EECS 358 – Introduction to Parallel Computing
Homework 2 (100 points, 15% of course grade)

EECS 358
G. Memik

DUE: Tuesday, May 16, 2017, 11:59pm

Similar to the first homework, you can work in groups of 2.

Problem 1 [20 pts.]

In this problem, you will analyze the performance of distributed memory message communication (similar to the analysis in Lecture 9). Given a 6x6 processor array, which are numbered as (i,j) for (i,j = 1, 2, 3, 4, 5, 6) for a processor in row i and column j, arranged as a two-dimensional torus[1]. Assuming cut-through routing, show the steps in the following operations and obtain an estimate for the time taken in microseconds using the following parameters:

Startup time t_s = 10 microseconds
Per hop time t_h = 2 microseconds
Per byte transfer time t_w = 0.01 microseconds

The operations are:

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 x 6 x 1000 bytes of data exchanged),
c) Circular shift by a stride of 4 of 1000 bytes (each part) of data among all processors.

Problem 2 [20 pts.]

This problem is meant to familiarize you with MPI and Condor on the computing cluster. You do not need to write any code.

Course Cluster: MPI & Condor

The applications will be started from the 358smp machine you have used for your previous homework. You can login to the 358smp machine and run the MPI program for the PI example.

% ssh 358smp.eecs.northwestern.edu

First, you have to change your Unix C shell startup configuration. Open the **.cshrc.linux** file and add the following two lines:

[1] A torus architecture is depicted in Lecture slides 8.9 through 8.11.

setenv PATH /usr/lib64/mpich/bin:$PATH
setenv MPDIR /usr/lib64/mpich/bin

Note that, you have to do this only once. You may have to source the file (% *source .cshrc.linux*) or logout/login. Now, copy the hw3.tar file to your directory and untar it

% cp ~memik/hw2.tar .
% tar –xvf hw2.tar

Then, compile the MPI program for the 358 machines:

% cd hw2
% mpicc -o cpi cpi.c

There are two files that you need to have in your directory to submit a job on the computing cluster. The first one is mp2script (which you don't need to modify during this homework), the other one is submit.cpi.mpi. This tells the scheduler how many processors are going to be used, the path to your executable, and the paths for the error and log files.

Open the submit.cpi.mpi file and change the arguments variable to point to the cpi in your directory. Finally, submit your job using the condor_submit call:

% condor_submit submit.cpi.mpi

Once you submit a job, you can monitor/query its progress through checking the condor queue:

% condor_q

When your job finishes, you should have cpi.log.XX and cpi.out.XX files in your submission directory where XX is the condor "cluster" (the job ID). The cpi.out.XX file will have the machines that your job with XX job ID has executed on, and then the result of the code.

Report runtimes for 1, 2, 4, 8, and 16 processors for the cpi program. Note that you can change the number of processors by setting the "machine_count = p" on the submit.cpi.mpi file. Do not expect speedups for this toy program.

Problem 3 [60 pts.]

In this problem, you are asked to write a parallel algorithm using MPI for solving a dense linear equations of the form A*x = b, where A is an n x n matrix and x and b are column vectors. You will use Gaussian elimination without pivoting. Note that you have written a shared memory parallel algorithm in Homework 1. Now you need to convert the algorithm to message passing distributed memory version using MPI. Assume that the data for the matrix A and the right hand side vector x is available only in processor 0. You can generate data randomly as was done in Homework 1. But note that you will need to distribute the data to all other processors before the

computation can start. Finally, the results will have to be collected into processor 0, which will print the final result.
The algorithm has two parts:

(a) Gaussian Elimination: The original system of equations is reduced to an upper triangular form

$$U\,x = y$$

where U is a matrix of size N x N in which all elements below the diagonal are zero, which are modified values of the A matrix. In addition, the diagonal elements have the value 1. The column 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.

(b) 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 Aji / Aii sa as to make the element Aji zero.

Part a) [40 points] Write a parallel algorithm using MPI and use 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 requires $O(n^2)$ operations, so you are not expected to parallelize back substitution.
- The algorithm should scale, assuming n is much larger than the number of processors.

Part b) [20 points] Evaluate the performance of the algorithm for a given matrix size (5000x5000) on the computing cluster using condor. Use the MPI timers for portability, i.e. the MPI_Wtime() calls, even though this measures the elapsed time instead of the CPU times.

Report runtimes for 1, 2, 4, 8, and 16 processors.

Do not forget to return your code and the timing results on Canvas.

Please include clear comments at the beginning of the code explaining your algorithm.

Good luck!