# Assignment 4, Spring 2017
# CS 6643, Parallel Processing
# K-Means Algorithm with Pthread

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

## Due Date
This assignment is due on Friday, 03/31/2017, at 11:55pm

## Materials to Review
Slides for "Shared Memory Programming with Pthreads" and Pthread code examples. Additionally, you may want to read the text book "Introduction to Parallel Computing" Chapter 7, and LLNL's Pthreads tutorial at https://computing.llnl.gov/tutorials/pthreads/.

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 zip file:
   1) `Makefile` – for compilation, you should NOT change this file.
   2) `main.c` – entry function, handles command line parameters, you need to change this file to create and control threads.
   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 may need to change this file to pass parameters to your threads.
   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 Pthreads in `main.c, k_mean.h` and `k_means.c`. Note that, for this assignment you will need to change THREE files. Please read the comments in these files to get an idea of what code you can or should put in these files.

4. In general, you will want to:
   1) put thread creation and parameter passing `main.c`;
   2) declare additional data structure in `k_mean.h` for passing parameters to your threads; and,
   3) implement the per-thread K_Mean algorithm in k_means.c within the pthread start routine `void *k_means(void *)`.
   4) For this assignment, you have more freedom in the implementation, such as adding more functions. However, make sure that you only change `main.c, k_mean.h` and `k_means.c`, and make sure that your code can be compile correctly with the supplied Makefile. When grading, we will ONLY use those three files and the supplied Makefile.

5. 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*.

6. After successfully compiling your implementation, you should test your implementation with the supplied examples to verify if your implementation is correct with only ONE thread. For this assignment, thread number is passed to the program with a command line option "-t."

   For example, the following examples execute the program with one thread using "-t 1."

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

7. If your results using ONE thread is correct, then you can go on to verify if your implementation is correct using more threads. First you need to set the thread number to more than one using command line option “-t”. For example, “-t 8” tells the K_MEANS program to use 8 threads:

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

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 -t 8

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 -t 8

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 -t 8

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 -t 8

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) and the ONE-THREAD results from your implementation. The order of your centers may be different from the example outputs.

8. 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
-t, --threads         specify the number of threads to use;
                      default 1 thread
-i, --iterations      specify the number of iterations of
                      clustering to run; default 10 iterations
```

9. 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 a reasonable execution time, we will use 10000 iterations. For example, the following command and outputs illustrates the execution of 8 threads with 10000 iterations,

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

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 0:03.00elapsed 795%CPU (0avgtext+0avgdata
1996maxresident)
k0inputs+0outputs (0major+199minor)pagefaults 0swaps
```

The green part of the above output gives the execution time (elapsed time, 3 seconds). You should change the number of threads (by varying the value of the “-t” option) 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 performs poorly.

## Hints

1. Data points, centers, and "points' cluster ids" are made global variables. All threads can read and write these global variables. You should also set your results (i.e., the final center coordinations and point assignments) to these global variables.

2. You will need to declare additionally data structure to pass in extra parameters to your threads. These extra parameters include data point count "m" and cluster count "k", as well as another variables that can help you decompose the K_Mean loops.

3. You will need to use Pthread barriers for this assignment. You may need to use Pthread mutex when writing to shared global variables.

4. Pthread code examples from the lecture can help you figure out how to properly use Pthread functions.

5. It will be helpful for this assignment to understand how C header file inclusion works, and what are "extern" variables.

6. 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 *main.c*, *k_means.h* and *k_means.c* to Blackboard. **THE FILENAMES MUST BE main.c, k_means.h and k_means.c.**
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.**