NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY
IT 301 Parallel Computing
LAB 7
28th September 2021
Faculty: Dr. Geetha V Dept of IT

NAME: BHUVANESWAR D
ROLL NO: 191IT107

MPI program 1: Simple Hello World program to find rank and size of communication world. (1 Mark)

```
#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;
}
```

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc helloworld.c -o hello
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 3 ./hello
Process 0 of 3, Hello World
Process 1 of 3, Hello World
Process 2 of 3, Hello World
```

MPI Program 2: MPI_Send() and MPI_Recv() for sending an integer. [Total 3 Marks]
(a) Note down source , destination and tag. (1 Marks)
(b) Modify the program to send the string "PCLAB" and add screenshot of the result. (1 marks)
c) Modify the program to send array of elements and add screenshot of the result. (1 marks)

```
#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;
printf("Process %d of %d, Value of x is %d sending the value x\n",myrank,size,x);
MPI_Send(&x,1,MPI_INT,1,55,MPI_COMM_WORLD);
}
```

```
else if(myrank==1)
{
printf("Value of x is : %d before receive\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;
}
```

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q2.c -o q2
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 3 ./q2
Process 0 of 3, Value of x is 10 sending the value x
Value of x is : 21952 before receive
Process 1 of 3, Value of x is 10
Source 0 Tag 55
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q2_s.c -o q2_s
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 3 ./q2_s
Process 0 of 3, Value of x is PCLAB sending the value x
Value of x is : PCLAB before receive
Process 1 of 3, Value of x is PCLAB
Source 0 Tag 55
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q2_ar.c -o q2_ar
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 3 ./q2_ar
Process 0 of 3, Value of x is sending the value x: 1 2 3 4 5
Value of x is before receive: 1 2 3 4 5
Process 1 of 3, Value of x is 1 2 3 4 5
Source 0 Tag 55
```

MPI Program 3: MPI_Send() and MPI_Recv() with MPI_ANY_SOURCE, MPI_ANY_TAG. Note down the results and write your observation. (2 Marks)

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x,i,y;
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=0;
do{
MPI_Recv(&x,1,MPI_INT,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&
status);
```

```
printf("Process %d of %d, Value of x is %d : source %d  tag %d error %d: \n \
n",myrank,size,x,status.MPI_SOURCE,status.MPI_TAG,status.MPI_ERROR);
}while(x>0);
}
else if(myrank>0)
{
y=myrank%5;
printf("Process %d of %d, Value of y is %d : sending the value y\n",myrank,size,y);
MPI_Send(&y,1,MPI_INT,0,(10+myrank),MPI_COMM_WORLD);
}
MPI_Finalize();
return 0;
}
```

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q3.c -o q3
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 2 ./q3
Process 1 of 2, Value of y is 1 : sending the value y
Process 0 of 2, Value of x is 1 : source 1  tag 11 error 138699120:

^CCtrl-C caught... cleaning up processes
```

Observation:

In the MPI_Recv() function, the source parameter is given the value of MPI_ANY_SOURCE which means accept the data from any source and the tag parameter is given the value of MPI_ANY_TAG which means accept the data with any tag value.

MPI Program 4: MPI_Send() and MPI_Recv() with mismatched tag. Record the result for mismatched tag and also after correcting tag value of send receive as same number (2 Marks)

```
#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);
printf("Verifying mistag send and receive\n");
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;
}
```

Mismatched tag:

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q4.c -o q4
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 2 ./q4
Verifying mistag send and receive
Verifying mistag send and receive
```

CORRECTED TAG:

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 2 ./q4
Verifying mistag send and receive
Verifying mistag send and receive
 Process 1 Recieved data from Process 0
1       2       3       4       5       6       7       8       9       10
```

MPI Program 5: MPI_Send() and MPI_Recv() standard mode:
Note down your observation on the content of x and y at Process 1 and Explain the importance of tag. (2 marks)

```
/* 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;
}
```

```
bhuvan@bhuvan-N550JK:~/Desktop$ mpicc q5.c -o q5
bhuvan@bhuvan-N550JK:~/Desktop$ mpiexec -n 3 ./q5
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array x : 2
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
Received Array y : 1
```

Explaination:
In standard mode, when P1 posts MPI_Recv(), it blocks until x is available in its application buffer. But since
the tags of send and recv are mismatched, P1 enters into deadlock. The 1st MPI_Send() in P0, returns as the
contents of x would reside in the system buffer. Then it posts the 2nd Send(). This tag matches with 1st receive in
P1. So x in P1 would recieve y values and Recv() returns and P1 resumes with 2nd Recv(). This tag matches with
the 1st Send() from P0. So this also executes. Y in P1 has contents of x in P0 and x in P1 Has contents of y in P0.

Tag is a non-negative integer assigned by the programmer to uniquely identify a message. Send and receive
operations should have matching message tags. Mismatch could lead to corrupt data, deadlock.