

## Assignment 2

**Roll no:** 43144

**Name:** Ruchika Pande

**Title:** Distributed application using Message Passing Interface (MPI).

### Problem Statement:

To develop a calculator application using Message Passing Interface (MPI).

### Objectives:

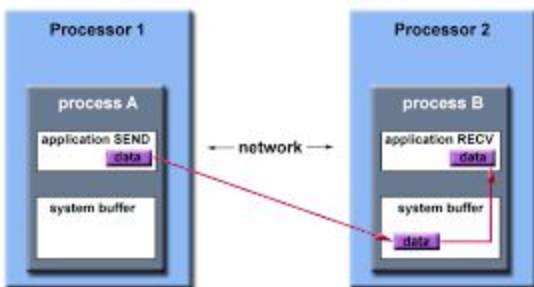
- To study about Message Passing Interface using JAVA

### Theory:

#### Message Passing Interface

Message Passing Interface (MPI) is a standardized and portable message-passing system developed for distributed and parallel computing. MPI provides parallel hardware vendors with a clearly defined base set of routines that can be efficiently implemented. As a result, hardware vendors can build upon this collection of standard low-level routines to create higher-level routines for the distributed-memory communication environment supplied with their parallel machines.

MPI gives user the flexibility of calling set of routines from C, C++, Fortran, C#, Java or Python. The advantages of MPI over older message passing libraries are portability (because MPI has been implemented for almost every distributed memory architecture) and speed (because each implementation is in principle optimized for the hardware on which it runs).



## Advantages of MPI :

- **Standardization:** MPI is the only message passing library which can be considered a standard
- **Portability:** No need to modify your source code when you port your application to a different platform
- **Functionality:** Many routines available to use
- **Availability:** A variety of implementations are available

## Disadvantages of MPI :

- Programmer is responsible:
  - to provide data in another processor
  - to explicitly define how and when data is communicated
  - to synchronize between tasks

## MPJ Express:

MPJ Express is a message passing library that can be used by application developers to execute their parallel Java applications on compute clusters or networks of computers. Compute clusters is a popular parallel platform, which is extensively used by the High Performance Computing (HPC) community for large scale computational work. MPJ Express is essentially a middleware that supports communication between individual processors of clusters. The programming model followed by MPJ Express is Single Program Multiple Data (SPMD). Although MPJ Express is designed for distributed memory machines like networks of computers or clusters, it is possible to efficiently execute parallel user applications on desktops or laptops that contain shared memory or multicore processors.

## Installing MPJ Express:

- Download MPJ Express (mpj.jar) and unpack it.
- Set environment variables MPJ\_HOME and PATH:
- export MPJ\_HOME=/path/to/mpj/
- export PATH=\$MPJ\_HOME/bin:\$PATH
- Create a new working directory for MPJ Express programs. e.g. /mpj-user directory.
- Compile the MPJ Express library: cd \$MPJ\_HOME; ant

## Core Routines :

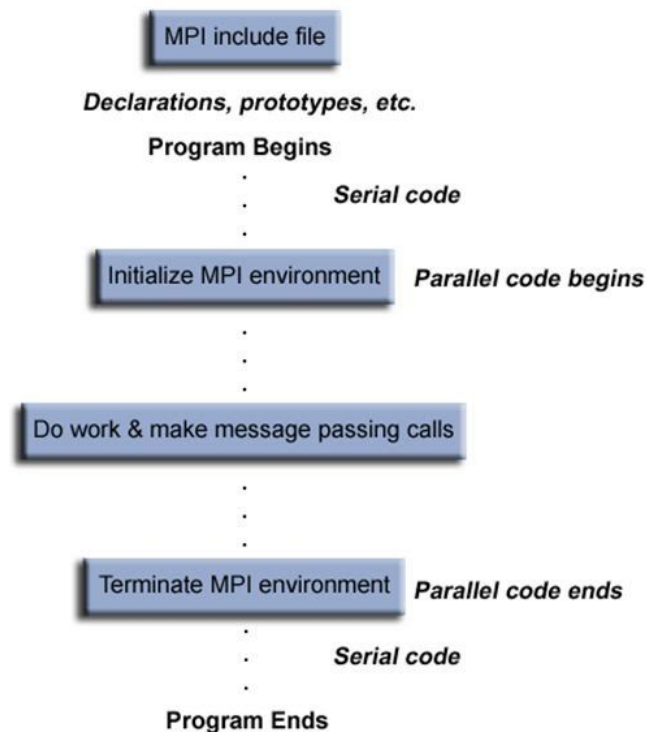
- **MPI.Init:** Initializes the MPI execution environment

- **MPI.Finalize:** Terminates the MPI execution environment
- **MPI.Comm\_size:** Returns the total number of MPI processes in the specified communicator
- **MPI.Comm\_rank:** Returns the rank of the calling MPI process within the specified communicator.

**Predefined Data Types :**

<b>MPI data type</b>	<b>Java data type</b>
MPI.CHAR	signed char
MPI.SHORT	signed short int
MPI.INT	signed int
MPI.LONG	signed long int
MPI.UNSIGNED.CHAR	unsigned char
MPI.UNSIGNED.SHORT	unsigned short int
MPI.UNSIGNED	unsigned int
MPI.UNSIGNED.LONG	unsigned long int
MPI.FLOAT	float
MPI.DOUBLE	double
MPI.LONG.DOUBLE	long double

## MPI Program Structure:



## Steps to write MPI program:

1. Firstly, we will create a file named, "my\_mpi.java".
2. At the beginning of the code, import mpi.MPI.  
import mpi.MPI
3. To import the above package you have to add MPJ.jar during compilation. This will provide support for multicore architecture and creates an MPI environment i.e. it allows to execute parallel java applications.
4. Create a class my\_MPI with main method
5. Root process initializes the sendbuff[ ] data array.
6. When Scatter() method is called, it sends data from the root process to other processes.
7. Each process multiplies the data received by itself.
8. Then root process gathers each individual number with Gather() method.
9. Gather() method takes elements from each process and gathers

them to the root process. The elements are ordered by the rank of the process from which they were received.

**Code:**

```
package mpiTest;

import mpi.MPI;

public class my_MPI {

    public static void main(String args[]) {

        //Initialize MPI execution environment

        MPI.Init(args);

        //Get the id of the process

        int rank = MPI.COMM_WORLD.Rank();

        //total number of processes is stored in size

        int size = MPI.COMM_WORLD.Size();

        int root = 0;

        //array which will be filled with data by root process

        int sendbuf[] = null;

        int sendadd[] = null;

        sendbuf = new int[size];

        //creates data to be scattered

        if (rank == root) {

            sendbuf[0] = 4;

            sendbuf[1] = 8;

            sendbuf[2] = 12;

            sendbuf[3] = 16;

            //print current process number
```

```

    System.out.print("Processor " + rank + " has data: ");
    for (int i = 0; i < size; i++) {
        System.out.print(sendbuf[i] + " ");
    }
    System.out.println();
}

//collect data in recvbuf
int recvbuf[] = new int[1];
// int recvadd[] = new int[1];
//following are the args of Scatter method
    //send, offset, chunk_count, chunk_data_type, recv, offset, chunk_count,
    chunk_data_type, root_process_id
MPI.COMM_WORLD.Scatter(sendbuf, 0, 1, MPI.INT, recvbuf, 0, 1, MPI.INT, root);
// MPI.COMM_WORLD.Scatter(sendbuf, 0, 1, MPI.INT, recvadd, 0, 1, MPI.INT, root);
System.out.println("Processor " + rank + " has data: " + recvbuf[0]);
//System.out.println("Processor "+rank+" is doubling the data");
recvbuf[0] = recvbuf[0] * recvbuf[0];
// recvadd[0] = recvadd[0] + recvadd[0];
//following are the args of Gather method
//Object sendbuf, int sendoffset, int sendcount, Datatype sendtype,
//Object recvbuf, int recvoffset, int recvcount, Datatype recvtype,
//int root)
MPI.COMM_WORLD.Gather(recvbuf, 0, 1, MPI.INT, sendbuf, 0, 1, MPI.INT, root);
// MPI.COMM_WORLD.Gather(recvadd, 0, 1, MPI.INT, sendadd, 0, 1, MPI.INT,
root);
//display the gathered result
if (rank == root) {

```

```

    System.out.println("After multiplying the data by itself");

    System.out.println("Process 0 has data: ");

    for (int i = 0; i < 4; i++) {

        System.out.print(sendbuf[i] + " ");

        //System.out.println("Added data : " + sendadd[i]+ " ");

    }

}

//Terminate MPI execution environment

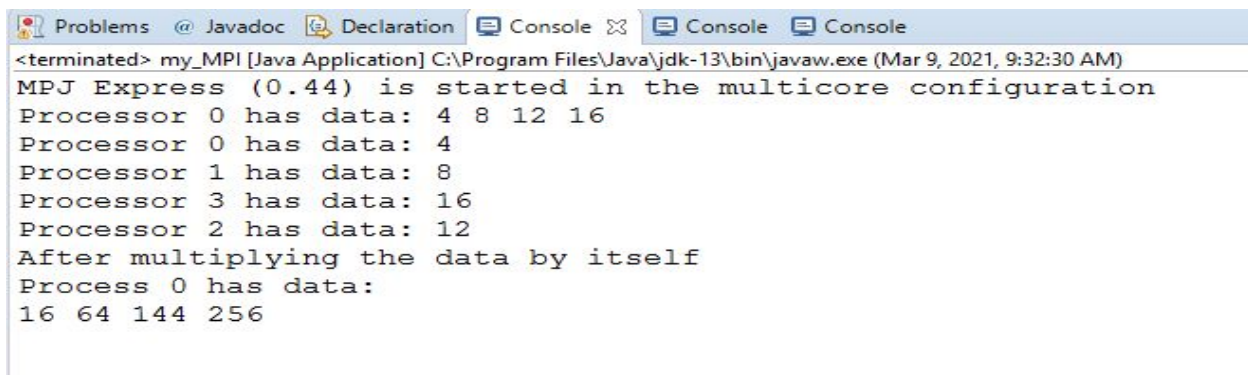
MPI.Finalize();

}

}

```

### Output:



```

<terminated> my_MPI [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Mar 9, 2021, 9:32:30 AM)
MPJ Express (0.44) is started in the multicore configuration
Processor 0 has data: 4 8 12 16
Processor 0 has data: 4
Processor 1 has data: 8
Processor 3 has data: 16
Processor 2 has data: 12
After multiplying the data by itself
Process 0 has data:
16 64 144 256

```

### Conclusion:

In this assignment, we learned about parallel programming using JAVA. We used MPJ express which is a JAVA API for message passing. Thus, we successfully implemented this assignment using mpj express in JAVA.