Assignment 5, Spring 2017
CS 6643, Parallel Processing
K-Means Algorithm with CUDA

Purpose
The goal of this assignment is to help you learn programming with CUDA.

Due Date
This assignment is due on Friday, 04/14/2017, at 11:55pm

Materials to Review
Slides for "GPGPU Computing and SIMD", "CUDA and GPGPU Computing" and "Advanced Topcis in GPGPU." Additionally, some online tutorials may also be helpful, such as https://llpanorama.wordpress.com/cuda-tutorial/ and http://supercomputingblog.com/cuda-tutorials/.

Instructions
1. Please read the instructions for AWS to learn how to do GPGPU programming on AWS Virtual Machines.

2. Please download the source code zip file, source.zip, from Blackboard. Copy the zip file to your AWS instance (I use scp or sftp), and decompress it with the following command:

```
$ unzip soruce.zip
$ cd source
```

3. There are 12 files from the zip file:
   1) `Makefile` – for compilation, you should NOT change this file.
   2) `main.cu` – entry function, handles command line parameters, you DO NOT need to change this file.
   3) `read_data.cu` – helper function for reading data points from data files you should NOT change this file.
   4) `random.cu` – random value generator, you should NOT change this file.
   5) `k_means.h` – common definitions; you can find the definition of "struct point" here; you Do NOT need to change this file.
   6) `k_means.cu` – you should implement the K-means algorithm in this file.
   7) `data/example1_k2_m10.txt` – sample input data file (2 clusters, 10 points)
      `example1.output.txt` – sample output file for example 1 (your outputs should match this file)
   8) `data/example2_k3_m30.txt` – sample input data file (3 clusters, 30 points)
      `example2.output.txt` – sample output file for example 2 (your outputs should match this file)
   9) `data/example3_k5_m500.txt` – sample input data file (5 clusters, 500 points)
      `example3.output.txt` – sample output file for example 3 (your outputs should match this file)

10) `data/example4_k8_m10000.txt` – sample input data file (8 clusters, 10000 points)
    `example4.output.txt` – sample output file for example 4 (your outputs should match this file)
11) `data/example5_k10_m10000.txt` – sample input data file (10 clusters, 10000points)
    `example5.output.txt` – sample output file for example 4 (your outputs should match this file)

4. Implement the K_Means algorithm using CUDA in `k_means.cu`. Note that, for this assignment you only need to change ONE files. Please read the comments in these files to get an idea of what code you can or should put in these files.

5. In general, you will want to:
   1) Allocate memory on GPU and copy all data (data points, centers and other necessary data) to GPU;
   2) Implement a GPU kernel that assigns data points to the correct centers.
   3) Compute the coordinates for the new centers, either on GPU or on CPU. If you choose to do this on CPU, then you also need to copy the data-point-to-center-assignments back to CPU. After computing the new centers on CPU, you also need to copy the newly-computed centers back to GPU.
   4) Repeat step 2) and 3) for the required iterations.
   5) If you choose to compute the new centers on GPU, you need to copy the new centers back to CPU after all iterations are finished.
   6) Note that the "k_means" function in `k_mean.cu` is just an interface for passing in basic parametersx. You need to add GPU kernels in `k_means.cu` and launch them in the "k_means" function.
   7) The "k_means" function has two input parameters for block count and thread count per block. Please use these two parameters when launching your kernels.

6. After you have implemented the K_Means algorithm, please compile the program using the following command, (this command should be issued in the directory with `Makefile` and other CUDA source files):

```
$ make
```

   You may encounter compilation errors and warnings. Correct your code to eliminate compilation errors. As a good programming practice, you should also try to eliminate as many warnings as possible (although CUDA compiler does issue incorrect warnings). Please MAKE SURE your implementation can compile without errors.

   If compilation is finished without error, you should have a new executable file named *k_means*.

7. After successfully compiling your implementation, you should test your implementation with the supplied examples to verify your implementation is correct with only ONE thread. For this assignment, block count and thread count is passed to the program with command line options "-b" (for block count) and "-t" (for thread count per block). For example, the following examples execute the program with one thread using "-b 1 -t 1."

```
$./k_means -f data/example1_k2_m10.txt -k 2  -i 100 -b 1 -t 1
Reading from data file: data/example1_k2_m10.txt.
Finding 2 clusters
centers found:
40.27, -185.25
-85.97, -13.16

$./k_means -f data/example2_k3_m30.txt -k 3 -i 100 -b 1 -t 1

Reading from data file: data/example2_k3_m30.txt.
Finding 3 clusters
centers found:
-176.25, -142.18
172.39, 43.03
66.01, 197.75

$./k_means -f data/example3_k5_m500.txt -k 5 -i 100 -b 1 -t 1

Reading from data file: data/example3_k5_m500.txt.
Finding 5 clusters
centers found:
-112.85, 138.49
120.73, 133.61
-174.82, -45.06
-27.18, 192.85
-70.18, -71.77

$./k_means -f data/example4_k8_m10000.txt -k8 -i 10000 -b 1
-t 1

Reading from data file: data/example4_k8_m10000.txt.
Finding 8 clusters
centers found:
-1804.35, 9354.77
10842.90, -13500.57
23334.93, 7272.14
-4122.96, -7844.89
21918.36, 10884.11
4425.72, 37811.02
3339.24, 17247.81
-4518.67, 20087.27

$./k_means -f data/example5_k10_m10000.txt -k 10 -i 10000 -b 1
-t 1

Reading from data file: data/example5_k10_m10000.txt.
Finding 10 clusters
```

```
centers found:
38507.34, 2184.57
699.67, 1023.90
2285.75, 16740.93
21669.52, -18506.64
7993.46, 23387.47
-3170.53, 32650.72
-35903.84, -20043.25
38436.87, 2134.21
-26618.41, 32616.35
5001.14, -29326.91
```

The output of each example should match the example outputs supplied in the zip file (under data directory). The order of your centers may be different from the example outputs.

8. If your results of using ONE thread is correct, then you can go on to verify if your implementation is correct using more threads. You need to set the block and thread counts to more than one using command line options “-b” and “-t”. For example, “-b 8 -t 32” tells the K_MEANS program to use 8 blocks and 32 threads per block:

```
$./k_means -f data/example1_k2_m10.txt -k 2  -i 100 -b 8 -t 32

Reading from data file: data/example1_k2_m10.txt.
Finding 2 clusters
centers found:
40.27, -185.25
-85.97, -13.16

$./k_means -f data/example2_k3_m30.txt -k 3 -i 100 -b 8 -t 32

Reading from data file: data/example2_k3_m30.txt.
Finding 3 clusters
centers found:
-176.25, -142.17
172.39, 43.03
66.01, 197.75

$./k_means -f data/example3_k5_m500.txt -k 5 -i 100 -b 8 -t 32

Reading from data file: data/example3_k5_m500.txt.
Finding 5 clusters
centers found:
-112.85, 138.49
120.73, 133.61
-174.82, -45.06
-27.18, 192.85
-70.18, -71.77

$./k_means -f data/example4_k8_m10000.txt -k 8 -i 10000 -b 8
-t 32
```

```
Reading from data file: data/example4_k8_m10000.txt.
Finding 8 clusters
centers found:
-1804.35, 9354.77
10842.90, -13500.57
23334.93, 7272.14
-4122.96, -7844.89
21918.36, 10884.11
4425.72, 37811.02
3339.24, 17247.81
-4518.67, 20087.27
$./k_means -f data/example5_k10_m10000.txt -k 10 -i 10000 -b 8
-t 32

Reading from data file: data/example5_k10_m10000.txt.
Finding 10 clusters
centers found:
38507.34, 2184.57
699.67, 1023.90
2285.75, 16740.93
21669.52, -18506.64
7993.46, 23387.47
-3170.53, 32650.72
-35903.84, -20043.25
38436.87, 2134.21
-26618.41, 32616.35
5001.14, -29326.91
```

The output of each example should match the example output supplied in the zip file (under data directory) and the ONE-THREAD results from your implementation. The order of your centers may be different from the example outputs.

9. The user interface to *k_means* remains the same. You can run *k_means* with -h to get an explanation of the command parameters.

```
$ ./k_means -h
Usage: ./this_command [-h] [-f DATA_FILE] [-k K] [-i ITERS]
                      [-t THREAD_CNT]

-h, --help            show this help message and exit
-f, --data-file       specify the path and name of input file
-k, --clusters        specify the number of clusters to
                      find;default 2 clusters
-i, --iterations      specify the number of iterations of
                      clustering to run; default 10 iterations
-b, --blocks          specify the number of blocks to use when
                      run on GPU; default 1 block
-t, --threads         specify the number of threads per block
                      to use when run on GPU; default 1 thread
                      per block
```

10. After you have verified that your implementation is correct, you can proceed to evaluate the performance of your implementation. You can use "time" command to get the execution time of your implementation. To get reasonable execution time, we will use 10000 iterations. For example, the following command and outputs illustrates the execution of 8 blocks X 32 Threads with 10000 iterations,

```
$/usr/bin/time ./k_means -f data/example4_k8_m10000.txt -k 8
-i 10000 -b 8 -t 32

Reading from data file: data/example4_k8_m10000.txt.
Finding 8 clusters
centers found:
-1804.35, 9354.77
10842.90, -13500.57
23334.93, 7272.14
-4122.96, -7844.89
21918.36, 10884.11
4425.72, 37811.02
3339.24, 17247.81
-4518.67, 20087.27
2.88user 2.45system 0:07.00elapsed 74%CPU (0avgtext+0avgdata
88148maxresident)k
0inputs+0outputs (0major+2279minor)pagefaults 0swaps
```

The green part of the above output gives the execution time (elapsed time, 7 seconds). You should change the number of blocks/threads (by varying the value of the "-b/-t" option) to get the execution times of your implementation. Your implementation should have performance improvement when using more threads. You will lose points if your submission performs extremely bad.

Hints

1. Copying data between CPU and GPU could be expensive. Therefore, you may want to reduce the number of data copies in your code.

2. I will use AWS P2.xlarge instance to grade. On P2.xlarge, there is one Nvidia K80 GPU, with 30 Stream processors and 64 double-precision FPU per SM.

3. I will use 8 blocks and 32 threads per block for grading. If your code runs better with other block/thread configurations. Please let me know by email.

4. CUDA does not provide global barriers. Therefore, A typical solution for global barrier is to partition computation tasks into kernels, with one kernel corresponds to one step/task. After finishing one task/step with a kernel, CUDA transfers the control back to CPU to synchronization the progress of all CUDA threads.

5. CUDA code examples from the lecture can help you write CUDA kernels.

6. As always, Google is your best friend. If you have strange compilation errors, you should try to Google the error to find solutions.

7. Example 4 will converge long before 10000 iterations. However, you should continue performing the clustering after convergence.

Submission Guideline

1. **WARNING: YOUR SUBMISSION ***MUST*** FOLLOW THIS GUIDELINE. THERE WILL BE 40% PENALTY FOR THOSE WHO FAIL TO FOLLOW THIS GUIDELINE.**
2. Submit your *k_means.cu* to Blackboard. **THE FILENAMES MUST BE k_means.cu.**
3. **Please DO NOT zip your submission.**
4. Your program will be compiled with the same Makefile that is given to you.
   You will receive 0 points if your code fails to compile.