CS 4230/6230
Programming Assignment 4

Due Wednesday, 12/4/2024, 11:59pm

This exercise should be performed on CADE.

1. (30 points) You are provided with MPI C code (**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 without causing a deadlock. Fix the problem by using non-blocking MPI send/receive calls. Report the time it takes (in nanoseconds) to perform a full ring communication (i.e., initially sent messages come back to sources after traversing all processes) with 8 processes, for the following message sizes: 1, 8, 64, 512, 4096, 32,768, 256K, and 1M double-precision elements.

2. (30 points) **Ping-pong:** Determine the time taken to perform a point-to-point communication between a pair of processes, for message sizes (*Msgsize*) 1, 8, 64, 512, 4096, 32Ki, 256Ki, and 1Mi 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, as depicted by the pseudocode below:

   ```
     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)
   ```

   Implement your ping-pong code using blocking MPI communication primitives MPI_Send and MPI_Recv primitives by filling in the “FIXME” in the provided template code **pingpong.c**.

   How does the measured data fit the alpha-beta model for message communication time? Attempt an estimate for alpha and beta (in nanoseconds and nanoseconds/byte, respectively), providing an explanation for your estimate [**Note:** the measured data will not perfectly match the theoretical alpha-beta model, but some clear trends should be apparent. Do your best to fit the alpha-beta model, with an accompanying explanation.]

3. (40 points) Implement a simple distributed-memory code for matrix-multi-vector multiplication (MMV) code by suitably modifying file **mat-multivec-mul.c**. Report speedup achieved for 2,4,8,16 processes. The code performs repeated MMV, 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 be space-inefficient and use fully replicated copies of all arrays on the MPI processes. [Note that MMV is just matrix-matrix multiplication.]

   ```
   for (i=0;i<1024;i++) for (j=0;j<16;j++) {y[i][j]=0; x[i][j] = sqrt(1.0*i+j);}
    for(iter=0;iter<10;iter++)
     {
      for(i=0;i<1024;i++)
       for(k=0;k<1024;j++)
        for(j=0;j<16;j++)
         y[i][j] += A[i][k]*x[k][j];

      for (i=0; i<N; i++) for (j=0;j<16;j++) x[i][j] = sqrt(y[i][j]);
     }
   ```

Include source code files, along with a report including performance data and explanation/interpretation.