CS 221 – Lab #11

Today we will experiment with timing our code. We'll make extensive use of a very simple MPI function: MPI_Wtime().

1. Return to the program you worked on earlier. For example, the recent program that calculated shortest distances among thousands of locations. Insert the following line of code immediately after the code that sets up MPI
```
start = MPI_Wtime();
```

2. Next, we are going to select strategic places in your code where we are going to insert more timing probes. Each one will be associated with its own distinct variable name. Consider the following locations:
   a. After reading the input file or generating the (random) input data to start with
   b. After conveying data to the other processes (e.g. Bcast or Scatter)
   c. After those processes have communicated their results (e.g. Reduce or Gather)
   d. The end of the program

   In each case, declare a new variable, and set it equal to the difference between MPI_Wtime() and the start value. This is because we want to keep track of the elapsed time.

3. Add some more variables to your program, so that instead of just measuring the amount of time since the beginning of the program, you also determine the amount of time since the previous checkpoint.

4. At the end of your program, print the values of each of your timing values. Compile your program. Run your program with 1 process. Which portion of your program seems to take the most time?

5. It would be interesting to see how long a single MPI function takes to execute. For example, at this point, you probably have a checkpoint immediately after a Scatter operation. Insert another one immediately before it, so we can see how long a single scattering does.

6. Re-run your program, but this time use 8 processes on 1 processor. Also run your program with 8 processes on 8 processors. Explain how have the times changed.

Matrix computations

7. Next, let's write a new program to practice matrix operations. We want to know how long it takes for us to add two 32x32 matrices of integers. To simplify your work, make the representation have a 1-D array of 32*32 elements, so we'll pretend it's a 2-D array. Then, the "row i, col j" element of a is found at a[32*i + j].

Initialize arrays a and b to contain random values in the range (0, 1, 2, 3) each.

Have your program compute the sum of the two matrices. How long does this operation take? When you parallelize this computation, what is the speedup?

If there is time remaining, repeat the same experiment on matrix multiplication as well.