Lab 12: Message Passing Interface (MPI)
Instructor: Vivek Sarkar

Course Wiki: http://comp322.rice.edu

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

Important tips and links

edX site : https://edge.edx.org/courses/RiceX/COMP322/1T2014R

Piazza site : https://piazza.com/rice/spring2016/comp322/home

Java 8 Download : https://jdk8.java.net/download.html

Maven Download : http://maven.apache.org/download.cgi

IntelliJ IDEA : http://www.jetbrains.com/idea/download/

HJ-lib Jar File : https://github.com/habanero-maven/hjlib-maven-repo/raw/mvn-repo/edu/rice/hjlib-cooperative/0.1.9/hjlib-cooperative-0.1.9.jar

HJ-lib API Documentation : http://pasiphae.cs.rice.edu/

HelloWorld Project : https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124

Goals for this lab

- Use MPI to distribute computation across multiple processes.

- Understand the parallelization of matrix-matrix multiply across multiple, separate address spaces.

- Complete an MPI implementation of matrix-matrix multiplication by filling in the correct communication calls.

1   Overview

In this lab you will use OpenMPI's Java APIs to gain experience with distributed computing using MPI. You will complete a dense matrix-matrix multiply implementation by filling in the missing MPI API calls in a partial MPI program.

Lab Projects

The template code and Maven project for this lab are located at:

- https://svn.rice.edu/r/comp322/turnin/S16/*NETID*/lab_12

Please use the subversion command-line client or IntelliJ to checkout the project into appropriate directories locally. For example, you can use the following command from a shell:

```
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S16/NETID/lab_12 lab_12
```

If you plan to submit manually rather than through the autograder, you should also check out the project on NOTS and complete the provided `myjob.slurm` file based on the contained TODOs.

For this lab, you will only be able to test your code on NOTS. It likely will not run locally. Local execution is not supported as this lab depends on compiled third-party binaries and a complex development environment that is only available on NOTS. However, you will still be able to compile locally as long as you import the project dependencies from the provided pom.xml. See step 6 at:

https://wiki.rice.edu/confluence/display/PARPROG/Using+IntelliJ+to+Download+and+Run+lab_1

if you need to recall how to import dependencies from Maven into Intellij.

2   Matrix Multiply using MPI

Your assignment today is to fill in incomplete MPI calls in a matrix multiply example that uses MPI to distribute computation and data across multiple processes. 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 33 and 34 for an overview of the MPI send() and recv() calls, and at http://fossies.org/dox/openmpi-1.10.2/namespacempi.html for the API details.

The provided parallel matrix-matrix multiply example works as follows:

1. The master process (`MPI.COMM_WORLD.getRank() == 0`) gets the size of the matrices to be multiplied and the number of processes to use from the unit tests.

2. Each MPI process allocates its own input matrices (`a`, `b`) and output matrix (`c`).

3. The master process initializes its local copies of each matrix and transmits their contents to all other MPI processes. At the same time the master process also assigns each process a set of matrix rows which that process is responsible for processing.

4. Each MPI process computes the contents of its assigned rows in the final output matrix `c`.

5. The master process collects the results of each worker process back to a single node and shuts down.

3   Tip(s)

- There are only two provided unit tests. One runs a small experiment and prints the input and output matrices to help with debugging. The other processes larger matrices and will be used to verify the performance and correctness of your implementation.

- If you run through the autograder, you should ignore all errors related to the running of correctness tests (because there are none for this lab).

4   Deliverables

Once you have completed the template MPI program by filling in the inter-process communication, submit myjob.slurm to NOTS or submit your code to the autograder for a final run. The teaching staff will want to see some performance improvement from 1 to 2 to 4 to 8 processes.