CONVEX-HULL PROBLEM

(Using Quick Hull and K-means Algorithms)

Snehal Jadhav

snehalja@buffalo.edu

50315111

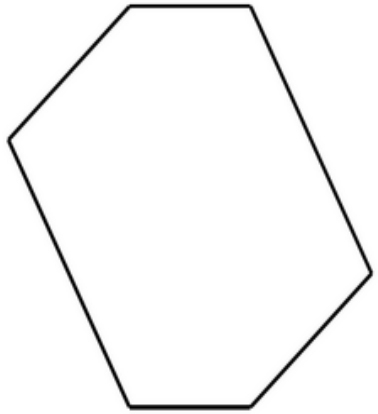
University at Buffalo The State University of New York

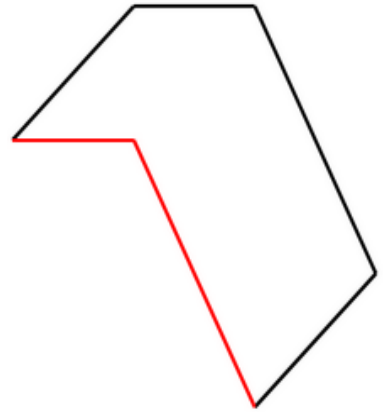
Contents of this presentation:

- Convex Hull Overview
- Applications
- The Quick Hull Algorithm
- K-means Algorithm
- The combinatory parallel approach
- Observations
- Inferences
- Challenges
- Future Scope
- References.

What is a convex hull?

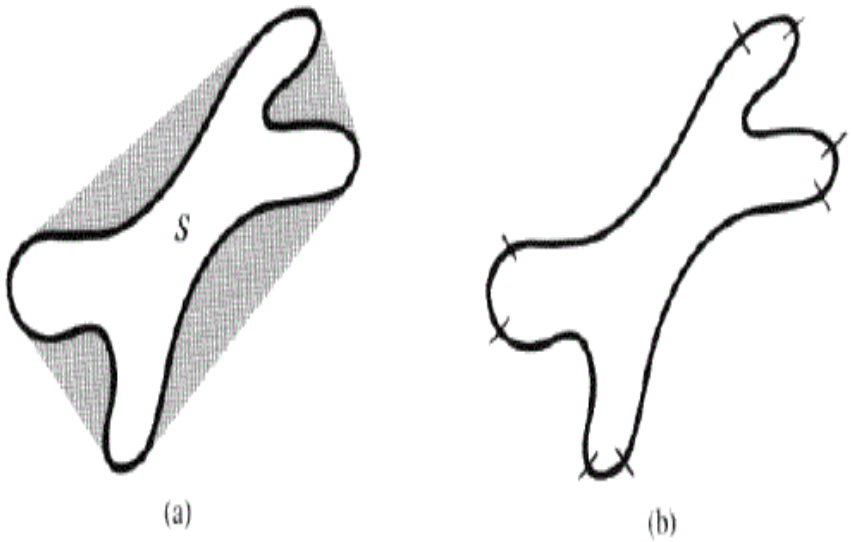
The convex hull of a set of points is defined as the smallest convex polygon, that encloses all of the points in the set.
Convex means that the polygon has no corner that is bent inwards.





Applications :

- **Shape analysis** : Shapes may be classified for the purposes of matching by their "convex deficiency trees", structures that depend for their computation on convex hulls.
- **Smallest box :** Finding the smallest three-dimensional box surrounding an object in space depends on the convex hull of the object.



(a) A region (S) and its convex deficiency (shaded); (b) partitioned boundary.

Applications :

- **Hand-gesture-recognition**" domain : Convex hull works as an envelope around the hand.
  - When the convex hull is drawn round the contour of the hand, it fits a set of contour points of the hand within the hull.
- **Collision avoidance** : Avoid collisions with other objects by defining the convex hull of the objects.

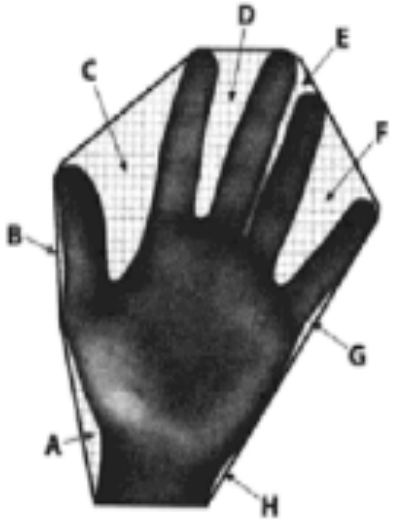


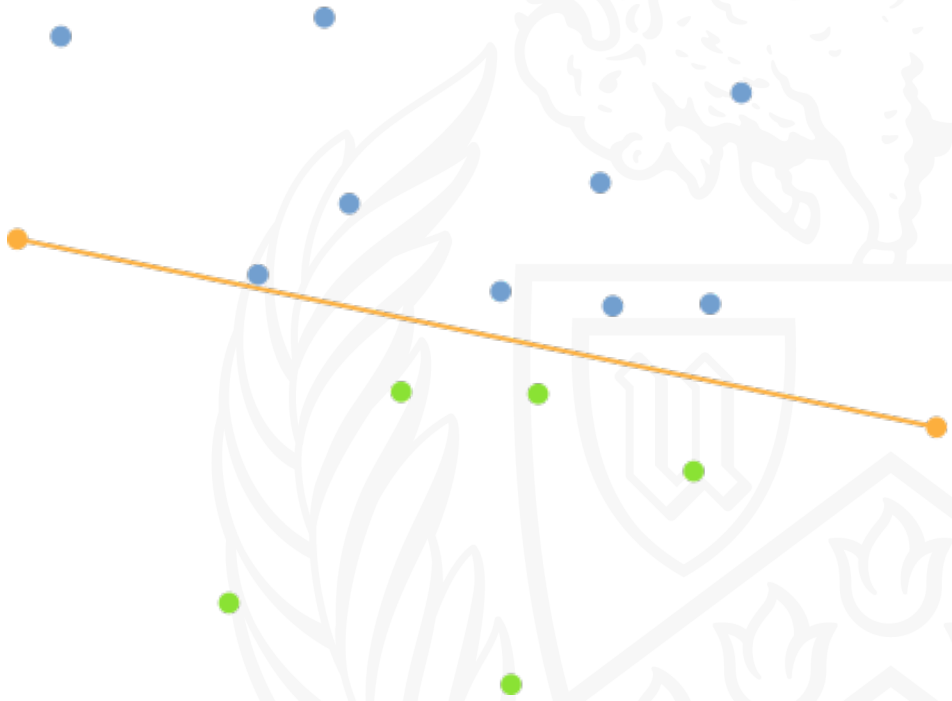
Figure 1 Convex hull of the contour of the hand and convex defect

Fig. 1 Detected convex and defect points in the image

Quick Hull Algorithm

1. Find the points with minimum and maximum x coordinates, as these will always be part of the convex hull.

2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.

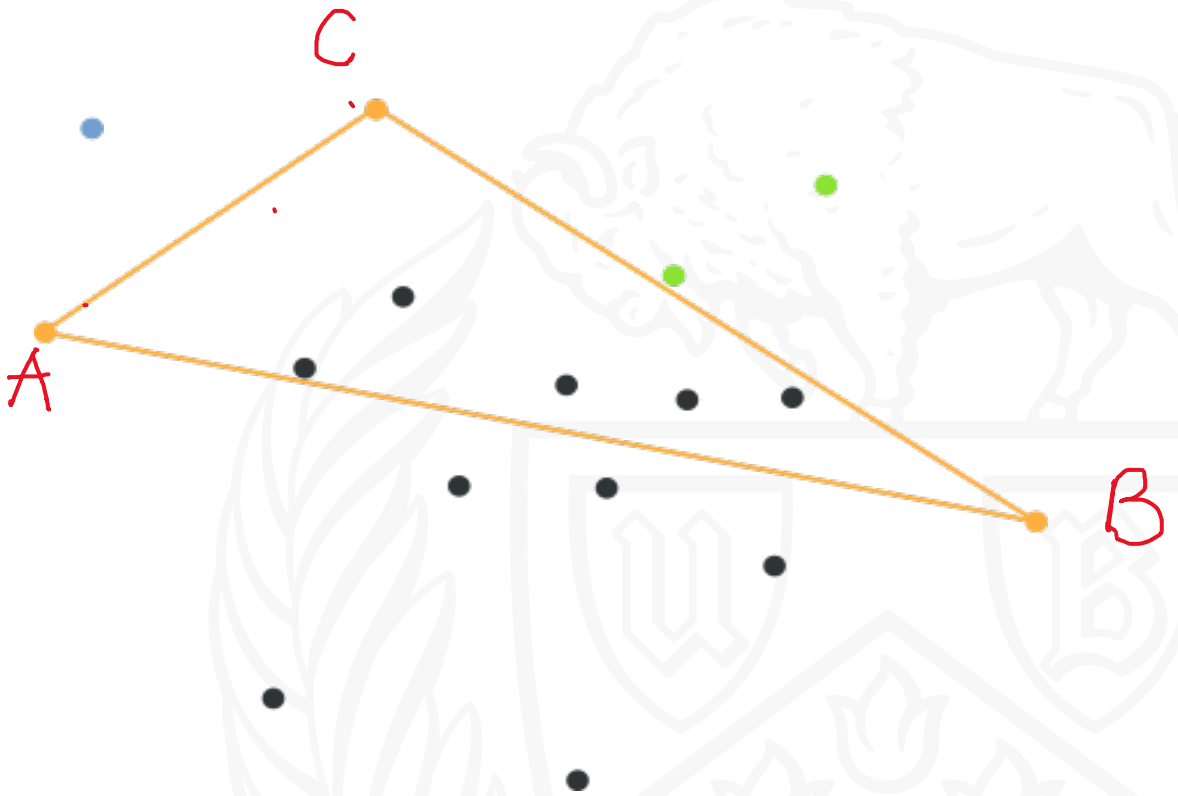


Quick Hull Algorithm

3. Determine the point, on one side of the line, with the maximum distance from the line. This point forms a triangle with those of the line.

4. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.

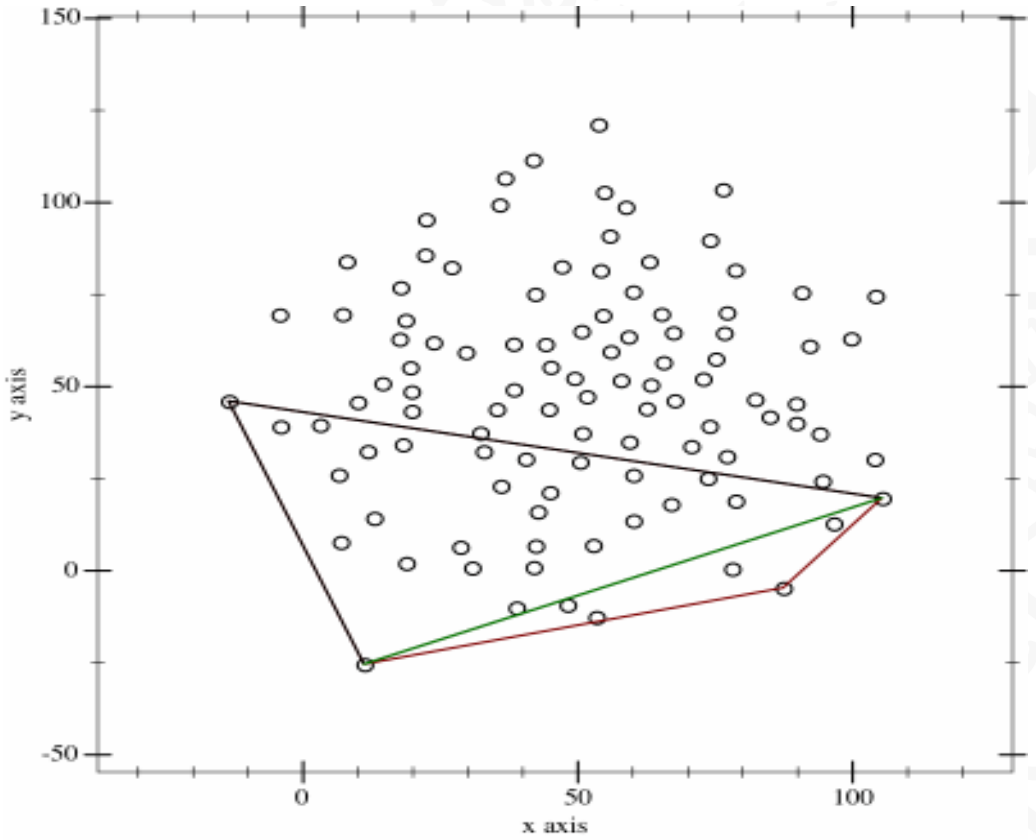
5. Repeat the previous two steps on the two lines formed by the triangle(AC and BC) (except the initial line- AB).



Quick Hull Algorithm

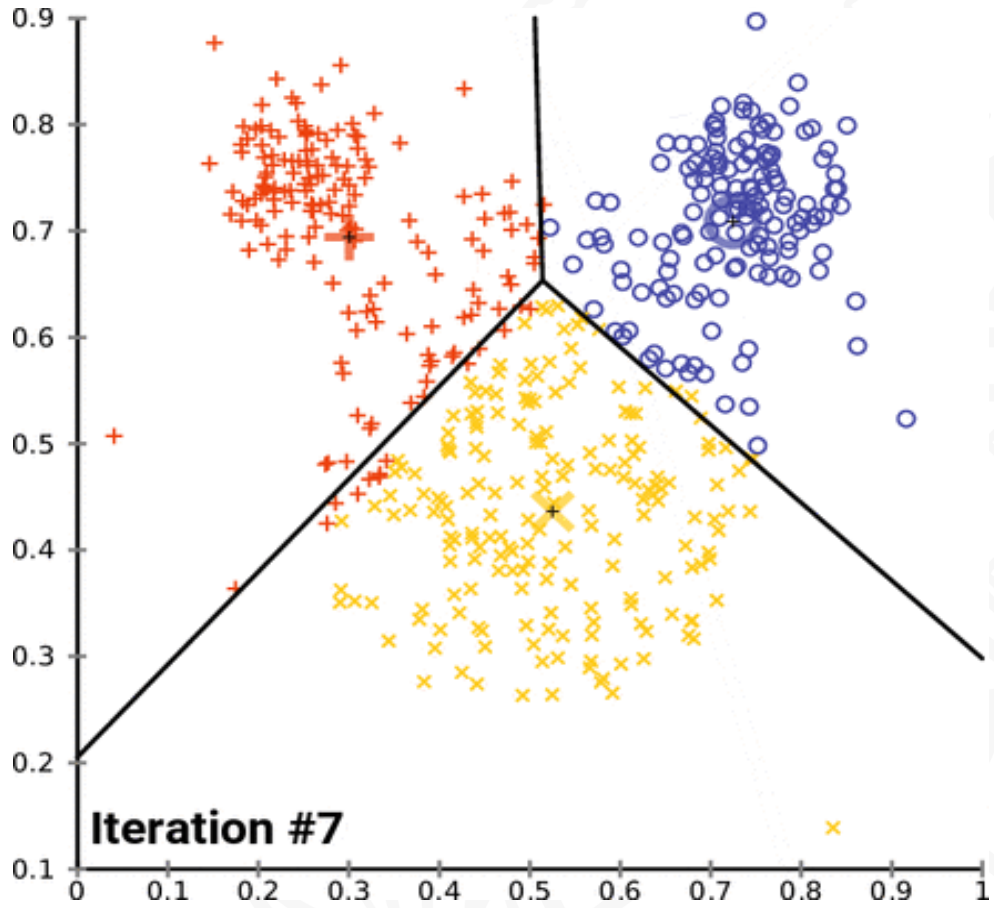
5. Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.

6. Just like the Quicksort algorithm, it has the expected time complexity of O(n log n), but may degenerate to O(nh) = O(n^2) in the worst case.

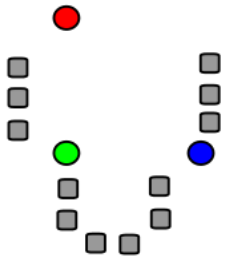


K-means Algorithm

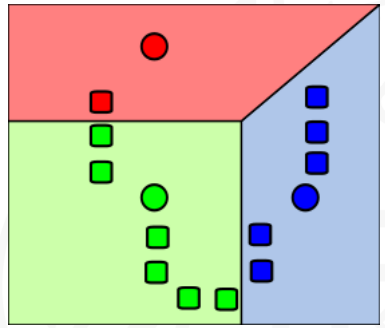
- You'll define a target number *k*, which refers to the number of centroids you need in the dataset.
- Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.
- The *'means'* in the K-means refers to averaging of the data; that is, finding the centroid.



K-means Algorithm

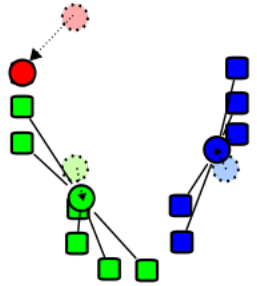


1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

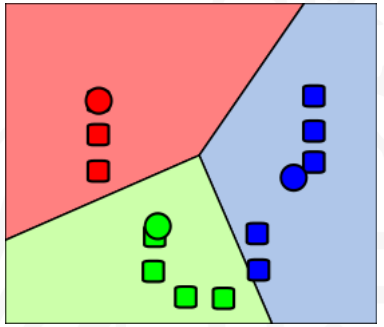


2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

K-means Algorithm



3. The centroid of each of the *k* clusters becomes the new mean.



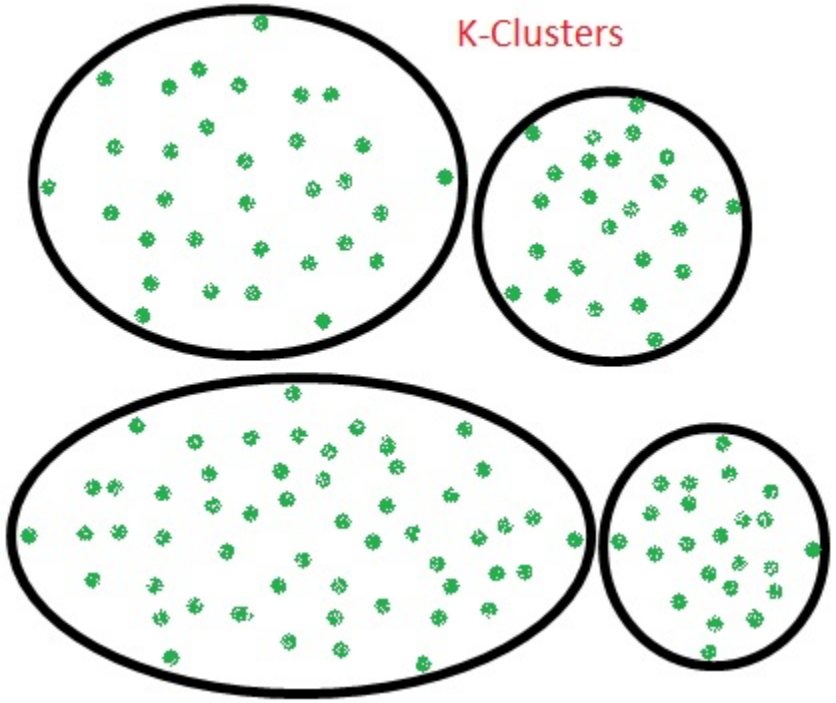
4. Steps 2 and 3 are repeated until convergence has been reached.

Parallel Convex Hull Using K-Means Clustering

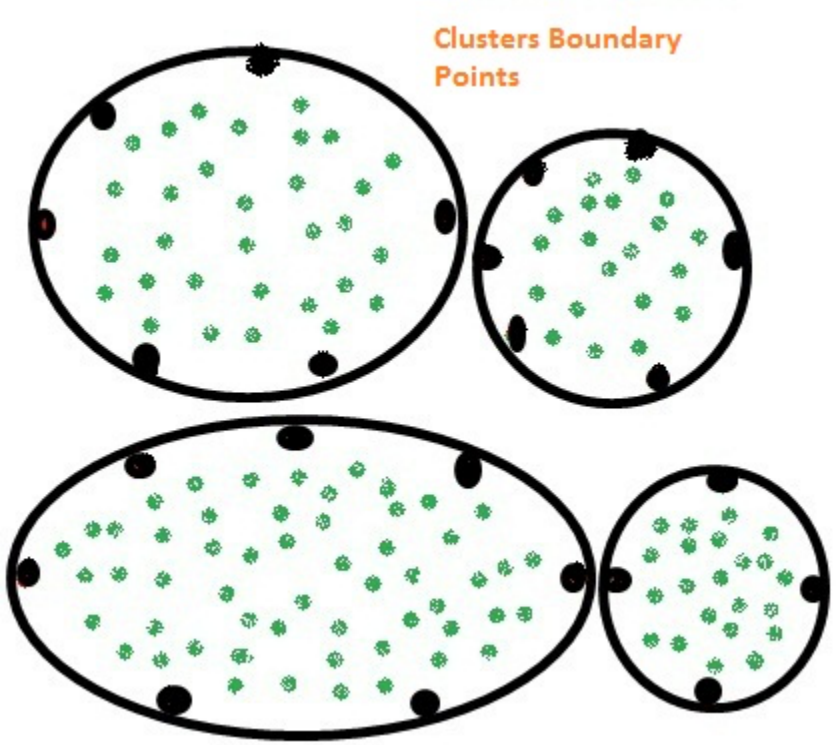
1. N points are divided into K clusters using K means.
2. Quick Hull is applied on each cluster (iteratively inside each cluster as well).
3. The convex hull points from these clusters are combined.
4. Quick Hull is applied again and a final Hull of all clusters is computed.

Convex Hull Using K-Means Clustering

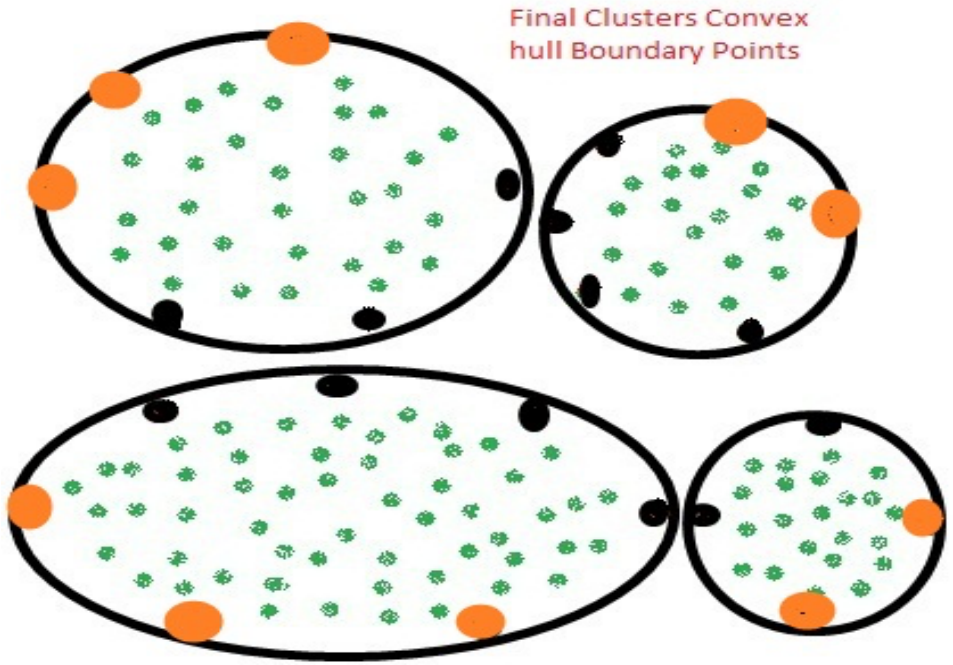
1



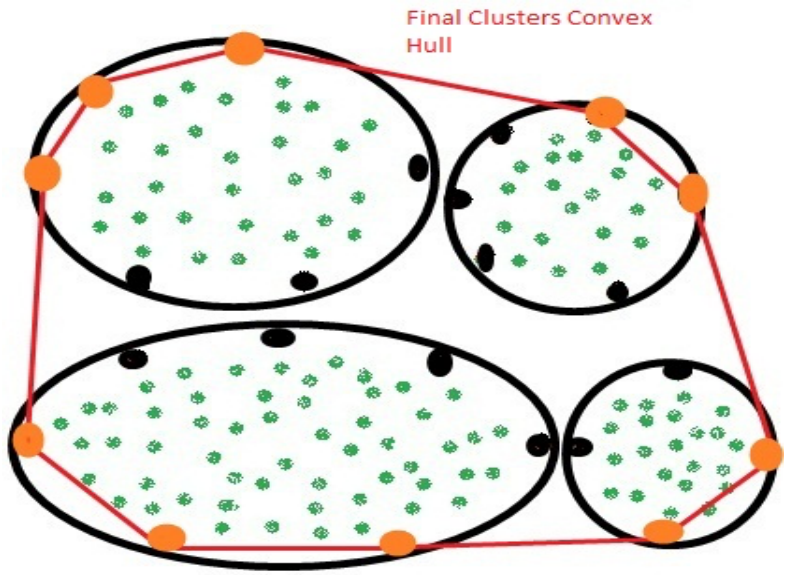
2



3



4



Performance on 1 node, 2 cores small set of points

Points	Time
100	0.0010
500	0.0014
1000	0.0015
5000	0.0025
10000	0.0050

8 clusters

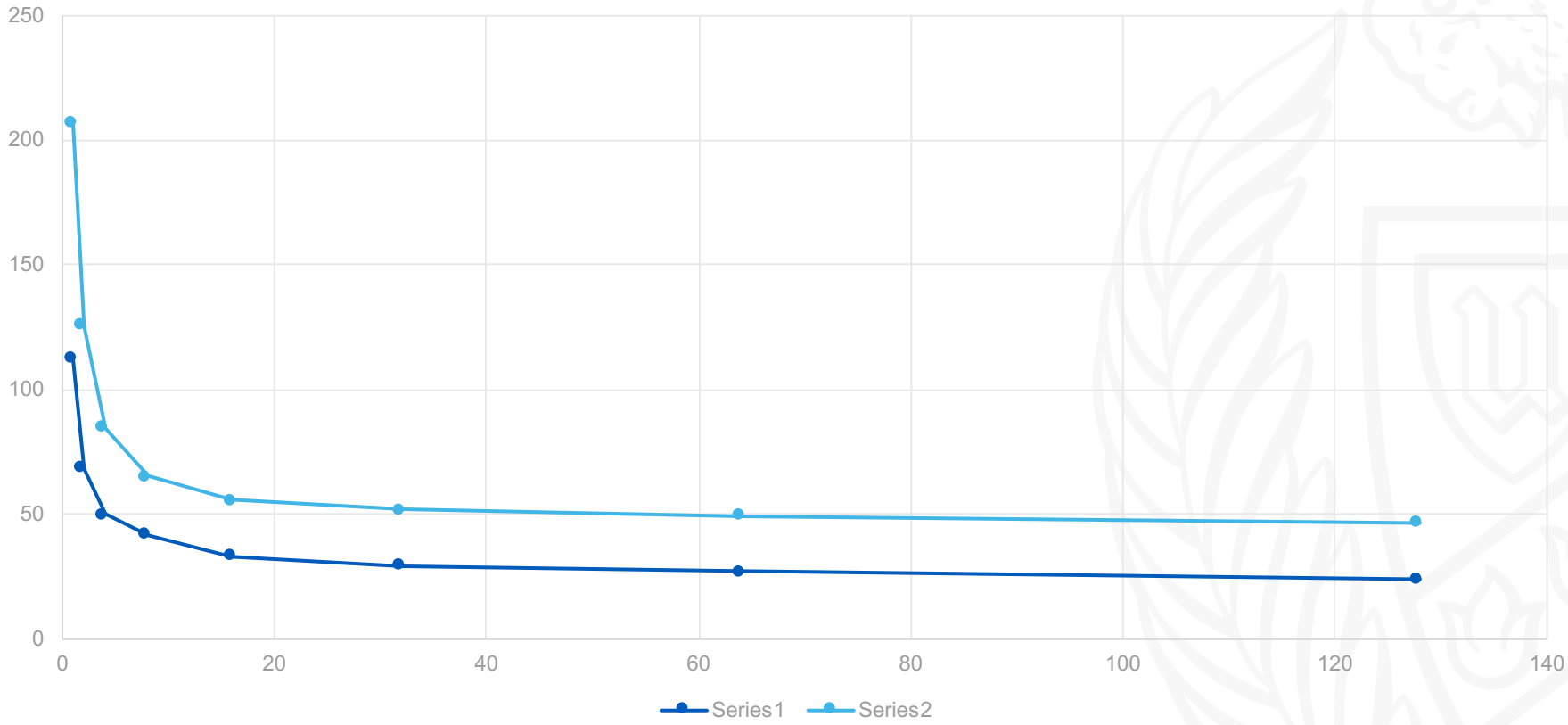
Points	Time
100	0.0012
500	0.0016
1000	0.0010
5000	0.0038
10000	0.0056

16 clusters

n	run1	run2	run3	avg	speedup	efficiency	cost
K = 128 AND 1 MILLION DATA POINTS TIME: SECONDS							
1	113.33	113.60	110.67	112.53			113
2	69.94	69.76	66.24	68.65	1.0	1.0	138
4	54.16	48.78	46.90	49.95	2.26	0.565	200
8	41.91	41.94	41.80	41.88	2.69	0.336	336
16	35.84	31.43	33.30	33.53	3.32	0.207	544
32	30.46	29.64	28.52	29.54	3.76	0.117	960
64	26.25	27.34	27.22	26.93	4.18	0.065	1728
128	23.92	24.16	24.08	24.05	4.7	0.036	3072

n	run1	run2	run3	avg	speedup	efficiency	cost
K = 256 AND 1 MILLION DATA POINTS TIME: SECONDS							
1	207.21	207.85	207.55	207.54			208
2	126.83	124.92	125.86	125.87	1.0	1.0	252
4	84.50	85.05	84.59	84.71	2.44	0.61	340
8	64.89	65.17	66.17	65.41	3.2	0.4	520
16	56.22	56.48	54.66	55.79	3.71	0.231	896
32	51.84	52.01	52.03	51.96	4	0.125	1728
64	49.39	49.74	49.42	49.52	4.16	0.065	3200
128	47.04	46.51	46.21	46.59	4.42	0.034	6016

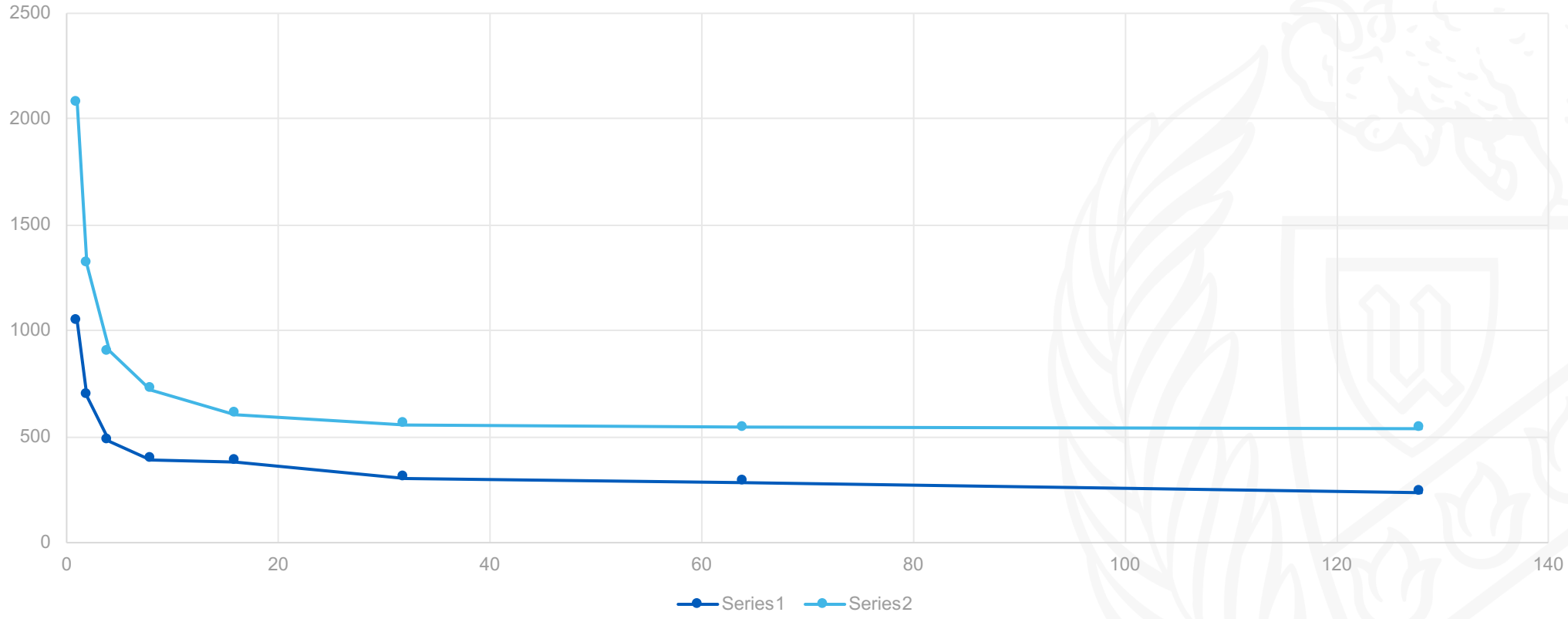
No. of processors (X) v/s time(Y)
Series 1:k=128 and Series 2: k=256 1M pts

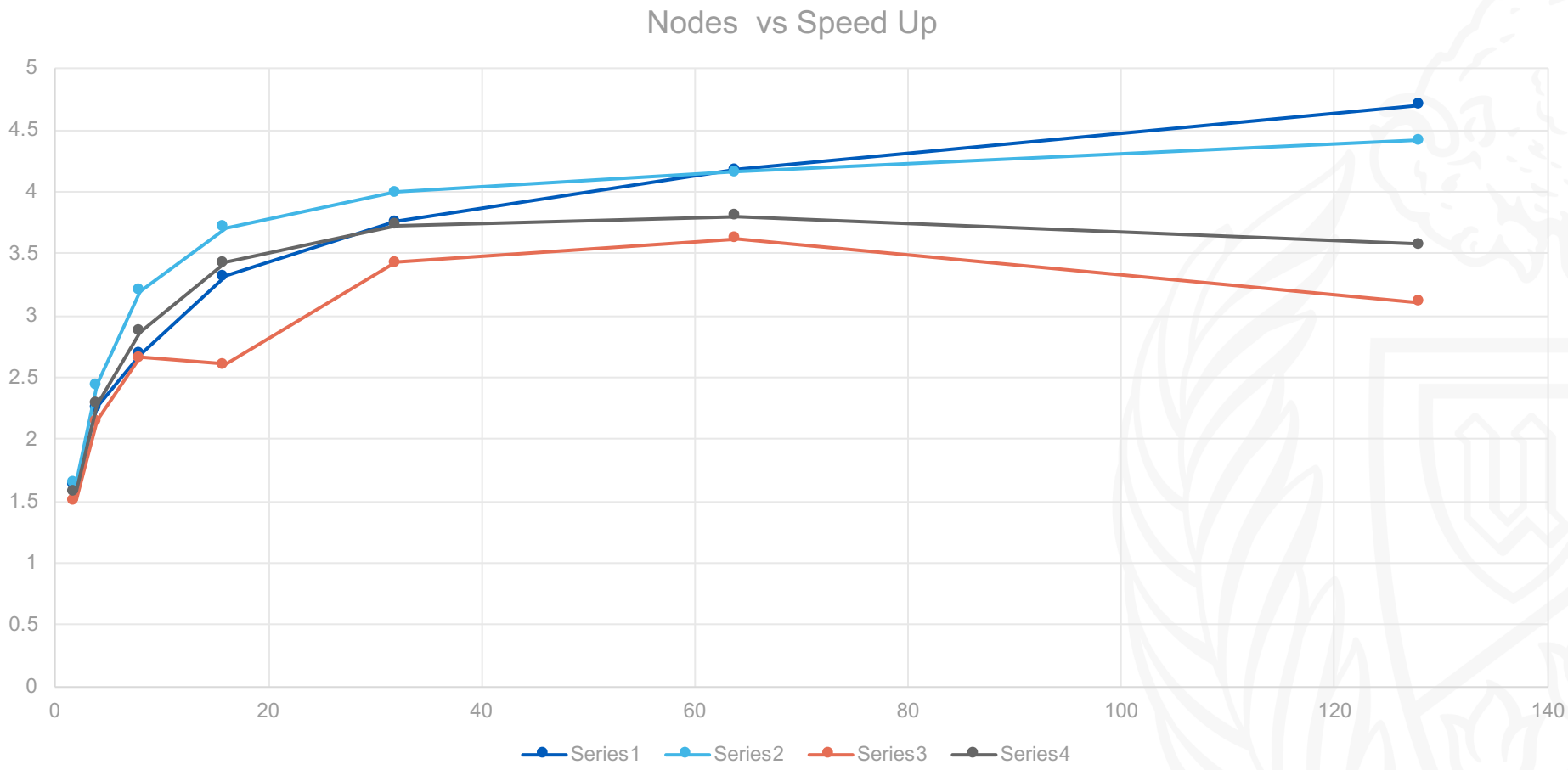


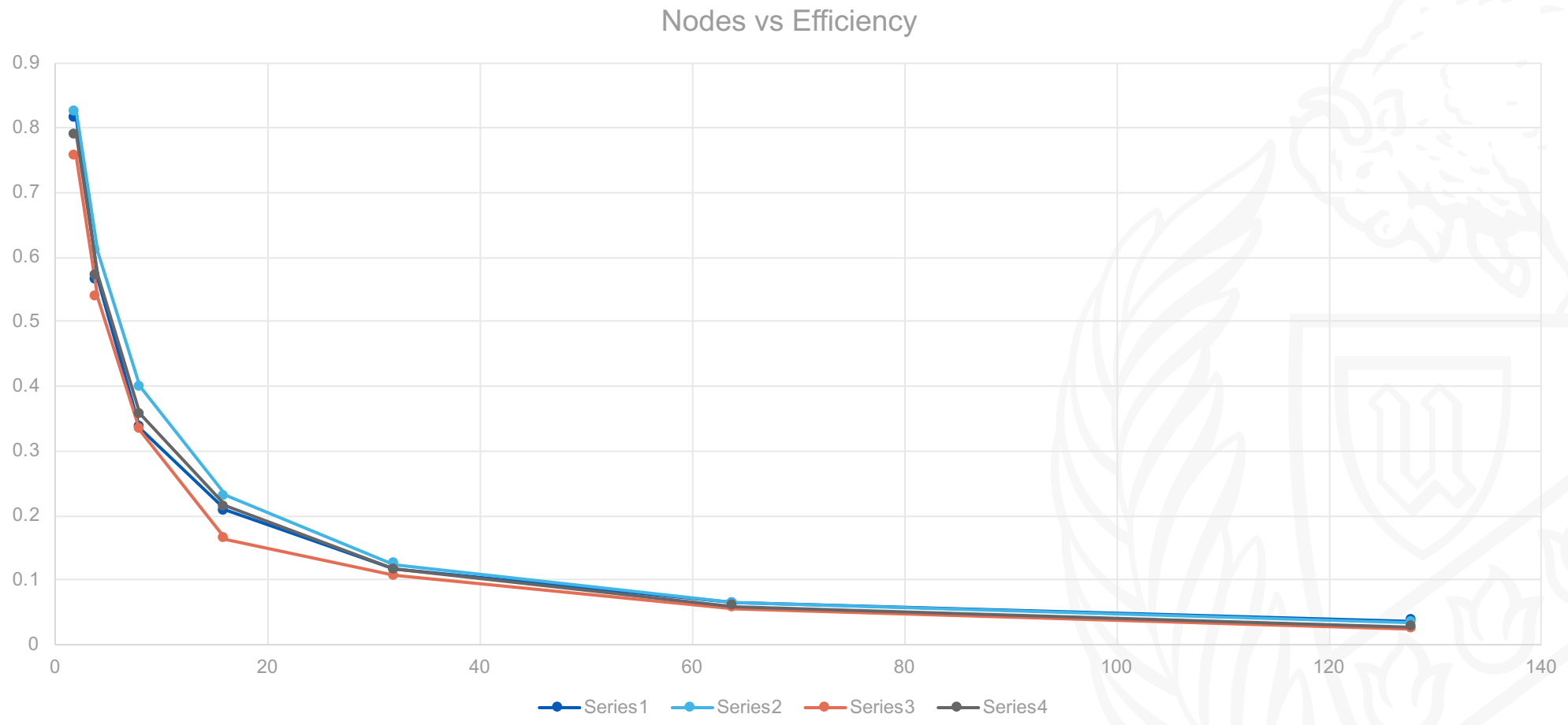
n	run1	run2	run3	avg	speedup	efficiency	cost
K = 128 AND 10 MILLION DATA POINTS TIME: SECONDS							
1	1058.82	1041.50	1039.49	1045.40			1045
2	693.01	691.87	692.23	692.37	1.0	1.0	1384
4	482.97	482.92	478.54	481.47	2.15	0.537	1940
8	393.13	390.74	392.13	392.00	2.66	0.332	3136
16	383.19	382.37	382.35	382.30	2.73	0.170	6112
32	313.55	310.20	288.31	304.02	3.43	0.107	9782
64	285.90	286.05	292.12	288.02	3.62	0.056	18432
128	237.32	235.97	234.92	236.07	4.41	0.034	30208

n	run1	run2	run3	avg	speedup	efficiency	cost
K = 256 AND 10 MILLION DATA POINTS TIME: SECONDS							
1	2063.62	2041.27	2126.57	2077.15			2077
2	1318.75	1319.66	1303.87	1314.1	1.0	1.0	2628
4	914.09	915.11	882.20	903.80	2.29	0.572	3616
8	723.37	722.27	723.23	722.96	2.87	0.358	5784
16	597.26	608.87	608.42	604.85	3.43	0.214	9680
32	552.50	564.53	551.43	556.15	3.73	0.116	17792
64	542.21	542.30	553.32	545.94	3.8	0.059	34944
128	540.02	541.20	535.58	538.27	3.85	0.030	68992

No. of processors (X) v/s time(Y)
Series 1:k=128 and Series 2: k=256 10 mil pts




Nodes vs Speed Up
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
140
Series1
Series2
Series3
Series4


Nodes vs Efficiency
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
120
140
Series1
Series2
Series3
Series4

Inferences:

- The ideal # of processors for 1 million data points is 16.
- The ideal # of processors for 10 million data points is 32.
- If # of clusters is less than the no of processors, the performance is degraded since PEs have < 1 (k/n) clusters to work with.
- After 32 the run time difference is almost negligible for the increase in the # of PEs.
  Reason: The communication time b/w PEs increases and significantly dominates the local computation time.

Challenges:

- Sequencing the Quick Hull and K-means parallel events.
- Queue time to access large memory servers.
- Parallelizing the Hybrid way.

Future scope:

- Implement the algorithm in Open MP.
- Implement the algorithm Hybrid (OpenMP+MPI).
- Compare scalability of MPI, OpenMP and Hybrid approaches.
- See the effect of choosing a different distance metric for clustering and/or a different strategy to initialize clusters.
- Implement the algorithm for 3+ D data.

References:

- Dr. Miller R. & Dr. Boxer L. (2012). Algorithms Sequential & Parallel: A Unified Approach
- Dr. Jones M. https://ubccr.freshdesk.com/support/solutions/articles/130000 26245-tutorials-and-training-documents
- Waghmare V. & Kulkarni D. (2010) Convex Hull Using K-Means Clustering in Hybrid (MPI/OpenMP) Environment.

Thank you.