OpenMPI – Clustering and DNA
Final Report

Yiming Zong (Jimmy)
Carnegie Mellon University
Pittsburgh, PA 15213
yzong@cmu.edu

Derek Tzeng
Carnegie Mellon University
Pittsburgh, PA 15213
dtzeng@andrew.cmu.edu

Abstract

This report reflects the final progress on the project *K-Means Clustering with OpenMPI* for 15-440, Fall 2014. It starts with an overview of K-Means Algorithm, followed by detailed descriptions of the algorithms applied to 2D data and on DNA strands, without and with parallelism. Then, it will cover the performance scalability of the parallel implementations, compared against the sequential versions as a baseline. Eventually, it will include building and testing instructions for the project.

1 Overview of K-Means Clustering

K-Means Clustering is an algorithm in the field of Machine Learning, which partitions n data points into k clusters, such that each data point is eventually associated to one of the k clusters that it is "nearest" to, and data points within a cluster should be "similar". The definition of "distance" between two elements in the sample universe can be customized depending on the dataset. Following is a basic outline of K-Means algorithm:

Algorithm 1 General Outline of Sequential K-Means Algorithm

1: **Input:** *Objects:* Data entries to classify; *N:* Number of data points; *K:* Number of target clusters.
2: **procedure** KMEANSCLUSTERING(Objects, N, K)
3: CentersOld $\leftarrow K$ random data entries
4: **repeat**
5: **for** i = 1 to N **do**
6: assoc$_i \leftarrow \arg\min_k$ ||Objects$_i$, CentersOld$_k$||
7: **for** k = 1 to K **do**
8: CentersNew$_k \leftarrow$ "mean" of data associated to CentersOld$_k$
9: Clusters_Converge $\leftarrow$ if CentersOld and CentersNew are close enough
10: CentersOld $\leftarrow$ CentersNew
11: **until** Clusters_Converge
12: **Output:** *assoc* holds final association of each element; *ClustersNew* holds value of each center.

Note that the outline above applies to any types of data, including data points in two-dimensional Cartesian plane ($\mathbb{R}^2$), and DNA strands data ($\mathbb{Z}_4^n$), both of which will be discussed in more details below. Note that in both cases, we mainly need to define the following:

- Distance measure between a data point and a cluster, i.e. $||\text{Objects}_i, \text{CentersOld}_k||$;
- Heuristic for finding the "mean" of data associated to a cluster; and
- Criteria for cluster convergence.

The following sections will describe how this algorithm can be applied to $\mathbb{R}^2$ and $\mathbb{Z}_4^n$ datasets. Also, since the calculations in *Line 5-6* above are independent of each other, the algorithm can be parallelized with OpenMPI framework to improve runtime. The details about message passing will also be covered.

2 K-Means Algorithm for 2D Dataset

For dataset of 2D data points ($\mathbb{R}^2$), the implementation uses *Cartesian Distance* between two points to measure the distance between a data point and a cluster, i.e.

$$||\vec{\mathbf{x}}, \vec{\mathbf{y}}|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

And, the "mean" of data points associated to some cluster is the "average" of the data vectors, i.e.

$$\vec{\text{Centers}}\text{New}_k = \frac{\sum_{i=1}^{N} \mathbb{1}_{\text{assoc}_i=k} \cdot \vec{\text{Objects}}_{\mathbf{i}}}{\sum_{i=1}^{N} \mathbb{1}_{\text{assoc}_i=k}}.$$

In the rare event that no data entries are associated to a cluster, instead of using the "average" (which is not defined), we set the new center to be a random data entry. As for the convergence criteria, the implementation allows the user to input a parameter $\varepsilon > 0$, such that when $||\vec{\text{Centers}}\text{Old}_k, \vec{\text{Centers}}\text{New}_k|| < \varepsilon$ for all $k \in \{0, 1, \cdots, K-1\}$, the algorithm terminates. Based on the specification above, the K-Means algorithm for 2D data points is as follows:

Algorithm 2 Sequential K-Means Algorithm on 2D Dataset

1: **Input:** *X, Y:* X- and Y- coordinates of dataset; *N:* Size of dataset; *K:* Number of clusters; ε: Convergence threshold.
2: **procedure** KMEANSSEQ2D(X, Y, N, K, ε)
3: $\mathcal{R}$[K] $\leftarrow$ indices for K random data entries
4: CenterXOld[K] $\leftarrow$ X-coordinates of data entries with indices in $\mathcal{R}$
5: CenterYOld[K] $\leftarrow$ Y-coordinates of data entries with indices in $\mathcal{R}$
6: **repeat**
7: XTotal[K], YTotal[K], count[K] $\leftarrow \vec{\mathbf{0}}$
8: **for** i = 0 to N-1 **do**
9: assoc$_i$ $\leftarrow \arg\min_k \sqrt{(\text{X}_i - \text{CenterXOld}_k)^2 + (\text{Y}_i - \text{CenterYOld}_k)^2}$
10: XTotal[assoc$_i$] $\leftarrow$ XTotal[assoc$_i$] + X$_i$
11: YTotal[assoc$_i$] $\leftarrow$ YTotal[assoc$_i$] + Y$_i$
12: count[assoc$_i$] $\leftarrow$ count[assoc$_i$] + 1
13: **for** k = 0 to K-1 **do**
14: **if** count$_k$ **not** 0 **then**
15: CenterXNew$_k$ $\leftarrow$ XTotal$_k$/count$_k$
16: CenterYNew$_k$ $\leftarrow$ YTotal$_k$/count$_k$
17: **else**
18: (CenterXNew$_k$,CenterYNew$_k$) $\leftarrow$ random data point *//When association is empty, reset center.*
19: Convergence $\leftarrow \sqrt{(\text{CenterXNew}_k - \text{CenterXOld}_k)^2 + (\text{CenterYNew}_k - \text{CenterYOld}_k)^2} < \varepsilon, \forall k < K$
20: CenterXOld $\leftarrow$ CenterXNew
21: CenterYOld $\leftarrow$ CenterYNew
22: **until** Convergence
23: **Output:** *assoc* holds final association of each element; *CenterXNew* and *CenterYNew* hold location of each center.

As mentioned in *Section One*, the algorithm can be parallelized with OpenMPI, especially *Line 8-12* above. In the implementation, there is a *Master Process (ID=0)* that specializes in summarizing results from the *Worker Processes (ID>0)*, and each *Worker Process* is allocated a range of data points to associate to one cluster each.

Whenever Master receives a *XTotal*, *YTotal*, or *count* array from a worker, it simply performs piecewise addition with the local version at Master. As for a portion of *assoc* array from Worker, Master needs to "plug" it into the correct section of *assoc* in the local version at Master, depending on the sender's ID. The corresponding psudocode is shown in *Algorithm 3* as follows:

Algorithm 3 Parallel K-Means Algorithm on 2D Dataset

1: **Input:** *X, Y:* X- and Y- coordinates of dataset; *N:* Size of dataset; *K:* Number of clusters; ε: Convergence threshold.
2: **procedure** KMEANSPARAL2D(X, Y, N, K, ε)
3: **all**: load input data, including X, Y, N, K, ε
4: **if self** is **master then**
5: $\mathcal{R}$[K] $\leftarrow$ indices for K random data entries

CenterXOld[K] ← X-coordinates of data entries with indices in $\mathcal{R}$
CenterYOld[K] ← Y-coordinates of data entries with indices in $\mathcal{R}$
repeat
XTotal[K], YTotal[K], count[K] ← $\vec{\mathbf{0}}$
if self is **master then**
Send CenterXOld and CenterYOld to all Workers.
else
Receive CenterXOld and CenterYOld from Master.
if self is **worker then**
for i **in** workerRange **do**
assoc$_i$ ← $\arg\min_k \sqrt{(\text{X}_i - \text{CenterXOld}_k)^2 + (\text{Y}_i - \text{CenterYOld}_k)^2}$
XTotal[assoc$_i$] ← XTotal[assoc$_i$] + X$_i$
YTotal[assoc$_i$] ← YTotal[assoc$_i$] + Y$_i$
count[assoc$_i$] ← count[assoc$_i$] + 1
Send XTotal[], YTotal[], count[], and relevant portion of assoc[] to Master, asynchronously
Wait for Convergence Result from Master. **break** if convergent
else
Wait till XTotal[], YTotal[], assoc[], and count[] arrive from all Workers. Merge arrays on the way.
for k = 0 to K-1 **do**
if count$_k$ **not** 0 **then**
CenterXNew$_k$ ← XTotal$_k$/count$_k$
CenterYNew$_k$ ← YTotal$_k$/count$_k$
else
(CenterXNew$_k$,CenterYNew$_k$) ← random data point *//When association is empty, reset center.*
Convergence ← $\sqrt{(\text{CenterXNew}_k - \text{CenterXOld}_k)^2 + (\text{CenterYNew}_k - \text{CenterYOld}_k)^2} < \varepsilon, \forall k < K$
Sends Convergence Result to all Workers.
CenterXOld ← CenterXNew
CenterYOld ← CenterYNew
until Convergence

Output: *assoc* holds final association of each element; *CenterXNew* and *CenterYNew* hold location of each center.

3 K-Means Algorithm for DNA Dataset

For dataset of DNA data points ($\mathbb{Z}_4^n$), the implementation uses the *Similarity Function* between two strings. Since it's given that all DNA strands have equal length, the *Similarity Function* can be calculated as:

$$F(s_1, s_2) := \text{number of indices } i \text{ where } s_1^{(i)} \text{ and } s_2^{(i)} \text{ are identical.}$$

We define the center of each cluster to also be a DNA strand, so the "similarity" between a data point and a center can be calculated with the *Similarity Function* above. And, the "mean" of data points associated to some cluster can be obtained "bit by bit", by taking the *mode* (most frequent type) of the bits at a certain index for all strands that belong to a cluster. In the rare event that no data entries are associated to a cluster, instead of using the "mean," we set the new center to be a random data entry. As for the convergence criteria, the implementation allows the user to input a parameter $\varepsilon \geq 1$, such that when $\mathcal{L} - F(\text{CentersNew}_k, \text{CentersOld}_k) < \varepsilon$ for all $k \in \{0, 1, \cdots, K-1\}$, the algorithm terminates ($\mathcal{L}$ is the common length of all DNA strands).

Based on the specification above, the sequential K-Means algorithm for DNA strands is as follows. In the psudocode below, note that ***Total* is an abbreviated notation for "any of ATotal, CTotal, GTotal, TTotal."

Algorithm 4 Sequential K-Means Algorithm on DNA Dataset

Input: *DNAs:* DNA strands; *N:* Size of input; *L:* Length of each strand; *K:* Number of clusters; ε: Convergence threshold.
procedure KMEANSSEQDNA(DNAs, N, L, K, ε)
CenterOld[K] ← K random data entries from DNAs[]
repeat
ATotal[K][L], CTotal[K][L], GTotal[K][L], TTotal[K][L], count[K] ← $\vec{\mathbf{0}}$
for i = 0 to N-1 **do**

```
7:            assoc_i ← arg max_k F(DNAs_i, CenterOld_k)
8:            count[assoc_i] ← count[assoc_i] + 1
9:            Increment *Total[assoc_i][l] if DNAs_i^(l) = *; repeat for each l ∈ {0, ··· , L − 1}
10:       for k = 0 to K-1 do
11:           if count_k not 0 then
12:               CenterNew_k^(l) ← * if *Total[k][l] is max among *Total[k][l]'s; repeat for each l ∈ {0, ··· , L − 1}
13:           else
14:               CenterNew_k ← random data point    //When association is empty, reset center.
15:       Convergence ← L − F(CenterNew_k, CenterOld_k) < ε, ∀k < K
16:       CenterOld ← CenterNew
17:   until Convergence
18: Output: assoc holds final association of each DNA; CenterNew holds value of each center.
```

Similar to previous section, the algorithm can be parallelized with OpenMPI, especially *Line 6-9* above. In the implementation, there is a *Master Process (ID=0)* that specializes in summarizing results from the *Worker Processes (ID>0)*, and each *Worker Process* is allocated a range of data points to associate to one cluster each.

Whenever Master receives a **Total* or *count* array from a worker, it simply performs piecewise addition with the local version at Master. As for a portion of *assoc* array from Worker, Master needs to "plug" it into the correct section of *assoc* in the local version at Master, depending on the sender's ID. The corresponding psudocode is shown in *Algorithm 5* as follows:

Algorithm 5 Parallel K-Means Algorithm on DNA Dataset

```
1: Input: DNAs: DNA strands; N: Size of input; L: Length of each strand; K: Number of clusters; ε: Convergence threshold.
2: procedure KMEANSPARALDNA(DNAs, N, L, K, ε)
3:     all: load input data, including DNAs, N, L, K, ε
4:     if self is master then
5:         CenterOld[K] ← K random data entries from DNAs[]
6:     repeat
7:         ATotal[K][L], CTotal[K][L], GTotal[K][L], TTotal[K][L], count[K] ← 0⃗
8:         if self is master then
9:             Send CenterOld to all Workers.
10:        else
11:            Receive CenterOld from Master.
12:        if self is worker then
13:            for i in workerRange do
14:                assoc_i ← arg max_k F(DNAs_i, CenterOld_k)
15:                count[assoc_i] ← count[assoc_i] + 1
16:                Increment *Total[assoc_i][l] if DNAs_i^(l) = *; repeat for each l ∈ {0, ··· , L − 1}
17:            Send *Total[], count[], and relevant portion of assoc[] to Master, asynchronously
18:            Wait for Convergence Result from Master. break if convergent
19:        else
20:            Wait till *Total[], assoc[], and count[] arrive from all Workers. Merge arrays on the way.
21:            for k = 0 to K-1 do
22:                if count_k not 0 then
23:                    CenterNew_k^(l) ← * if *Total[k][l] is max among *Total[k][l]'s; repeat for each l ∈ {0, ··· , L−1}
24:                else
25:                    CenterNew_k ← random data point    //When association is empty, reset center.
26:            Convergence ← L − F(CenterNew_k, CenterOld_k) < ε, ∀k < K
27:            Sends Convergence Result to all Workers.
28:            CenterOld ← CenterNew
29:    until Convergence
30: Output: assoc holds final association of each DNA; CenterNew holds value of each center.
```

4 Performance Analysis for K-Means on 2D Dataset

The following section analyzes the performance of K-Means algorithm on 2D dataset, with and without parallelism. For consistency, the runtimes listed below are all average of 20 runs on `ghc30` (for sequential version), or `ghc50` and `ghc52` (for OpenMPI-parallel version). The size of dataset for both cases are 150 000, with 10 000 belonging to each cluster. *Degrees of Parallelism* below mean the number of processes participating in OpenMPI framework, *including* the Master process.

The runtimes of K-Means on 2D Dataset without or with different degrees of parallelism are listed as follows (run with default parameters as in `./runSeq2D.sh` and `./runParal2D.sh`):

Degree of Parallelism:	Sequential	2	4	8	12	16	20	24
Overall Runtime (s):	24.997	34.122	14.393	13.270	10.839	9.498	11.485	11.735
Total Process Time (s):	24.997	68.244	57.572	106.16	130.068	151.966	229.694	281.634
Total time / Sequenial Time:	1.000	2.730	2.303	4.247	5.203	6.079	9.189	11.267

Graphical presentation of the benchmark is as follows:

Figure 1: Plot of Overall Runtime & Total Process Time against Sequential Time

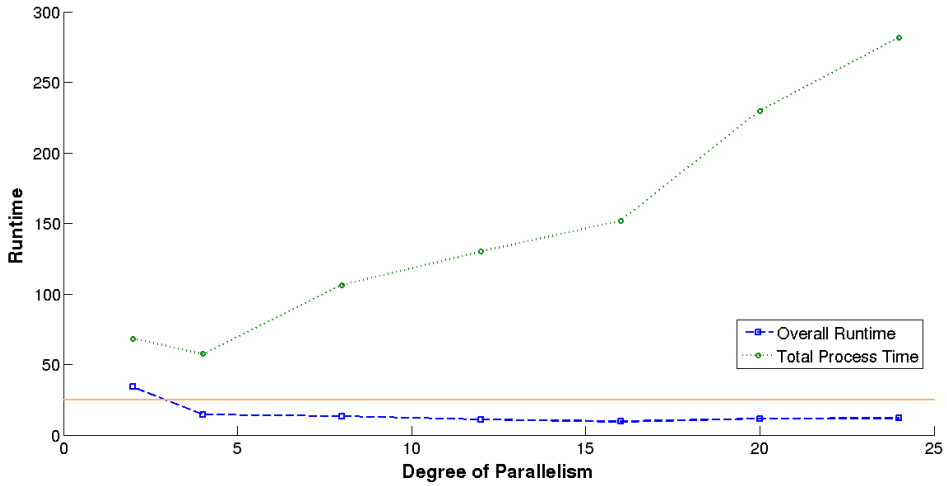
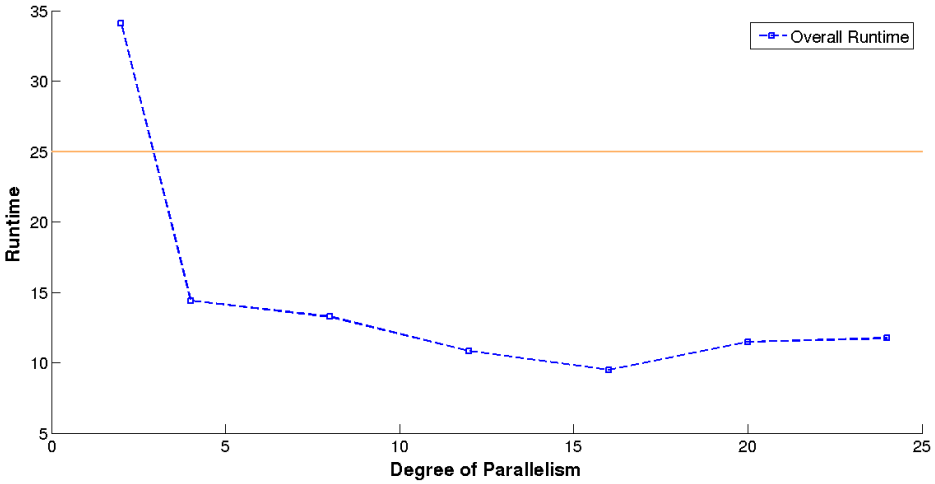


Figure 2: Plot of Overall Runtime against Sequential Time



According to the results above, following are the key observations and their interpretations:

- *Overall Runtime* reaches its minimum when *degree of parallelism* is around 16, where the runtime of parallel algorithm is merely 38% of the sequential one;
- While the *degree of parallelism* increases from 12 to 24, overall runtime of the algorithm does not decrease much – this means the increased overhead of OpenMPI is roughly equal to the speed-up from parallelism during that range;
- When *degree of parallelism* increases over 16, an increase in overall runtime is observed, which can be explained by the increased overhead of MPI that exceeds the benefit of parallelism, and by the saturation of actual physical machines' processing power;
- In general, as the degree of parallelism increases, the *total process time* also increases due to MPI overhead. And, the rather unexpected decrease of total process time when degree of parallelism increases from 2 to 4 is most likely due to the waiting time that Master thread wastes without doing any calculation.
- Overall, from *Figure One*, it is clearly shown that while OpenMPI enables parallelism acrosses processes and machines, it does introduce a non-trivial amount of communication overhead, which needs to be managed carefully by application developer, as the green dotted *Total Process Time* appears far above the organge reference line that represents the runtime for sequential K-Means.

5 Building & Testing Instructions

To re-build the project, simply `cd` into the project directory and run `./rebuildAll.sh`, which will remove existing class files and rebuild them from source. Note that `mpijavac` needs to be installed on the building machine.

Sample valid input files for 2D dataset and DNA dataset are located at `./input/rand2D.csv` and `./input/randDNA.csv` respectively. And, dataset generation scripts are located in `utils/` directory. To use them, first `cd ./utils`, and then `./gen2D.sh` would generate 2D dataset to be passed in K-Means. Similarly, `./genDNA.sh` would generate DNA dataset. Note that the output files would be written to `../input/` of the current directory, so make sure to `cd` into `./utils/` *before* running the bash scripts. Also, *generating random DNA dataset might take a long time – generating 150000 data entries in 15 clusters takes around 30 minutes*.

After the previous two steps, the sequential and parallel K-Means algorithms can be run by running one of the `./run*.sh` bash scripts in the project root directory. Default parameters that work with the provided datasets have been provided in the bash scripts. Feel free to modify them if necessary.

The output of the alrogithm will be written to the same directory as input, whereas the output file name has `.out` appended to the original input file name. Since the output can be long depending on the dataset, it is recommended to open it with `less -N` instead of with conventional text editors.