NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 8

14th October 2020

Faculty: Dr. Geetha V and Mrs. Thanmayee

NOTE:

1) For each program, you must add a screenshot of the output. Write analysis for each observation.

2) Steps to execute :

mpicc helloworld.c -o hello

mpiexec -n 2 ./hello

n is the number of processes to be launched.

1. MPI “Hello World” program :

```
#include<mpi.h>

#include<stdio.h>

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

{

int size,myrank;

MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&size);

MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

printf("Process %d of %d, Hello World\n",myrank,size);

MPI_Finalize();

return 0;

}
```

2. Demonstration of MPI_Send() and MPI_Recv(). Sending an Integer.

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x,i;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if(myrank==0)
{
x=10;
MPI_Send(&x,1,MPI_INT,1,55,MPI_COMM_WORLD);
}
else if(myrank==1)
{
printf("Value of x is : %d\n",x);
MPI_Recv(&x,1,MPI_INT,0,55,MPI_COMM_WORLD,&status);
printf("Process %d of %d, Value of x is %d\n",myrank,size,x);
printf("Source %d Tag %d \n",status.MPI_SOURCE,status.MPI_TAG);
}
MPI_Finalize();
return 0;
}
```

Modifications:

USE wild cards : **MPI_ANY_SOURCE, MPI_ANY_TAG** in the place of tag and source in **MPI_Recv()**. To check the content in **status** structure.

Note: Add a screenshot of the modified code and output.

3. Demonstration of MPI_Send() and MPI_Recv(). Sending a string.

```
#include <mpi.h>

#include<stdio.h>

#include<string.h>

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

{

char message[20];

int myrank;

MPI_Status status;

MPI_Init(&argc,&argv);

MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

if(myrank==0) /* code for process zero */

{

strcpy(message,"Hello world");

MPI_Send(message,strlen(message)+1,MPI_CHAR,1,10, MPI_COMM_WORLD);

}

else if(myrank==1) /* code for process one */

{

MPI_Recv(message,20,MPI_CHAR,0,10,MPI_COMM_WORLD,&status);

printf("Received : %s\n", message);

}

MPI_Finalize();

return 0;
```

```
}
```

4. Demonstration of MPI_Send() and MPI_Recv(). Sending elements of an array.

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x[50],y[50],i;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if(myrank==0)
{
for(i=0;i<50;i++)
x[i]=i+1;
MPI_Send(x,10,MPI_INT,1,20,MPI_COMM_WORLD);
}
else if(myrank==1)
{
MPI_Recv(y,10,MPI_INT,0,20,MPI_COMM_WORLD,&status);
printf(" Process %d Recieved data from Process %d\n",myrank, status.MPI_SOURCE);
for(i=0;i<10;i++)
printf("%d\t",y[i]);
}
MPI_Finalize();
return 0;
```

```
}
```

5. Demonstration of Blocking Send and Receive with mismatched tags.

Here Send and Receive will be posted by Process 0 and Process 1 respectively. The execution will not complete as the Send and Receive does not have matching tags.

Basically this is a Standard mode of Send and Receive. In the next program you will learn that Send will buffer the data and continue execution but receive will block if matching send is not posted.

```
#include<mpi.h>

#include<stdio.h>

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

{

int size,myrank,x[50],y[50],i;

MPI_Status status;

MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&size);

MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

if(myrank==0)

{

for(i=0;i<50;i++)

x[i]=i+1;

MPI_Send(x,10,MPI_INT,1,10,MPI_COMM_WORLD);

}

else if(myrank==1)

{

MPI_Recv(y,10,MPI_INT,0,1,MPI_COMM_WORLD,&status);

printf(" Process %d Recieved data from Process %d\n",myrank, status.MPI_SOURCE);

for(i=0;i<10;i++)

printf("%d\t",y[i]);
```

```
}
MPI_Finalize();
return 0;
}
```

NOTE: Check in the system monitor if the processes are running. Process name is the name of the executable file. Add the screenshot.

6. MPI_Send() and MPI_Recv() standard mode:

/* Demonstration of Blocking send and receive.*/

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x[10],i,y[10];
MPI_Status status;
MPI_Request request;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if(myrank==0)
{
        for(i=0;i<10;i++)
        {
        x[i]=1;
        y[i]=2;
        }
        MPI_Send(x,10,MPI_INT,1,1,MPI_COMM_WORLD);
```

```
        //Blocking send will expect matching receive at the destination

        //In Standard mode, Send will return after copying the data to the system buffer. The
call will block if the buffer is not available or buffer space is not sufficient.

        MPI_Send(y,10,MPI_INT,1,2,MPI_COMM_WORLD);

        // This send will be initiated and matching receive is already there so the program will
not lead to deadlock

}

else if(myrank==1)

{

        MPI_Recv(x,10,MPI_INT,0,2,MPI_COMM_WORLD,&status);

        //P1 will block as it has not received a matching send with tag 2

        for(i=0;i<10;i++)

        printf("Received Array x : %d\n",x[i]);

        MPI_Recv(y,10,MPI_INT,0,1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);

        for(i=0;i<10;i++)

        printf("Received Array y : %d\n",y[i]);

}

MPI_Finalize();

return 0;

}
```

a) Note down your observation on the content of x and y at Process 1.

b) Explain the importance of tag.

c) Write your analysis about Blocking Send and Receive. Whether it is advantageous?

d) What is the need for Non blocking Send and Receive.