Practical 1

Distributed Operating System

Name - Ritesh Parkhi

Roll No. – 46

Batch – 3

AIM:

To study basic theory of MPI and MPI commands**.**

THEORY:

MPI stands for Message Passing Interface. MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.

MPI primarily *addresses the message-passing parallel programming model*: data is moved from the address space of one process to that of another process through cooperative operations on each process.

Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be:

- Practical
- Portable
- Efficient
- Flexible

The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.x

Interface specifications have been defined for C and Fortran90 language bindings:

- C++ bindings from MPI-1 are removed in MPI-3
- MPI-3 also provides support for Fortran 2003 and 2008 features

Actual MPI library implementations differ in which version and features of the MPI standard they support. Developers/users will need to be aware of this.

Reasons for Using MPI:

- **Standardization** - MPI is the only message passing library that can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.
- **Portability** - There is little or no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.
- **Performance Opportunities** - Vendor implementations should be able to exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.
- **Functionality** - There are over 430 routines defined in MPI-3, which includes the majority of those in MPI-2 and MPI-1.
    - NOTE: Most MPI programs can be written using a dozen or less routines
- **Availability** - A variety of implementations are available, both vendor and public domain.

Functionality:

The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way, with language-specific syntax (bindings), plus a few language-specific features. MPI programs always work with processes, but programmers commonly refer to the processes as processors. Typically, for maximum performance, each CPU (or core in a multi-core machine) will be assigned just a single process. This assignment happens at runtime through the agent that starts the MPI program, normally called mpirun or mpiexec.

MPI library functions include, but are not limited to, point-to-point rendezvous-type send/receive operations, choosing between a Cartesian or graph-like logical process topology, exchanging data between process pairs (send/receive operations), combining partial results of computations (gather and reduce operations), synchronizing nodes (barrier operation) as well as obtaining network-related information such as the number of processes in the computing session, current processor identity that a process is mapped to, neighbouring processes accessible in a logical topology, and so on. Point-to-point operations come in synchronous, asynchronous, buffered, and ready forms, to allow both relatively stronger and weaker semantics for the synchronization aspects of a rendezvous-send. Many outstanding [clarification needed] operations are possible in asynchronous mode, in most implementations.

MPI-1 and MPI-2 both enable implementations that overlap communication and computation, but practice and theory differ. MPI also specifies thread safe interfaces, which have cohesion and coupling strategies that help avoid hidden state within the interface. It is relatively easy to write multithreaded point-to-point MPI code, and some implementations support such code. Multithreaded collective communication is best accomplished with multiple copies of Communicators, as described below.

MPI COMMANDS:

Although MPI is a complex and multifaceted system, we can solve a wide range of problems using just six of its functions! We introduce MPI by describing these six functions, which initiate and terminate a computation, identify processes, and send and receive messages:

MPI_INIT: Initiate an MPI computation.

MPI_FINALIZE: Terminate a computation.

MPI_COMM_SIZE: Determine number of processes.

MPI_COMM_RANK: Determine my process identifier.

MPI_SEND: Send a message.

MPI_RECV: Receive a message.

MPI PROGRAM:

```
#include <stdio.h>
#include <mpi.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello World\n" );
    MPI_Finalize();
    return 0;
}
```

Compilation: mpicc hello.c -o hello

Run: mpirun -np 4 hello

OUTPUT:

```
Hello World from process 0 of 4
Hello World from process 2 of 4
Hello World from process 3 of 4
Hello World from process 1 of 4
```

CONCLUSION:

Hence we successfully studied about MPI and MPI commands.