CS451 "Introduction to Parallel and Distributed Computing"
Homework 3

Submission:
- ***Due by 11:59pm of 10/31/2019 (Thursday midnight)***
- ***Late penalty: 20% penalty for each day late***
- ***Please upload your assignment on the Blackboard with the following name: CS451_ LastName_FirstName_HW3***
- ***Please also upload your code on Blackboard***
- ***Please do NOT email your assignment to the instructor and TA!***

1. **Calculation of communication cost**
In this problem you will analyze the performance of distributed memory message communication (lecture 7). Given a 6*6 processor array which are numbered as (i,j) (i,j=1,2,3,4,5,6) for a processor in row i and column j, arranged as a **2-dimensional torus**, and assuming **cut-through routing**, *show the steps* in the following operations and obtain an estimate for the time taken in microseconds assuming the following parameters:
Startup time t_s = 10 microseconds
Per hop time t_h =2 microseconds
Per byte time t_w = 0.2 microseconds
   a) **One-to-all broadcast** of 1000 bytes of data from processor (2, 2) to all the processors.
   b) **All-to-all scatter** of sections of 1000 bytes from each processor to every other processor (i.e. a total of 6*6*1000 bytes of data exchanged overall)

2. **MPI environment**
In this problem, you are required to get yourself familiar with MPI compilation and execution on Chameleon. Please check the file "*Run MPI program on chameleon.pptx*" on the Blackboard for the detailed information. Once you set up your environment for running MPI on chameleon, you can compile and run a sample MPI code (*mpi-pi.c*) on chameleon. In this code, it is process 0 that is going to collect data from other processes. You are required to modify the program to make the process with the largest rank to collect data and output the final result.

3. **Gaussian elimination MPI programming**
In this problem you are asked to write a parallel algorithm using MPI for solving a dense linear equation 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. You had written a shared memory parallel algorithm in previous assignments (using pthread and OpenMP). Now you need to convert the algorithm to a message passing distributed memory version using MPI.

Assume that the data for the matrix A and the right hand-side vector x is available in processor 0. You can generate the data randomly as was done in HW2.

But note that you will need to distribute the data to all other processors. Finally the results will have to be collected into processor 0 which will print the final results.

The algorithm has two parts:

- *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.

(Part a) Write a parallel algorithm using MPI using static interleaved scheduling. The whole point of parallel programming is performance, **so you will be graded partially on the efficiency of your algorithm.**

Suggestions:

- Consider carefully the data dependencies in Gaussian elimination, and the order in which tasks may be distributed.
- 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.
- The algorithm should scale, assuming N is much larger than the number of processors.
- There should be clear comments at the beginning of the code explaining your algorithm.

(Part b) Report running times for 1, 2, 4, 8, 12 and 16 processors. Evaluate the performance of the algorithm for a given matrix size (3000*3000) on the cluster. USE THE MPI TIMERS FOR PORTABILITY, i.e. the MPI_Wtime() calls even though this measures elapsed times and not CPU times.