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 implementation of K-Means algorithm using CUDA 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. CUDA

CUDA in this program was mainly used in cuda_kmeans.cu and cuda_main.cu files. This program creates kernels for Euclid distance, nearest cluster, and computing delta. The Euclid distance kernel takes coordinates, objects, clusters, object id, and cluster id as arguments and returns the Euclid distance between the two multi-dimensional points. The find nearest cluster kernel takes coordinates, objects, clusters, membership, and intermediates as arguments and computes the nearest cluster. The compute delta kernel takes in the number of intermediates from two objects and computes the delta value. Each of these kernels create a single grid consisting of blocks that can run a maximum of 1024 threads each.

In the cuda_main.cu file, pointers for membership (thread ID), objects, and clusters were made as arguments, so that the three kernels can be created. After computing the data on the GPU and transferring the results back to the CPU, the pointers needs to be freed up to prevent wasteful memory consumption.

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 (cuda_main) for CUDA. 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 executable (without any flags). The CUDA executable prints out the following:

```
Usage: omp_main [switches] -i filename -n num_clusters
         -i filename    : file containing data to be clustered
```

-b : input file is in binary format (default no)

-n num_clusters: number of clusters (K must > 1)

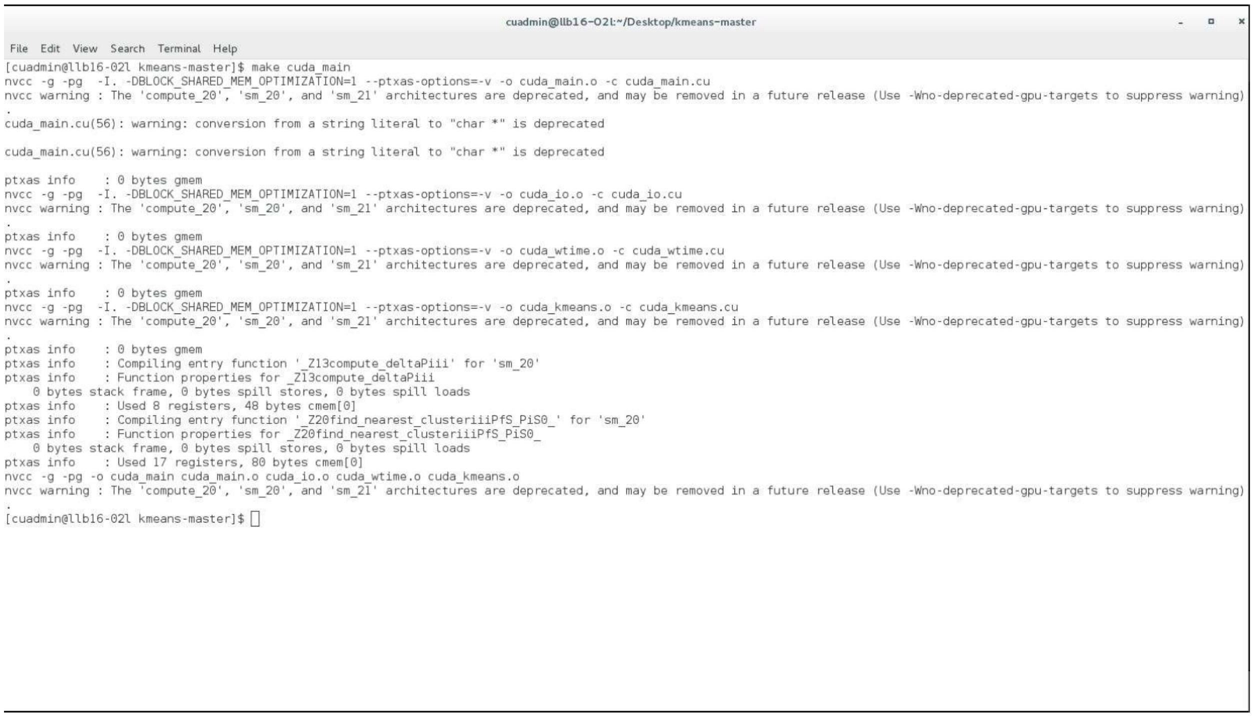
-t threshold : threshold value (default 0.0010)

-o : output timing results (default no)

-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 nvcc and the necessary flags to compile the program without any critical errors.



cuadmin@llb16-02l:~/Desktop/kmeans-master

File Edit View Search Terminal Help

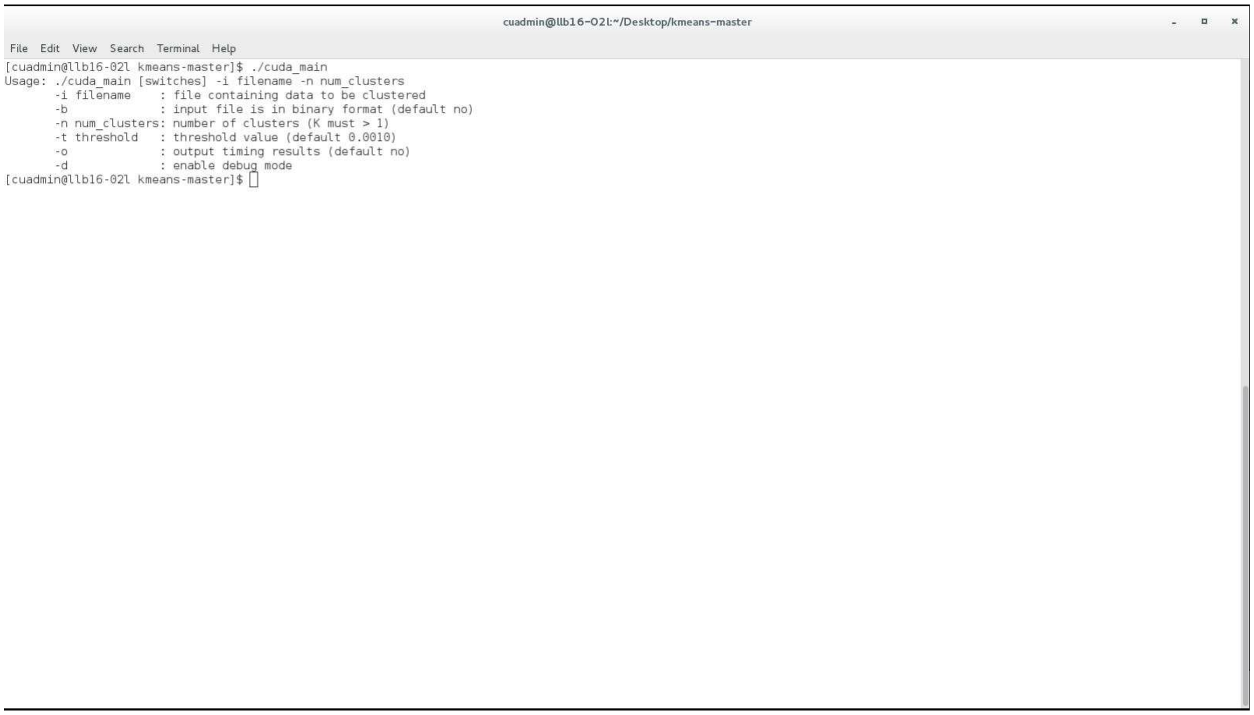
```
[cuadmin@llb16-02l kmeans-master]$ make cuda_main
nvcc -g -pg  -I. -DBLOCK_SHARED_MEM_OPTIMIZATION=1 --ptxas-options=-v -o cuda_main.o -c cuda_main.cu
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning)
.
cuda_main.cu(56): warning: conversion from a string literal to "char *" is deprecated

cuda_main.cu(56): warning: conversion from a string literal to "char *" is deprecated

ptxas info    : 0 bytes gmem
nvcc -g -pg  -I. -DBLOCK_SHARED_MEM_OPTIMIZATION=1 --ptxas-options=-v -o cuda_io.o -c cuda_io.cu
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning)
.
ptxas info    : 0 bytes gmem
nvcc -g -pg  -I. -DBLOCK_SHARED_MEM_OPTIMIZATION=1 --ptxas-options=-v -o cuda_wtime.o -c cuda_wtime.cu
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning)
.
ptxas info    : 0 bytes gmem
nvcc -g -pg  -I. -DBLOCK_SHARED_MEM_OPTIMIZATION=1 --ptxas-options=-v -o cuda_kmeans.o -c cuda_kmeans.cu
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning)
.
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z13compute_deltaPiii' for 'sm_20'
ptxas info    : Function properties for _Z13compute_deltaPiii
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 8 registers, 48 bytes cmem[0]
ptxas info    : Compiling entry function '_Z20find_nearest_clusteriiiPfS_PiS0_' for 'sm_20'
ptxas info    : Function properties for _Z20find_nearest_clusteriiiPfS_PiS0_
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 17 registers, 80 bytes cmem[0]
nvcc -g -pg -o cuda_main cuda_main.o cuda_io.o cuda_wtime.o cuda_kmeans.o
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning)
.
[cuadmin@llb16-02l kmeans-master]$ 
```

Fig 1: Compiling the CU Program using the provided makefile

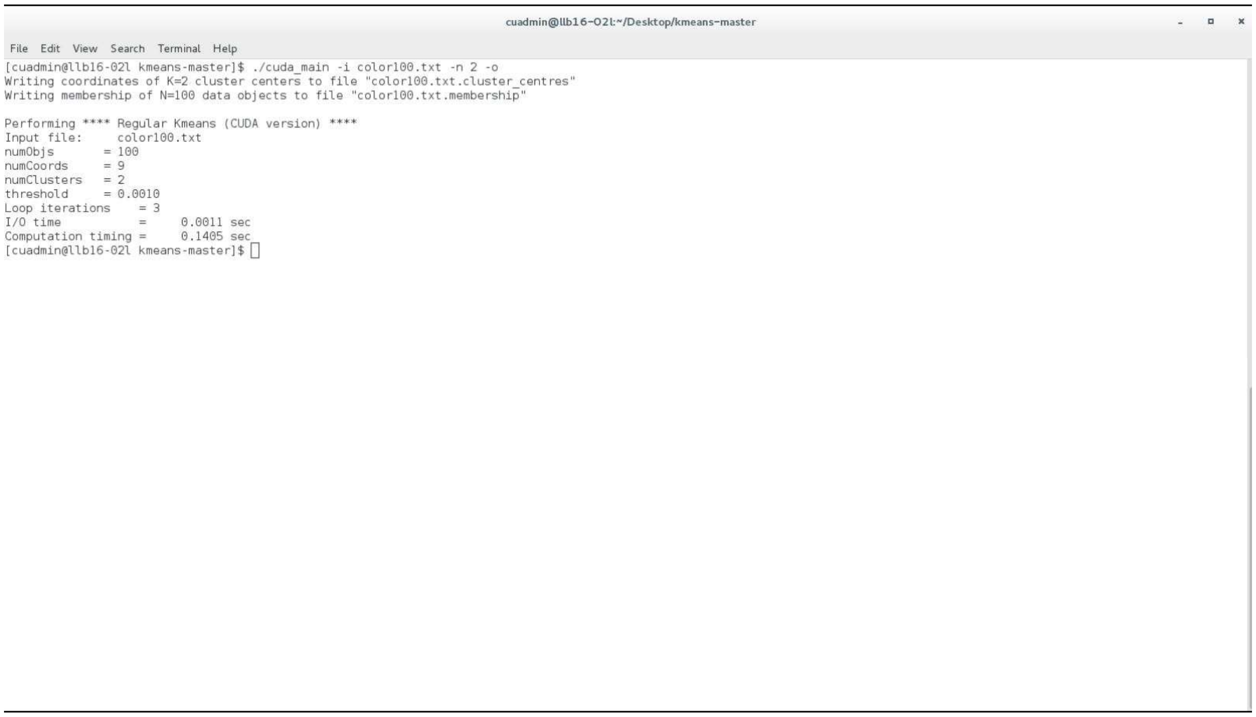
The compiler warning mainly revolved around the shared memory portion of the code found in cuda_kmeans.cu. The warning is not critical if there is not too many data objects and clusters during a single execution of the algorithm (In our case, this warning is not critical). Next, I ran the cuda_main to display the acceptable flags I can use to compute the algorithm.


cuadmin@llb16-02l:~/Desktop/kmeans-master
File Edit View Search Terminal Help

```
[cuadmin@llb16-02l kmeans-master]$ ./cuda_main
Usage: ./cuda_main [switches] -i filename -n num_clusters
       -i filename    : file containing data to be clustered
       -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)
       -o             : output timing results (default no)
       -d             : enable debug mode
[cuadmin@llb16-02l kmeans-master]$
```

Fig 2: Appropriate flags that can be use to find the kmeans of a file

Finally, I ran cuda_main with the flags to read the color100.txt file, indicate that I wanted two clusters, and output the computational speed of the algorithm. My results are found in Fig 3.


cuadmin@llb16-02l:~/Desktop/kmeans-master
File Edit View Search Terminal Help

```
[cuadmin@llb16-02l kmeans-master]$ ./cuda_main -i color100.txt -n 2 -o
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 (CUDA version) ****
Input file:     color100.txt
numObjs       = 100
numCoords     = 9
numClusters   = 2
threshold     = 0.0010
Loop iterations    = 3
I/O time           =     0.0011 sec
Computation timing =     0.1405 sec
[cuadmin@llb16-02l kmeans-master]$ 
```

Fig 3: Executing the file by using the appropriate flags

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

VI. Challenges

One of the biggest challenges I ran across was trying to compile and run this program on my windows machine. Because this code uses UNIX specific libraries and functions (unistd.h, getopt(), etc.), windows did not recognized the library/methods or the Makefile. Running bash on windows fixed this problem, but created an additional problem. Running the Linux bash on Windows 10 failed to compile because the Linux bash on Windows 10 did not detect the Nvidia Video Card on my laptop. As a result, I compiled and ran my code in one of the lab Nvidia-enabled GPU computers. With more time, I could manually code the window's version of getopt() to avoid using the unistd.h library.

VII. References

1. **https://en.wikipedia.org/wiki/K-means_clustering**
2. https://github.com/serban/kmeans
3. **http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html**