

PDC LAB 10

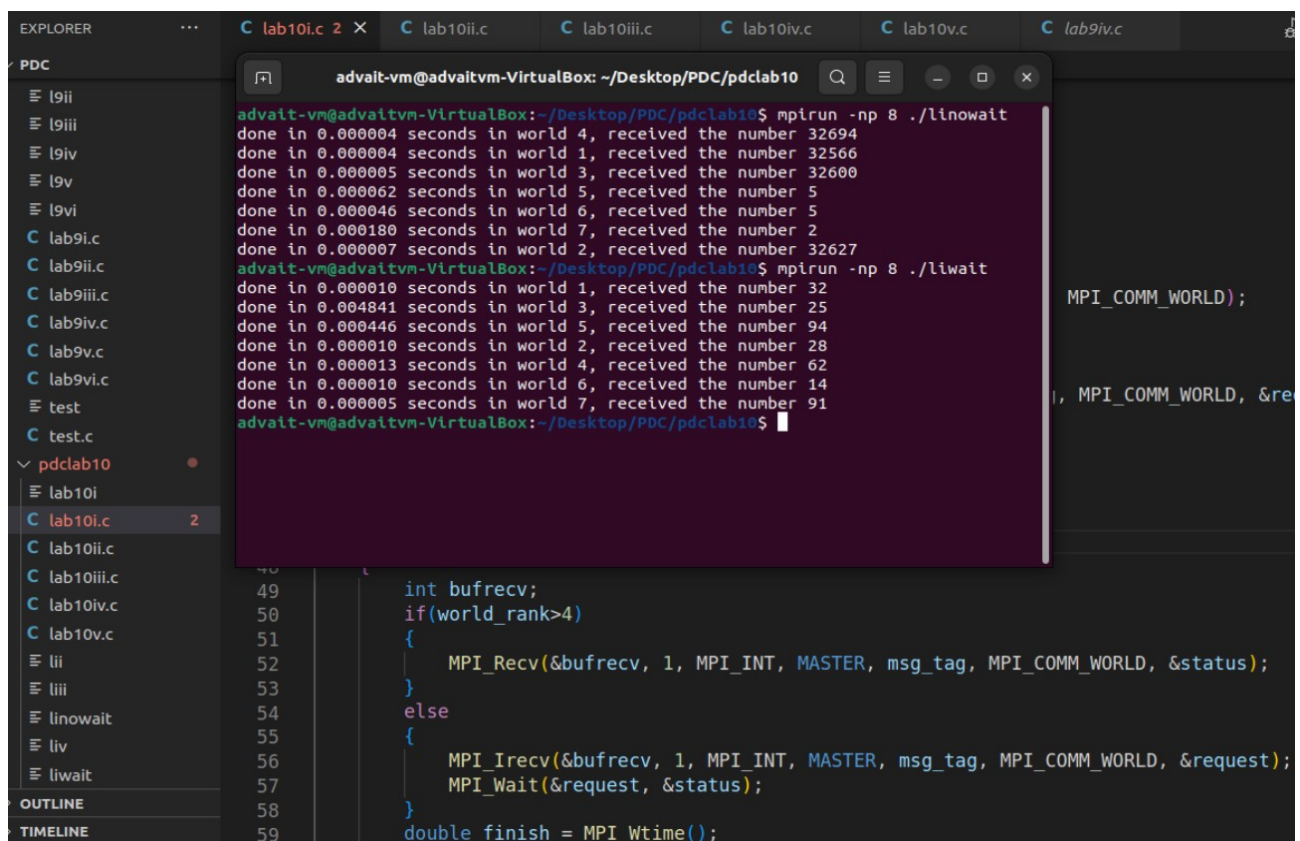
Advait Deochakke

20BCE1143

Sample isend, ireceive with mpi_wtime

Sample send and receive with mpi_wtime

Both combined in one, easily showcasing the difference



The screenshot shows a Visual Studio Code editor with a terminal window and a C code file. The terminal window displays the output of two MPI programs. The first program, `linowait`, uses `MPI_Send` and `MPI_Recv` without `MPI_Wait`, resulting in a race condition where data is received before it is sent. The second program, `liwait`, uses `MPI_Irecv` and `MPI_Wait`, ensuring that data is received only after it has been sent. The C code file, `lab10i.c`, shows the implementation of these two programs.

```
advait-vm@advaitvm-VirtualBox: ~/Desktop/PDC/pdclab10
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$ mpirun -np 8 ./linowait
done in 0.000004 seconds in world 4, received the number 32694
done in 0.000004 seconds in world 1, received the number 32566
done in 0.000005 seconds in world 3, received the number 32600
done in 0.000062 seconds in world 5, received the number 5
done in 0.000046 seconds in world 6, received the number 5
done in 0.000180 seconds in world 7, received the number 2
done in 0.000007 seconds in world 2, received the number 32627
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$ mpirun -np 8 ./liwait
done in 0.000010 seconds in world 1, received the number 32
done in 0.004841 seconds in world 3, received the number 25
done in 0.000446 seconds in world 5, received the number 94
done in 0.000010 seconds in world 2, received the number 28
done in 0.000013 seconds in world 4, received the number 62
done in 0.000010 seconds in world 6, received the number 14
done in 0.000005 seconds in world 7, received the number 91
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$
```

```
int bufrecv;
if(world_rank>4)
{
    MPI_Send(&bufrecv, 1, MPI_INT, MASTER, msg_tag, MPI_COMM_WORLD);
}
else
{
    MPI_Irecv(&bufrecv, 1, MPI_INT, MASTER, msg_tag, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);
}
double finish = MPI_Wtime();
```

In the above example, we can also see that not using `MPI_Wait` with `Isend` and `Irecv` can lead to instantly sending data which may or may not be ready for sending and receiving (`linowait`)

Code ->

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

```

#include<stdlib.h>
#include<time.h>

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

int main()
{

    MPI_Status status;
    MPI_Request request = MPI_REQUEST_NULL;

    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    double start = MPI_Wtime();

    int msg_tag=1729;

    if(world_rank==0)
    {
        srand(time(NULL));
        int buffer[100]={0};
        for(int dest=1; dest<world_size; dest++)
        {
            buffer[dest]=rand()%100;
            if(dest>4)
            {
                MPI_Send(&buffer[dest], 1, MPI_INT, dest, msg_tag,
MPI_COMM_WORLD);
            }
            else
            {
                MPI_Isend(&buffer[dest], 1, MPI_INT, dest, msg_tag,
MPI_COMM_WORLD, &request);
                MPI_Wait(&request, &status);
            }
        }
    }
    else
    {
        int bufrecv;
        if(world_rank>4)
        {

```

```

        MPI_Recv(&bufrecv, 1, MPI_INT, MASTER, msg_tag, MPI_COMM_WORLD,
&status);
    }
    else
    {
        MPI_Irecv(&bufrecv, 1, MPI_INT, MASTER, msg_tag, MPI_COMM_WORLD,
&request);
        MPI_Wait(&request, &status);
    }
    double finish = MPI_Wtime();
    printf("done in %f seconds in world %d, received the number %d\n",
finish-start, world_rank, bufrecv);
}

```

```

MPI_Finalize();

return 0;

}

```

Implementing the broadcast using send and receive
(also implementing MPI_Broadcast and checking time diff)

The screenshot shows the Visual Studio Code interface with a C program in `lab10ii.c` and its execution output in the terminal. The program implements a broadcast using `MPI_Irecv` and `MPI_Wait` for slave processes, and `MPI_Bcast` for the master process. The terminal output shows the execution of `mpirun -np 8 ./lii` on a system with 8 processes. The master process (rank 0) starts the broadcast, and the slave processes (ranks 1-7) receive the data. The output shows the time taken for each process to finish and the value received.

```

lab10ii.c - PDC - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
PDC
  l9iv
  l9v
  l9vi
  lab9i.c
  lab9ii.c
  lab9iii.c
  lab9iv.c
  lab9v.c
  lab9vi.c
  test
  test.c
  pdclab10
    lab10i
    lab10i.c
    lab10ii.c
    lab10iii.c
    lab10iv.c
    lab10v.c
    lii
    liii
    linowait
    liv
    liwait
    lv
OUTLINE
TIMELINE

```

```

advait-vm@advaitvm-VirtualBox: ~/Desktop/PDC/pdclab10
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$ mpirun -np 8 ./lii
time taken to finish the full set of scuffcast 0.000041
master here, starting real MPI_Bcast. Check by value updating to 0
i am world 0, my buffer value is 0, and i took 0.000036 secs
done in 0.000004 seconds in world 1, received the scuffcast. My number is 4
i am world 1, my buffer value is 0, and i took 0.000020 secs
done in 0.000004 seconds in world 2, received the scuffcast. My number is 54
i am world 2, my buffer value is 0, and i took 0.000033 secs
done in 0.000012 seconds in world 3, received the scuffcast. My number is 60
i am world 3, my buffer value is 0, and i took 0.000143 secs
done in 0.006842 seconds in world 4, received the scuffcast. My number is 71
i am world 4, my buffer value is 0, and i took 0.000040 secs
done in 0.000014 seconds in world 5, received the scuffcast. My number is 80
i am world 5, my buffer value is 0, and i took 0.004083 secs
done in 0.000464 seconds in world 6, received the scuffcast. My number is 37
i am world 6, my buffer value is 0, and i took 0.000309 secs
done in 0.000005 seconds in world 7, received the scuffcast. My number is 17
i am world 7, my buffer value is 0, and i took 0.000013 secs
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$

```

```

46     printf("time taken to finish the full set of scuffcast %f\n", finish-start);
47     printf("master here, starting real MPI_Bcast. Check by value updating to 0\n");
48     for(int i=0; i<world_size; i++)
49         buffer[i]=0;
50 }
51 else
52 {
53     MPI_Recv(&buffer[0], world_size, MPI_INT, MASTER, msg_tag, MPI_COMM_WORLD, &statu
54     double finish = MPI_Wtime();
55     printf("done in %f seconds in world %d, received the scuffcast. My number is %d\n
56 }
57
58 double start1=MPI_Wtime();
59
60 MPI_Bcast(&buffer[0], world_size, MPI_INT, MASTER, MPI_COMM_WORLD);

```

Here, 'scuffcast' refers to MPI_Broadcast being implemented using MPI_Send and MPI_Recv in a pseudo manner

whereas, the second line which mentions buffer value being zero corresponds to real MPI_Bcast

Code ->

```
//implementing MPI_Bcast with MPI_Send/MPI_Recv
//then comparing with MPI_Bcast

#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<stdbool.h>
#include<unistd.h>

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

int main()
{
    MPI_Status status;
    MPI_Request request = MPI_REQUEST_NULL;

    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    double start = MPI_Wtime();

    int msg_tag=1729;
    int buffer[100]={0};

    if(world_rank==0)
    {
        srand(time(NULL));
        for(int i=0; i<world_size; i++)
            buffer[i]=rand()%100;

        for(int dest=1; dest<world_size; dest++)
```

```

    {
        //sends to all (including itself in MPI_Bcast)
        MPI_Send(&buffer[0], world_size, MPI_INT, dest, msg_tag,
MPI_COMM_WORLD);
    }
    double finish = MPI_Wtime();
    printf("time taken to finish the full set of scuffcast %f\n", finish-
start);
    printf("master here, starting real MPI_Bcast. Check by value updating
to 0\n");
    for(int i=0; i<world_size; i++)
        buffer[i]=0;
}
else
{
    MPI_Recv(&buffer[0], world_size, MPI_INT, MASTER, msg_tag,
MPI_COMM_WORLD, &status);
    double finish = MPI_Wtime();
    printf("done in %f seconds in world %d, received the scuffcast. My
number is %d\n", finish-start, world_rank, buffer[world_rank]);
}
double start1=MPI_Wtime();

MPI_Bcast(&buffer[0], world_size, MPI_INT, MASTER, MPI_COMM_WORLD);

double finish1=MPI_Wtime();

printf("i am world %d, my buffer value is %d, and i took %f secs\n",
world_rank, buffer[world_rank], finish1-start1);
MPI_Finalize();

return 0;
}

```

Ring communication

The image shows a Visual Studio Code editor window with the following components:

- File Explorer (Left):** Displays the project structure. The file `lab10iii.c` is selected under the `pdclab10` folder.
- Code Editor (Center):** Shows the source code for `lab10iii.c`. The code is a C program that uses MPI to communicate between 8 processes. It includes headers for MPI and stdio, and defines a function `nextPrime` to find the next prime number. The main function initializes MPI, gets the world size and rank, and then sends and receives prime numbers in a loop.
- Terminal (Right):** Shows the output of the program. The output displays the sequence of prime numbers sent and received by each process, confirming that the communication is working correctly.

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

int nextPrime(int n) {
    int i = n + 1;
    while (1) {
        if (i % 2 == 0) i++;
        else {
            int j = 3;
            while (j * j <= i) {
                if (i % j == 0) i++;
                j++;
            }
            return i;
        }
    }
}

int main() {
    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    double start = MPI_Wtime();

    if (world_rank != 0) {
        //worlds 1-8 start receiving
        MPI_Recv(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        printf("world %d received %d\n", world_rank, msg);
    }

    msg = nextPrime(msg);

    MPI_Send(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
    printf("world %d send prime number %d\n", world_rank, msg);

    if (world_rank == 0) {
        MPI_Recv(&msg, 1, MPI_INT, world_size - 1, msg_tag, MPI_COMM_WORLD, &status);
        printf("world %d received prime number %d from world %d\n", world_rank, msg, world_rank + 1);
    }

    double finish = MPI_Wtime();
    printf("world %d finished in time %f\n", world_rank, finish - start);

    MPI_Finalize();
}
```

The terminal output shows the following sequence of events:

```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$ mpirun -np 8 ./liii
world 0 send prime number 2 to world 1
world 1 received prime number 2 from world 0
world 1 send prime number 3 to world 2
world 1 finished in time 0.000079
world 2 received prime number 3 from world 1
world 2 send prime number 5 to world 3
world 2 finished in time 0.000253
world 3 received prime number 5 from world 2
world 3 send prime number 7 to world 4
world 3 finished in time 0.004081
world 4 received prime number 7 from world 3
world 4 send prime number 11 to world 5
world 4 finished in time 0.004355
world 5 received prime number 11 from world 4
world 5 send prime number 13 to world 6
world 5 finished in time 0.006343
world 6 received prime number 13 from world 5
world 6 received prime number 19 from world 7
world 6 finished in time 0.006331
world 7 received prime number 17 from world 6
world 7 send prime number 19 to world 0
world 7 finished in time 0.011127
world 0 send prime number 17 to world 7
world 0 finished in time 0.006285
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$
```

Through clever placement, we are able to set every node except starting node to MPI_Recv mode, and then as the communication progresses, the nodes become un-blocked and communicate to the next node in the ring.

Here we send an increasing prime across the ring

Code ->

```
//implementing MPI_Bcast with MPI_Send/MPI_Recv
//then comparing with MPI Bcast
```

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<stdbool.h>
#include<unistd.h>

#define MASTER 0
#define FROM_MASTER 1
```



```

#define FROM_WORKER 2
int msg_tag=1729;
int msg=1;

int nextPrime(int msg);
bool isPrime(int msg);

int main()
{
    MPI_Status status;
    MPI_Request request = MPI_REQUEST_NULL;

    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    double start = MPI_Wtime();

    if(world_rank!=0)
    {
        //worlds 1-8 start receiving msg from previous
        MPI_Recv(&msg, 1, MPI_INT, world_rank-1, msg_tag, MPI_COMM_WORLD,
&status);
        printf("world %d received prime number %d from world %d\n",
world_rank, msg, world_rank-1);
    }

    msg=nextPrime(msg);

    MPI_Send(&msg, 1, MPI_INT, (world_rank+1)%world_size, msg_tag,
MPI_COMM_WORLD);
    printf("world %d send prime number %d to world %d\n", world_rank, msg,
(world_rank+1)%world_size);
    if(world_rank==0)
    {
        MPI_Recv(&msg, 1, MPI_INT, world_size-1, msg_tag, MPI_COMM_WORLD,
&status);
        printf("world %d received prime number %d from world %d\n",
world_rank, msg, world_size-1);
    }

    double finish=MPI_Wtime();
    printf("world %d finished in time %f\n", world_rank, finish-start);

    MPI_Finalize();

    return 0;
}

```

```
}
```

```
bool isPrime(int n)
```

```
{
```

```
    if(n<=1)
```

```
        return false;
```

```
    if(n<=3)
```

```
        return true;
```

```
    if(n%2 == 0 || n%3 == 0)
```

```
        return false;
```

```
    for(int i=5; i*i<=n; i=i+6)
```

```
        if(n%i==0 || n%(i+2)==0)
```

```
            return false;
```

```
    return true;
```

```
}
```

```
int nextPrime(int msg)
```

```
{
```

```
    if(msg <=1)
```

```
        return 2;
```

```
    int prime=msg;
```

```
    bool found=false;
```

```
    while(!found)
```

```
    {
```

```
        prime++;
```

```
        if(isPrime(prime))
```

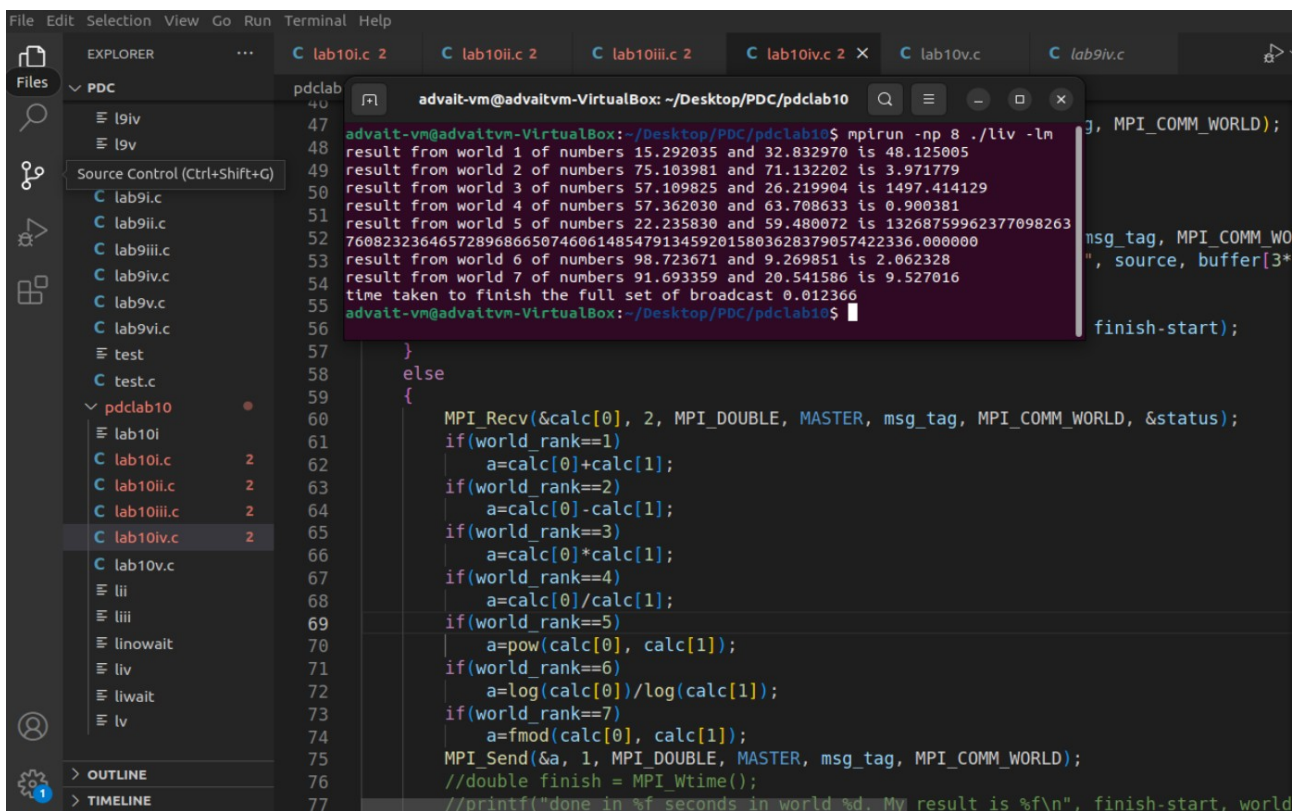
```
            found=true;
```

```
    }
```

```
    return prime;
```

```
}
```


rank0 - sends randsum, rank1 - Add const , rank2 - sub const, rank3 - mul const



The screenshot shows a Visual Studio Code editor with a file explorer on the left displaying a project structure under 'PDC'. The main editor window shows a C program with MPI. A terminal window at the bottom displays the output of running the program with 8 processes. The output shows results for each world rank, including calculations and a final time taken to finish the full set of broadcast.

```
47 pdclab
48 40
49 47
50 48
51 49
52 50
53 51
54 52
55 53
56 54
57 55
58 56
59 57
60 58
61 59
62 60
63 61
64 62
65 63
66 64
67 65
68 66
69 67
70 68
71 69
72 70
73 71
74 72
75 73
76 74
77 75
```

```
advait-vm@advaitvm-VirtualBox: ~/Desktop/PDC/pdclab10
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$ mpirun -np 8 ./liv -lm
result from world 1 of numbers 15.292035 and 32.832970 is 48.125005
result from world 2 of numbers 75.103981 and 71.132202 is 3.971779
result from world 3 of numbers 57.109825 and 26.219904 is 1497.414129
result from world 4 of numbers 57.362030 and 63.708633 is 0.900381
result from world 5 of numbers 22.235830 and 59.480072 is 13268759962377098263
7608232364657289686650746061485479134592015803628379057422336.000000
result from world 6 of numbers 98.723671 and 9.269851 is 2.062328
result from world 7 of numbers 91.693359 and 20.541586 is 9.527016
time taken to finish the full set of broadcast 0.012366
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab10$
```

Here we execute different operations at each slave node

Code ->

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<stdbool.h>
#include<unistd.h>
#include<math.h>

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

int main()
{
    MPI_Status status;
    MPI_Request request = MPI_REQUEST_NULL;
```

```

MPI_Init(NULL, NULL);

int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

double start = MPI_Wtime();

int msg_tag=1729;
double buffer[100]={0};
double calc[2]={0};
double a=0;

if(world_rank==0)
{
    srand(time(NULL));
    for(int i=0; i<world_size*3; i++)
    {
        buffer[i]=((float)rand()/(float)(RAND_MAX))*100;
        //printf("%f\n", buffer[i]);
    }

    for(int dest=1; dest<world_size; dest++)
    {
        //sends to all
        MPI_Send(&buffer[3*(dest-1)], 2, MPI_DOUBLE, dest, msg_tag,
MPI_COMM_WORLD);
    }
    for(int source=1; source<world_size; source++)
    {
        //receives from all
        MPI_Recv(&buffer[3*(source-1)+2], 1, MPI_DOUBLE, source, msg_tag,
MPI_COMM_WORLD, &status);
        printf("result from world %d of numbers %f and %f is %f\n", source,
buffer[3*(source-1)], buffer[3*(source-1)+1], buffer[3*(source-1)+2]);
    }
    double finish = MPI_Wtime();
    printf("time taken to finish the full set of broadcast %f\n", finish-
start);
}
else
{
    MPI_Recv(&calc[0], 2, MPI_DOUBLE, MASTER, msg_tag, MPI_COMM_WORLD,
&status);
    if(world_rank==1)
        a=calc[0]+calc[1];
    if(world_rank==2)

```

```
    a=calc[0]-calc[1];
    if(world_rank==3)
        a=calc[0]*calc[1];
    if(world_rank==4)
        a=calc[0]/calc[1];
    if(world_rank==5)
        a=pow(calc[0], calc[1]);
    if(world_rank==6)
        a=log(calc[0])/log(calc[1]);
    if(world_rank==7)
        a=fmod(calc[0], calc[1]);
    MPI_Send(&a, 1, MPI_DOUBLE, MASTER, msg_tag, MPI_COMM_WORLD);
    //double finish = MPI_Wtime();
    //printf("done in %f seconds in world %d. My result is %f\n", finish-
start, world_rank, a);
}

MPI_Finalize();

return 0;
}
```
