**Distributed Memory Parallelism with MPI**

**NOTE:**

**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().**

#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;

} Modifications:

You can try using 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.

**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); **// no status information**

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;

}

**6. 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;

}

**7. 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;

**}**

**8. 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;

}

**9. 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.

**10. In a smart agriculture system in a large area like a state, sensors are deployed to collect temperature and humidity. The sensed information are stored in a server in the cloud. A query on calculating the average temperature and average humidity of the complete state needs the processing of 10 lakh data elements. Write a parallel program using MPI in which N number of processes run in parallel to calculate the average of 10 lakh elements stored in an array, in order to improve response time.**

• Note: You may use number of elements to be smaller than 10 lakh for testing, as you have to initialize that many elements.