PARALLEL ALGORITHMS K-MEANS CLUSTERING

Presenter:
Divya Srivastava
divyasri@buffalo.edu
Person No.- 50290383

University at Buffalo The State University of New York

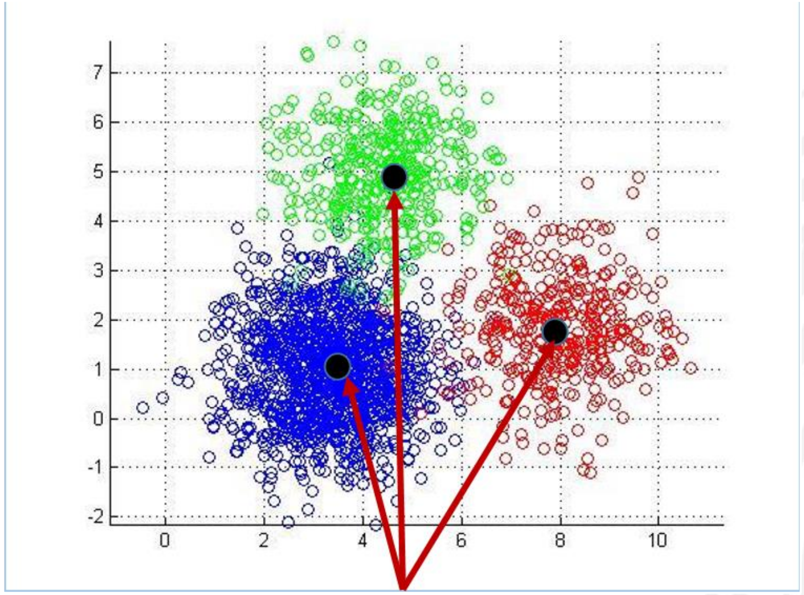
Outline

University at Buffalo The State University of New York

Problem

K-means Clustering

Dividing a large vector filled of points into smaller groups which are organized according to a centroid point, each group must have almost the same number of components.



Centroids (K)

Algorithm

The algorithm as described in Andrew Ng’s Machine Learning class over course era works as follows:

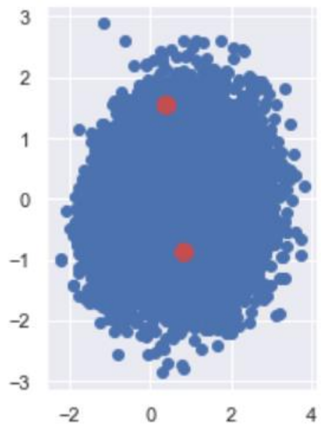
- Initialize K cluster centroids randomly
- Repeat the following until there is no further change in the centroid values or for a specific number of maximum iterations-
    - For each point, compute which centroid is nearest to it
    - For each centroid, move its location to the mean location of the points assigned to it
- The distance between a centroid and a point is calculated by: - Euclidean Distance: Point – K = |Distance| (Absolute value result)

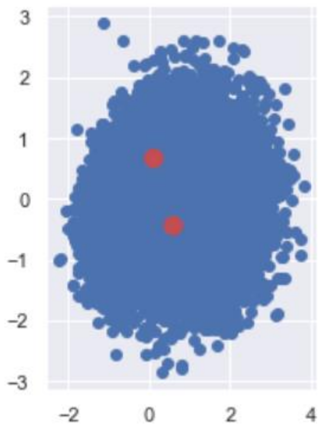
Parallel Algorithm Implementation (MPI)

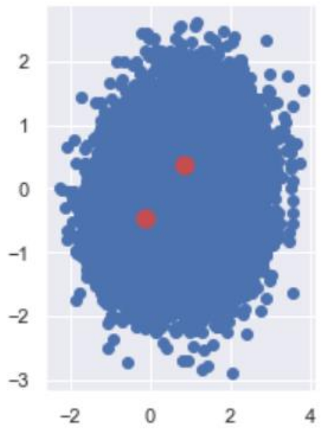
- The parallel implementation uses data parallelism.
- Data objects to be clustered are evenly partitioned among all processes.
- The cluster centers are replicated.
- Global-average reduction for all cluster centers is performed at the end of each iteration in order to generate the new cluster centers.
  - Calculate local sums for each cluster in each processor
  - Send these sums to processor 0 and compute the global average
  - Broadcast these new centroid values to all processors
- Repeat these steps until n iterations

University at Buffalo The State University of New York

Serial Implementation Results

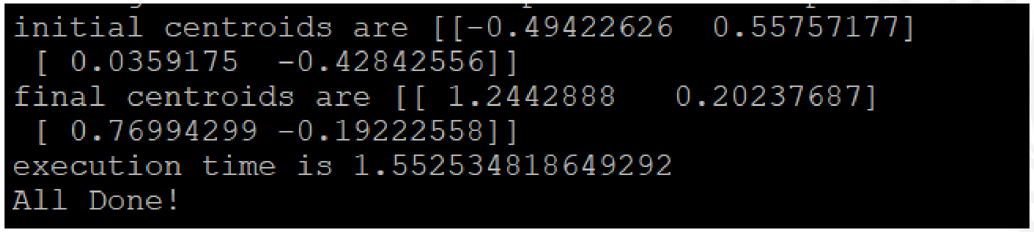

Initial Centroids


Centroids after 1st Iteration


Final Centroids after 30th iteration

```
initial centroids are [[ 0.27657513  0.06587827]
 [-0.25956531 -0.46897107]]
final centroids are [[ 0.77043796  0.21717402]
 [-0.41681164 -0.48771682]]
execution time is 4.567802667617798
```

Parallel Implementation Results (3 Processors)

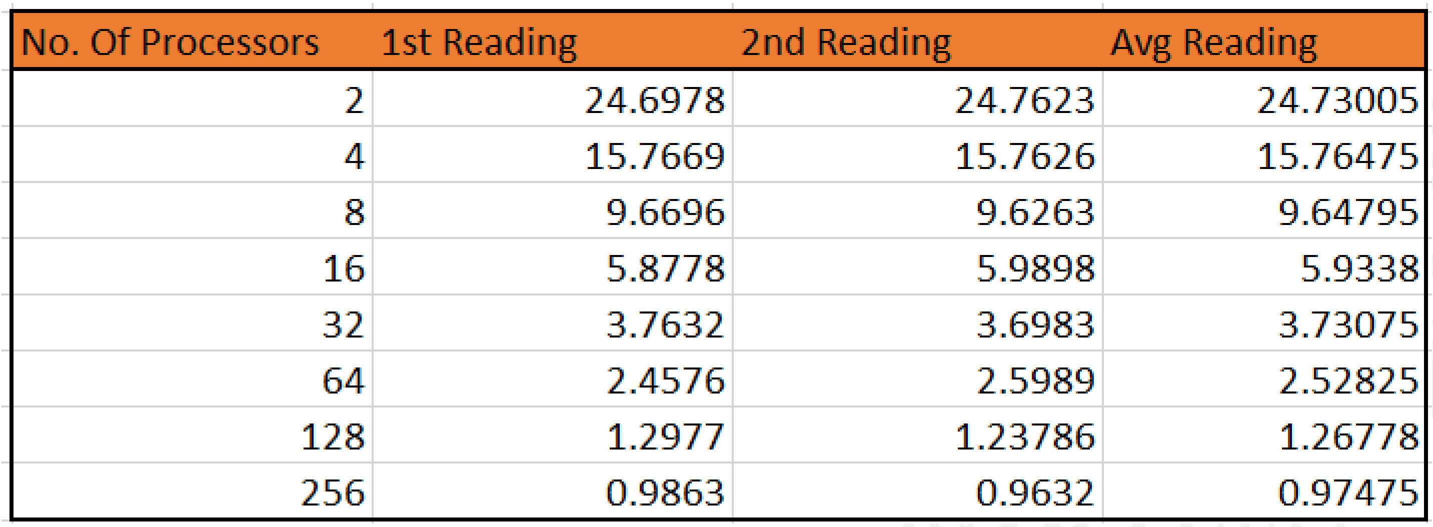


```
initial centroids are [[-0.49422626  0.55757177]
 [ 0.0359175  -0.42842556]]
final centroids are [[ 1.2442888   0.20237687]
 [ 0.76994299 -0.19222558]]
execution time is 1.552534818649292
All Done!
```

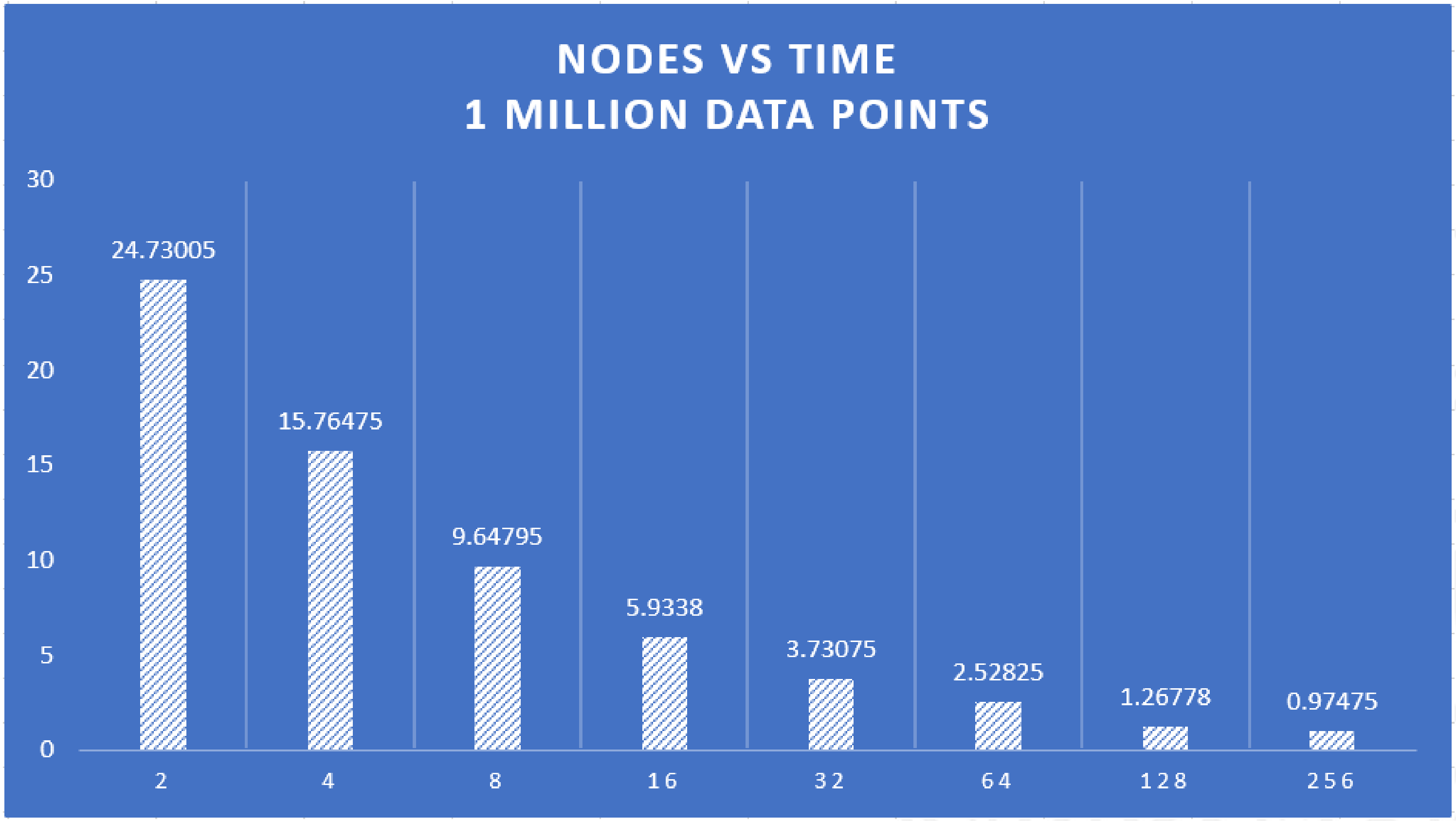
University at Buffalo The State University of New York

EXPERIMENTS

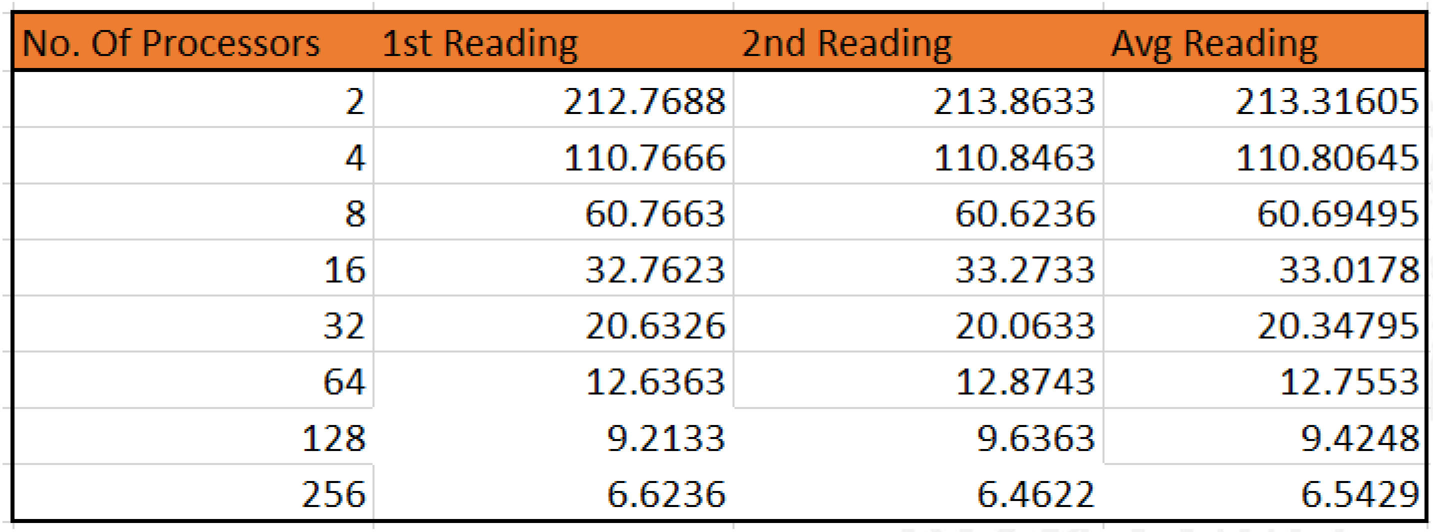
Readings for 1 Million Data Points



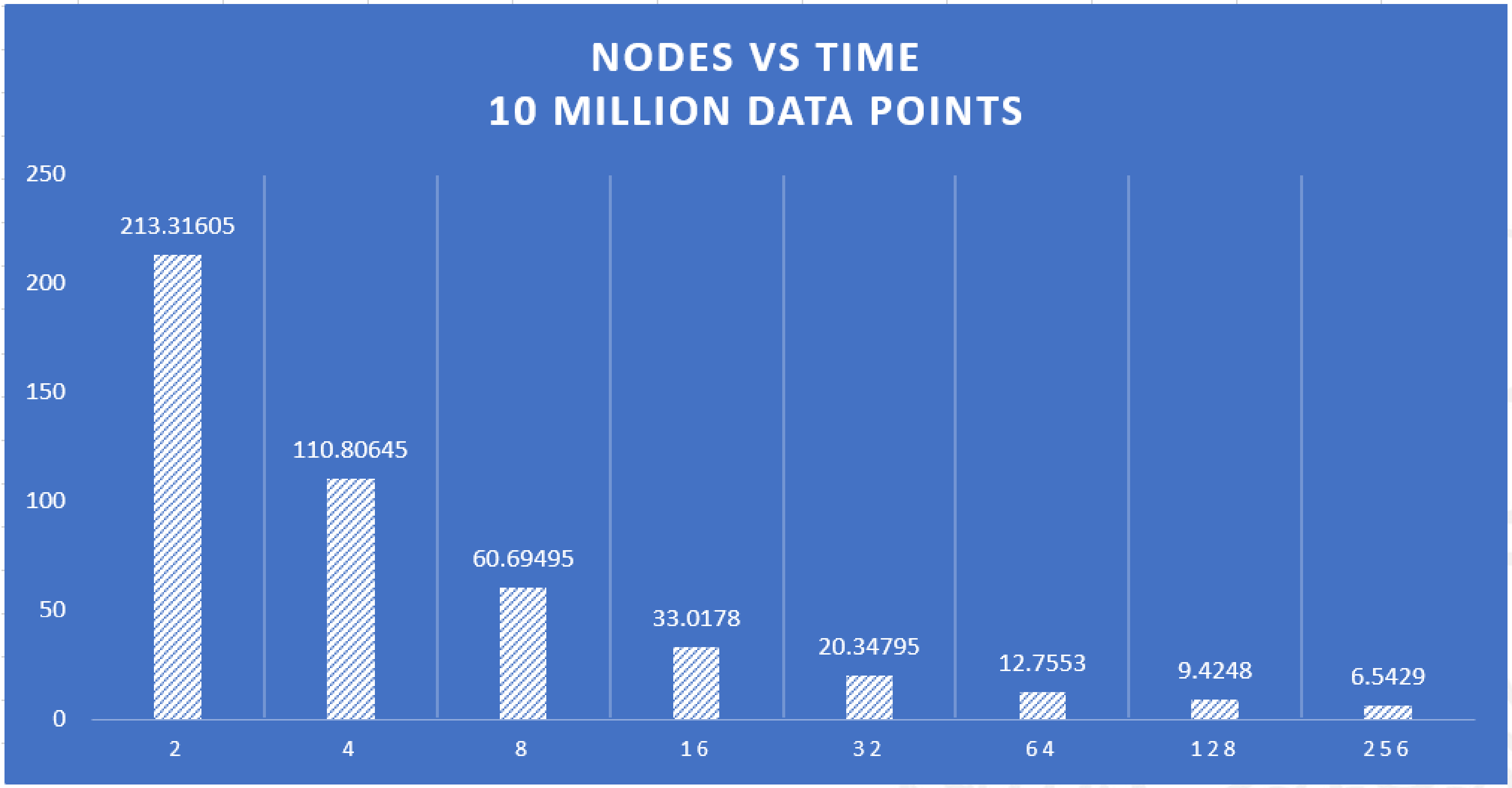
No. Of Processors	1st Reading	2nd Reading	Avg Reading
2	24.6978	24.7623	24.73005
4	15.7669	15.7626	15.76475
8	9.6696	9.6263	9.64795
16	5.8778	5.9898	5.9338
32	3.7632	3.6983	3.73075
64	2.4576	2.5989	2.52825
128	1.2977	1.23786	1.26778
256	0.9863	0.9632	0.97475


NODES VS TIME
1 MILLION DATA POINTS
30
25
20
15
10
5
0
24.73005
15.76475
9.64795
5.9338
3.73075
2.52825
1.26778
0.97475
2
4
8
16
32
64
128
256

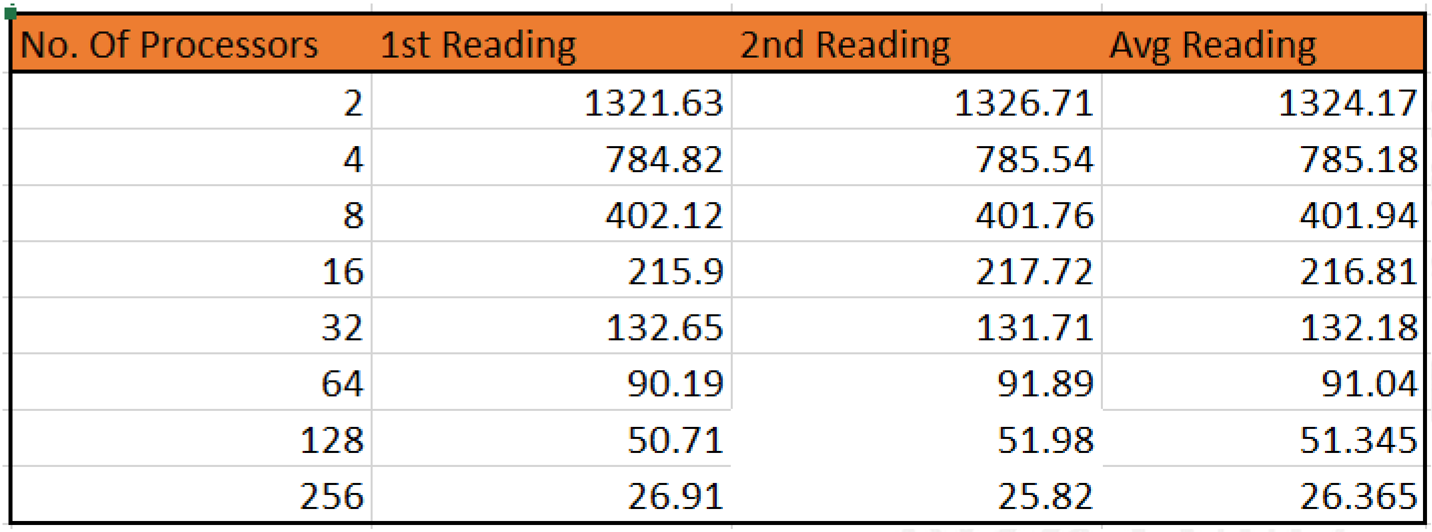
Readings for 10 Million Data Points



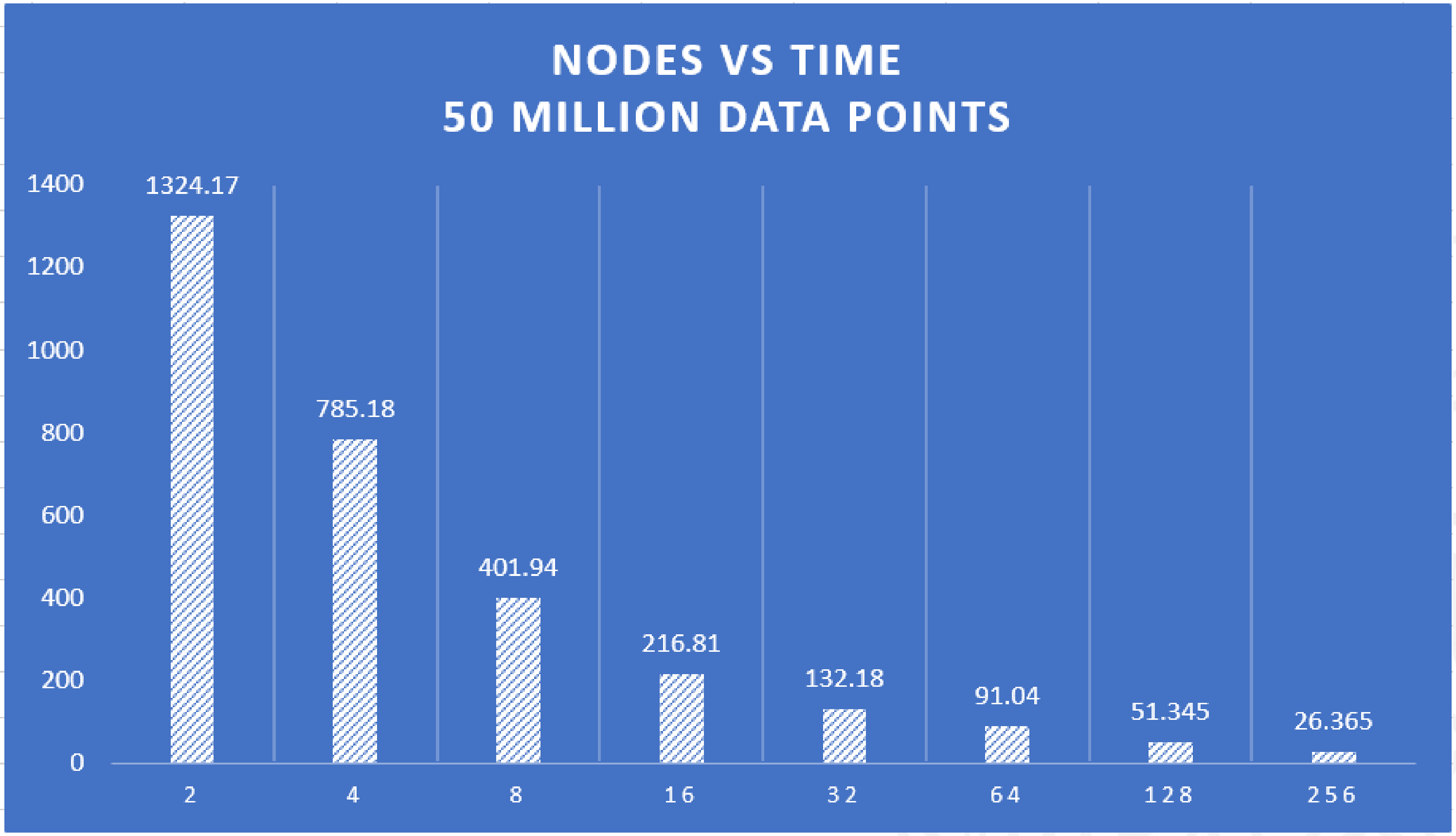
No. Of Processors	1st Reading	2nd Reading	Avg Reading
2	212.7688	213.8633	213.31605
4	110.7666	110.8463	110.80645
8	60.7663	60.6236	60.69495
16	32.7623	33.2733	33.0178
32	20.6326	20.0633	20.34795
64	12.6363	12.8743	12.7553
128	9.2133	9.6363	9.4248
256	6.6236	6.4622	6.5429


NODES VS TIME
10 MILLION DATA POINTS
250
200
150
100
50
0
213.31605
110.80645
60.69495
33.0178
20.34795
12.7553
9.4248
6.5429
2
4
8
16
32
64
128
256

Readings for 50 Million Data Points



No. Of Processors	1st Reading	2nd Reading	Avg Reading
2	1321.63	1326.71	1324.17
4	784.82	785.54	785.18
8	402.12	401.76	401.94
16	215.9	217.72	216.81
32	132.65	131.71	132.18
64	90.19	91.89	91.04
128	50.71	51.98	51.345
256	26.91	25.82	26.365


NODES VS TIME
50 MILLION DATA POINTS
1400
1200
1000
800
600
400
200
0
1324.17
785.18
401.94
216.81
132.18
91.04
51.345
26.365
2
4
8
16
32
64
128
256

Observations

- In order to achieve better performance its critical to identify the optimal number of processors that would be required for any given computation as more processors doesn't mean significant gains in speedup.
- For 1 million data points, the gain observed from 126 to 256 processors was just 20% which increased to 30% in case of 10 million data points and to 50% in case of 50 million data points. Hence, increasing the no. of processors to 256 is worth only in case of huge data.
- The communication time cannot be ignored. When number of processors is increasing, the efficiency of parallel algorithm drops, cost of communication increases (This was observed when we increased the number of processors from 128 to 256 with 1 million total data-points).
- For small data size, a graph of valley shape would have been observed where instead of gain, loss would have been observed as communication cost would have been more prominent with increasing no. of processors.
- The difference in execution times for different processors is noticeable when the data size is large.

Challenges

- Faced deadlock problems and had to make sure that the blocking functions in MPI were used correctly.
- Experiments on same setup yielded slightly different values. Took averages to obtain the final results.

References

- Algorithms Sequential and Parallel, A Unified Approach 3rd edition

  ~Russ Miller, Laurence Boxer
- Mpi4Py official documentation

Thank you