NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 9

21st October 2020

Faculty: Dr. Geetha V and Mrs. Thanmayee

---

1. Non-Blocking Send and Receive.

```
#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.

printf(“Send returned immediately\n”);

}

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(“Receive returned immediately\n”);
```

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

Observation:

a) Note the difference between standard mode send and non blocking send.

b) Note the observation by placing MPI_Wait() routine after MPI_Isend() and MPI_Irecv().

2. 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;
```

}

3. Demonstration of MPI_Reduce with Sum Operation

- **You may use MPI_PROD to get product of elements in each process.**
- **You may also try using array of elements instead of single element x.**
- **Try to understand the working of Reduce.**

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,i,x,y;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
x=myrank; // Note the value of x in each process.
MPI_Reduce(&x,&y,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
        if(myrank==0)
        {
        printf("Value of y after reduce : %d\n",y);
        }
MPI_Finalize();
return 0;
}
```

4. Demonstration of MPI_Gather():

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

```
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x=10,y[5],i;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
MPI_Gather(&x,1,MPI_INT,y,1,MPI_INT,0,MPI_COMM_WORLD); // Value of x at each process is copied to array y in Process 0
if(myrank==0)
{
for(i=0;i<size;i++)
printf("\nValue of y[%d] in process %d : %d\n",i,myrank,y[i]);
}
MPI_Finalize();
return 0;
}
```

5. Demonstration of MPI_Scatter()

- **Note that the program is hard coded to work with 4 processes receiving two chunks from the array.**
- **You may change according to what you want to explore.**

```
#include<mpi.h>
#include<stdio.h>
int main(int argc,char *argv[ ])
{
int size,myrank,x[8],y[2],i;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if(myrank==0)
{
printf("Enter 8 values into array x:\n");
```

```
for(i=0;i<8;i++)
scanf("%d",&x[i]);
}
MPI_Scatter(x,2,MPI_INT,y,2,MPI_INT,0,MPI_COMM_WORLD);
for(i=0;i<2;i++)
printf("\nValue of y in process %d : %d\n",myrank,y[i]);
MPI_Finalize();
return 0;
}
```

6. Write an MPI program to find the smallest element in a given array of size N.

- Try to find out how many processes you may need for parallel computation based on N.
- Use MPI_Reduce routine. Identify which routine you would use to find the minimum number in a given array.