National Institute of Technology Karnataka Surathkal,
Department of Information Technology,
IT 300 - Parallel Computing

Lab 5: 01/10/2018

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;
}
```

Steps to execute :

mpicc helloworld.c

mpiexec -n 2 ./a.out

2. Demonstration of MPI_Send() and MPI_Recv().

Write the syntax of MPI_Send() and MPI_Recv(). Describe the parameters.

```
#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,1,1,MPI_COMM_WORLD);
}
else if(myrank==1)
{
printf("Value of x is : %d\n",x);
MPI_Recv(&x,1,MPI_INT,0,1,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;
}
```

3. Non-Blocking Send and Receive.

Also check the behavior of the program by replacing Isend() and Irecv() with Send() and Recv() respectively.

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x,i;
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)
{
x=10;
MPI_Isend(&x,1,MPI_INT,1,20,MPI_COMM_WORLD,&request); // Tag is different at receiver.
for(i=0;i<5;i++)
MPI_Send(&i,1,MPI_INT,1,i,MPI_COMM_WORLD);
}
else if(myrank==1)
{
printf("Value of x is : %d\n",x);
MPI_Irecv(&x,1,MPI_INT,0,25,MPI_COMM_WORLD,&request);
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);
for(i=0;i<5;i++)
{
MPI_Recv(&i,1,MPI_INT,0,i,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
printf("Received i : %d\n",i);
}
}
MPI_Finalize();
return 0;
}
```

4. MPI_Send() standard mode:

```
/* Demonstration of Blocking send and receive.*/
//  No Deadlock in Standard mode as it uses buffer when necessary.
//  Deadlock occurs if Synchronous mode send is used (MPI_Ssend() instead of MPI_Send())
#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 buffer
        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;
}
```

5. Demonstration of Broadcast operation : MPI_Bcast().

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if(myrank==0)
{
scanf("%d",&x);
}
MPI_Bcast(&x,1,MPI_INT,1,MPI_COMM_WORLD);
printf("Value of x in process %d : %d\n",myrank,x);
MPI_Finalize();
return 0;
}
```