**PRACTICAL # 04**

**OBJECT:**

Hello world program using openMPI

**THEORY:**

OpenMPI is the platform for parallel computing. The core of OpenMPI is the Message Passing Interface feature, which is used for communication between the parallel programs executing on different machines.

OpenMPI program starts with the function *MPI\_Init.* This function creates the OpenMPI processes.

*MPI\_Init(&argc, &argv)*

The argc and argv are the optional command line arguments to the OpenMPI program when it is run.

The next important function is MPI\_Comm\_size that takes two parameters, *MPI\_COMM\_WORLD* and *&numprocs*. The first argument is an OpenMPI constant while the second is a user defined integer variable that is filled by the number of processors available in this parallel system.

*MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);*

The id of each of the process is called its rank, which is unique for everyone. To find the rank, MPI\_Comm\_rank is used.

*MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);*

*This function fills the second argument with rank of the current process in the parallel system running this code.*

*Opposite to MPI\_Init is MPI\_Finalize function that terminates the OpenMPI environment and ends the processes.*

*MPI\_Finalize();*

*The MPI\_Send and MPI\_Recv functions have similar arguments except an extra MPI\_Status argument in the MPI\_Recv function.*

*MPI\_Send(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_WORLD);*

The first argument is the data buffer to send. Second argument is the size of this buffer and third argument is the data-type of this buffer. Forth argument is the id of the processes to which message is to be sent.

Fifth argument, the TAG is the optional group id in the OpenMPI world. The last argument, *MPI\_COMM\_WORLD, is the OpenMPI constant.*

*MPI\_Recv(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_WORLD, &stat);*

**code:**

In the first OpenMPI simple example program below, we send a "hello" message to each processor (or processor core), using MPI\_Send function. Each of the parallel processes receives the message, using MPI\_Recv function, and appends its id alongwith string “Reporting for duty” to the message.

*// "Hello World" MPI Test Program*

*#include <iostream>*

*#include <mpi.h>*

*#include <string.h>*

*#define BUFSIZE 128*

*#define TAG 0*

*using namespace std;*

*int main(int argc, char \*argv[])*

*{*

*char idstr[32]; char buff[BUFSIZE];*

*int numprocs, myid, i;*

*MPI\_Status stat;*

*// MPI programs start with MPI\_Init; all 'N' processes exist thereafter*

*MPI\_Init(&argc, &argv);*

*// find out how big the SPMD world is*

*MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);*

*// and this processes' rank is*

*MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);*

*/\* At this point, all processes are running equivalently, the rank*

*distinguishes the roles of the processes in the parallel model, with*

*rank 0 often used specially...*

*\*/*

*if(myid == 0)*

*{*

*printf("%d: We have %d processors\n", myid, numprocs);*

*//send message to other processes*

*for(i=1;i<numprocs;i++)*

*{*

*sprintf(buff, "Hello %d! ", i);*

*MPI\_Send(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_WORLD);*

*}*

*// receive messages from other processes*

*for(i=1;i<numprocs;i++)*

*{*

*MPI\_Recv(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_WORLD, &stat);*

*cout<<myid<< ":"<< buff;*

*}*

*}*

*else // for all other processes*

*{*

*// receive from rank 0:*

*MPI\_Recv(buff, BUFSIZE, MPI\_CHAR, 0, TAG, MPI\_COMM\_WORLD, &stat);*

*sprintf(idstr, "Processor %d Reporting for duty", myid);*

*strncat(buff, idstr, BUFSIZE-1);*

*// send to rank 0:*

*MPI\_Send(buff, BUFSIZE, MPI\_CHAR, 0, TAG, MPI\_COMM\_WORLD);*

*}*

*// MPI programs end with MPI Finalize; this is a weak synchronization point*

*MPI\_Finalize();*

*return 0;*

*}*

Save the above program with .cpp extension.

The command to compile the program on command line is:

***mpic++ program.cpp -o program***

To run the program type:

***mpirun program***

When run with two processors this gives the following output

0: We have 2 processors

0: Hello 1! Processor 1 reporting for duty

In mpirun command, we can also specify the number of processes to create. The syntax is

***mpirun -np 3 program***

The **-np** switch in the above command specifies number of processes to create.

Note: if you get an error like:

“There are not enough slots available in the system to satisfy the 3 slots

that were requested by the application:

program

Either request fewer slots for your application, or make more slots available

for use.”

Then you can use the switch –oversubscribe as below:

***mpirun --oversubscribe -np 3 program***

**ACTIVITIES**

**Activity 1**

Setup the openMPI development environment and execute the above program.

**REVIEW QUESTIONS**

1. What is the difference between a sequential program and a parallel program?
2. What is the purpose of the function MPI\_Comm\_size ?
3. What is meant by the Rank of a processor and why is it used?
4. What is Rank of master processor in openMPI environment?
5. What is purpose of MPI\_Send and MPI\_Recv functions?