Lab #8 – Practice with collective communication

The techniques that we are using in section 3.4 are so important, that we need to be sure we can use them routinely. So today we are going to continue working with collective communication functions in MPI.

First of all, can you describe in your own words what each of these functions does? Think about why you would use one function instead of another.

MPI_Reduce, MPI_Bcast, MPI_Barrier, MPI_Scatter, MPI_Gather

The first task today is to finish the program you were working on yesterday. This was the program that generated random integers and counted how many times a target value appeared. Show me the output of your program before starting the next exercise.

Now, let's write a program that does a little analysis of the game Yahtzee. (Do people on yachts play Yahtzee?) Here, we will simulate the rolling of 5 dice. We want to compute the histogram of the results. In other words, how many times did a sum of 17 appear, a sum of 25 appear, etc., for all possible sum values.

1. This is going to be a parallel program using MPI, so start off your program by including the MPI background functions:
    a. MPI_Init
    b. MPI_Comm_rank
    c. MPI_Comm_size
    d. MPI_Get_processor_name
    e. And finally MPI_Finalize

    At a minimum, we are going to need variables to keep track of "my rank," i.e. my process number, the total number of processes, and the name of the processor we are running on.

    You will also need to declare an array to hold the histogram. In fact, we should have two arrays, one for a local_count to store the values computed by a process, and a total_count that process 0 will report our final totals at the end. How large should our arrays be? Hint: consider the meaning of local_count[10]. Don't be concerned if some of your cells go unused later.

    As we work through the program, you may be adding additional variables, include files and comments.

2. Because this program is going to feature random number generating, be sure to include <stdlib.h>. Immediately after you call the MPI setup functions, initialize the random number generator with the call: srand(time(0)) as we have done in previous programs that used random numbers.

3. We need to interactively ask the user how many trials should be performed per process. The number of processes has already been determined at the command line. Once you have the number of trials per process, make sure all the processes get this number.

   It would be a good idea to have a printf statement that forces each process to identify itself, the processor it resides on, and how many trials it thinks it needs to do. Don't continue unless the output here makes sense.

4. Next, write a loop that sets all of the cells in the local_count array to 0.

5. Write a function called do_trials(). The purpose of this function is for a process to simulate the rolls of dice, add them up and update the histogram. What parameters need to be passed to this function? Make sure that this function is called appropriately from main().

6. At this point, we can assume that all of the processes have computed their local_count histograms. Use the appropriate MPI function to add them all up and put the result into total_count. Note that we are adding entire arrays, not just adding scalar values.

7. Output the results. Print out the histograms from each process as well as grand total histogram. Run your program, and make sure your results are readable, and that the arithmetic was done correctly.

8. Did each process come up with the same histogram? If so, you should change the way the random number seed was set. We should make sure each process has a different seed! How would we change the call to srand() to make sure this is the case?

9. Let's have fun running your program a few times. How many trials do we need before all possible sums are nonzero? How many total trials is your program able to run in 1 minute on your cluster?

10. In addition to a basic histogram of what sums can be achieved, we can also find out how often certain events take place when we roll the five dice. Modify your program so that it also determines how many times these events happen:
    a. Five of a kind
    b. Four of a kind (i.e. exactly 4)
    c. Three of a kind (i.e. exactly 3)

    One nice way to determine whether you have an n-of-a-kind is to count how many of each value has come up in a roll. In other words, by computing a histogram of a single roll. For example, if you roll 5, 2, 3, 2, 6, then the histogram of this roll is (0, 0, 2, 1, 0, 1, 1). You can then scan this histogram to see if the number 3, 4, or 5 is present.

A full house is a situation where one number appears 3 times and another number appears twice. How would you scan your roll's histogram to see if this has happened?

Another thing to look for is a "straight" where all five dice form a continuous sequence as in 1,2,3,4,5 or 2,3,4,5,6. How would you scan your roll's histogram to quickly determine if a straight has occurred?

The next question for you is to decide where these results should be stored by each process and in a grand total. Do you want to add elements to your existing arrays, or create a new array of these miscellaneous results?

Homework Assignment #1, due 9:00am on Tuesday 5/23

I'd like to give you at least 3 homework assignments this term, so that means we ought to do one now. This program will make use of the words2.txt file available on the class Web site. As you work on this program, hopefully some of the techniques you have used recently will come in handy.

1. Write a parallel program that reads this dictionary, and counts how many 9 letter words there are. You must use appropriate MPI functions to delegate this work to the various processes.

2. Your program should have enough diagnostic output so that it's clear that each process is counting its share of 9 letter words.

3. Next, generalize your program so that it counts the number of n letter words, for all legitimate values of n. By doing so we can see which words lengths are more common.

4. Next, modify your program so that for the 9-letter words only, report on the distribution of the number of vowels. In other words, how many 9 letter words have 3 vowels, 5 vowels, etc.

5. Include documentation inside your source file that explains which MPI function(s) you are using and why.

6. Please e-mail me your source code before the deadline. At the beginning of Tuesday's lab I will ask you to run your program for me on your system. Don't procrastinate, because your next homework may follow on the heels of this one.