Assignment 3, Spring 2017
CS 6643, Parallel Processing
K-Means Algorithm with OpenMP

Purpose
The goal of this assignment is twofold: 1) help you determine which loops of the K-Means algorithm can be parallelized; 2) help you learn OpenMP programming

Due Date
This assignment is due on Friday, 02/24/2017, at 11:55pm

Materials to Review
Slides for "Introduction to OpenMP" and "OpenMP Programming." Additionally, you may want to read the text book "Introduction to Parallel Computing" Chapter 7.10, and LLNL's OpenMP tutorial at https://computing.llnl.gov/tutorials/openMP/.

Instructions

1. Please download the source code zip file, source.zip, from Blackboard. Copy the zip file to CS department servers fox01~fox06 (I use scp or sftp), and decompress it with the following command:

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

2. There are 12 files from the tarball:
   1) `Makefile` – for compilation, you should NOT change this file.
   2) `main.c` – entry function, handles command line parameters, you should NOT change this file.
   3) `read_data.c` – helper function for reading data points from data files you should NOT change this file.
   4) `random.c` – 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 should NOT change this file.
   6) `k_means.c` – 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)

3. Implement the K_Means algorithm using OpenMP in `k_means.c`. You should finish the implementation of function "`void k_means(struct point p[MAX_POINTS], int m, int k, int iters, struct point u[MAX_CENTERS], int c[MAX_POINTS])`" in `k_means.c`. Please read the comments in k_means.c for the explanation of input and output parameters.

   To start, copy your code from Assignment 1 to k_means.c. Then add OpenMP directives and clauses to parallelize your loops.

4. 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 C 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 all warning. 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*.

5. After successfully compiling your implementation, you should test your implementation with the supplied examples to verify your implementation is correct with only ONE thread. First you need to set the environment variable OMP_NUM_THREADS to 1 so that OpenMP run-time knows to only use on thread. If you are in tcsh, please do:

```
$ set OMP_NUM_THREADS 1
```

   If you are in bash, please do:

```
$ export OMP_NUM_THREADS=1
```

   Please run the following commands to test your implementation:

```
$./k_means -f data/example1_k2_m10.txt -k 2  -i 100
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
```

```
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

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

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

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

6. If your results using ONE thread is correct, then you can go on to verify is your implementation is correct using more threads. First you need to set the environment variable

OMP_NUM_THREADS to 8 so that OpenMP run-time knows to only use on thread. If you are in tcsh, please do:

```
$ set OMP_NUM_THREADS 8
```

If you are in bash, please do:

```
$ export OMP_NUM_THREADS=8
```

After setting the environment variables, you can then test your program:

```
$./k_means -f data/example1_k2_m10.txt -k 2  -i 100

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

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

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

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

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

7. 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]

-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
```

8. 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 threads with 10000 iterations (you need to set the OMP_NUM_THREADS to the correct thread numbers as demonstrated previously),

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

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
14.04user 0.01system 0:02.00elapsed 795%CPU (0avgtext+0avgdata
```

```
1996maxresident)
k0inputs+0outputs (0major+199minor)pagefaults 0swaps
```

The green part of the above output gives the execution time (elapsed time, 2 seconds). You should change the number of threads (by varying the value of OMP_NUM_THREADS) to get the execution times of your implementation. Your implementation should have performance improvement when using more threads (ideally, near linear speedup). You will lose points if your submission has no performance improvement using more threads.

Hints

1. There are only a few loops need to be parallelized. You only need to add directives and clauses to parallelize these loops. A typical implementation only needs adding less than 20 lines of code.

2. For nested loops, a "#pragma omp (parallel) for" will only parallelize the outer loop.

3. Be careful with variables. For any parallel region, you should list all the variables used in this region, and determine what variables can be shared and what variables must be private. If your multi-threaded outputs are wrong, it is most likely that some variables are mis-scoped (i.e., variable should be private, but declared/default-ed to be shared).

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

5. Note that the interface to "random_center()" function has changed. You need to update your code to use the newest interface.

6. 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.c* to Blackboard. **THE FILENAME MUST BE k_means.c. Only k_means.c should be submitted.**
3. 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.**