CPSC 479 Project 2: Introduction to HPC - Data Science project

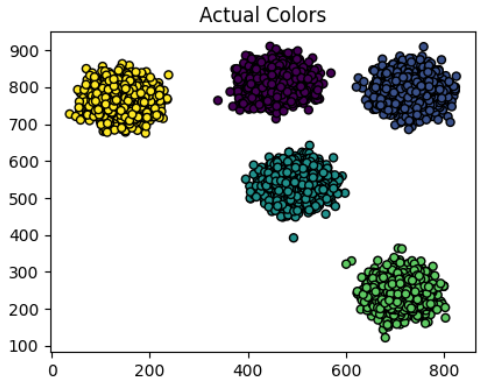
Prof. Doina Bein, CSU Fullerton

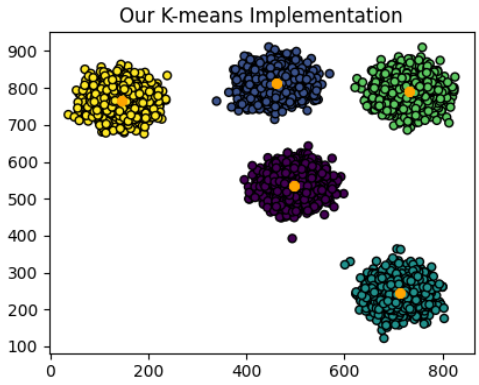
dbein@fullerton.edu

Summary

For our project we chose to do a parallel implementation of the k-means problem using Lloyd's heuristics. Implemented with Message passing interface (MPI) and C++. This file gives a problem statement, the contributors, pseudocode, instructions on how to run the code, and screenshots of the code running

Problem: To take a given dataset and separate these observations into a number of K clusters

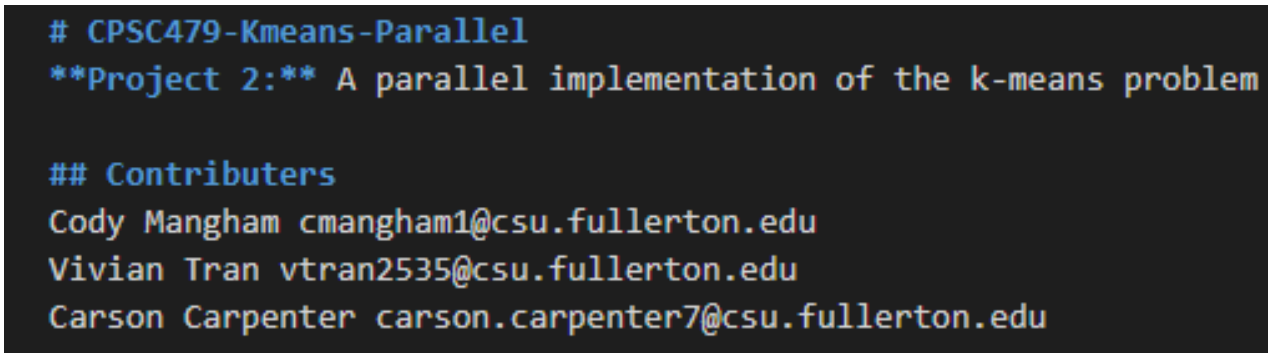




1. Contributors

Cody Mangham : cmangham1@csu.fullerton.edu
Vivian Tran: vtran2535@csu.fullerton.edu
Carson Carpenter: carson.carpenter7@csu.fullerton.edu

2. A full-screen screenshot with your group member names shown clearly. One way to make your names appear in Atom is to simply open your README.md.



3. Pseudocode

```
def Kmeans_Parallel():
      dim := the number of dimensions of data
      N := the size of the data set (number of points)
      K := the amount of clusters in the data
      DP := the intialized 2d (N x dim) array with all the points
      rank := number of current process 0 to size - 1
      size := number of processes
      MSE := ∞ // Mean square error
      centroids_sum := sum of points closest to centroid
      no_centroids  := number of points closest to centroid
      centroids := list of points in centroids K x dim
      if rank == 0:
            Initialize centroids to random points
            MPI_broadcast(centroids&, K)
      Centroids_sum_t := sum of points closest to centroid for rank
      no_centroids_t := number of points closest to centroid for rank
      MSE_t := 0 // mean square error for rank
      while MSE != previousMSE:
            previousMSE := MSE
            MSE_t = 0
            for i=0 to K - 1:
```

```
      centroids_sum_t[i] = 0
      no_centroids_t[i] = 0
for i = rank * (N/size) to (rank+1) * (N/size) - 1:
      l := 0
      min_dis := dist(cluster_points[l],DP[i])
      for j = 1 to K - 1:
            if dist(cluster_points[j],DP[i]) < min_dis:
                  min_dis = dist(cluster_points[j],DP[i])
                  l = j
      centroids_sum_t[l] = centroids_sum_t[l] + datapoints[i]
      no_centroids_t[l] = no_centroids_t[l] + 1
      MSE_t = MSE_t + min_dis
for j = 0 to K - 1:
      /* Find the average of each centroids */
      MPI_Allreduce(no_centroids_t[j], no_centroids[j],MPI_SUM)
      MPI_Allreduce(centroids_sum_t[j], centroids_sum[j],MPI_SUM)
      // Prevent divide by zero
      no_centroids[j] = min(no_centroids[j],1)
      centroids_sum[j] = centroids_sum[j] / no_centroids[j]
MPI_Allreduce(MSE_t, MSE,MPI_SUM)
```

4. A brief description on how to run the code.

Compile With: make
Run With: mpirun -np <process num> main
Or
mpirun --oversubscribe -np <process num> main

5. Two snapshots of code executing for some two distinct values of N.

N = 100000

```
cody@DESKTOP-QGVNI6R:project-2-data-science-cody-carson-vivian$ mpirun -np 5 ./main
722.808,2015.52
2668.98,2852.04
748.694,328.727
2927.83,1001.69
150.344,165.149
```

N = 1000000

```
cody@DESKTOP-QGVNI6R:project-2-data-science-cody-carson-vivian$ make
mpic++ -std=c++11 -c main.cpp
mpic++ -o main main.o
cody@DESKTOP-QGVNI6R:project-2-data-science-cody-carson-vivian$ mpirun --oversubscribe -np 20 ./main
140.696,23.8765
557.481,252.435
719.532,732.04
82.2256,274.368
751.08,767.443
```