EE/CSCI 451
Spring 2017
Programming Homework 2

Assigned: January 27, 2017
Due: February 10, 2017, before 11:59 pm, submit via blackboard
Total Points: 50

Example program

print_msg_with_join.c is the example we discussed in Discussion #2. You can refer it for thread creation, joining and passing arguments to threads.
To compile it, type: gcc -lrt **-lpthread** -o go print_msg.c
To run it, type: ./go

1 Parallel Matrix Multiplication [10 points]

Parallelize the **naive** matrix multiplication which you implemented in PHW 1 using Pthreads. One simple way to parallelize the algorithm is evenly partitioning and distributing the computation works of the elements of output matrix among the threads and letting the threads work independently in parallel. For example, assuming the matrix size is $n \times n$ and the number of threads is p; let thread #1 compute the elements of column $0 \sim \frac{n}{p} - 1$ of the output matrix, thread #2 compute the elements of column $\frac{n}{p} \sim \frac{2n}{p} - 1$ of the output matrix, and so on.

- The matrix initialization is the same as that in PHW 1. The matrix size is $4K \times 4K$. Print out the execution time and the value of $C[100][100]$ in your program.
- Pass the number of threads p as a command line parameter [1].
- Report the execution time for $p = 1$ (the serial version you did in PHW1), $4, 16, 64$ and 256. Draw a diagram to show the execution time under various p. An example diagram is shown in Figure 1. Briefly discuss your observation.
- In your report, discuss how you partition the computation works of the elements of the output matrix among the threads.

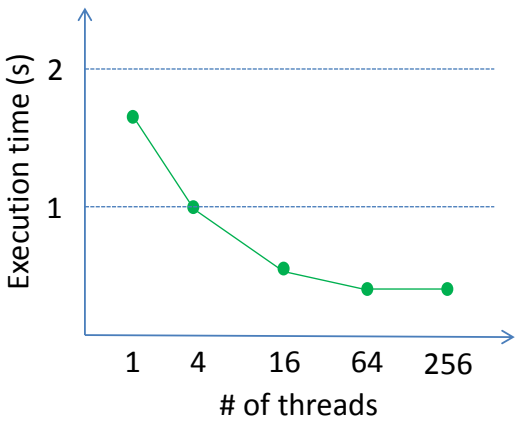

Figure 1: Example diagram

- Name the program as p1.c

2 Parallel K-Means [15 points]

In PHW 1, you implemented the K-Means algorithm. In this problem, you will need to parallelize your implementation using Pthreads. Let p denote the number of threads you create. The parallel version of K-Means algorithm has the following steps:

1. Initialize a mean value for each cluster.
2. Partition and distribute the data elements among the threads. Each thread is responsible for a subset of data elements.
3. Each thread assigns its data elements to the corresponding cluster. Each thread also locally keeps track of the number of data element assigned to each cluster in current iteration and the corresponding sum value.
4. Synchronize the threads.
5. Recompute the mean value of each cluster.
6. Check convergence; if the algorithm converges, replace the value of each data with the mean value of the cluster which the data belongs to, then terminate the algorithm; otherwise, go to Step 2.

In this problem, the input data is a **1024×1024** matrix which is stored in the 'input.raw'; the value of each matrix element ranges from 0 to 255. Thus, the matrix can be displayed as an image. We will have **6** clusters (K=6). The initial mean values for the clusters are 0, 65, 100, 125, 190 and 255, respectively. To simplify the implementation, you do not need check the convergence; run **50** iterations (Step 2-5) then terminate the algorithm and output the matrix into the file named 'output.raw'.

- Implement a serial version without using Pthreads (This is the program you submitted for PHW 1).
- Implement a parallel version using Pthreads. Name the program as p2.c. Pass the number of threads p as a command line parameter
- In your report, you need report the execution time for the **50** iterations (excluding the read/write time) for $p = 4, 8$. Compare with the serial version with respect to execution time, discuss your observation.
- Show the image of the output matrix in your report. You can display the output image, 'output.raw', using the imageJ [3] software or the given Matlab script file, 'show_raw.m'. If you use 'show_raw.m', remember to specify the file name in the script.

3 Producer-consumer [25 points]

The producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-thread synchronization problem. In this assignment, you will simulate the producer-consumer problem using mutex and condition variable. Producer makes cookie and puts cookie on the shelf. Consumer buys cookie and takes cookie away from the shelf.

- There are **two** consumers and one producer. Each producer or consumer is implemented as a thread.
- The capacity of the shelf is 10.
- When the producer checks the number of cookies on the shelf, (1) if the number is less than 9, he puts 2 cookies on the shelf; (2) if the number is 9, he puts 1 cookie on the shelf.
- When the shelf is not empty, a consumer thread can only take 1 cookie at a time.
- If the producer thread has put ≥ 30 cookies on the shelf, it can be terminated.
- If a consumer thread has taken 15 cookies away from the shelf, the consumer thread can be terminated.
- Whenever the number of cookies on the shelf changes, you need print a message in the following format.

- Producer has put 2 cookies; # of cookies on the shelf changes from 0 to 2.
- Consumer 2 has taken 1 cookies; # of cookies on the shelf changes from 2 to 1.
- Producer has put 4 cookies; # of cookies on the shelf changes from 1 to 3.
- Consumer 1 has taken 1 cookies; # of cookies on the shelf changes from 3 to 2.
- Consumer 1 has taken 2 cookies; # of cookies on the shelf changes from 2 to 1.
- Consumer 1 has taken 3 cookies; # of cookies on the shelf changes from 1 to 0.
- ...

1. **Using mutex only [10 points]**
   In this version, when each thread intends to check/update the number of cookies on the shelf, the thread needs to acquire the mutex first in order to avoid any race condition. Here, only mutex is used. Name this program as p3a.c.

2. **Using mutex and condition variable [15 points]**
   In this version, when the producer checks the number of cookies on the shelf, if the number is 10, producer goes to sleep; if the number is 0, producer puts 2 cookies on the shelf and sends a broadcast signal to wake up the consumers. When a consumer checks the number of cookies on the shelf, if the number is 0, the consumer goes to sleep. Initially, all the threads are awake. Name this program as p3b.c.

4 Submission

You may discuss the algorithms. However, the programs have to be written individually. Submit the code (p1.c, p2.c, p3a.c and p3b.c) and the report via **Blackboard**. Your program should be written in C or C++. Make sure your program is runnable. Write clearly how to compile and run your code in the report as well. It is recommended to include a Makefile [2] along with your submission. If your program has error when we compile or run your program, you will lose at least 50% of credits.

References

[1] "Command Line Parameter Parsing,"
`http://www.codingunit.com/c-tutorial-command-line-parameter-parsing`

[2] "Using make and writing Makefiles,"
`http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html`

[3] "imageJ,"
`http://rsb.info.nih.gov/ij/download.html`