CSE 160/260 Programming Assignment #1

Due Date: 4/13/99 11:59 PM

Goal:
The goal of this assignment is to introduce you to parallel programming (message-passing model) with MPI. MPI (Message Passing Interface) is a library of routines that will enable you to write a SPMD-style program.

Resources:

- Check out the hello_world example in the $PUBLIC directory. I will go over this simple example in section as well.
- There are several examples in the $MPI/examples directory for you to look at and run.
- MPI information and links:
  http://www-cse.ucsd.edu/classes/sp99/cse160/programming/mpi_info.html
- How to set up your environment before using MPI:
  http://www-cse.ucsd.edu/classes/sp99/cse160/programming/setup.html
- Be sure to check the announcements page often for hints, clarification of assignment, etc.
  http://www-cse.ucsd.edu/classes/sp99/cse160/announcements.html

Assignment description: Ring program
Your program will take a command line input of a string of any length. This string will get passed around to all processes. Process 0 will read in the command line input string and pass it to process 1, process 1 will pass the string to process 2, and so on. Process N - 1 will pass the string back to process 0. Your program should be able to handle any number of processes, N. Your command line should be of the following format:

  mpirun -np <number of processes> -machinefile <file of machine names> ring <input string>

(More information on mpirun is presented in the section below) Hint: work on one step at a time and build your program up in increments.

1. On startup, each process will discover its rank (i.e. a unique process identifier) using MPLComm_rank. For each message the process outputs, it will use its rank to identify itself. For example, the process of rank 2 (or process 2) would output messages like this:

   2:  This is an output message.
   Note: there is one space between the colon and the message

2. Use the call MPLGet_processor_name to find out what machine you are running on. Print out a message in the following format:

   <rank>: Starting up on host <machine name>...

For example, the first message process 1 prints out would be:

   1: Starting up on host uape-1.ucsd.edu...
3. Use the call MPLComm_size to find out the total number of processes (i.e. the value passed in with -np). Use the following format:

<rank>: Number of total processes = <total number of processes>

4. Process 0 will check that there is an input string and that there are at least 2 processes. If there is an error, process 0 will send a message (using MPLBcast) to the other processes telling them to exit. Otherwise, process 0 will send a message to the other processes telling them that the input was correct. In other words, your program should not hang if there is an error.

5. Process 0 will begin passing the string. It will first send the length of the string and then the actual input string (i.e. 2 separate messages) using MPLSend. Each of the remaining processes will receive the length of the input string using MPLRecv, dynamically allocate space for a string of that size, and then receive the input string. Each process should print out the following:

<rank>: Receiving input string length <length> from process <rank of right neighbor>
<rank>: Receiving input string 'input string' from process <rank of right neighbor>
<rank>: Sending input string length <length> to process <rank of left neighbor>
<rank>: Sending input string 'input string' to process <rank of left neighbor>

So, for example, your messages would look something like this:

3: Received input string length 5 from process 2
3: Received input string 'hello' from process 2
3: Sending input string length 5 to process 0
3: Sending input string 'hello' to process 0

6. When process 0 receives the string back (to make it a complete ring), it will print out the time (using MPLWtime) it took for the string to travel around the ring. Process 0 should print out a message of the following format:

<rank>: total time around loop = <time> s

7. Create a barrier before exiting using MPLBarrier. Each process should print out the time that it spent waiting at the barrier in the following format:

<rank>: time waiting at barrier = <time> s

8. Have process 0 print out the average time that each process spent waiting using MPLReduce and process 0 as the root process. The message should be of the following format:

<rank>: avg barrier wait time = <time> s

**Brief description of mpirun:**

To run an MPI program, you should prefix the command line with `mpiexec`. The following command line options will be useful to you when using `mpiexec`:

- `-np number of processes`: Specifies the number of processes you want started up. By default, `number of processes = 1`. 
- `machinefile file of machine names`: Specifies the machines you want the processes to be started up on.

**Brief description of MPI calls that you will use in this program:**

- `MPLInit`: Initializes the MPI environment. Call this before using any MPI calls.
- `MPLFinalize`: Exits the MPI environment. Call this function before exiting your program.
- `MPLComm_rank`: Retrieves the process’s unique identifier.
- `MPLComm_size`: Finds the total number of processes involved in this invocation of the program.
- `MPLGet_processor_name`: Finds the machine name the process is running on.
- `MPLSend`: Sends a blocking message from calling process to a destination process.
- `MPLRecv`: Receives a message from another process.
- `MPLWtime`: Timer.
- `MPLBarrier`: Blocks calling process until all processes have checked in.
- `MPLReduce`: Collects a value from all processes. An operation is performed on all values and returned to root process.

**How to compile your MPI program:**

Copy the makefile from the $PUBLIC directory. Name your program ‘ring.c’ and then type ‘make’.

**Things to try out:**

- What happens when you specify more processes with `-np` than you have machines in `file of machine names`?
- How is the loop time affected by changing the length of the input string?

**Instructions for turning in programming assignment #1:**

You only need to turn in your ring.c file (i.e. don’t turn in your makefile).

**160 students:** Use the turnin program. So, the command to turnin your program would be the following:

```
    turnin -c cs160s ring.c
```

**260 students:** Since there is no turnin program for you to use, cshelp provided us with another solution. There is now a directory 'turnin' that was created for you all at `/net/bintjua/class/sp99/cse260/turnin`. This directory is writable only by those of you in this class and you can only delete files you write in there (so, basically, this should be pretty secure). Rename your ring.c to your login name and stick a copy in this directory.

**Grading:**

*Note: This program will be partially graded by scripts, therefore it is important that the output of your messages matches the formats specified above.*

If your program passes the scripts and you use all of the MPI calls that I asked for, you will receive full credit. Otherwise, I will check your code and assign the appropriate partial credit. If your program is correct, but your output does not match mine, you will be docked 10%.