**ARYAMAN MISHRA**

**19BCE1027**

**LAB 8**

1. **Simple MPI Send and Recv**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char\*\* argv) {

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Find out rank, size

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// We are assuming at least 2 processes for this task

if (world\_size < 2) {

fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

int number;

if (world\_rank == 0) {

// If we are rank 0, set the number to -1 and send it to process 1

number = -1;

MPI\_Send(

/\* data = \*/ &number,

/\* count = \*/ 1,

/\* datatype = \*/ MPI\_INT,

/\* destination = \*/ 1,

/\* tag = \*/ 0,

/\* communicator = \*/ MPI\_COMM\_WORLD);

} else if (world\_rank == 1) {

MPI\_Recv(

/\* data = \*/ &number,

/\* count = \*/ 1,

/\* datatype = \*/ MPI\_INT,

/\* source = \*/ 0,

/\* tag = \*/ 0,

/\* communicator = \*/ MPI\_COMM\_WORLD,

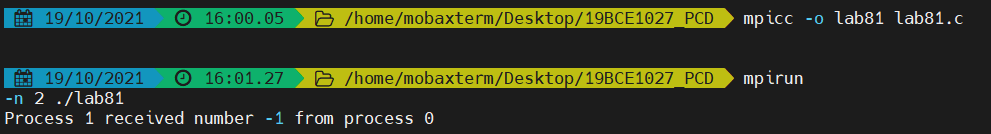
/\* status = \*/ MPI\_STATUS\_IGNORE);

printf("Process 1 received number %d from process 0\n", number);

}

MPI\_Finalize();

}



#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char\*\* argv) {

const int PING\_PONG\_LIMIT = 10;

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Find out rank, size

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// We are assuming 2 processes for this task

if (world\_size != 2) {

fprintf(stderr, "World size must be two for %s\n", argv[0]);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

int ping\_pong\_count = 0;

int partner\_rank = (world\_rank + 1) % 2;

while (ping\_pong\_count < PING\_PONG\_LIMIT) {

if (world\_rank == ping\_pong\_count % 2) {

// Increment the ping pong count before you send it

ping\_pong\_count++;

MPI\_Send(&ping\_pong\_count, 1, MPI\_INT, partner\_rank, 0, MPI\_COMM\_WORLD);

printf("%d sent and incremented ping\_pong\_count %d to %d\n",

world\_rank, ping\_pong\_count, partner\_rank);

} else {

MPI\_Recv(&ping\_pong\_count, 1, MPI\_INT, partner\_rank, 0, MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE);

printf("%d received ping\_pong\_count %d from %d\n",

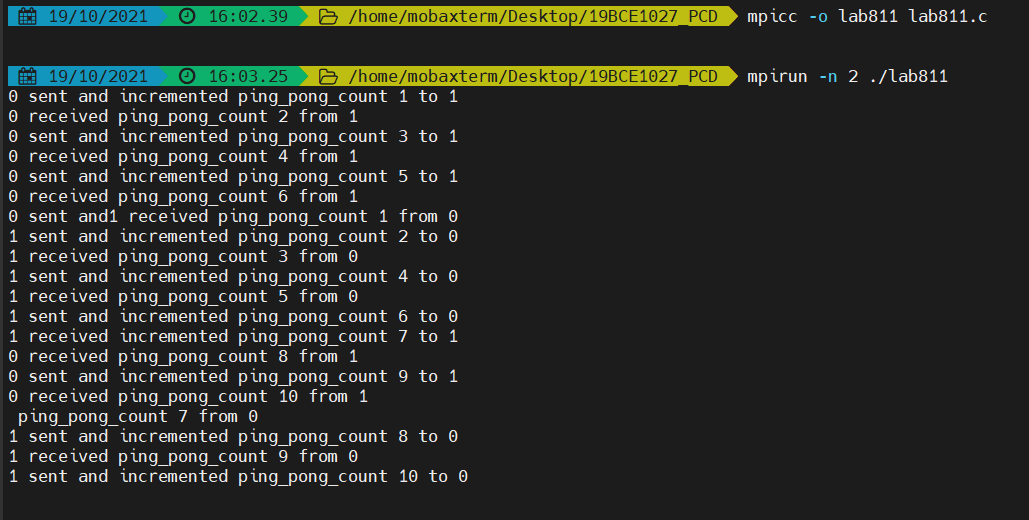
world\_rank, ping\_pong\_count, partner\_rank);

}

}

MPI\_Finalize();

}



1. **Simple MPI iSend and iRecv**

#include<stdio.h>

#include<stdlib.h>

#include"mpi.h"

//int MPI\_Isend(void \*buf,int count , MPI\_Datatype datatype,int dest , int tag ,MPI\_Comm comm,MPI\_Request \*request);

//int MPI\_Irecv(void \*buf,int count , MPI\_Datatype datatype,int source , int tag , MPI\_Comm comm,MPI Request \*request)

//int MPI\_Wait(MPI\_Request \*request , MPI\_Status \*status)

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

{

int numtasks,rank,next,prev,buf[2],tag1=1,tag2=2;

tag1=tag2=0;

MPI\_Request reqs[4];

MPI\_Status stats[4];

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&numtasks);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

prev=rank-1;

next=rank+1;

if(rank==0)

prev=numtasks-1;

if(rank==numtasks-1)

next=0;

MPI\_Irecv(&buf[0],1,MPI\_INT,prev,tag1,MPI\_COMM\_WORLD,&reqs[0]);

MPI\_Irecv(&buf[1],1,MPI\_INT,prev,tag2,MPI\_COMM\_WORLD,&reqs[1]);

MPI\_Isend(&rank,1,MPI\_INT,next,tag2,MPI\_COMM\_WORLD,&reqs[2]);

MPI\_Isend(&rank,1,MPI\_INT,next,tag1,MPI\_COMM\_WORLD,&reqs[3]);

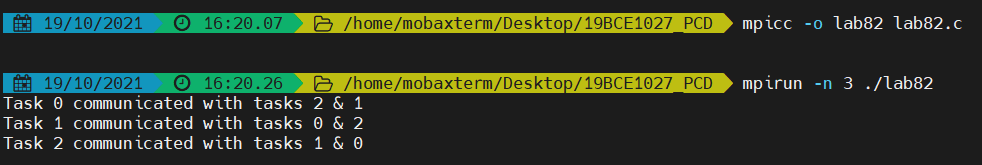
MPI\_Waitall(4,reqs,stats);

printf("Task %d communicated with tasks %d & %d\n",rank,prev,next);

MPI\_Finalize();

return 0;

}



1. **Simple MPI Bcast**

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

int rank;

int buf;

const int root=0;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if(rank == root) {

buf = 777;

}

printf("[%d]: Before Bcast, buf is %d\n", rank, buf);

/\* everyone calls bcast, data is taken from root and ends up in everyone's buf \*/

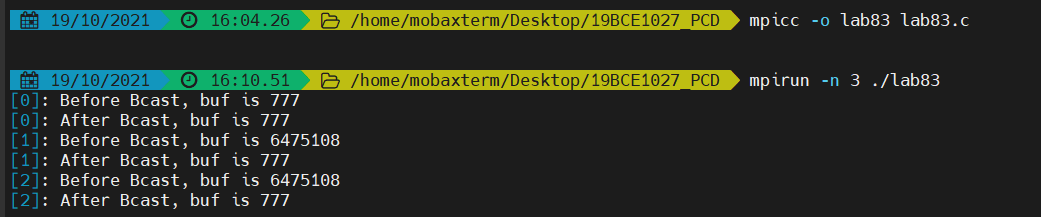
MPI\_Bcast(&buf, 1, MPI\_INT, root, MPI\_COMM\_WORLD);

printf("[%d]: After Bcast, buf is %d\n", rank, buf);

MPI\_Finalize();

return 0;

}



1. **Implementation of Bcast using MPI send and recv**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

void my\_bcast(void\* data, int count, MPI\_Datatype datatype, int root,

MPI\_Comm communicator) {

int world\_rank;

MPI\_Comm\_rank(communicator, &world\_rank);

int world\_size;

MPI\_Comm\_size(communicator, &world\_size);

if (world\_rank == root) {

// If we are the root process, send our data to everyone

int i;

for (i = 0; i < world\_size; i++) {

if (i != world\_rank) {

MPI\_Send(data, count, datatype, i, 0, communicator);

}

}

} else {

// If we are a receiver process, receive the data from the root

MPI\_Recv(data, count, datatype, root, 0, communicator, MPI\_STATUS\_IGNORE);

}

}

int main(int argc, char\*\* argv) {

MPI\_Init(NULL, NULL);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int data;

if (world\_rank == 0) {

data = 100;

printf("Process 0 broadcasting data %d\n", data);

my\_bcast(&data, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

} else {

my\_bcast(&data, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Process %d received data %d from root process\n", world\_rank, data);

}

MPI\_Finalize();

}

