Lab 11: Message Passing Interface (MPI)

Instructor: Vivek Sarkar

1 Turning in your lab assignments

We're asking all COMP 322 students to turn in their lab assignments before leaving. You will need to do the following:

1. Create a directory called `lab_11/` in your SUGAR account.
2. Do all your work for today's lab in this directory.
3. Before you leave, create a zip file of your work by changing to the parent directory for `lab_11/` and issuing the following command, " `zip -r lab_11.zip lab_11`".
4. Use the turn-in script to submit the contents of the `lab_11.zip` file as a new `lab_11` directory in your turnin directory. (Transfer the file to your CLEAR account if needed.)

2 Setup on SUGAR

1. Download the lab11.zip file provided on the course wiki, and unzip its contents in the `lab_11/` directory.
2. To request a dedicated *compute node*, you should use the following command (as usual) from a SUGAR login node, "*qsub -q commons -I -V -l nodes=1:ppn=8,walltime=00:30:00*".

   When successful, it will give you a command shell on a dedicated 8-core compute node for your use for 30 minutes at a time. Your home directory is the same on both the login and compute nodes.
3. Run the following command in the `lab_11/` directory to set up the environment for executing mpiJava programs, "*source setup.txt*".

3 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. Also, look at the slides for Lectures 33 and 34 for the syntax of mpiJava calls.

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?