Parallel K means clustering using MPI

Author: Gautam Shende

CSE 633: Parallel Algorithms

Instructor: Dr. Russ Miller

Date: 05/08/2018

OVERVIEW

1) CLUSTERING

CLUSTERING

1. Partitioning of data into subsets called clusters

2. Similar elements placed in same cluster. Similarity is calculated based on some distance metric such as euclidean distance or hamming distance.

3. Example :
   Dataset = {US, CHN, IN, CA}
   No of clusters = 2

   Cluster 1: US, CA
   Cluster 2: CHN, IN

2) K-Means

K-MEANS FOR CLUSTERING

1. Select k i.e. the number of clusters

2. Use any strategy* to select k points to be cluster centers.

3. Put each point in the data set in the cluster which has its center closest to the point

4. Calculate new cluster centers by taking means of all points in a cluster

5. Repeat 3 and 4 until convergence

EXAMPLE

- U = {1,6,10,18,3,14} , K=2
- ASSUME CLUSTER CENTERS TO BE C1 = 1, C2 = 6
- CLUSTER C1: {1,3}
  CLUSTER C2: {6,10,18,14}
- UPDATE CENTRE C1 = AVG {1,3} = 2
  UPDATE CENTRE C2 = AVG {6,10,18,14} = 12
- UPDATED CLUSTER C1: {1,3,6}
  UPDATED CLUSTER C2: {10,18,14}
- UPDATE CENTRE C1 = AVG {1,3,6} = 3.333
  UPDATE CENTRE C2 = AVG {10,18,14} = 14
- UPDATED CLUSTER C1: {1,3,6}
  UPDATED CLUSTER C2: {10,18,14}
- NO CHANGE IN CLUSTER CONFIGURATION (CONVERGENCE)
  -> STOP <-

TM

PARALLEL K-MEANS

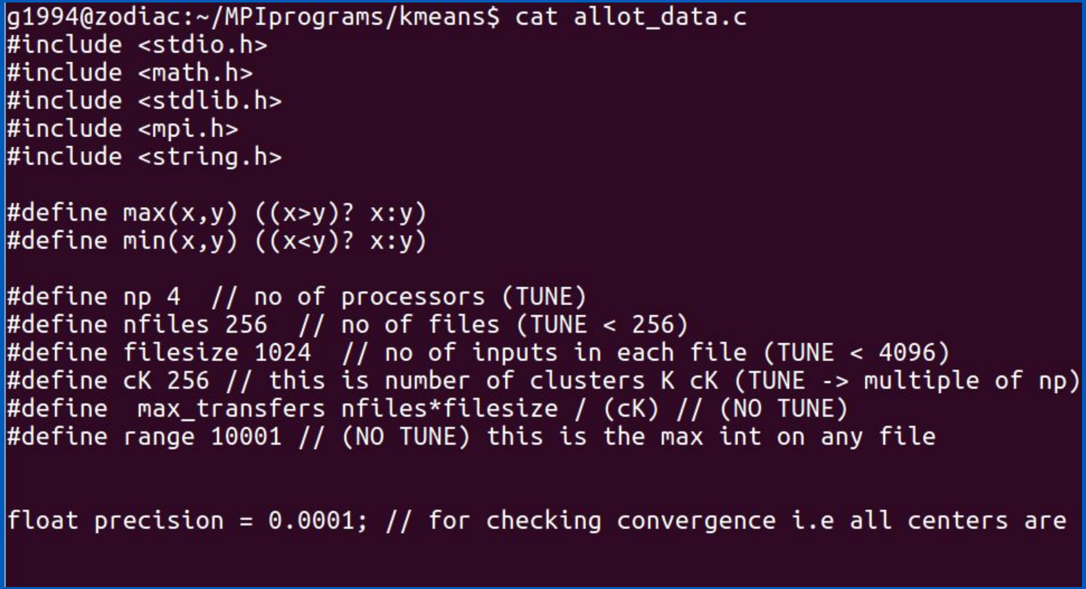
1. Allot k cluster centers to the nodes(n) equally such that each node is responsible for (k/n) clusters.

2. Now Each node does the following
   a. Calculate centers of (k/n) clusters by mean
   b. Broadcast (k/n) centers to all other nodes
   c. Receive (k/n) centers from every other node
   d. Calculate distance of all points from all centers and find closest cluster
   e. Send and receive points (internal and external transfers)

3. Repeat until Convergence (stopping condition)
   - No internal/external transfers
     <-> Centers remain constant

3) PARALLEL MODEL & FLOW

MODEL PARAMETERS



```
g1994@zodiac:~/MPIprograms/kmeans$ cat allot_data.c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <mpi.h>
#include <string.h>

#define max(x,y) ((x>y)? x:y)
#define min(x,y) ((x<y)? x:y)

#define np 4  // no of processors (TUNE)
#define nfiles 256  // no of files (TUNE < 256)
#define filesize 1024  // no of inputs in each file (TUNE < 4096)
#define cK 256 // this is number of clusters K cK (TUNE -> multiple of np)
#define  max_transfers nfiles*filesize / (cK) // (NO TUNE)
#define range 10001 // (NO TUNE) this is the max int on any file


float precision = 0.0001; // for checking convergence i.e all centers are
```

*local parameter = Max iterations (=300)

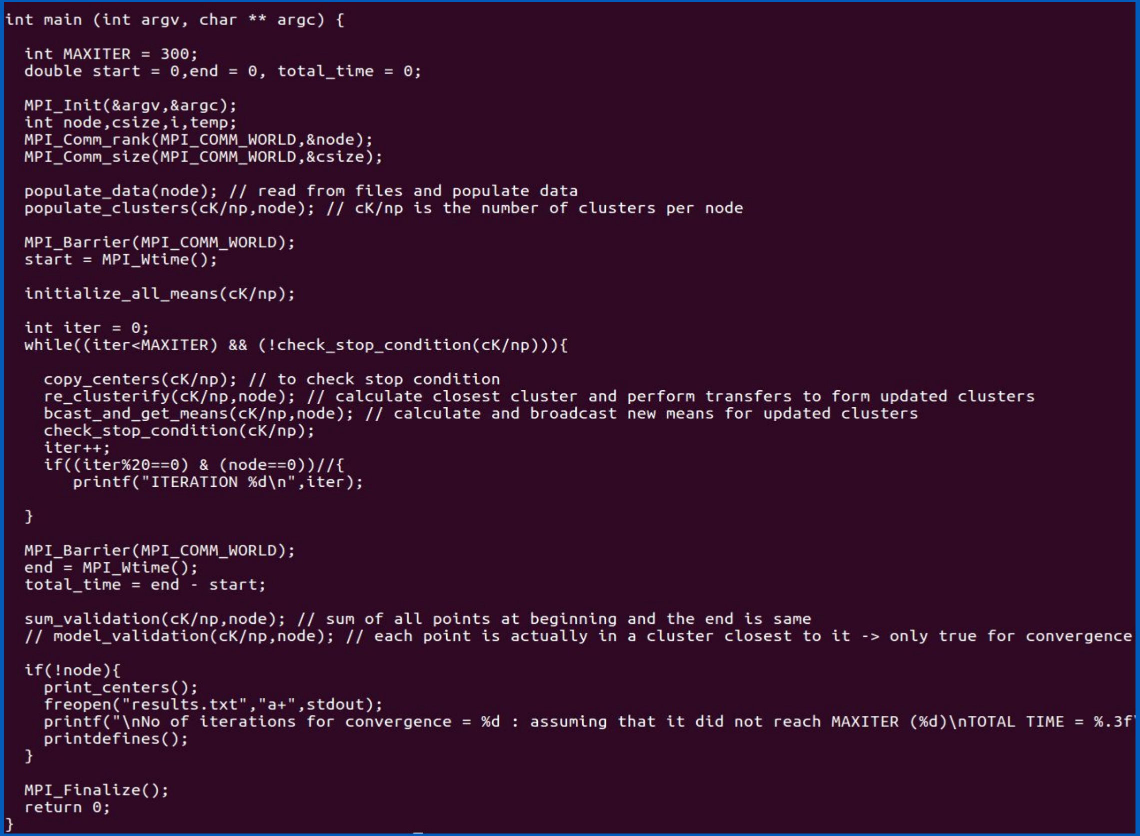
Complexity: O(input*K*iterations*dimensions)

= O(nfiles*filesize*cK*max_iterations*1)

= ~(256*1024*256*300) = ~2*10^10 =~20 billion calculations

Repository: https://github.com/thezodiac1994/Parallel-Alogrithms

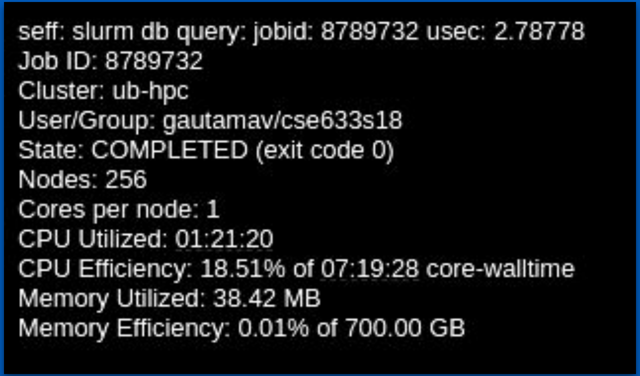
FLOW OF PARALLEL PROGRAM



```
int main (int argv, char ** argc) {

  int MAXITER = 300;
  double start = 0,end = 0, total_time = 0;

  MPI_Init(&argv,&argc);
  int node,csize,i,temp;
  MPI_Comm_rank(MPI_COMM_WORLD,&node);
  MPI_Comm_size(MPI_COMM_WORLD,&csize);

  populate_data(node); // read from files and populate data
  populate_clusters(cK/np,node); // cK/np is the number of clusters per node

  MPI_Barrier(MPI_COMM_WORLD);
  start = MPI_Wtime();

  initialize_all_means(cK/np);

  int iter = 0;
  while((iter<MAXITER) && (!check_stop_condition(cK/np))){

    copy_centers(cK/np); // to check stop condition
    re_clusterify(cK/np,node); // calculate closest cluster and perform transfers to form updated clusters
    bcast_and_get_means(cK/np,node); // calculate and broadcast new means for updated clusters
    check_stop_condition(cK/np);
    iter++;
    if((iter%20==0) & (node==0))//{
       printf("ITERATION %d\n",iter);

  }

  MPI_Barrier(MPI_COMM_WORLD);
  end = MPI_Wtime();
  total_time = end - start;

  sum_validation(cK/np,node); // sum of all points at beginning and the end is same
  // model_validation(cK/np,node); // each point is actually in a cluster closest to it -> only true for convergence

  if(!node){
    print_centers();
    freopen("results.txt","a+",stdout);
    printf("\nNo of iterations for convergence = %d : assuming that it did not reach MAXITER (%d)\nTOTAL TIME = %.3f
    printdefines();
  }

  MPI_Finalize();
  return 0;
}
```

4) RESULTS & INFERENCES

RESULTS & INFERENCES



```
seff: slurm db query: jobid: 8789732 usec: 2.78778
Job ID: 8789732
Cluster: ub-hpc
User/Group: gautamav/cse633s18
State: COMPLETED (exit code 0)
Nodes: 256
Cores per node: 1
CPU Utilized: 01:21:20
CPU Efficiency: 18.51% of 07:19:28 core-walltime
Memory Utilized: 38.42 MB
Memory Efficiency: 0.01% of 700.00 GB
```

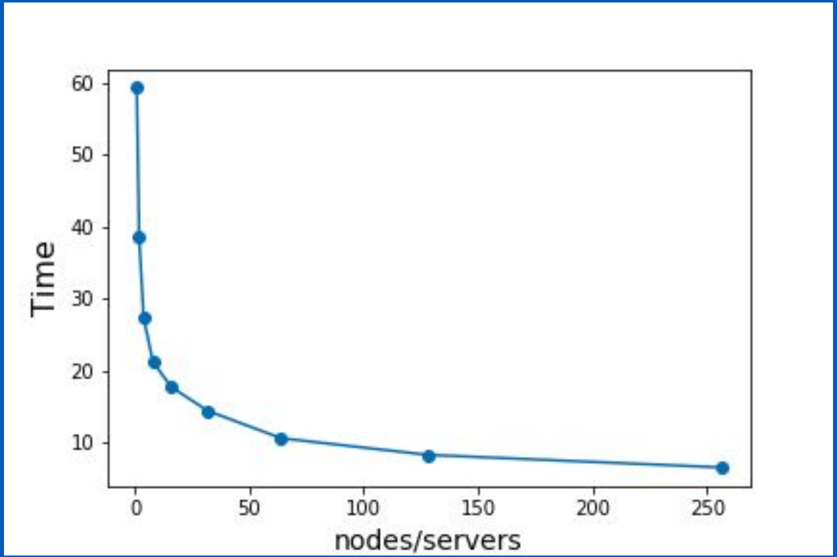
(First things first)

a) Nodes vs Time

cK = 256, inpsize = 262,144, iterations = 106 (convergence)

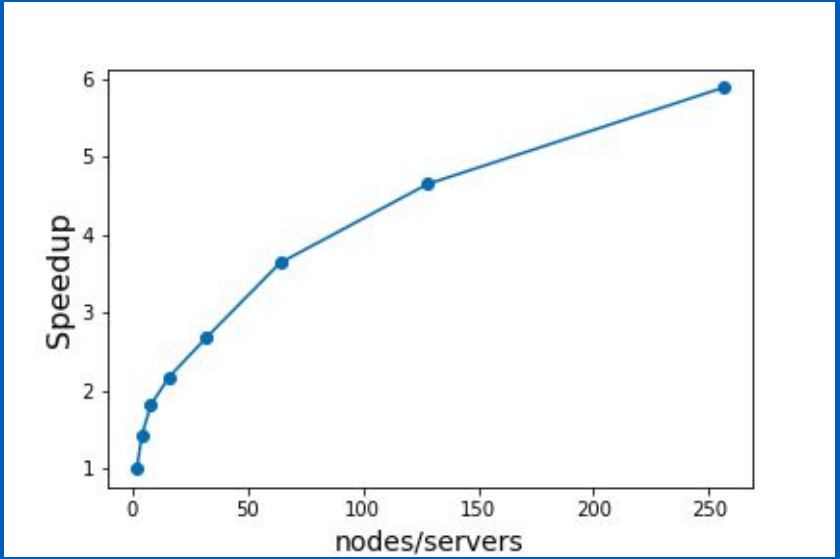
nodes	t0	t1	t2	t3	t4	avg
256	6.531	6.538	6.674	6.536	6.53	6.562
128	8.28	8.428	8.242	8.298	8.282	8.306
64	10.526	10.54	10.698	10.548	10.78	10.618
32	14.42	14.405	14.393	14.543	14.392	14.431
16	18.104	18.114	18.092	18.09	17.388	17.746
8	21.214	21.201	21.201	21.211	21.578	21.281
4	27.329	27.305	27.328	27.312	27.353	27.352
2	38.61	38.632	38.64	38.62	38.6	38.621
1	59.317	59.378	59.425	59.352	59.349	59.364

a) Nodes vs Time



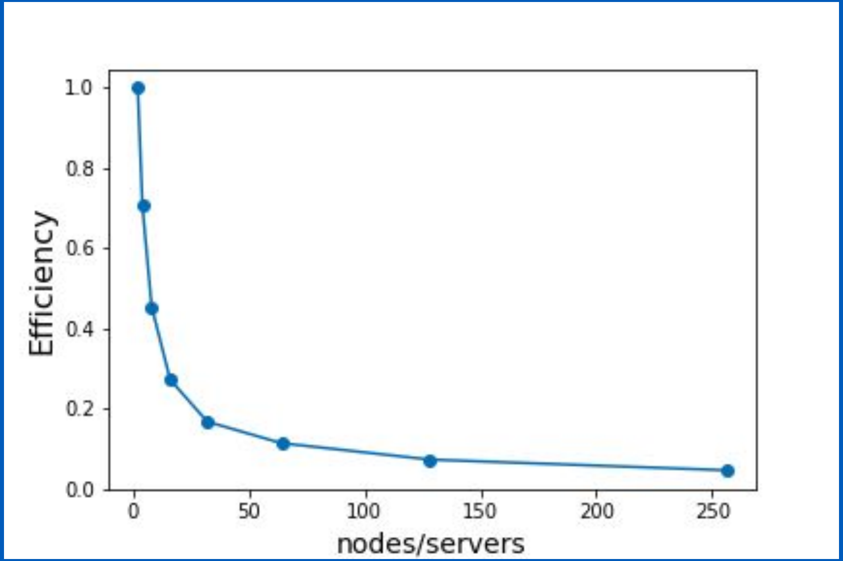
Nodes	1	2	4	8	16	32	64	128	256
Time(s)	59.364	38.621	27.352	21.281	17.746	14.431	10.618	8.306	6.562

b) SPEEDUP (starting n=2)



Nodes	2	4	8	16	32	64	128	256
Speedup	1.0	1.411	1.814	2.176	2.676	3.637	4.649	5.886

c) EFFICIENCY (starting n=2)



Nodes	2	4	8	16	32	64	128	256
Efficiency	1.0	0.705	0.453	0.272	0.167	0.114	0.073	0.046

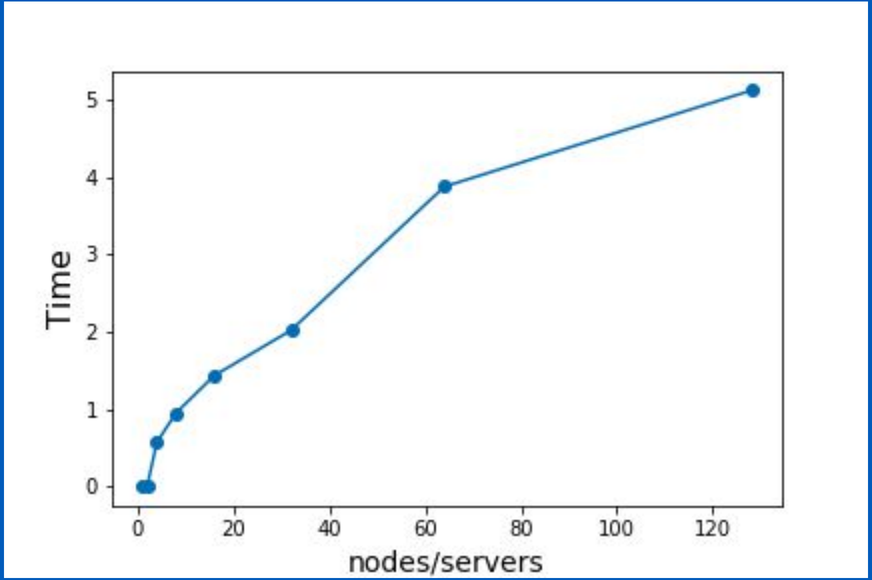
TM

d) Nodes vs Time
- increasing data with nodes

cK = 256

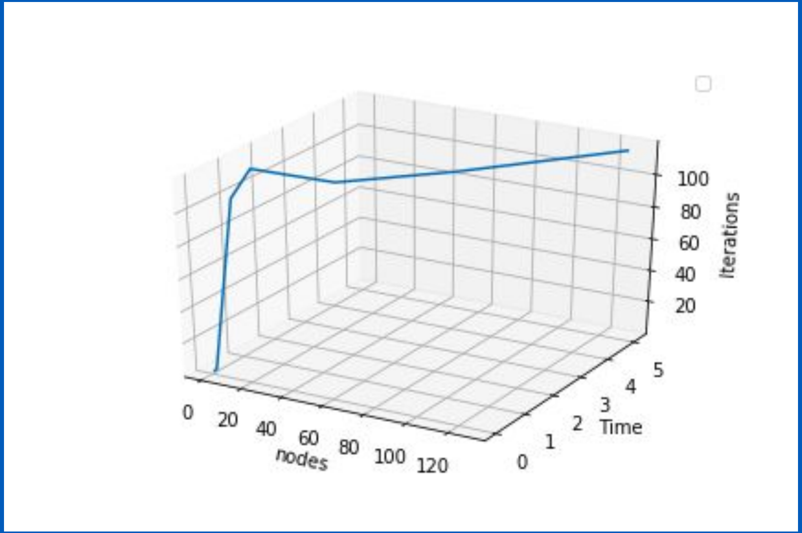
nodes	data	#iter	t0	t1	t2	t3	t4	avg
128	131072	116	5.173	5.198	5.211	5.089	5.064	5.1185
64	65536	101	3.879	3.878	3.88	3.88	3.874	3.8765
32	32768	105	2.025	2.019	2.024	2.025	2.017	2.021
16	16384	112	1.429	1.432	1.434	1.431	1.445	1.437
8	8192	118	0.965	0.97	0.966	0.969	0.941	0.953
4	4096	103	0.574	0.575	0.579	0.573	0.574	0.574
2	2048	2	0.011	0.011	0.011	0.011	0.01	0.0105
1	1024	1	0.009	0.005	0.007	0.008	0.008	0.0085

d) Nodes vs Time
- increasing data with nodes



Nodes	1	2	4	8	16	32	64	128
Time(s)	0.008	0.010	0.574	0.953	1.437	2.021	3.876	5.118
Data (x = 2^10)	x	2x	4x	8x	16x	32x	64x	128x

d) Nodes vs Time
- increasing data with nodes (considering iterations)



Nodes	1	2	4	8	16	32	64	128
Time	0.008	0.010	0.574	0.953	1.437	2.021	3.876	5.118
Data (x = 2^10)	x	2x	4x	8x	16x	32x	64x	128x
Iterations	1	2	103	118	112	105	101	116

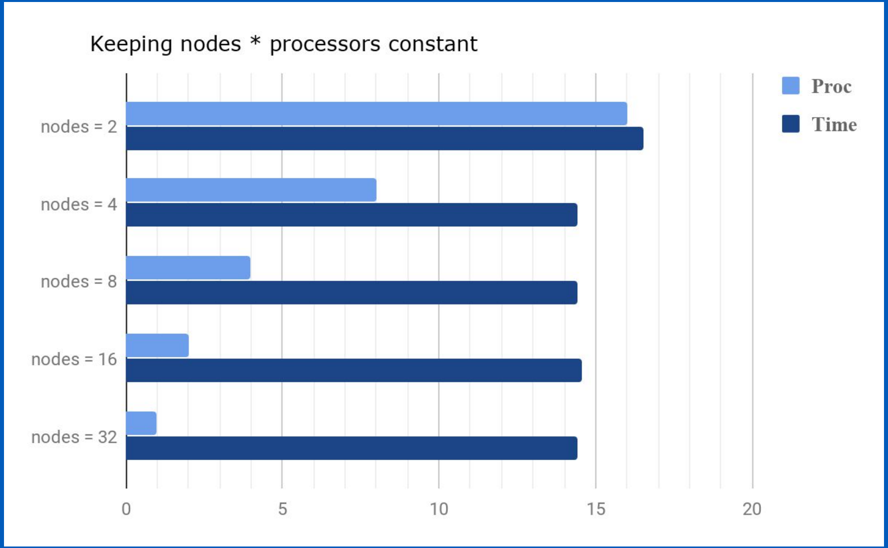
e) Varying cpus per node

cK = 256, inpsize = 262144, nodes*cpus = 32

nodes	cpus	t0	t1	t2	t3	avg
2	16	16.026	18.446	17.73	17.035	16.531
4	8	14.457	14.46	14.439	14.423	14.44
8	4	14.396	14.411	14.394	14.419	14.408
16	2	14.68	14.617	14.656	14.396	14.538
32	1	14.42	14.405	14.426	14.419	14.418

TM

e) Varying cpus per node



Nodes , cpus	2,16	4,8	8,4	16,2	32,1
Time(s)	16.531	14.44	14.408	14.538	14.418

f) Varying input size

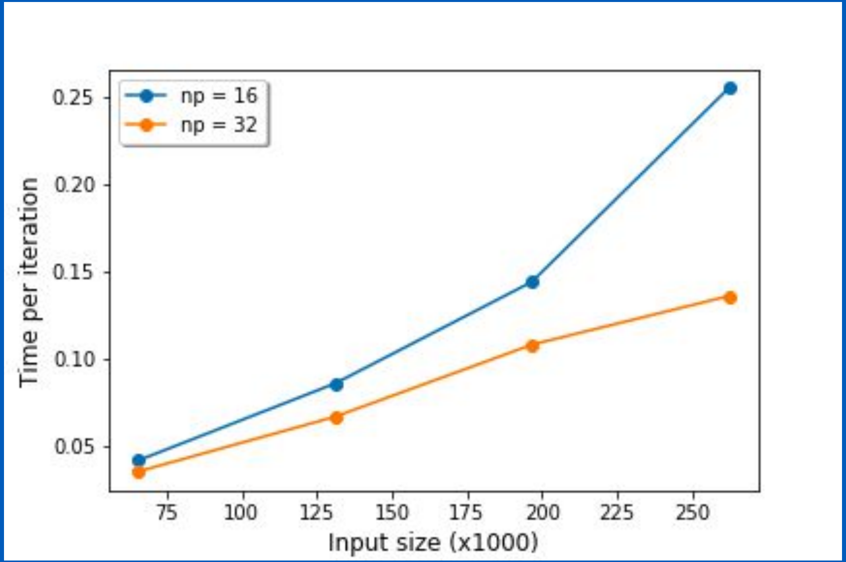
np = 32, cK = 256

inpsize	t0	t1	t2	t3	avg	iterations	avg/iter
4x = 262144	14.404	14.404	14.406	14.399	14.40325	106	0.136
3x = 196608	13.273	13.264	13.255	13.275	13.26675	123	0.108
2x = 131072	8.622	8.627	8.648	8.625	8.6305	130	0.066
x = 65536	4.494	4.491	4.497	4.497	4.49475	129	0.035

np = 16, cK = 256

inpsize	t0	t1	t2	t3	avg	iterations	avg/iter
4x = 262144	27.091	27.093	27.081	27.094	27.08975	106	0.256
3x = 196608	17.698	17.73	17.732	17.716	17.719	123	0.144
2x = 131072	11.121	11.12	11.123	11.118	11.1205	130	0.085
x = 65536	5.317	5.31	5.318	5.315	5.315	129	0.041

f) Varying input size



InpSize	x	2x	3x	4x
Time/Iteration (np = 16)	0.041	0.085	0.144	0.256
Time/Iteration (np = 32)	0.035	0.066	0.108	0.136

*time/iter so we don't have to consider the third variable/dimension (iterations) separately

TM

g) Varying cK

inpsize = 262144, nodes = 16, iterations = c

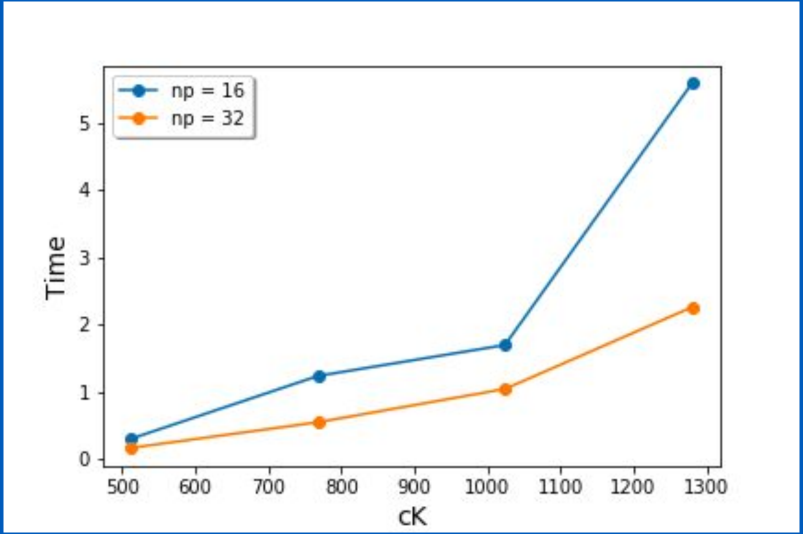
ck	t0	t1	t2	avg
512	0.303	0.306	0.301	0.302
768	1.237	1.234	1.238	1.2375
1024	1.694	1.701	1.701	1.6975
1280	5.583	5.585	5.599	5.591

inpsize = 262144, nodes = 32, iterations = c

ck	t0	t1	t2	avg
512	0.167	0.17	0.166	0.1665
768	0.547	0.548	0.55	0.5485
1024	1.043	1.044	1.044	1.0435
1280	2.258	2.26	2.263	2.2605

* Ran for low no of fixed iterations, because the motive was to see the effect of k and not reach convergence. Constant iterations eliminates the need to consider it explicitly as an extra variable/dimension affecting the graph

g) Varying cK



cK (x=256)	2x	3x	4x	5x
Time (np = 16)	0.302	1.237	1.697	5.591
Time (np = 32)	0.1665	0.5485	1.0435	2.2605

h) Inferences

1) For my model, 16-32 nodes are ideal from the point of view of efficiency and improvement in running times.

2) When keeping nodes * cpus = 32, the combination., there is not much deviation but nodes = 4, cpus = 8 was found to work best. Too many cpus on one node saturates it and too few increases communication cost.

3) Increasing cK has more impact on running time as in comparison to adding more uniformly distributed data.

4) It is important to also consider the number of iterations while scaling the model (input).

5) CHALLENGES

CHALLENGES

1) Getting access to servers
   - 5 days -> 256 nodes
   - 4 days -> 128 nodes
   - 2 days -> 64 nodes

2) Sequencing of Parallel events

3) Hyper tuning parameters
   - cK, nP, cores/node, input size, etc

4) Validation of the algorithm

6) FUTURE SCOPE

FUTURE SCOPE

1) Implement on a multidimensional dataset

2) Apply to a real world clustering problem

3) See how different variations of the algorithm perform in terms of time and number of iterations (like choosing a different distance metric, different strategy to initialise clusters, etc)

4) Implement a similar model on OpenMP and CUDA

7) REFERENCES

REFERENCES

1) Algorithms Sequential & Parallel: A Unified Approach (Dr. Russ Miller, Dr.Laurence Boxer)

2) https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-and-training-documents (Dr. Matthew Jones)

3) A Parallel K-Means Clustering Algorithm with MPI (Jing Zhang, Gongqing Wu, Xuegang Hu, Shiying Li, Shuilong Hao)

4) https://www.buffalo.edu/ccr/support/ccr-help.html (UB CCR help)

5) Stackoverflow (for general MPI questions)