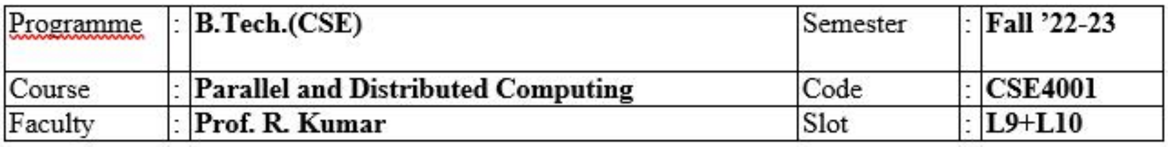
NAME – ANSH GOEL

REGISTER NO.- 20BCE1798

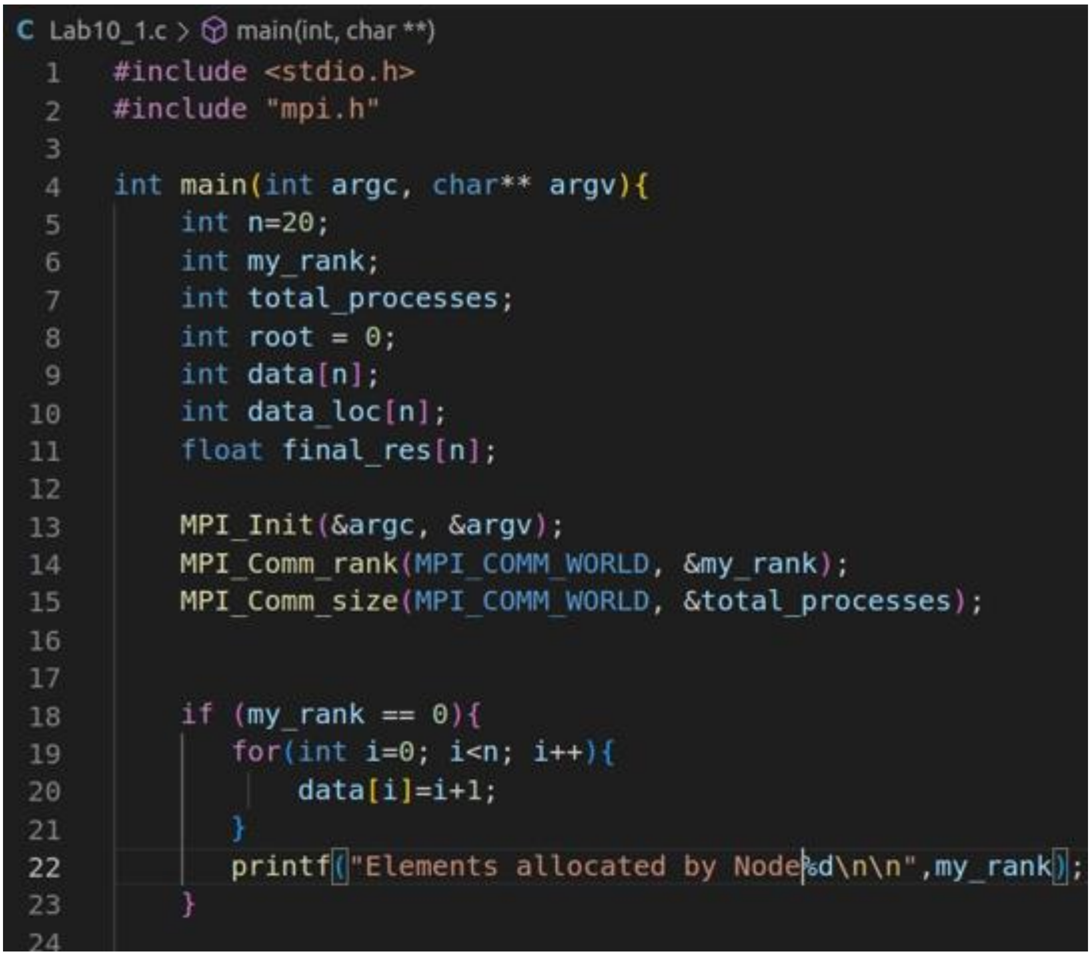
VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore - Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127



Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**Prof. R. Kumar**	Slot	:	**L9+L10**

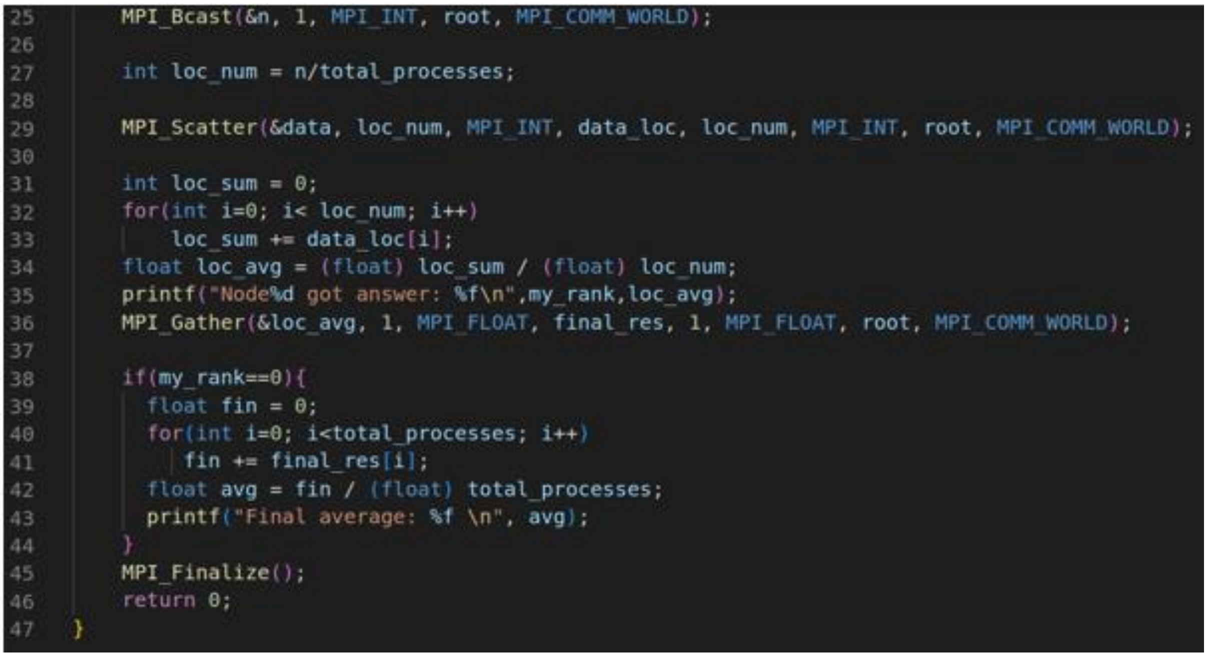
1. Write a program in MPI to generate 'n' random float numbers and send' k' of those to each node and make them compute the average and send it back to the master which computes the average of those averages.

CODE:



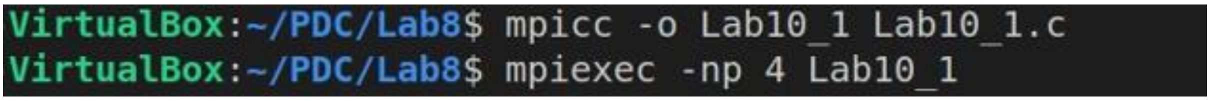
C Lab10_1.c > main(int, char **)

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

int main(int argc, char** argv){
    int n=20;
    int my_rank;
    int total_processes;
    int root = 0;
    int data[n];
    int data_loc[n];
    float final_res[n];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &total_processes);


    if (my_rank == 0){
        for(int i=0; i<n; i++){
            data[i]=i+1;
        }
        printf("Elements allocated by Node%d\n\n",my_rank);
    }

```

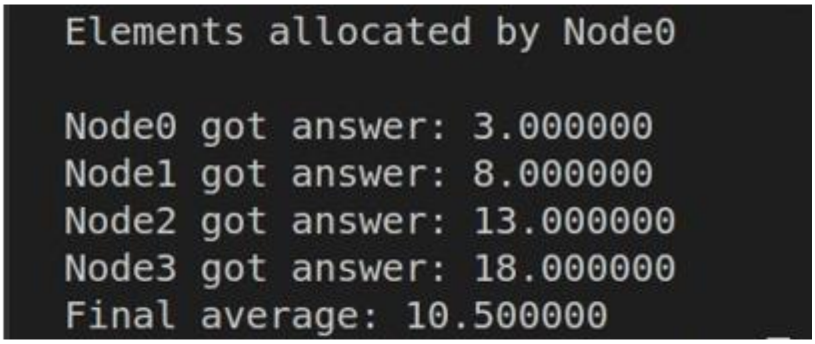


```c
    MPI_Bcast(&n, 1, MPI_INT, root, MPI_COMM_WORLD);

    int loc_num = n/total_processes;

    MPI_Scatter(&data, loc_num, MPI_INT, data_loc, loc_num, MPI_INT, root, MPI_COMM_WORLD);

    int loc_sum = 0;
    for(int i=0; i< loc_num; i++)
        loc_sum += data_loc[i];
    float loc_avg = (float) loc_sum / (float) loc_num;
    printf("Node%d got answer: %f\n",my_rank,loc_avg);
    MPI_Gather(&loc_avg, 1, MPI_FLOAT, final_res, 1, MPI_FLOAT, root, MPI_COMM_WORLD);

    if(my_rank==0){
      float fin = 0;
      for(int i=0; i<total_processes; i++)
         fin += final_res[i];
      float avg = fin / (float) total_processes;
      printf("Final average: %f \n", avg);
    }
    MPI_Finalize();
    return 0;
}
```

OUTPUT:



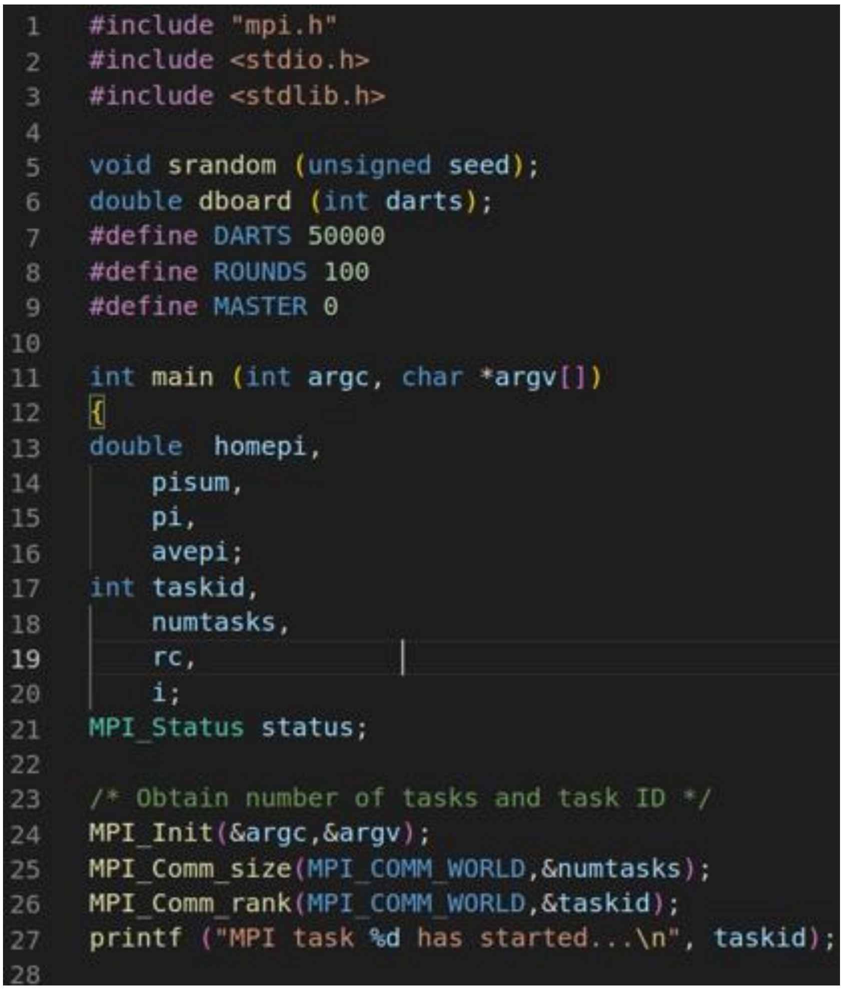
```
VirtualBox:~/PDC/Lab8$ mpicc -o Lab10_1 Lab10_1.c
VirtualBox:~/PDC/Lab8$ mpiexec -np 4 Lab10_1
```



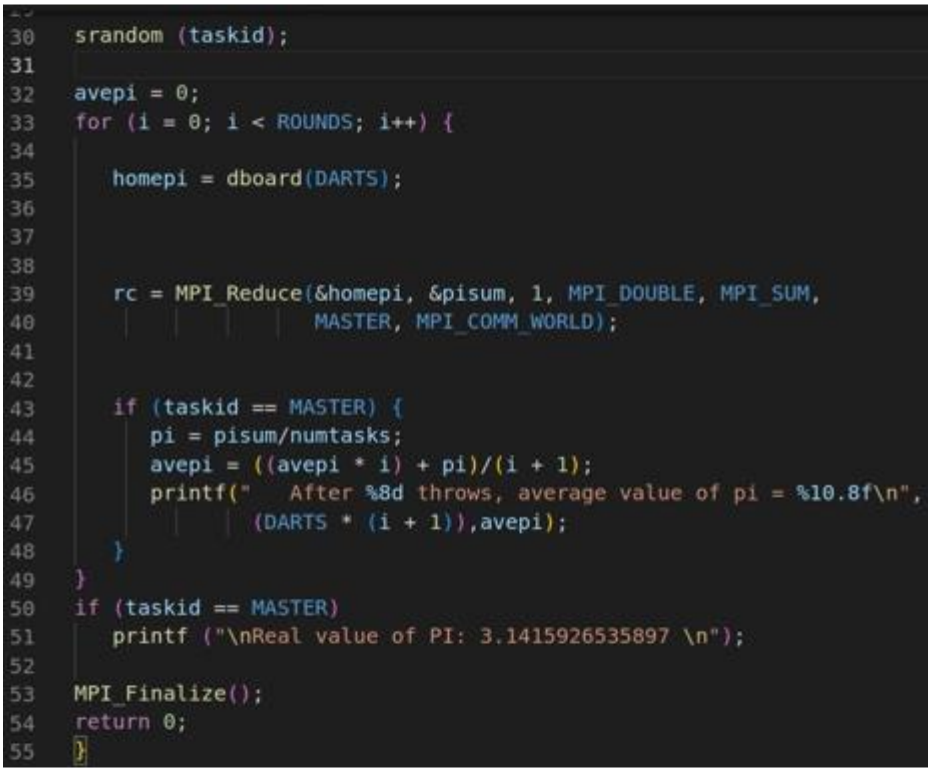
```
Elements allocated by Node0

Node0 got answer: 3.000000
Node1 got answer: 8.000000
Node2 got answer: 13.000000
Node3 got answer: 18.000000
Final average: 10.500000
```

2. Write a MPI program to compute PI using “dartboard” technique for 1000 rounds by using reduction collective computation.

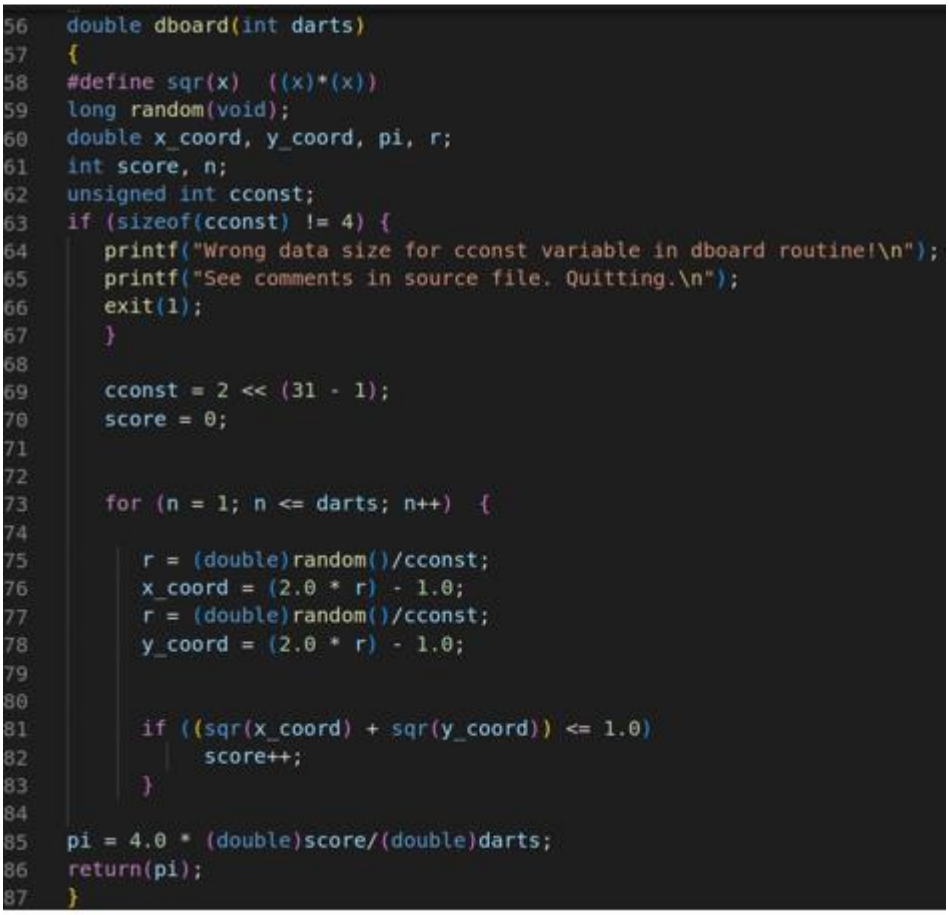
CODE:



```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

void srandom (unsigned seed);
double dboard (int darts);
#define DARTS 50000
#define ROUNDS 100
#define MASTER 0

int main (int argc, char *argv[])
{
double  homepi,
        pisum,
        pi,
        avepi;
int taskid,
        numtasks,
        rc,
        i;
MPI_Status status;

/* Obtain number of tasks and task ID */
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
printf ("MPI task %d has started...\n", taskid);

```

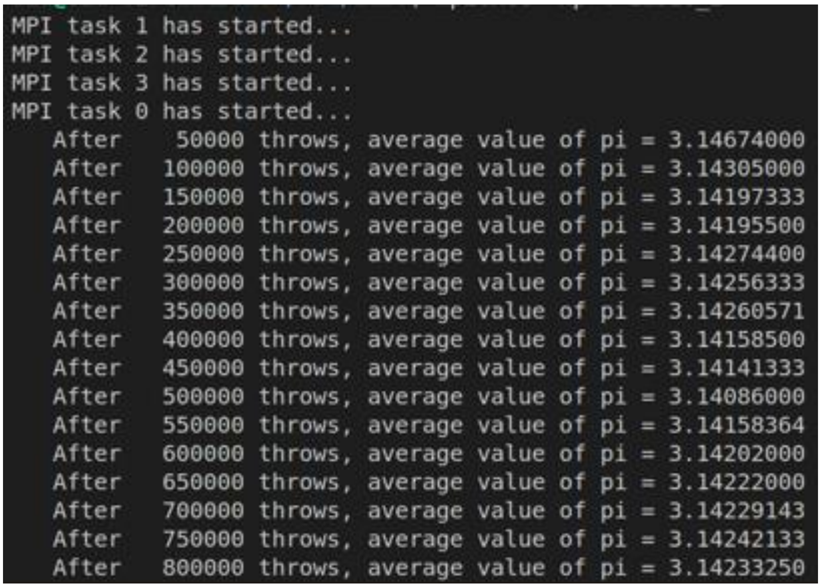


```c
srandom (taskid);

avepi = 0;
for (i = 0; i < ROUNDS; i++) {

   homepi = dboard(DARTS);



   rc = MPI_Reduce(&homepi, &pisum, 1, MPI_DOUBLE, MPI_SUM,
                   MASTER, MPI_COMM_WORLD);


   if (taskid == MASTER) {
      pi = pisum/numtasks;
      avepi = ((avepi * i) + pi)/(i + 1);
      printf("   After %8d throws, average value of pi = %10.8f\n",
              (DARTS * (i + 1)),avepi);
   }
}
if (taskid == MASTER)
   printf ("\nReal value of PI: 3.1415926535897 \n");

MPI_Finalize();
return 0;
}
```

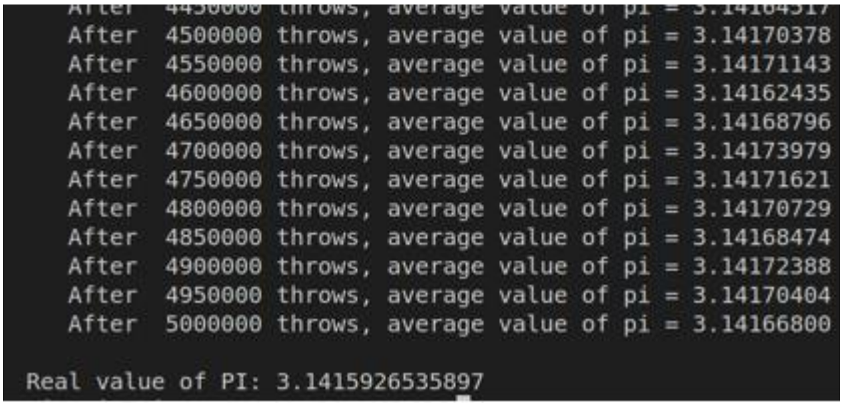


```c
double dboard(int darts)
{
#define sqr(x)  ((x)*(x))
long random(void);
double x_coord, y_coord, pi, r;
int score, n;
unsigned int cconst;
if (sizeof(cconst) != 4) {
   printf("Wrong data size for cconst variable in dboard routine!\n");
   printf("See comments in source file. Quitting.\n");
   exit(1);
   }

   cconst = 2 << (31 - 1);
   score = 0;


   for (n = 1; n <= darts; n++)  {

      r = (double)random()/cconst;
      x_coord = (2.0 * r) - 1.0;
      r = (double)random()/cconst;
      y_coord = (2.0 * r) - 1.0;


      if ((sqr(x_coord) + sqr(y_coord)) <= 1.0)
           score++;
      }

pi = 4.0 * (double)score/(double)darts;
return(pi);
}
```

OUTPUT:



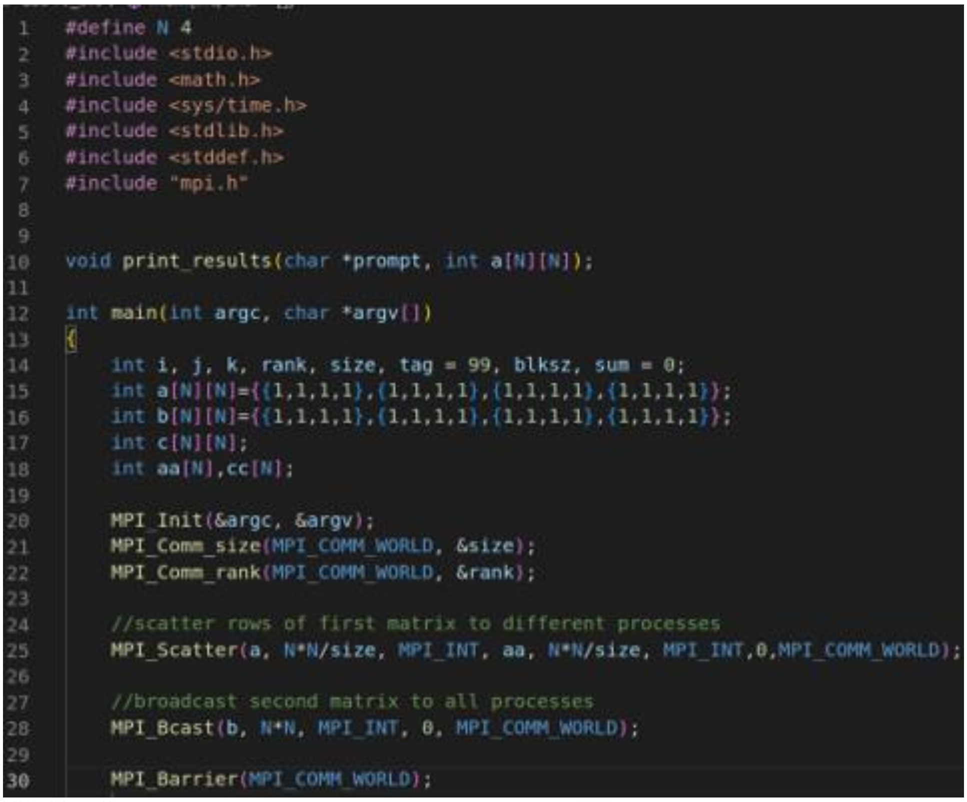
```
MPI task 1 has started...
MPI task 2 has started...
MPI task 3 has started...
MPI task 0 has started...
   After    50000 throws, average value of pi = 3.14674000
   After   100000 throws, average value of pi = 3.14305000
   After   150000 throws, average value of pi = 3.14197333
   After   200000 throws, average value of pi = 3.14195500
   After   250000 throws, average value of pi = 3.14274400
   After   300000 throws, average value of pi = 3.14256333
   After   350000 throws, average value of pi = 3.14260571
   After   400000 throws, average value of pi = 3.14158500
   After   450000 throws, average value of pi = 3.14141333
   After   500000 throws, average value of pi = 3.14086000
   After   550000 throws, average value of pi = 3.14158364
   After   600000 throws, average value of pi = 3.14202000
   After   650000 throws, average value of pi = 3.14222000
   After   700000 throws, average value of pi = 3.14229143
   After   750000 throws, average value of pi = 3.14242133
   After   800000 throws, average value of pi = 3.14233250
```



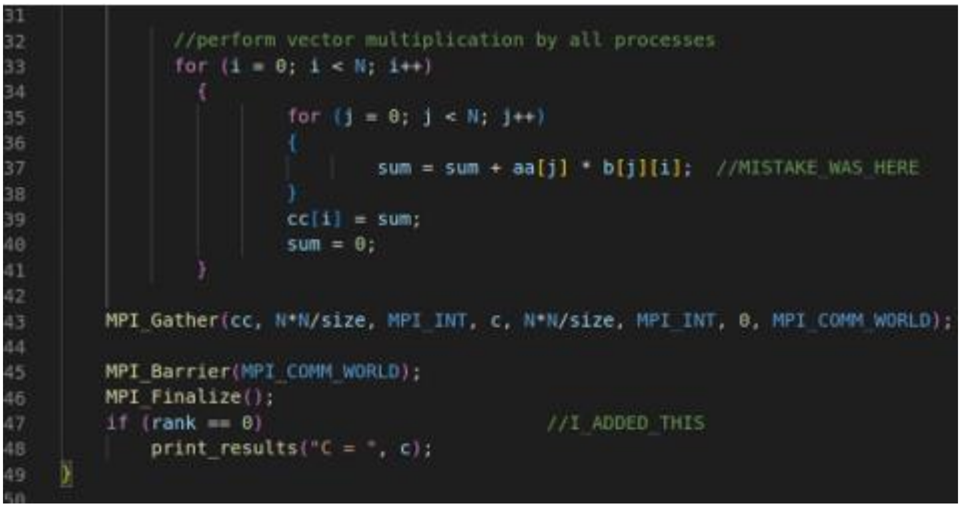
```
   After  4450000 throws, average value of pi = 3.14164317
   After  4500000 throws, average value of pi = 3.14170378
   After  4550000 throws, average value of pi = 3.14171143
   After  4600000 throws, average value of pi = 3.14162435
   After  4650000 throws, average value of pi = 3.14168796
   After  4700000 throws, average value of pi = 3.14173979
   After  4750000 throws, average value of pi = 3.14171621
   After  4800000 throws, average value of pi = 3.14170729
   After  4850000 throws, average value of pi = 3.14168474
   After  4900000 throws, average value of pi = 3.14172388
   After  4950000 throws, average value of pi = 3.14170404
   After  5000000 throws, average value of pi = 3.14166800

 Real value of PI: 3.1415926535897
```

3. Write a MPI program to perform matrix multiplication (1000x1000) using scatter and gather routines.

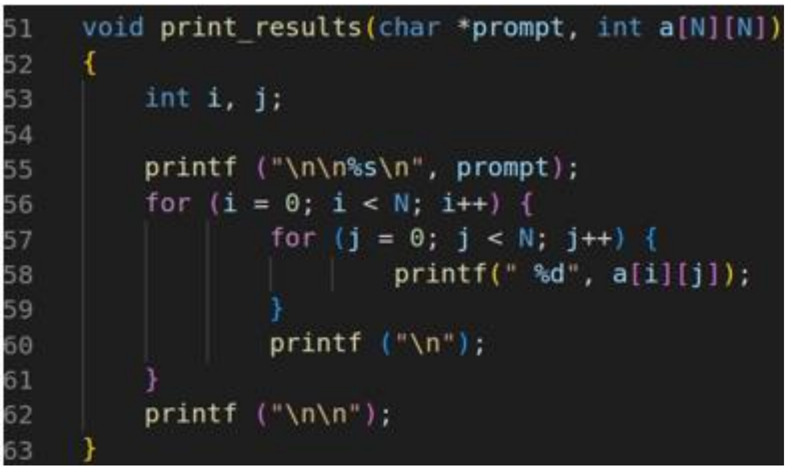
CODE:



```
#define N 4
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <stdlib.h>
#include <stddef.h>
#include "mpi.h"


void print_results(char *prompt, int a[N][N]);

int main(int argc, char *argv[])
{
    int i, j, k, rank, size, tag = 99, blksz, sum = 0;
    int a[N][N]={{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};
    int b[N][N]={{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};
    int c[N][N];
    int aa[N],cc[N];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //scatter rows of first matrix to different processes
    MPI_Scatter(a, N*N/size, MPI_INT, aa, N*N/size, MPI_INT,0,MPI_COMM_WORLD);

    //broadcast second matrix to all processes
    MPI_Bcast(b, N*N, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
```

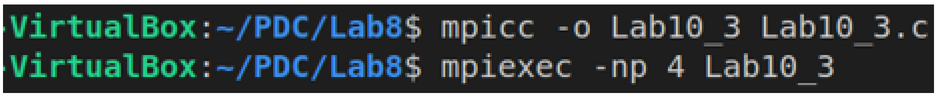


```

        //perform vector multiplication by all processes
        for (i = 0; i < N; i++)
          {
                for (j = 0; j < N; j++)
                {
                        sum = sum + aa[j] * b[j][i];  //MISTAKE_WAS_HERE
                }
                cc[i] = sum;
                sum = 0;
          }

    MPI_Gather(cc, N*N/size, MPI_INT, c, N*N/size, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    if (rank == 0)                          //I_ADDED_THIS
        print_results("C = ", c);
}
```

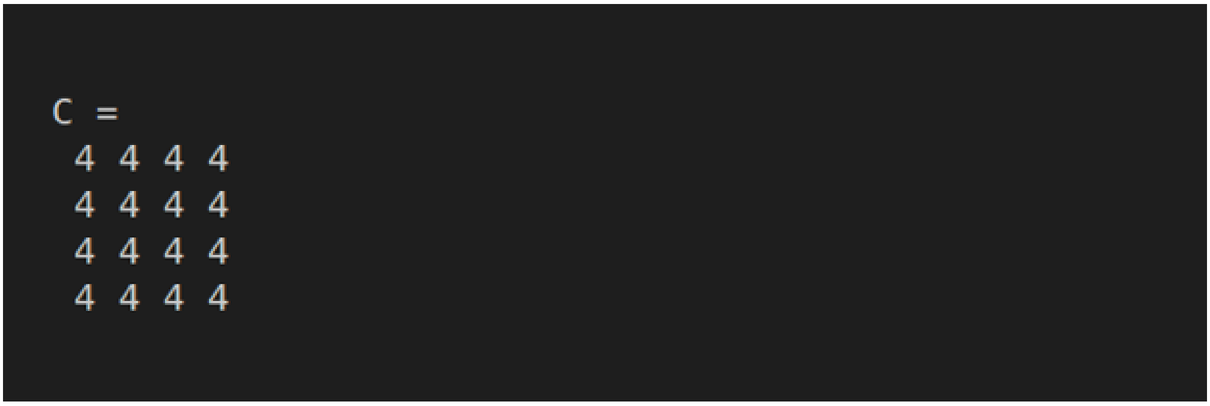


```c
void print_results(char *prompt, int a[N][N])
{
    int i, j;

    printf ("\n\n%s\n", prompt);
    for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                    printf(" %d", a[i][j]);
            }
            printf ("\n");
    }
    printf ("\n\n");
}
```

OUTPUT:



```
VirtualBox:~/PDC/Lab8$ mpicc -o Lab10_3 Lab10_3.c
VirtualBox:~/PDC/Lab8$ mpiexec -np 4 Lab10_3
```



```


C =
 4 4 4 4
 4 4 4 4
 4 4 4 4
 4 4 4 4
```