SIENAcollege
Computer Science

Computer Science 335
Parallel Processing and HPC
Siena College
Fall 2024

Lab 4: Point to Point Communication

Due: 9:00 AM, Monday, September 30, 2024

For this lab, you will write your first real message passing programs using MPI. We focus here on the fundamental building blocks of message passing: sends and receives, known as point-to-point communication.

You may work alone or with a partner on this lab.

Learning goals:

1. To learn about basic MPI point-to-point communication

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `p2p-lab-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

You may answer the lab questions right in the `README.md` file of your repository, or use the `README.md` to provide a link to a Google document that has been shared with your instructor or the name of a PDF of your responses that you would upload to your repository.

Point-to-Point Communication

Pacheco Chapter 3, in Sections 1, 2 and 3, introduces MPI's basic point-to-point communication capabilities. Note that you can obtain copies of all examples from the book's web site, and a copy of the examples has been placed on noreaster in `/home/cs335/pacheco/ipp-source-use`. The ones we are using today are in the `pacheco-ch3` directory in your repository.

First, let's look at Pacheco's `mpi_hello,c` program. This is a more complicated version of a "Hello, World" program than we looked at in the previous lab. Instead of having each process print out its message directly with `printf`, only the process whose `MPI_Comm_rank` is 0 prints anything. All others send a message to process 0.

Question 1: Briefly summarize the concept of a *communicator* in MPI. (1 point)

Question 2: Which communicator is used by this program, and where is is created? (1 point)

Question 3: When this program is run with 4 processes, how many messages are sent/received? (1 point)

Question 4: The program includes one `MPI_Send` and one `MPI_Recv` call. Briefly describe the purpose of each parameter to each of those calls. (6 points)

Pacheco describes what is required for a send to match a corresponding receive. Modify the `mpi_hello.c` program so that one of the parameters to `MPI_Send` does not match the intended corresponding `MPI_Recv`.

Question 5: What happens when you run this modified version? (1 point)

The `mpi_trap1.c` program uses MPI to parallelize the problem of using the trapeziodal rule to calculate an integral of a function between two endpoints. The math isn't our main concern, but it's worth a refresher on this as provided in Pacheco Section 3.2.1.

`mpi_trap2.c` performs the same computation but rather than hard-coding in the parameters for the endpoints of the interval and the number of trapezoids, it prompts for and reads those in at run time.

Question 6: We previously discussed three ways to assign tasks to processes: domain decomposition by slices, domain decomposition by interleaving, and a bag of tasks. Which of these more closely corresponds to the way `mpi_trap1.c` and `mpi_trap2.c` assign tasks to the processes? Briefly justify your answer. (2 points)

Question 7: Briefly describe the "phases" of the computation done by the `mpi_trap2.c` program. Phases would be parts of the program that perform tasks such as interprocess communication, input/output, and solution computation. (5 points)

Gathering Timings

The trapezoidal rule program, if run with a sufficiently large number of trapezoids (*e.g.*, large enough n), can take a significant amount of processing time. Let's measure it!

First, take a look at the `man` page for the `MPI_Wtime` function. This function is commonly used by MPI programs for timings, much in the way we used `gettimeofday` for our previous C programs. `MPI_Wtime` conveniently returns an elapsed time from a fixed point in time in the past, measured in seconds.

> **Practice Program:** Insert timers using `MPI_Wtime` around the computation part of `mpi_trap2.c`. (3 points)

Question 8: Run `mpi_trap2` for 2, 4, 8, 16, 32, and 64 processes on noreaster for a=-100, b=100, n=1000000000. Report your timings. Are the extra processes taking advantage of processor cores and speeding up the computation? At what point do extra processes stop helping? (10 points)

Some Practice

Practice Program: Write a C program using MPI called `mpiexchange.c` that runs for exactly 2 processes. The rank 0 process places an arbitrary `int` value in a variable, sends that value to the rank 1 process, and then receives a new value for the variable from the rank 1 process. Therefore, the rank 1 process should receive that value, modify it in some way (in my solution, I just double it), and sends it back to the rank 0 process. If run with a number of processes not equal to 2, the program should print an error message and exit. Write your program in the `mpiexchange` subdirectory of your repository. (10 points)

Here is my program's output:

```
0 sending 49152 to 1
0 received 98304 from 1
1 received 49152 from 0
1 sending 98304 to 0
```

Submission

Commit and push your code. Make sure your answers to lab questions provided using one of the mechanisms mentioned in the "Getting Set Up" part of the lab.

Grading

This assignment will be graded out of 40 points.

Feature	Value	Score
Question 1	1	
Question 2	1	
Question 3	1	
Question 4	6	
Question 5	1	
Question 6	2	
Question 7	5	
timers	3	
Question 8	10	
`mpiexchange.c`	10	
Total	40	