CSE 6230/5960
Programming Assignment 4

Due Friday, 4/24/2020, 11:59pm

1. (30 points) **Ping-pong:** Determine the time taken to perform a point-to-point communication between a pair of processors, for message sizes (*Msgsize*) 1, 8, 64, 512, 4096, 32,768, 256K, and 1M elements (double precision floating-point elements occupying 8 bytes each).

   Use a ping-pong test, where two processes (say P0 and P1) pass a message between themselves a number of times:

```
  P0                            P1
barrier                       barrier
start timer                   start timer
repeat niter times            repeat niter times
{                             {
  send(A,P1)                    recv(A,P0)
  recv(B,P1)                    send(A,P0)
  send(B,P1)                    recv(B,P0)
  recv(A,P1)                    send(B,P0)
}                             }
stop timer                    stop timer
time = totaltime/(4*niter)    time =totaltime/(4*niter)
```

   The above pseudo-code is shown using blocking send/recv primitives. Implement your ping-pong code:

   (a) Using blocking MPI communication primitives MPI_Send and MPI_Recv

   (b) Using only standard non-blocking send/receive primitives

   Report performance in Gigabytes/second, dividing total volume of data communicated by the total time measured on P0. Since the CHPC notchpeak cluster is a cluster of SMP nodes, it is of interest to examine difference in message communication time between processes mapped to processors on the same node versus processes mapped to processors on different nodes. This can be done by allocating two processors in different ways (batch execution templates will be provided). In your report, comment on differences in observed performance for within-node and across-node communication.

2. (20 points) You are provided with MPI C code (pa4-ring.c) implementing a ring communication pattern among processes. It suffers from the deadlock problem discussed in class. Find the largest message size that can be run on Notchpeak nodes without causing a deadlock. Fix the problem by using non-blocking MPI send/receive calls. Report the time it takes to perform a full ring communication (i.e., initially sent messages come back to sources after traversing all processes) with 16 processes on a single notchpeak node (use a batch submission for the measurement) for the following message sizes: 1, 8, 64, 512, 4096, 32,768, 256K, and 1M double-precision elements.

3. (50 points) Implement a simple distributed-memory code for matrix-vector multiplication (MV) code by suitably modifying file pa4-mv.c. Report speedup achieved for 2,4,8,16 processes, using a batch job allocating one notchpeak node, and speedup for 24, 32 processes, using 2 nodes. The code performs repeated MV, where the output vector from each MV is element-wise updated and becomes the input vector for the next iteration. For ease of programming, you may use fully replicated copies of all arrays.

```
for (i=0;i<N;i++) {y[i]=0; x[i] = sqrt(1.0*i);}
 for(iter=0;iter<10;iter++)
  {
   for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      y[i] += A[i][j]*x[j];

   for (i=0; i<N; i++) x[i] = sqrt(y[i]);
  }
```

4. (Extra credit: 50 points) Implement the distributed-memory MV algorithm with 2D partitioning discussed in class. Experiment with different problem sizes and number of MPI processes (going beyond 32 processes) to characterize when the 2D-partitioned parallel version is faster than 1D-partitioned one.

Include source code files, along with a report on Canvas including performance data from execution of the codes and explanation/interpretation.