Programming Project #2

CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing
Spring 2020

Due 3/31 Tue., 12:30 p.m. Please submit a soft copy on Blackboard. Late submissions are accepted till 4/7 Tue., 12:30 p.m. with 10% penalty each day. No submissions accepted after 4/7 Tue., 12:30 p.m.

Please submit a single tarball/zipped file containing all your source codes and documents. Please name your submission file starting as "LastName_FirstName_PP2".

Please note that we may request an in-person, 5-10 mins quick demo for grading.

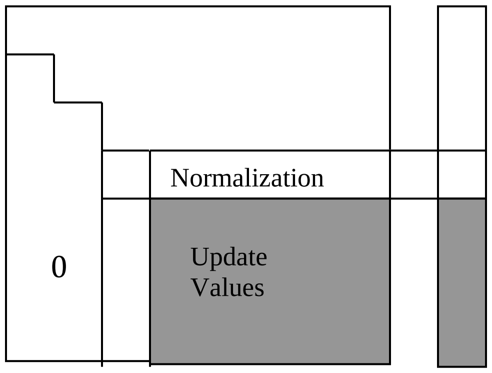
Pthreads Programming

Source code samples: Please checkout source code samples with executing the following command:

```
git clone https://discl.cs.ttu.edu/gitlab/yongchen/cs4379cs5379.git
```

Q1 (40%). In this part of the programming project, you are asked to experiment Example 7.3 and Example 7.7 codes in the reference book "Introduction to Parallel Computing" (also can be found from Blackboard, "Readings" section). Please note Example 7.7 code needs the read-write lock construct that is created and shown in Sections 7.8.1. Please try both codes on the HPCC cluster and report the performance with 1, 2, 4, 8, 16, and 32 threads for finding a minimum entry of 100 millions of integers (you can generate randomly) with both codes. Develop a Makefile to automate the compilation. Please submit your codes and performance result.

Q2 (60%). In this part of the programming project, you are asked to write a parallel program using PTHREADS for solving a set of dense linear equations of the form A*x=b, where A is an n*n matrix and b is a vector. You will use Gaussian elimination without pivoting described as below.



The algorithm has two parts (an example code gauss.c is provided):

- *Gaussian Elimination*: the original system of equation is reduced to an upper triangular form Ux=y, where U is a matrix of size n*n in which all elements below the diagonal are zeros which are the modified values of the A matrix, and the diagonal elements have the value 1. The vector y is the modified version of the b vector when you do the updates on the matrix and the vector in the Gaussian elimination stage.
- *Back Substitution*: the new system of equations is solved to obtain the values of x.

The Gaussian elimination stage of the algorithm comprises (n-1) steps. In the algorithm, the ith step eliminates nonzero subdiagonal elements in column i by subtracting the ith row from row j in the range [i+1,n], in each case scaling the ith row by the factor A_{ji}/ A_{ii} so as to make the element A_{ji} zero. See the figure above.

Hence the algorithm sweeps down the matrix from the top corner to the bottom right corner, leaving subdiagonal elements behind it.

In the reference book "Introduction to Parallel Computing" Chapter 8, Section 8.3 (also can be found from Blackboard, "Readings" section) contains more details about Gaussian elimination algorithm.

Requirements and deliverables:

1. Come up with a parallel version of this program using PThreads. A critical point of parallel programming is performance. Therefore, once you achieve a correct solution, please consider performing some experimentation with your program to try to optimize its performance.
2. There should be clear comments explaining your code.
3. Develop a Makefile to automate the compilation.
4. You need to report and plot the performance of your program with 1, 2, 4, 8, 16, and 32 threads on the HPCC cluster.
5. In addition, you may describe some of different versions of your program that you have tried, what performance results you got out of them, and why you think your current version is reasonably efficient (or what more you could do to make it more efficient) to show the level of effort.

Suggestions:

- Consider carefully the data dependencies in Gaussian elimination in your multithreaded program.
- Gaussian elimination involves $O(n^3)$ operations. The back substitution stage requires only $O(n^2)$ operations, so you are not expected to parallelize back substitution.
- You may find a barrier is useful and you may implement the barrier.
- You may observe performance slowdowns because of the locking overhead.

Grading Criteria:

Please note that we may request an in-person, 5-10 mins quick demo for grading.

Q1	Percentage %	Criteria
40%	40	Correct Example 7.3 experiment (code can be compiled, executed, and produce correct results) and successfully carry out specified test cases
	40	Correct Example 7.7 experiment (code can be compiled, executed, and produce correct results) and successfully carry out specified test cases
	20	Correctness and features of Makefile (at least automate compilation and cleanup)
Q2	**Percentage %**	**Criteria**
60%	10	Inline comments to briefly describe your code
	30	Correct use of primitives to create threads, correct use of primitives for synchronization and mutual exclusion, including mutex locks and conditional variables. Correct Makefile (at least automate compilation and cleanup).
	30	Correct parallelization and correct results.
	10	Successfully carry out specified test cases
	20	A brief document to report plotted results, any other solutions you have tried, and your findings in general.

Note on cheating: We have zero-tolerance policy for cheating. Working in groups is fine for discussing approaches and techniques. Copying problem solutions or code is cheating. Both the person copying and the person giving the answers will be equally penalized. Make sure you do your own work.

Reference Materials:

- PThreads**:**
  - POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads/
  - Linux Tutorial: POSIX Threads, http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
  - Complete Pthreads API: http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html

- Makefile**:**
  - http://www.gnu.org/software/make/manual/make.pdf or
  - http://www.gnu.org/software/make/manual/html_node/index.html