In this exercise you are to characterize point-to-point MPI communication performance and use measured data to estimate performance of parallel dense matrix vector multiplication on the OSC Pentium Cluster. Use batch parallel execution for obtaining all performance data (including sequential performance data in problem 3; for larger sizes of the matrix, interactive execution may fail due to memory limits).

1. **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)
   bw = 8*msgsize/time
   ```

   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 Megabytes/second, dividing total volume of data communicated by the total time measured on P0. Use \(niter = 100\) for message sizes up to 32K elements and \(niter=10\) for larger messages.

   The test is to be performed for the following two situations:

   - Both processors are on the same dual-SMP node (#PBS -l nodes=1:ppn=2:ib)
   - The processors are on different cluster nodes (#PBS -l nodes=2:ppn=1)

   Testing of performance within a dual node and across different nodes can be achieved by having the program run on four processors (using #PBS -l nodes=2: ppn=2), with ping-pong being done respectively between 0-1, 0-2, and 0-3. One out of the three processes with ranks \{1,2,3\} will be on the same dual node as the process with rank 0 and the other two will be on a different node. The MPI_Get_processor_name function can be used to determine which processes get mapped to the same dual-node as that with rank 0.
2. **Ring**: Measure communication performance (using only non-blocking send/receive) for processors communicating in a ring. In each iteration, each processor receives a message from one neighbor while sending its message to the other neighbor. The message received at the current step is sent out in the next step. Thus all processors should be simultaneously active in communicating with neighbor processors. After $P$ steps, each processor should have received the message it originally sent to its neighbor. Measure performance for $P=2, 4, 8$ and the same message sizes as for the ping-pong test. Map all processes to distinct nodes (i.e. $ppn=1$).

3. **Sequential matrix-vector multiply**: Characterize the performance (in MFLOPS) for repeated matrix-vector multiply, for $n \times n$ matrices for $n$ ranging from 4 to 1024 (4, 8, 16, 32, 64, 128, 256, 512, 1024). Use the SAXPY form for Fortran; Dot-Product form for C. Use “mpicc -O” or “mpif77 -O” for compiling; do not use any explicit unrolling or tiling. In order to ensure sufficiently high granularity for timing, use $niter$ to be $1024^2/n^2$.

   - Initialize each component of $x$ and $y$ to 1.0
   - Initialize $A[i,j]$ to $((i+j) \mod n)/(n*(n-1)/2)$
   - start timer
   - repeat $niter$ times
     - $y = Ax$
     - $x = Ay$
   - stop timer
   - output min and max components of $x$

4. **Parallel matrix-vector multiply**: 
   - Assuming that no overlap of communication with computation occurs, use the measured data for sequential matrix-vector multiply performance and point-to-point communication performance to predict the size of the smallest square matrix (need not be power-of-two size) that can achieve at least 50% efficiency for parallel matrix-vector multiply (with row-block partitioning) on 4 processors and 8 processors, respectively. Assume each process is mapped to a distinct cluster node and that the P-1 step ring algorithm is used for the all-gather.
   - Implement parallel matrix-vector multiplication using MPI and compare measured and predicted performance (use repetitions as before, to ensure sufficient granularity for accurate timing).
     - Using the implementation of point-to-point ring communication for the all-gather (all-to-all-broadcast).
     - Using an MPI all-gather instead of the point-to-point communication

Submit a report containing analysis, performance data (including the actual outputs from the runs, showing timing data as well as the min and max of vector $x$) and source code. Information about using the OSC Pentium Cluster may be found at http://www.osc.edu/hpc/computing/p4/.