**CSC 338 Parallel and Distributed Computing**

**Exercise No. 6a, March 27, 2017**

**Calculating a global sum with MPI**

**Goal**

Learn to use collective communication in MPI

**Procedure**

Copy the exercise folder from the class server and open mpi\_global\_sum.c with your favorite text editor. You will have to compile and execute this program from a \*nix or Cygwin command line but you can, if you want to, edit the source code in Windows. You will find the PowerPoint slides for Chapter 3 useful. **NOTE:** You will run your program with 4 processes.

Comments in the skeleton program tell you what to do, step by step. Following is some clarification.

1. global\_n is the number of integers that will be added. These numbers will be read into an array that you will declare in main().
2. local\_n is the number of integers that each process will add; it must be evenly divisible by the number of processes (comm\_sz). These numbers will be held in an array that you will declare in main().
3. You will declare an int to hold the local sum for each process and an int that will hold the global sum in process 0.
4. The MPI boiler plate includes ints for the number of processes (comm\_sz) and the local rank of each process (my\_rank). It also includes calls to MPI\_Init(), MPI\_Comm\_size(), and MPI\_Comm\_rank().
5. Process 0 will do the following:
   1. read the numbers from randnums.txt by calling readfile(), which is provided in the skeleton code. You will pass the filename and the array that will hold all the numbers to readfile().
   2. Call MPI\_Scatter() with the appropriate arguments. MPI\_Scatter() is described on slides 64 and 65 of Chapter\_3.ppt. Remember that send\_buf\_p is a pointer to the array holding all the numbers, send\_count is the number of elements being sent to each process (not the total number of elements!), MPI data types are on slide 18, recv\_buf\_p is a pointer to the local array receiving the numbers, recv\_count is the number of elements to be received (must match send\_count), src\_proc is the rank of the process sending the data, and comm is the communicator used by the processes.
   3. Calculate its own local sum.
   4. Call MPI\_Reduce() with the appropriate arguments. MPI\_Reduce() is described on slide 44. input\_data\_p is a pointer to the variable holding the local sum, output\_data\_p is a pointer to the variable receiving the global sum, count is the number of data elements being passed (1, in this case), and MPI ops are listed on slide 45.
   5. Print the global sum.
6. Other processes will:
   1. Call MPI\_Scatter(); the call should look exactly like the call for process 0.
   2. Calculate the local sum.
   3. Call MPI\_Reduce(); the call should look exactly like the call for process 0 except you can use a different variable for the second argument (output\_data\_p) because that argument is only used in process 0.
7. Compile and execute your program with 4 processes (mpiexec –n 4 ./prog\_name).
8. If you have time, modify the program to execute with 8 processes.