Use "turnin HW3@cs140 LastName_FirstName_filename" in a CSIL machine. List the group members in the submitted file. If there is a difficulty in file submission, email to sbluen@cs.ucsb.edu.

Due on May 13, 2014 before class. No late submission on or after May 16. The programming assignment is graded based on the correctness of the code functionality (you need to allow TAs to test), the performance of the code (allow TA to observe what you see), and the analysis of performance results. The point distribution is marked for each problem below.

The code should be compiled with optimization flag (-O1). The source code for textbook Chapter 4 is in http://www.cs.usfca.edu/~peter/ipp. It contains time.h that specifies a time measurement function. Steven will copy your submitted code to TSCC for a quick test, and thus a) donot submit binary files; b) include a README on how to compile, how to test for correctness and performance with suggested parameters, and the expected test results; c) include a performance report on time, efficiency and scalability. See the description of each problem for more details. In choosing the problem size for analysis, you can choose the sizes that take ~1 minute or less to complete. That should sufficiently demonstrate parallel performance while not spending too much time in experimentation.

1. (5 points) The producer-consumer threads can be synchronized using a semaphore as follows.
   - Consumer thread:
     sem_wait(s);
     Consume an item
   - Producer thread:
     Produce an item
     sem_post(s);

Describe 3 application cases in which it makes sense to set the initial semaphore value s as 0, positive value, or negative value, respectively. Notice that pthread semaphore implementation does not allow a negative initial value. You can refer the semaphore definition discussed in class on the impact of having a negative value.

2. (5 points) The producer-consumer code discussed in class can coordinate multiple producers and consumers using unlimited buffer space that holds items produced. Each consumer checks if there is an item available before consumption. Extend this code under the limited buffer space constraint. Namely the shared memory system has memory space for at most n items. Before a producer produces an item, it needs to check if there is 1 unit of space available and then use that unit of space. After a consumer consumes an item, it releases one unit of space and a producer waiting for space can use such space.
3. (5 points) Sketch the pseudo code for space allocation malloc() and free() functions that contain synchronization using conditional variables. An example of threads that use malloc() and free() is in the class Pthread slides (Part I).

4. (5 points) Textbook 4.8

5. (Total 20 points) Based on HW2 Problem 6, the following program involves the matrix vector multiplication in its loop, given $n \times n$ matrix $A$ and column vectors $x$ and $c$.

   \[
   \text{For } k = 0 \text{ to } t-1 \\
   y = Ax + c \\
   x = y
   \]

   EndFor

   Develop two Pthreads versions: one uses condition variables and the second version uses Pthreads barrier function for coordination. One thread generates the initial values for $x$ and $c$ vectors. Each thread generates the initial values for matrix $A$'s elements it owns. Values for matrix $A$ are $A[i][j] = 0$ if $i=j$ otherwise $-1/n$. Use row-wise block mapping to assign rows to threads. The initial value of vector $x$ is 0. The value of vector $c$ is $c[i] = i/n$ when $i$ varies from 0 to $n-1$. Problem 4.9 of the textbook explains the syntax of Pthread barrier and you may find additional information from the web.

   1) (10 points) The submitted code should contain a test option with $n$, $t$, # of threads, version number as input. The main thread should print the first 30 elements of the final result vector $x$, and that allows a checkup to see if the result vector is correct. The code should also print the measured parallel time.

   2) (4 points) Suggest a problem size setting for the TA to test less than 1 minute and see if there is a performance gain using 4 or 8 cores compared to 1 core. Or explain if there is no performance gain.

   3) (4 points) Also submit a performance report for 3 larger problem sizes to assess the speedup, efficiency, scalability, and coordination cost paid for two versions. Allocate one machine which contains up to 16 cores. Write a sentence or more to compare with the MPI code performance from your HW2 report.

   4) (2 points) In the report for condition variable implementation, summarize your high-level thread synchronization using pseudo code and provide an explanation on how the threads are coordinated from one iteration to another.
6. (Total 20 points) Based on HW2 Problem 7. Given \( n \) numbers and \( p \) threads, the odd-even transposition scheme can sort the list in fewer than \( p \) phases if the input list is partially sorted. In an extreme case, if the input list is already sorted, the algorithm requires 0 phases. The program can be modified to follow this consideration. We will use one list to test.

a. An almost sorted list. The list is divided into 4 quarters with the even size. The numbers inside each quarter are not sorted, but the numbers between quarters are sorted. For example, \( 2, 1, 5, 3, 6, 7, 9, 8 \).

Convert your HW2 MPI solution to a Pthread solution and use Pthread semaphore variables for coordination. Avoid local sorting by checking if it is already sorted, and skip data exchange between a pair of processes if it is not necessary during the even or odd phase.

1) (10 points) The submitted code should contain a test option with \( n \) and \( p \) as input. Thread 0 should print the first 50 elements of the input list and final list, and that allows a checkup to see if the sorted vector is correct. The code should also print the measured parallel time.

2) (4 points) Suggest a problem size setting for the TA to test in less than 1 minute and see if there is a performance gain using 4 or 8 cores compared to 1 core. Or explain if there is no performance gain.

3) (4 points) Also submit a performance report for 3 large problem sizes (e.g. \( n=10 \) millions, 100 millions, or any reasonable size). Assess speedup, efficiency, and scalability when the number of cores used is 4, 8, 16. Write a sentence or more to compare with the MPI code performance from your HW2 report.

4) (2 points) Summarize with the pseudo code and explain how threads are coordinated with condition variable going through the multiple odd-even phases of sorting.

7. (Total 10 points) Develop two code versions based on Text book 4.18 (a) and (b). The sample source code is called pth_mat_vect from Text book chapter 4 code collection. Following the textbook notation, let \( m \) be the number of rows and \( n \) be the number of columns. Modify this code so that the value of vector \( x \) is \( x[i]=i/n \) when \( i \) varies from 0 to \( n-1 \). Values for matrix \( A \) are \( A[i][j]=0 \) if \( i=j \) otherwise \( -1/n \).

1) (5 points) The submitted code should contain a test option with \( m, n, p, and version number \) as input. Thread 0 should print the some elements of \( A \) and first 50 elements of the multiplied results, that allows a correctness inspection. The code should also print the measured parallel time.

2) (5 points) Write how you change the text book source code to address false sharing with the two options described. Also submit a performance report similar as Table 4.5 in Text book Chapter 4 and explain your findings.