PDC LAB 10

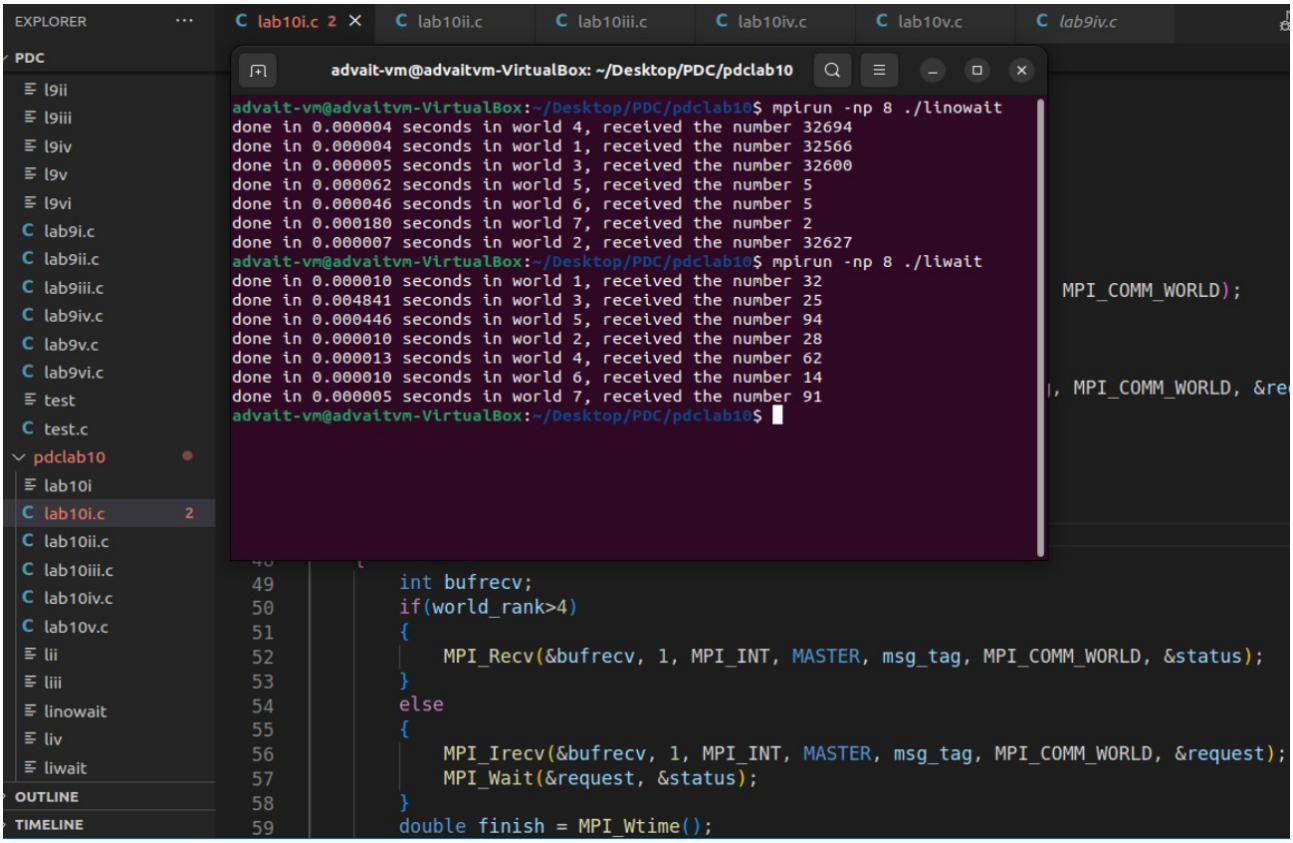
Advait Deochakke

20BCE1143

Sample isend, ireceive with mpi_wtime

Sample send and receive with mpi_wtime

Both combined in one, easily showcasing the difference



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 ->

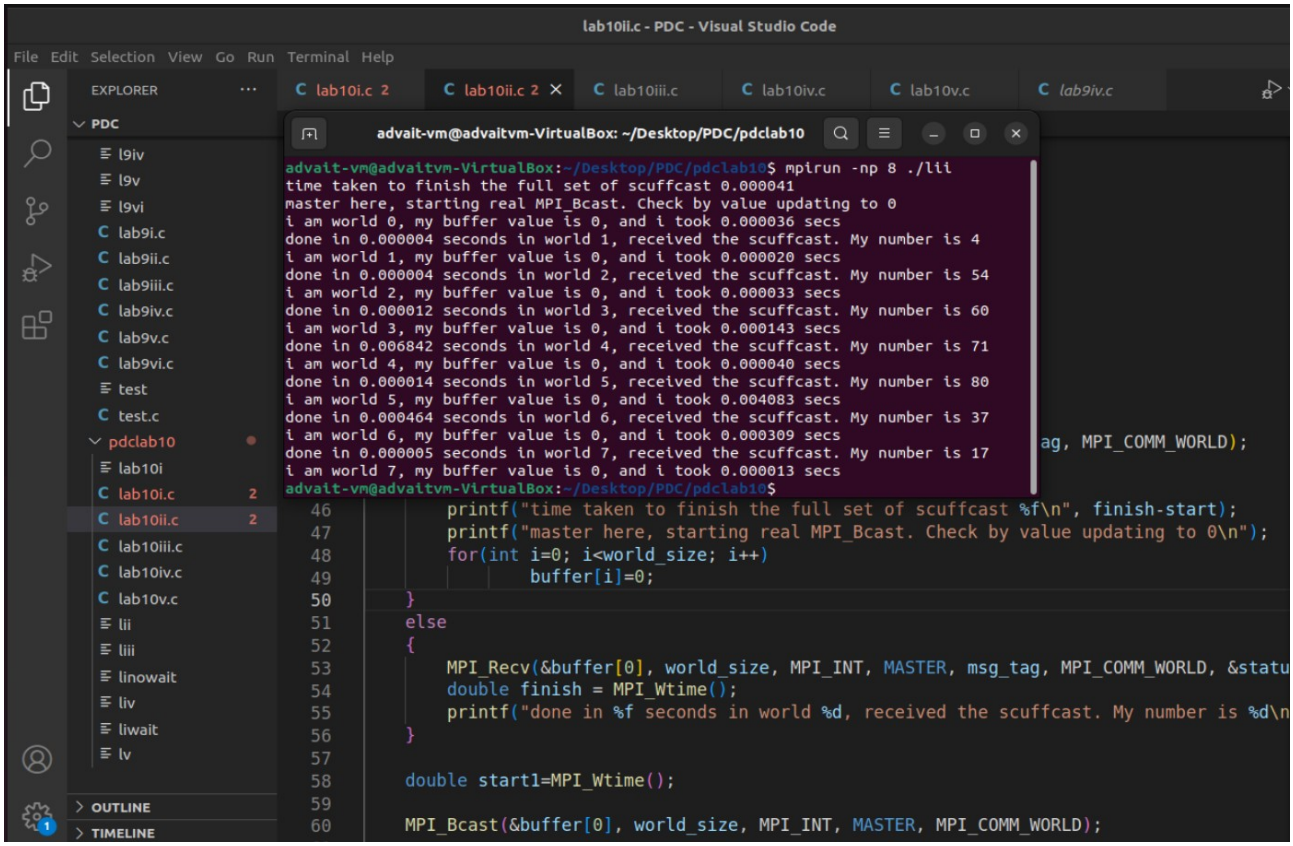
```
#include<mpi.h>
#include<stdio.h>
```

```c
#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)



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

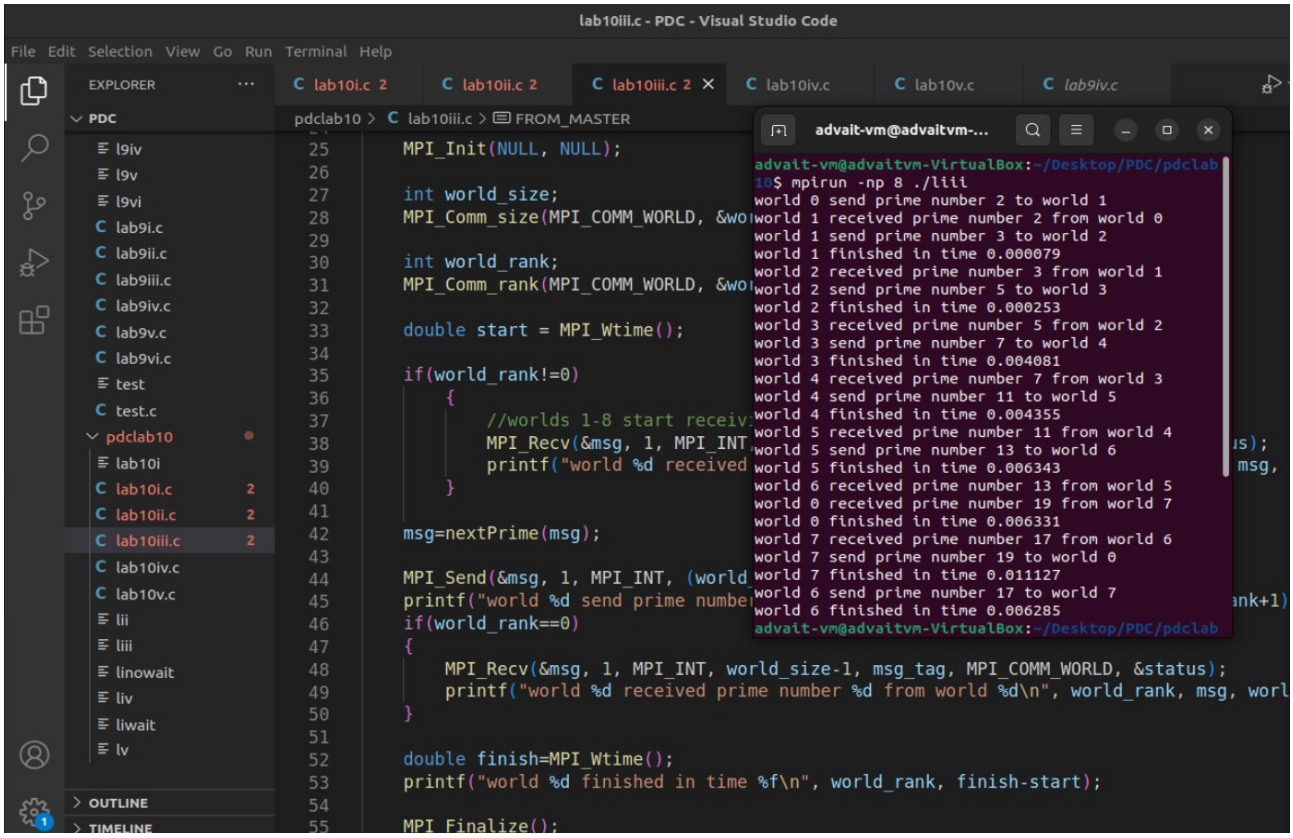
whereas, the second line which mentions buffer value being zero corresponds to real MPI_Bcast

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



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

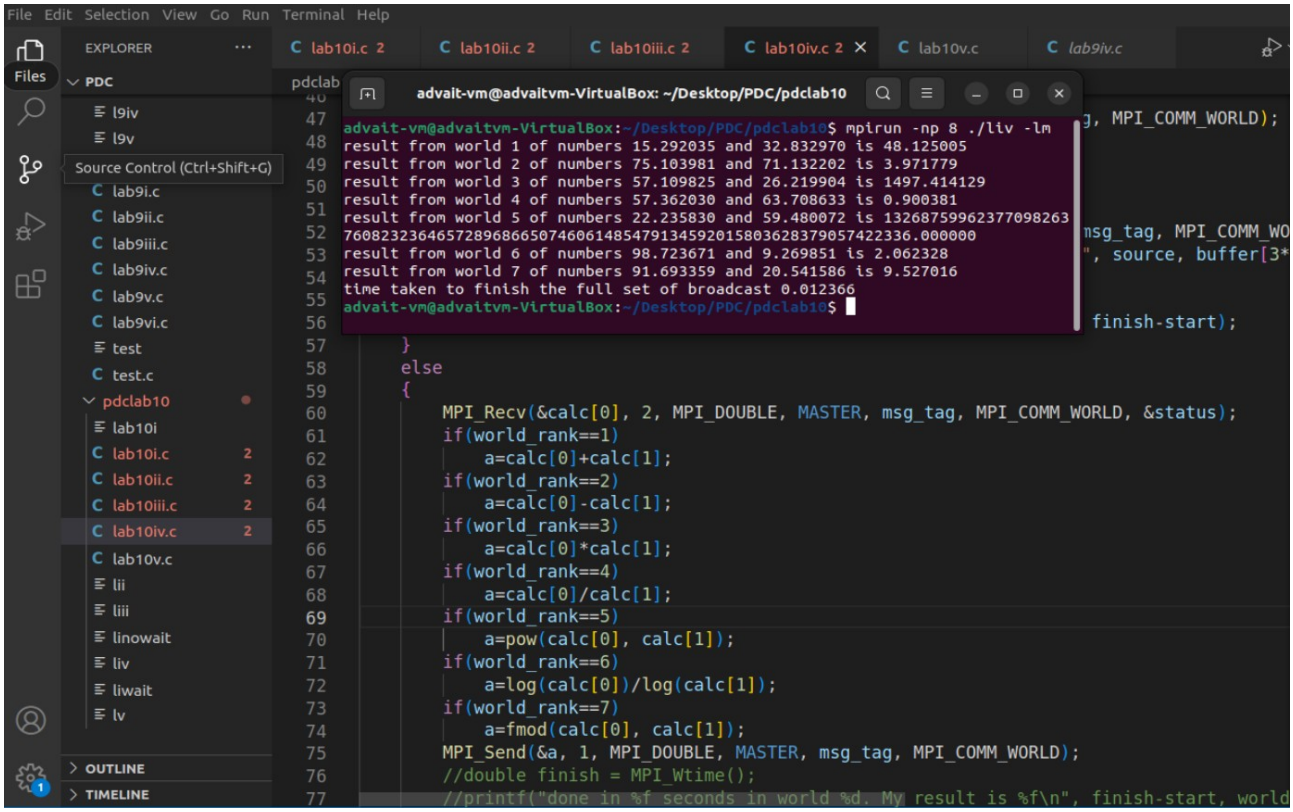
Code ->

```
//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



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;
}
```
