MPI Exercises

1. Simplest MPI Program: Program helloWorld in MPI. Use only 2 processors and start with the template provided, either in Fortran, `helloWorld.f`, or in C, `helloWorld.c`, and modify it to print out the processor rank and number of processors. The program contains instructions and hints embedded in the comments (in ALL CAPS).

2. Basic Message Passing: Modify the existing serial code to sum the elements of a vector by adding mpi calls to produce a parallel version. Use only 2 processors and start with either the serial version in Fortran, `serialsum.f`, or in C, `serialsum.c`, and modify it to pass the upper half of the array to processor 1, sum each half, and combine the answers. The program contains instructions and hints embedded in the comments.

3. Basic Message Passing (Intermediate): Try to generalize the previous example to run with an arbitrary number of processors. Perform the global sum via the fan-in method, whereby at each stage a pair of partial sums are added on half the previous number of processors, until the final sum is achieved. For instance, if the sum is originally split across 8 processors, then the first stage will produce 8 partial sums, the second stage will spread these across 4 processors (leaving the other 4 idle) and perform 4 sums, etc.

4. Basic Message Passing: Modify the existing serial code to compute π by adding mpi calls to produce a parallel version. Use only 2 processors and start with either the serial version in Fortran, `serialpi.f`, or in C, `serialpi.c`, and modify it to divide the work among the processors. The program contains instructions and hints embedded in the comments.

5. Basic Message Passing (Challenge): Perform a filtering operation on a 2D grid with periodic boundary conditions by applying the Laplace operator. That is, update the value of a particular grid point, say $\mathrm{u_{ij}}$, for one iteration as follows:

   $$\mathrm{u_{i,j}(t_{n+1}) = u_{i+1,j}(t_n) + u_{i-1,j}(t_n) + u_{i,j+1}(t_n) + u_{i,j-1}(t_n) - 4.0 * u_{i,j}(t_n)}$$

   Note this algorithm is not stable and will diverge. *class_directory* as your results to the serial version.

6. Basic Message Passing: Simple exercise in functional parallelism. Have one processor compute $\mathrm{e^x}$ using builtin function calls while the other sums the series

   $$\mathrm{e^x = 1 + x + x^2/2 + x^3/3! + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}}.$$

   Stop the series when the next term only contributes some arbitrary fraction to the sum (you choose).

7. Collective Communication: Modify the computation of pi problem to read in the number of intervals (nint) on one processor and broadcast the value to the other processors. Use a collective operation to get the solution rather than the send and receive operations.

8. Collective Communication: Initialize an array with values from 1 to 24. Scatter this out to all processors. Then add the value of 1000*(processor_rank) to each element on that processor. Now gather the results back and print them out. Try this on 2, 3, 4, 6 ... processors. Do you get back what you expect? You can start with the disperse C or Fortran program and modify it or write your own.

9. MPI Communicators: Generate an array using random numbers. Split the tasks into two groups, even and odd processors. Distribute the array to both groups. The odd group should find the minimum value of the array, the even group the maximum value of the array. Use MPI reduce operations to find the extrema across the processors.

10. MPI Derived Datatypes: Create a data type representing a row of an array and distribute a different row to all processes. Now repeat this for a column. (Note: If you do this in Fortran rather than C, then do columns first and then do rows).

11. MPI Derived Datatypes (intermediate): Use derived datatypes to transpose a matrix between two processors.

12. PTP Communication: The jacobi program is similar to the smoothing algorithm discussed in class. What problems does this algorithm have? How would you change it? Implement these changes.

13. PTP Communication (Advanced): Modify the jacobi algorithm to do a block-block (checkerboard) distribution rather than the blocking by columns that it does.

14. PTP Communication (Advanced): Test the following hypothesis regarding communication time for the jacobi algorithm.Hypothesis: Since the communication occurs at block boundaries, communication volume is minimized by the 2D, block-block, partition (second case) which has a better area to perimeter ratio. However, in this partition, each processor communicates with four neighbors, rather than two neighbors in the 1D partition. When the ratio of n/P (n array size, P number of processors) is small, communication time will be dominated by the fixed overhead per message, and the first partition will lead to better performance. When the ratio is large, the second partition will result in better performance.