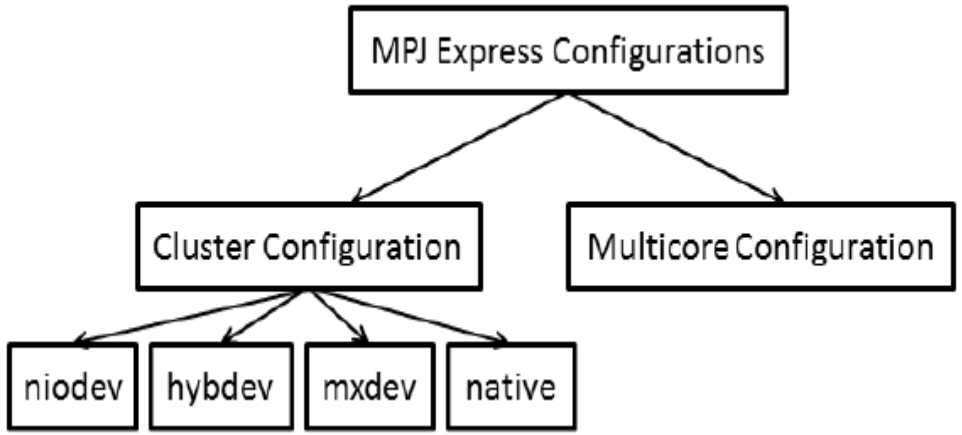
ASSIGNMENT NO. 2	
Assignment No. 2	To develop any distributed application using Message Passing Interface (MPI).
Objective(s):	By the end of this assignment, the student will be able to implement any distributed applications based on MPI.
Tools	MPJ Express Software (Version 0.44), Java 8.

MESSAGE PASSING INTERFACE

- **Message passing** is a popularly renowned mechanism to implement parallelism in applications; it is also called MPI.
- A basic prerequisite for message passing is a good communication API.
- **MPJ Express** is a message passing library that can be used by application developers to execute their parallel Java applications on compute clusters or network of computers.
- **MPJ is a familiar Java API for MPI implementation.**
- 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).
- **MPJ Express** is designed for distributed memory machines like network of computers or clusters, it is possible to **efficiently execute parallel user applications on desktops or laptops t**hat contain shared memory or multicore processors.

MPJ: MPI using JAVA

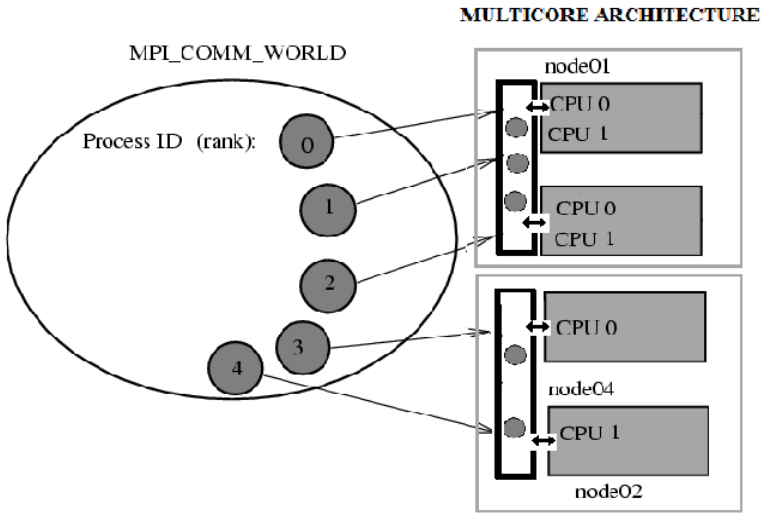
- **MPJ Express Configuration:**
- The MPJ Express software can be configured in two ways as shown in Figure.
- Multicore configuration—is used to execute MPJ Express user programs on laptops and desktops.
- The cluster configuration—is used to execute MPJ Express user programs on clusters or network of computers.



MPJ: MPI using JAVA

- **Multicore configuration:**
- The multicore configuration is meant for users who plan to write and execute parallel Java applications using MPJ Express on their desktops or laptops—typically such hardware contains shared memory and multicore processors.
- In this configuration, users can write their message passing parallel application using MPJ Express and it will be ported automatically on multicore processors.

  Also this configuration is preferred for teaching purposes since students can execute message passing code on their personal laptops and desktops.
  The user applications stay the same when executing the code in multicore or cluster configuration.



GETTING STARTED WITH MPJ Express

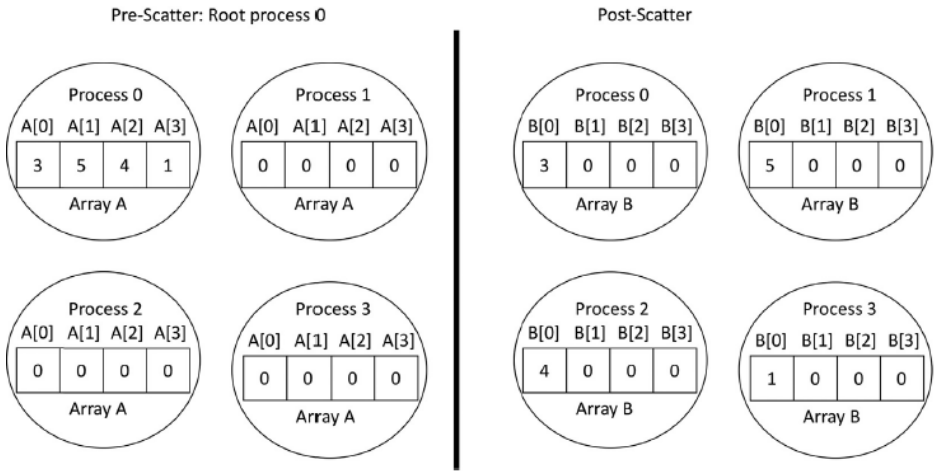
- **Installing MPJ Express:**
- Download MPJ Express (mpj.jar) and unpack it.
- Set environment variables MPJ_HOME and PATH:
  - `export MPJ_HOME=/path/to/mpj/`
  - `export PATH=$MPJ_HOME/bin:$PATH`
- Create a new working directory for MPJ Express programs. e.g. /mpj-user directory.
- Compile the MPJ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

MPI ENVIRONMENT

- MPI is for communication among processes, which have separate address spaces.
- **Group** is the set of processes that communicate with one another.
- **Communicator** is the central object for communication in MPI.
- There is a default communicator whose group contains all initial processes, called **`MPI_COMM_WORLD`**.
- Every MPI program must contain `import mpi.MPI`
- **MPI_Init** initializes the execution environment for MPI.
- A process is identified by its ***rank*** in the group associated with a communicator.
- How many processes are participating in this computation?
  - **`MPI_Comm_size`** function reports the number of processes.
  - **`MPI_Comm_rank`** function reports the *rank*, a number between 0 and size-1, identifying the calling process.
- **MPI_Finalize** cleans up all the extraneous mess that was first put into place by MPI_Init.

COMMUNICATION COLLECTIVES:
Collective Operations in MPI

- *Collective communications* refer to set of MPI functions that transmit data among all processes specified by a given communicator.
- *Scatter-gather:* Ability to “scatter” data items in a message into multiple memory locations and “gather” data items from multiple memory locations into one message.
- In MPI_Gather, each process sends content of send buffer to the root process. Root receives and stores in rank order.
- IN sendbuf starting address of send buffer.
- OUT recvbuf (address of receive buffer).



MPI IMPLEMENTATION

- Firstly, we will create a file named, "`ScatterGather.java`".
- At the beginning of the code, import `mpi.MPI`.

```
import mpi.MPI
```

- 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.
- Create a class `ScatterGather` with main method

```
public class ScatterGather {
    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;
```

MPI IMPLEMENTATION

- Root process initializes the `sendbuff[ ]` data array.

```
//array which will be filled with data by root process
    int sendbuf[]=null;

    sendbuf= new int[size];

//creates data to be scattered
    if(rank==root){
        sendbuf[0] = 10;
        sendbuf[1] = 20;
        sendbuf[2] = 30;
        sendbuf[3] = 40;

    //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();
    }
```

MPI IMPLEMENTATION

- When Scatter() method is called, it sends data from the root process to other processes.

```
//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);
    System.out.println("Processor "+rank+" has data: "+recvbuf[0]);
    System.out.println("Processor "+rank+" is doubling the data");
```

- Each process doubles the data received.

```
//logic for doubling the number
recvbuf[0]=recvbuf[0]*2;
```

MPI IMPLEMENTATION

- Then root process gathers each individual doubled number with `Gather()` method.
- `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.

```
//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);


//display the gathered result
    if(rank==root){
       System.out.println("Process 0 has data: ");
        for(int i=0;i<4;i++){
            System.out.print(sendbuf[i]+ " ");
        }
    }
//Terminate MPI execution environment
    MPI.Finalize();
}
```

STEPS FOR COMPILATION AND EXECUTION

- **Install**ing MPJ Express Programs in the Multicore Configuration

1. Download MPJ Express and unpack it.
2. Set MPJ_HOME and PATH environment variables:

   export MPJ_HOME=/path/to/mpj/

   export PATH=$MPJ_HOME/bin:$PATH

   (These above two lines can be added to ~/.bashrc)

- **Compile:** `ScatterGather.java`

  `javac -cp $MPJ_HOME/lib/mpj.jar ScatterGather.java` (`mpj.jar` is inside lib folder in the downloaded MPJ Express)
- **Execute:** $MPJ_HOME/bin/`mpjrun.sh -np 4 ScatterGather`

EXPECTED OUTPUT

```
pict@ubuntu:~/Documents/Lab/A2$ export MPJ_HOME=/home/pict/Documents/Lab/A2/mpj-v0_44
pict@ubuntu:~/Documents/Lab/A2$ export PATH=$MPJ_HOME/bin:$PATH
pict@ubuntu:~/Documents/Lab/A2$ javac -cp $MPJ_HOME/lib/mpj.jar ScatterGatherTest.java
pict@ubuntu:~/Documents/Lab/A2$ $MPJ_HOME/bin/mpjrun.sh -np 4 ScatterGatherTest
MPJ Express (0.44) is started in the multicore configuration
Processor 0 has data: 10 20 30 40
Processor 0 has data: 10
Processor 0 is doubling the data
Processor 1 has data: 20
Processor 1 is doubling the data
Processor 2 has data: 30
Processor 2 is doubling the data
Processor 3 has data: 40
Processor 3 is doubling the data
Process 0 has data:
20 40 60 80
```

CONCLUSION

- There has been a large amount of interest in parallel programming using Java. mpj is an MPI binding with Java along with the support for multicore architecture so that user can develop the code on it's own laptop or desktop. This is an effort to develop and run parallel programs according to MPI standard.