Fall 2020 CIS 345/545 Project 1

(Due Sep. 24)

In this project, you are asked to write two independent programs, `thr_atomic.c` and `thr_reduce.c`, each of which uses m computational threads to concurrently calculate the sum of square roots from 1 to n, where m and n are powers of 2 and are specified in the command line.

For the program `thr_atomic.c`, it gets the values m and n from the command line arguments and converts them to two integers, respectively. Next, it creates m threads using `pthread_create()` and each thread computes the sum of n/m square roots. Namely, the first thread (i.e. thread 0) computes the sum of square roots from 1 to n/m, the second thread (i.e. thread 1) computes the sum of the square roots from $n/m+1$ to $2n/m$, etc. When a thread finishes its computation, it should print its partial sum and atomically add it to a **shared global variable**. Note that your program needs to use `pthread_barrier_wait()` to let the main thread know that all of the m computational threads have done the atomic additions and hence it can print the result. Below is an example of running your thread program:

```
bach> ./thr_atomic 2 65536
thr 0: 3954518.036356
thr 1: 7230420.422587
sum of square roots: 11184938.458943
```

The program `thr_reduce.c` is similar to `thr_atomic.c` except that you need to use the parallel reduction approach to combine the partial sums. That is, your program uses a **shared global array** and each computational thread stores its partial sum, but not prints, in an array element indexed on its thread ID. Then, half of these threads call `pthread_join()` to wait for their corresponding partner threads completion, and then each of these threads can add two numbers in the array together. This reduction procedure will be performed $\log(m)$ times and each time the number of the active threads will be reduced half. Finally, there will be only one active thread and this thread should print the whole array.

For example, assume that there are 8 computational threads. In the first reduction step (i.e. adding two partial sums in the array), thread 0 should wait for thread 4 done, thread 1 has to wait for thread 5 done, threads 2 needs to wait for thread 6 done, and thread 3 should wait for thread 7 done. In the second reduction step, thread 0 waits for thread 2 finished, and thread 1 waits for thread 3 finished. In the third step, thread 0 waits for thread 1 done and then prints the the whole array. *Hint*: to find its partner thread ID during the ith reduction step, a thread can use its ID XORed with 2^{r-i}, where $r = \log(m)$. Note that the main thread needs to use a reverse loop to create threads. That is, the thread with the highest ID (m-1) will be created first and the thread with ID 0 last. The main thread just calls `pthread_exit()` after creating m threads. It does not need to wait for these threads. Calling `pthread_exit()` in the main thread will allow other threads to continue execution.

Turning it in

Each group (at most two CIS345 students, CIS545 students should work alone) has to submit your program and report electronically by using the following command on grail:

```
turnin -c cis345s -p proj1 makefile thr_atomic.c thr_reduce.c p1_report.pdf
```

Your report should include the description of your code, experiences in debugging and testing, etc. The cover page should contain your picture(s), name(s) and the login id you used to turnin the project. Start on time and good luck. If you have any questions, send e-mail to `sang@cis.csuohio.edu`.