Use "turnin HW2@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. For some problem, hand drawing may be easier and a hardcopy of drawing can be submitted.

Due on April 29, 2014 before class. No late submission on or after May 2. 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 see what you see), and the analysis of performance results. The point distribution is marked for each problem below.

The MPI code should be compiled with optimization flag (-O2). The MPI source code for textbook chapter 3 is in http://www.cs.usfca.edu/~peter/ipp. 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 few minutes to complete. That should sufficiently demonstrate parallel performance while not spending too much time in experimentation.

1. (5 points) Text book 3.2.
   Clarification: Write your solution in an SPMD pseudo code style. You need to modify the solution discussed in section 3.2.2 and derive the start and end points of each interval.

2. (5 points) Textbook 3.6

3. (5 points) Textbook 3.8 (b). [TA discusses 3.8 (a)]

4. (5 points) Derive the dependence graph pattern for the following program. Also perform loop interchange if it is legal.

   For i= 2 to n
   For j= 2*i to n
     a(i,j)= a(i-1,j-1)+a(i-1,j);
   EndFor
   EndFor

5. (Total 15 points) Implement the parallel tree summation solution for adding n vectors using MPI.
Each vector is of dimension $k$. The first vector is $[0, 1, 2, \ldots, k-1]$. The second vector is $[k, k+1, \ldots, 2k-1]$. The $j$-th vector is $[j^*k, j^*k+1, \ldots, j^*k+k-1]$. Those vectors are evenly assigned to $p$ MPI processes using a block mapping. Each process generates the initial values for its local vectors that it owns. Then you develop two version of implementation. One is to use basic MPI send and receive functions to communicate. The second version uses MPI reduce function.

1) (7 points) The submitted code should contain a test option with $n$ and $k$ as input. Process 0 should print the first 30 elements of the final result vector, and that allows a checkup to see if the result vector is correct. Process 0 should also print the measured parallel time.

2) (3 points) Suggest a problem size setting for the TA to test within 1 minute if there is a performance gain using 4 or 8 cores compared to 1 core. Or explain if there is no performance gain.

3) (5 points) Also submit a performance report for a large problem size (e.g. $n=1$ millions and $k=100,000$ or any reasonable size) Report the parallel time, speedup, and efficiency when the number of cores used is 4, 8, 16 for each version. Explain if the performance observed makes sense.

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

\[ y = Ax + c \]

\[ x = y \]

Extend the textbook MPI solution for matrix vector multiplication for this problem. Process 0 generates the initial values for $x$ and $c$ vectors and distribute to all processes. Each process generates the initial values for matrix $A$’s elements it owns. Finally Process 0 collects the final result for vector $x$ and print the first 30 elements of vector $x$. The initial value of vector $x$ is 0. The value of vector $c$ is $c[i] = i/n$ when $i$ varies from 0 to $n-1$.

There are two test matrices for $A$ to be used and you are asked to develop two versions of code considering the characteristics of these two matrices.

a) Values for matrix $A$ are $A[i][j] = 0$ if $i=j$ otherwise $-1/n$. Use row-wise block mapping to assign rows to processes.

b) Values for matrix $A$ are $A[i][j] = 0$ if $i > j$ otherwise $-1/n$. Namely $A$ is an upper triangular matrix. Code can be modified since we know the lower triangular elements are always 0. To balance the load, cyclic mapping should be used.
1) **(10 points)** The submitted code should contain a test option with \( n \), \( t \), and matrix type as input.

Process 0 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 and communication cost.

2) **(3 points)** Suggest a problem size setting for the TA to test within 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) **(7 points)** Also submit a performance report for 3 larger problem sizes to assess the speedup, efficiency, scalability, and communication cost paid. For example, consider \( n=25,000, 50,000 \) and \( 100,000 \). \( t=1000 \), or any reasonable size. The number of cores varies from 1 to 32 cores.

7. **(20 points)** Given \( n \) numbers and \( p \) processes, 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 two lists to test.

   a. A randomly generated list. This represents a general situation and the odd-even sort may still have to go through \( p \) phases.
   b. 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, \( 2, 1, 5, 3, 6, 7, 9, 8 \).

Modify the MPI odd-even code from the textbook chapter 3 to 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 the test list choice as input.

Process 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) **(3 points)** Suggest a problem size setting for the TA to test within 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) **(7 points)** Also submit a performance report for 3 large problem sizes (e.g. \( n=10 \) millions, 100 millions, or any reasonable size) before and after the use of phase skipping technique. Assess the difference in parallel time, speedup, efficiency, and scalability when the number of cores used is 4, 8, 16, and 32.