Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 14: Message Passing
Spring Semester 2020

Assigned on: **25.05.2020** Due by: **01.06.2020**

Overview

This week's assignment is about message passing.

Message passing enables parallel communication between independent processes that do not share memory, but where data is always explicitly transferred between processes. This assignment gives you a hands-on introduction on how to use message passing. In this exercise we will use MPI, the message passing interface library, which is the dominant message passing interface run on super computers with millions of cores. As not every student has a super computer and to remain within the Java language environment, we will be using in this exercise MPJ-Express, an implementation of MPI for Java.
Note: For performance evaluations MPI experiments should be run with high-performance MPI implementations (e.g., MVAPICH) available for C and Fortran.

Exercise 1 – Hello World

For this exercise you are asked to implement a simple "Hello World" program with MPJ Express (http://mpj-express.org/). MPJ express is already part of assignment13.zip, but to use it in eclipse a specific "Run Configuration" must be set up, where the VM arguments are set to "-jar libs/starter.jar -np 2" and the environment variable MPJ_HOME is set to "mpj".

The basic programming constructs you need for writing the "Hello world" program are:

```
public class MPI extends java.lang.Object {
        static Intracomm                COMM_WORLD
        static java.lang.String[]       Init(java.lang.String[] argv)
        static void                     Finalize()
}

public class public class Intracomm {
        public int                      Rank() throws MPIException
        public int                      Size() throws MPIException
}
```

Implement a new class `HelloWorld` which initializes an MPI object, retrieves the 'rank' and 'size' of the current process, and prints in each process:

```
Hi from <rank>
Running with size <size> processes
```

Run this program with 4 concurrent ranks. The expected output is:

```
Hi from <3>
Running with size <4> processes
Hi from <0>
Hi from <2>
Running with size <4> processes
Running with size <4> processes
Hi from <1>
Running with size <4> processes
```

To run MPJ programs from Eclips, right click on HelloWorld.java in Project Explorer → Run As → Run Configurations → Double-click Java Application → Switch to tab Arguments → In the VM arguments field, enter "-jar libs/starter.jar -np 2" → Switch to tab Environment → Select Add → In the Name field, enter "MPJ_HOME" → In the Value field, enter "mpj" → Select OK → Select Apply → Select Run

Exercise 2 – Ping Pong

Implement a new MPI program `PingPong` that is expected to be run with two ranks. “Rank 0” initializes an integer data value to ’zero’ and then sends this data value to process “Rank 1”. “Rank 1” increments the received item by one and sends it back to “Rank 0”. “Rank 0” again increments the data item and sends it back to “Rank 1”. This process is implemented in a loop with `Iterations` iterations.

```
public class public class Intracomm {

        public void Send(java.lang.Object buf,
                         int offset,
                         int count,
                         Datatype datatype,
                         int dest,
                         int tag) throws MPIException

        public Status Recv(java.lang.Object buf,
                           int offset,
                           int count,
                           Datatype datatype,
                           int source,
                           int tag) throws MPIException
}
```

Run your program and measure (use `System.nanoTime`) how long a single “ping-pong” takes on your system? What is the minimal, maximal, median, and mean time of a “ping-pong”?

Run your program again and this time transfer instead a single integer an array of size 128, 1024, 1024^2. How does the size of the data affect the time of a ping-pong? Which property are you measuring with a “ping-pong”? Does the property change with the size of the data transferred?

Exercise 3 – Sieve of Eratosthenes

In this exercise, you distribute the computation of prime numbers using MPI. The Sieve of Eratosthenes computes prime numbers within a range [0, MaxNumber[as follows. First, an array is allocated which keeps track for each number if this number is possibly prime. Initially it is assumed all numbers might be prime (except of 0 and 1). The algorithm preserves the invariant that the first number in the array that is marked as possibly prime is indeed prime. In the initial state, this number if '2', which is the smallest prime number. To eliminate numbers that are not prime, we iterate over the list of numbers and cross out all multiples of the current prime number, then look for the next prime number (smallest number marked as possibly prime), and cross again all numbers out that are multiples of the current prime number. This process continues until a prime number that is larger than $sqrt(MaxNumber)$ is reached. In this exercise, we are only interested in how many prime numbers exist, not their actual values.

Parallelize the sieve of Erathosthenes by assigning each process a partition of the overall number array. Each process initializes its partition of the number array and marks numbers that are multiples of a given prime only within its partition of the number array. To obtain the number of prime numbers, each process counts the primes in its partition of the number array. These individual results are the *reduced* to a single value that is reported by “Rank 0”.

The following functions might be useful:

```
public class public class Intracomm {

        public void Bcast(java.lang.Object buf,
                          int offset,
                          int count,
                          Datatype type,
                          int root) throws MPIException

        public void Reduce(java.lang.Object sendbuf,
                           int sendoffset,
                           java.lang.Object recvbuf,
                           int recvoffset,
                           int count,
                           Datatype datatype,
                           Op op,
                           int root) throws MPIException
}
```

Note: Assume $sqrt(MaxNumber)$ still belongs to “Rank 0”. This allows Rank 0 to distribute the work to all other processes.

Measure the performance of sequential and parallel execution of your MPISieve implementation. How does performance change with the number of processors? What did you expect to see? Can you explain your results? What would you expect to see on a distributed memory super computer?

Exercise 4 – Custom Reduce

Implement a reduction operation, similar to the MPI Reduce function. To make things simple, assume that op is MPJSUM (the ”+” operation) and your function is called on MPI_COMM_WORLD only. Your algorithm should not use more than $O(Plog(P))$ messages.

Submission

In order to receive feedback for your exercises, you need to submit your code to the Git repository. You will find detailed instructions on how to install and set-up Eclipse for use with Git in Exercise 1.

Once you have completed the skeleton, commit it to Git by following the steps described below. For the questions that require written answers, please write them on paper and bring them to the next exercise session where the solutions will be discussed.

- **Check-in your project for the first time**
  - Right click your created project called **assignment14**.
  - In the menu go to **Team**, then click **Share Project**.
  - You should see a dialog "Configure Git Repository". Here, next to the Repository input field click on **Create...**
  - Select a root git directory or your projects that you have created in Execise 1. Note for all your assignments you should use the same directory.
  - click **Finish**.
- **Commit changes in your project**
  - Now that your project is connected to your git repository, you need to make sure that every time you change your code or your report, at the end you commit your changes and send (push) them to the git server.
  - Right click your project called **assignment14**.
  - In the menu go to **Team**, then click **Commit...**.
  - In the Comment field, enter a comment that summarizes your changes.
  - In the Files list, select all the files that you changed and want them to be committed. This typically includes all the Java files but not necessarily all the files (e.g., you dont have to commit setting files of our eclipse installation).
  - Then, click on **Commit** to store the changes locally or **Commit and Push** to also upload them to the server. Note that in order to submit your solution you need to **both** commit and push your changes to the server.
- **Push changes to the git server**
  - Right click your project called **assignment14**.
  - In the menu go to **Team**, then click **Push Branch 'master'**. Note if this is not your fist push you can also use **Push to Upstream** to speed up the process.
  - A new dialog appears, now fill in for the URL field:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG20/<nethz-username>.git`
  - Click **Next**
  - Keep the default values and click **Next**
  - An authentication dialog should appear. Fill in your nethz username and password and click **OK**.
  - Click **Finish** to confirm your changes. Note that eclipse might ask for authentication again.
- **Browse your repository online**
  - you can access and browse the files in your repository online on GitLab at:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG20/<nethz-username>`