Marcus Chong

Technical Write-Up

I. Introduction

K-means clustering algorithm (Lloyd's Algorithm) is a method of vector quantization that is popular for cluster analysis in data mining. It aims to partition *n* objects into *k* clusters in which each object belongs to the cluster to the nearest mean, serving as a "prototype" of the cluster. Given a set of objects (x_1, x_2, …, x_n), where each object is a d-dimensional real vector, k-means clustering aims to partition the n points into k ($\leq n$) sets S = {S_1, S_2, …, S_k} to minimize the within-cluster sum of squares.

II. Algorithm

This parallel implementation of K-Means algorithm using OpenMP was developed by Northwestern University. The executable created by the program read input files that stores the data objects to be clustered. Northwestern University provided several example files in the Image_data sub-directory. The input files can be in ASCII text, raw binary, or netCDF. This program does the following:

1. Reads the points from the file (ex. Colors100.txt).
2. The program then picks the first *k* objects as the initial cluster centers.
3. The program finds the nearest cluster for data object. If the object's membership changes, the data object increments delta (% of objects that change their clusters) by 1.
4. It then averages the centroids of the new cluster using the objects inside the cluster.
5. Finally, the program checks whether the ratio between the delta and the number of data objects is greater than the threshold. If yes, the program repeats itself (repeat step 2)
6. If no, the program output the cluster results

III. OpenMP

OpenMP in this program was mainly used in omp_kmeans.c file. In this file, many environment variables were parallelized with their clauses of *private, firstprivate, shared, schedule, and reduction*. *Private* specifies that each thread should have its own instance of a variable. *Firstprivate* specifies that each thread should have its own instance of a variable, and that the variable should be initialized with the value of the variable, because it exists before the parallel construct. *Shared* specifies that one or more variables should be shared among all threads. *Schedule* dives the loop into equal-sized chunks or as equal as possible in case where the number loop iterations is not evenly divisible by number of threads multiplied by the chunk size. *Reduction* specifies that one or more variables that are private to each thread are the subject of a reduction operation at the end of the parallel region.

IV. Compiling and Execution

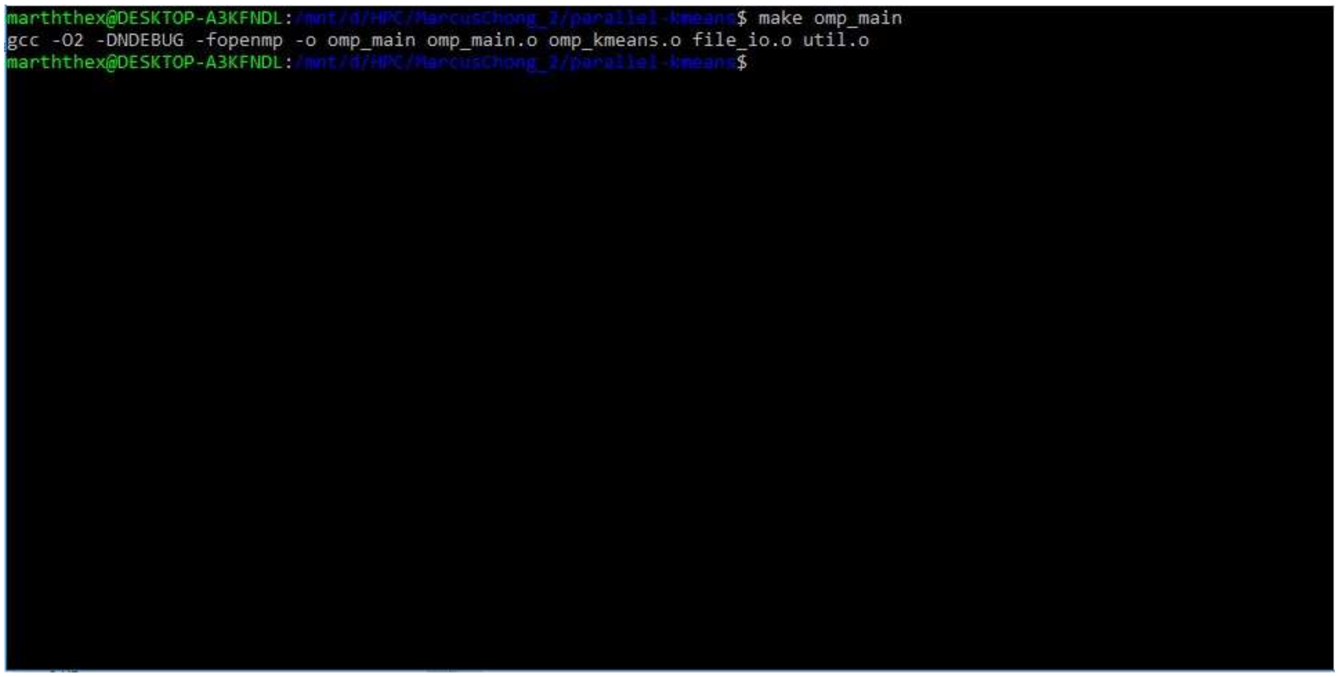
Northwestern University provided a Makefile to make it easier for the user to compile the program. Specifically, the makefile produces an executable (omp_main) for OpenMP. They also provided a README file which provided instructions for how to use the make file and how to run the program. The program provides a list of available command-line arguments that can be obtained by running the -h option. The OpenMP executable prints out the following:

```
Usage: omp_main [switches] -i filename -n num_clusters
        -i filename    : file containing data to be clustered
        -c centers     : file containing initial centers. default: filename
        -b             : input file is in binary format (default no)
        -n num_clusters: number of clusters (K must > 1)
        -t threshold   : threshold value (default 0.0010)
```

```
-p nproc     : number of threads (default system allocated)
-a           : perform atomic OpenMP pragma (default no)
-o           : output timing results (default no)
-v var_name  : using PnetCDF for file input and output and var_name
             : is variable name in the netCDF file to be clustered
-d           : enable debug mode
```

V. Results

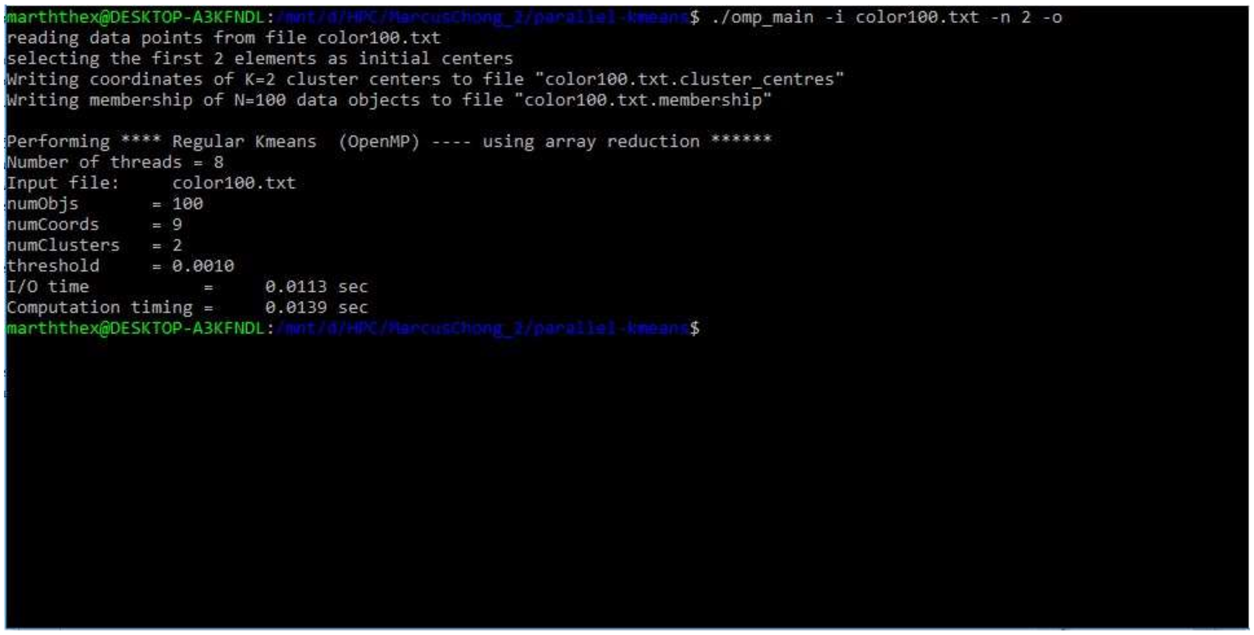
For my test run, I compiled the program by using Northwest University "easy-to-use" makefile (Fig 1). The makefile then executed gcc and the necessary flags to compile the program without any errors.



```
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$ make omp_main
gcc -O2 -DNDEBUG -fopenmp -o omp_main omp_main.o omp_kmeans.o file_io.o util.o
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$
```

Fig 1: Compiling the C Program using the provided makefile

I wanted to compare the speed of the k-means algorithm parallelized version to that of the sequential version to see difference in efficiency and performance. I first ran the program in sequential with the following command in Fig 2.

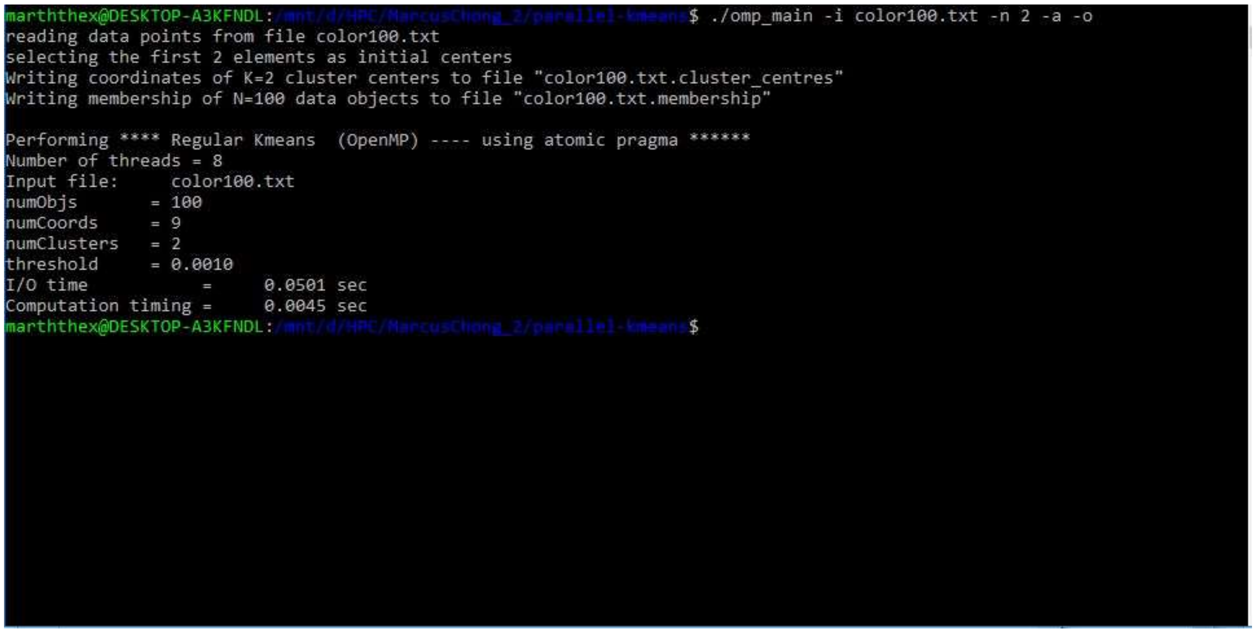


```
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$ ./omp_main -i color100.txt -n 2 -o
reading data points from file color100.txt
selecting the first 2 elements as initial centers
Writing coordinates of K=2 cluster centers to file "color100.txt.cluster_centres"
Writing membership of N=100 data objects to file "color100.txt.membership"

Performing **** Regular Kmeans  (OpenMP) ---- using array reduction ******
Number of threads = 8
Input file:     color100.txt
numObjs       = 100
numCoords     = 9
numClusters   = 2
threshold     = 0.0010
I/O time           =     0.0113 sec
Computation timing =     0.0139 sec
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$
```

Fig 2: Executing the file by using array reduction (Sequential)

I then ran the program using parallel by performing the same command along with *-a* in the command line (performs atomic OpenMP pragma). The result is shown in Fig 3.



```
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$ ./omp_main -i color100.txt -n 2 -a -o
reading data points from file color100.txt
selecting the first 2 elements as initial centers
Writing coordinates of K=2 cluster centers to file "color100.txt.cluster_centres"
Writing membership of N=100 data objects to file "color100.txt.membership"

Performing **** Regular Kmeans  (OpenMP) ---- using atomic pragma ******
Number of threads = 8
Input file:     color100.txt
numObjs       = 100
numCoords     = 9
numClusters   = 2
threshold     = 0.0010
I/O time           =     0.0501 sec
Computation timing =     0.0045 sec
marththex@DESKTOP-A3KFNDL:/mnt/d/HPC/MarcusChong_2/parallel-kmeans$
```

Fig 3: Executing the file by using atomic pragma (Parallel)

As you can see in this example, the parallel algorithm is approximately **three times** faster than the sequential algorithm.

The program outputs a coordinates of cluster centers file and a membership file. The Cluter Centers File (.center_centres) contains the coordinates for the cluster centers. The Membership files (.membership) identifies which point belongs to which cluster.

VI. Challenges

Challenges include limitations of data types and substantial number of data points. This program only uses C float data type for all coordinates and other read numbers. The program will not run with any other data type. Data objects also cannot exceed 2 GB due to the 4-byte integers used in the program. Changes in the program's code will be needed to accommodate data object files that exceed 2 GB.

VII. References

1. **https://en.wikipedia.org/wiki/K-means_clustering**
2. **http://bobweigel.net/wiki/images/Tboggs_project_report_csi702.pdf**
3. **http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html**