Due Date: 5:00pm, Thursday, 3/3/2016

The purpose of this lab is to gain experience in thinking about parallelizing programs and in using Pthreads to implement parallel programs. In doing this, you will also gain experience in mechanisms for managing synchronization and communication between threads.

Step 1: Obtain c code and data files from the web

Download the following files from the web to a folder on bally using these three commands:

```
wget http://csweb.cs.wfu.edu/~turketwh/346/etime.h
wget http://csweb.cs.wfu.edu/~turketwh/346/etime.c
wget http://csweb.cs.wfu.edu/~turketwh/346/kmeansSmall.txt
```

Before the project is due (but not on 2/18), you will be provided two additional challenge input files:

```
wget http://csweb.cs.wfu.edu/~turketwh/346/kmeansMedium.txt
wget http://csweb.cs.wfu.edu/~turketwh/346/kmeansLarge.txt
```

The `etime.h` and `etime.c` files are the header (function declaration) and implementation (function definition) files for the timing routines we will make use of in this lab. The three text files are data files for testing your program. You should start by only working with `kmeansSmall.txt`. Each file has the following format: On the top line is the number of samples S followed by the number of features per sample F. The rest of the file is S rows of F values, where each value is a real number between 0 and 1, with each value in the file separated by a space. As an example, an input data file with 4 samples of 2 features each would look like this:

4 2
1.0 1.0
0.98 0.98
0.25 0.40
0.25 0.48

Step 2: Implementing K-Means Clustering using P-Threads

Develop a program that can employ K-Means clustering on data from an input file formatted as described above and that prints to an output file the cluster membership for each sample. Thinking about the program in the most general terms possible, it should do the following:

- Be executed using the following command line: `kmeans K P inputFile outputFile`
- Read in the data from the input data file suggested by the user on the command line
- Use K-Means clustering to cluster the data items, using the user's provided (on the command line) number of clusters K and number of processors P
- Write the final cluster values in the appropriate format to the output data file suggested by the user on the command line
- Record the timing of the parallel parts of the program

The first line of the output file should hold the value S (the number of samples) and then each line afterwards should hold the cluster number (0..K-1) for each sample, written in the order the samples were read in. As an example, the output that might result from the example data file above could look like the output on the next page, suggesting there were 4 samples and the first two samples ended up in cluster 0 and the other two samples ended up in cluster 1.

4
0
0
1
1

Here is a basic description of the K-means clustering algorithm as was reviewed in class:

- Randomly choose K of your provided samples as initial cluster centers
- For each sample, determine which of the centers the sample is closest to using Euclidean distance and associate the sample with an appropriate center index (0..K-1)
- Repeat
    - Determine K new center values, by averaging the sample features for each sample associated with a given center
    - For each sample, determine which of the centers the sample is closest to using Euclidean distance and associate it with the appropriate center index (0..K-1)

  until an iteration occurs when no samples change the center index it is associated with. Stop at 100 iterations if convergence has not occurred by that point.

An explanation and example of K-Means can be found in this Youtube video:
https://www.youtube.com/watch?v=_aWzGGNrcic

To parallelize the program, you should allocate a fair (evenly distributed) number of samples to each thread. Be as fair as possible if the number of samples is not evenly divisible by the number of threads. Carefully think through whether/where there are any points in the program that need thread synchronization or other thread management.

To time your program, add the hooks to the *tic()* and *toc()* functions just before the *pthread_create* and just after the *pthread_join* function calls in your main function, similar to how it was done in the *triplets_threads.c* program. The timing should not encompass any I/O (input/output) so that should all be completed before the threads start or performed after the threads end.

The small dataset has twelve 2-dimensional points. Use it for developing and debugging your code. I would suggest you plot the twelve points to get a sense of how they should cluster (I developed it so there should be nice clusters when K = 2, K = 3, or K = 6). Since K-Means has a random element to it (initial choice of centers), you may not always get the same clusters, but you should expect over multiple runs to fairly consistently get the same items clustered together if the clusters are well-defined.

Step 3: Recording parallel processing times

The small dataset should be used for debugging of your program during development. The to-be-released-soon medium and large datasets will be used for timing. For both the medium and large datasets, you should evaluate the following values of K and P: K = {2,5,10,25}, P={1,2,4,8,16} (try all 20 combinations). Record your times as an average over 3 runs in a table. A template table will be provided to you. Ultimately, you will run (20 parameter settings x 2 files x 3 timings per setting&file) ➔ 120 instances of your program.

Step 4: Questions

Address the questions on the next page about the K-Means clustering problem and your parallel implementation of it.

1. Even though you only have a small sample size (two different values for S, the dimension upon which you are distributing the work), how does the *speedup* change for fixed P as the S parameter changes? Explain why you believe you are seeing such changes in the speedup.
2. How does the *speedup* change when you increase P for a given file? Explain why you believe you are seeing such changes in the speedup.
3. How does the *efficiency* change when you increase P for a given input file? Explain why you believe you are seeing such changes in the speedup.

Step 5: Grading Rubric
Correctness (30%): Your program clusters correctly on the small dataset you were provided and a separate small dataset I will generate independently. I will run your program multiple times to account for the inherent randomness in the program.

Questions (30%): Accurate and detailed answers to the questions in Step 4.

Coding and Documentation (30%): Clean, readable code, with documentation describing any major algorithmic steps you are implementing and, most importantly, documentation explaining the purpose of any thread-based code you write.

Timings (10%): Complete and reasonable timings submitted.

Step 5.5: Prizes
The 3 students with the fastest correct code on a data file that is the size of the large data file will receive a small, non-grade related, prize. I will do the timing!

Step 6: Submission
Submit your commented C code, your answers to the Questions in Word/PDF/txt format, and your timing table via Sakai under *Assignments/Program 3* by the deadline.

Step 7: Grad Students
You will be assigned (soon) a few additional tasks and/or questions.