**CSC 338 Parallel and Distributed Computing**

**Exercise No. 6b, April 3, 2017**

**Calculating a global sum with MPI**

**Goal**

Learn to use collective communication with memory allocation in MPI

**Background**

In the previous exercise, you wrote a program to compute a global sum of 40 integers using four processes. In this exercise, you’ll make the program more general by allowing it to sum a different number of integers and use different numbers of processes, although the number of elements to be added will still have to be evenly divisible by the number of processes.

**Procedure**

Copy the exercise folder from the class server and open mpi\_global\_sum2.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.

The program you are given, mpi\_global\_sum2.c, is based on the solution to the previous exercise, a program that reads 40 integers and calculates the sum using four processes. The reason that program is tied to 40 integers and four processes is because global\_n is set to 40, local\_n is set to global\_n/4, and the two arrays (global\_nums and local\_nums) were set to corresponding lengths.

In order to generalize the program, you will have to discover the number of input elements and allocate memory to handle that number of elements. A function (findlen()) has been added to the given program—it returns the number of integers in a text file. Slide number 65, in Chapter\_3.ppt, shows an example function that allocates memory to handle a vector of n doubles then distributes that vector among some number of processes.

Your task is to modify mpi\_global\_sum2.c to discover the number of integers in the file, allocate the appropriate memory, and compute and print the global sum. Compile and test your program using various numbers of processes; you may also want to change the number of integers in the input file. Be aware that the number of elements to be added must be evenly divisible by the number of processes (what happens when you violate that condition?).

While this program is a big improvement on the one that was hard-coded for four processes and 40 integers, it would be better to have a program that can handle arbitrary numbers of processes and integers. If you have time, see if you can find out how to use MPI\_Scatterv() to accomplish that.