Lab 11: Message Passing Interface (MPI)

Instructor: Vivek Sarkar

Resource Summary

Course wiki: `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`

Staff Email: comp322-staff@mailman.rice.edu

Clear Login: ssh *your-netid*@ssh.clear.rice.edu and then login with your password

Sugar Login: ssh *your-netid*@sugar.rice.edu and then login with your password

Linux Tutorial visit `http://www.rcsg.rice.edu/tutorials/`

IMPORTANT: Please refer to the tutorial on Linux and SUGAR from Lab 5, as needed. Also, if you edit files on a PC or laptop, be sure to transfer them to SUGAR before you compile and execute them (otherwise you may compile and execute a stale/old version on SUGAR).

As in past labs, create a text file named `lab_11_written.txt` *in the* `lab_11` *directory, and enter your timings and observations there.*

1 MPI Environment Setup

1. Download the lab11.zip file provided on the course wiki, and unzip its contents in the `lab_11/` directory.
2. Run the following command in the `lab_11/` directory to set up the environment for executing mpiJava programs, "*source setup.txt*".

2 Matrix Multiply using MPI-Java

Your assignment today is to fill in incomplete MPI calls in a matrix multiply example that uses mpiJava. You should complete all the necessary MPI calls in `MatrixMult.java`, to make it work correctly. There are comments (TODOs numbered 1 to 14) in the code that will help you with modifying these MPI calls. You can look at the slides for Lectures 32 and 33 for an overview of the mpiJava calls, and at `http://www.hpjava.org/mpiJava/doc/api` for the API details (click on the "Comm" link).

Though MPI is designed for execution on distributed-memory machines, we will create multiple sequential MPI Processes within a single SUGAR node for the purpose of this lab. Thus, all parallelism will stem from the use of multiple MPI processes within a single SUGAR node.

The steps to compile and run the updated `MatrixMult.java` file on the command line are as follows:

1. Compile the program with the Makefile provided: *make*
2. Run the program with the Makefile provided, using 8 processes: *make run8*
3. Repeat with 1, 2 and 4 processes: *make run1*
   make run2
   make run4

What performance differences do you see for different numbers of processes?