processors.
- Each processor then locally calculates distance of the point from each centroid and is divided into K clusters.
- Then the processor locally calculates the sum of each cluster and returns the sum and length of each cluster to the processor with rank 0.
- Then, Processor with rank 0, receives the sum and length of the clusters and calculate the new clusters centroids, a broadcast it to all the processors.
- Above procedure of clustering continues for n iterations.

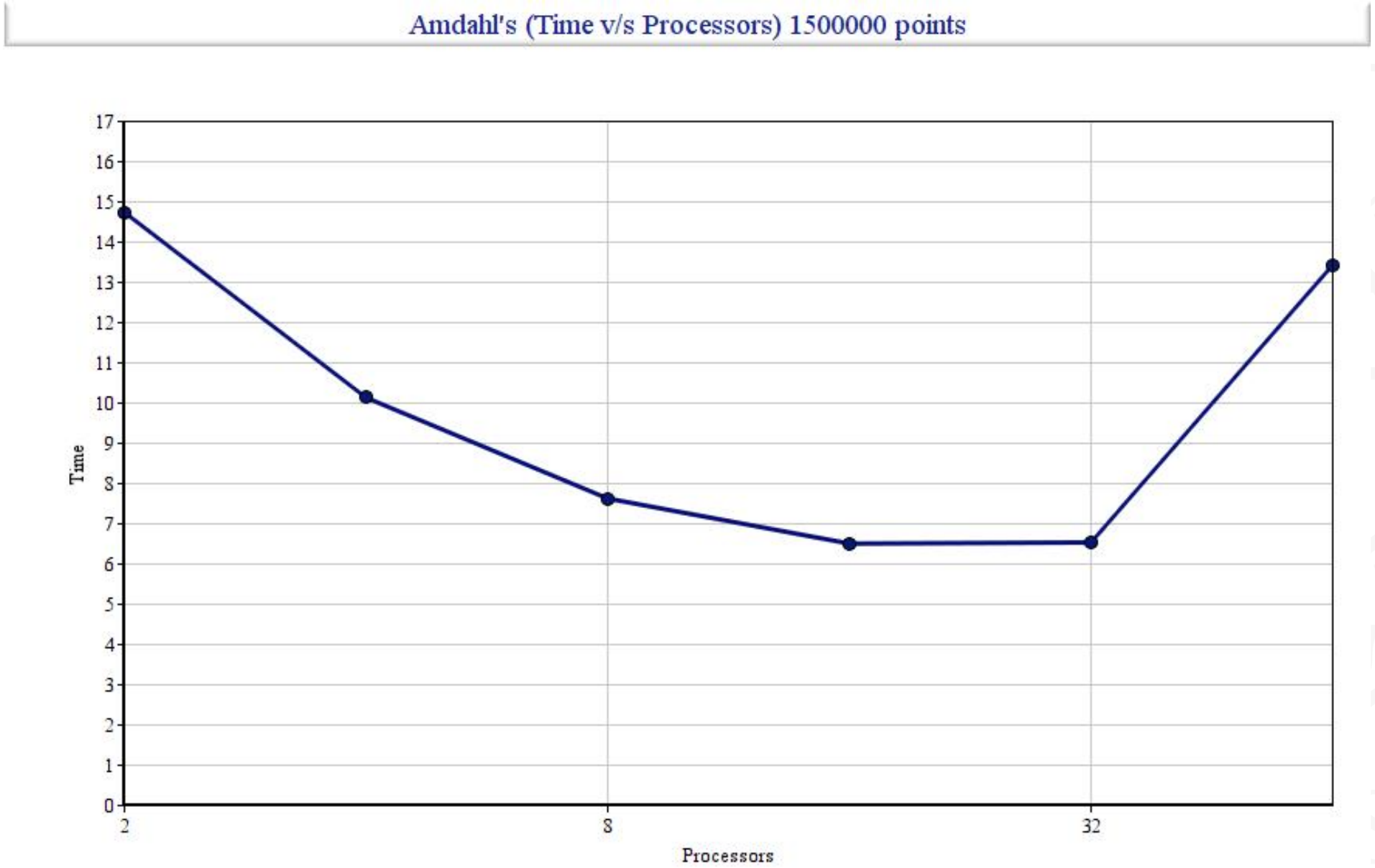
Readings:

Below readings show the new cluster centers calculated for different number of data points with increasing number of processors.

```
[mrunalna@vortex2:~]$ ./kmeans-parallel.sh
 Anaconda Python 2.7 version 2019.10 has been loaded.
 Intel-MPI is in your path. This is adequate for compiling and running most codes. Source the
/util/academic/intel/17.0/compilers_and_libraries_2017/linux/mpi/intel64/bin/mpivars.sh file for more features.
[[8.02073608 1.04728374]
 [0.97921048 0.94142644]
 [5.02002949 5.05160844]]
0.0372860431671
[mrunalna@vortex2:~]$ ./kmeans-parallel.sh
 Anaconda Python 2.7 version 2019.10 has been loaded.
 Intel-MPI is in your path. This is adequate for compiling and running most codes. Source the
/util/academic/intel/17.0/compilers_and_libraries_2017/linux/mpi/intel64/bin/mpivars.sh file for more features.
[[1.00343614 0.93228404]
 [4.96220196 4.95370008]
 [7.90394033 0.9955709 ]]
0.0279741287231
[mrunalna@vortex2:~]$ ./kmeans-parallel.sh
 Anaconda Python 2.7 version 2019.10 has been loaded.
 Intel-MPI is in your path. This is adequate for compiling and running most codes. Source the
/util/academic/intel/17.0/compilers_and_libraries_2017/linux/mpi/intel64/bin/mpivars.sh file for more features.
[[1.01064311 0.92602857]
 [4.88073738 5.01299327]
 [7.97499368 1.03243626]]
0.0527241230011
[mrunalna@vortex2:~]$
```

Sample Readings of Data Points and Number of Processors

Number of Data Points	2 processors	4 processors	8 processors	16 processors	32 processors	64 processors
1500	0.03410450617	0.0239983	0.03170152505	0.128037254	0.331446768	2.418035666
15000	0.14970914	0.1061520179	0.0765351804	0.07758331295	0.06714200974	4.067548831
150000	1.318409486	0.9297121763	0.7582879464	0.6338289976	0.7354993025	3.061531308
1500000	14.74032735	10.13659803	7.62879169	6.509320339	6.537833664	13.43311969
15000000	152.7570831	103.5258799	81.64824339	70.10960893	69.4403375	132.9132652


Amdahl's (Time v/s Processors) 1500000 points
Time
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
2
8
32
Processors


Gustafon's (Time v/s Processors)
Time
11.0
10.5
10.0
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
8
32

Conclusion:

- We observe significant speedup unto 32 processors.
- Cost of communication affects the speedup significantly when the number of processors exceeds 64.
- The significant change in the speedup is observed for a large number of data and more number of clusters

References:

- https://mpi4py.readthedocs.io/en/stable/tutorial.html
- https://arxiv.org/pdf/1608.06347.pdf

Thank You!