Name	:	Abhishek N N
Reg.No	:	20BCE1025
Email	:	abhishek.nn2020@vitstudent.ac.in

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**

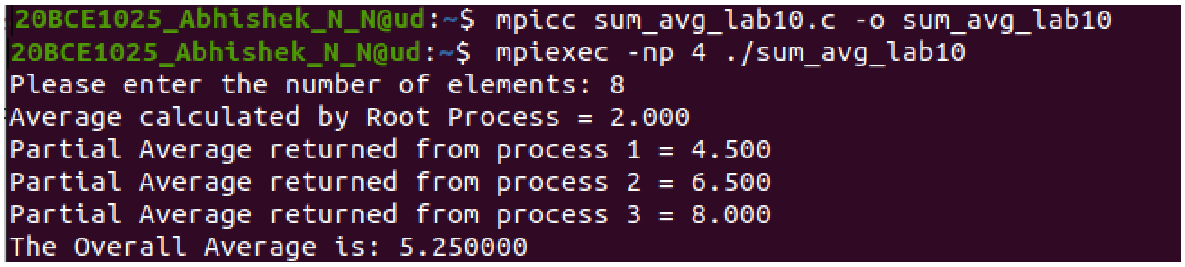
1. Write a program in MPI to generate 'n' random float numbers and send' k' of those to each node and make them compute the average and send it back to the master which computes the average of those averages.

```
#include <mpi.h>

#include <stdio.h>
#include <stdlib.h>

#define max_n 10000
#define send_data_tag 2001
#define return_data_tag 2002

float array[max_n];
float array2[max_n];
int main(int argc, char** argv) {
     float sum, partial_sum;
     float avg, partial_avg;
     MPI_Status status;
     int my_id, root_process, ierr, i, num_procs, an_id, sender,
     num_rows_to_send, hum_rows_to_receive, num_rows_received;
     int n, k;

     int start_n, end_n;
```

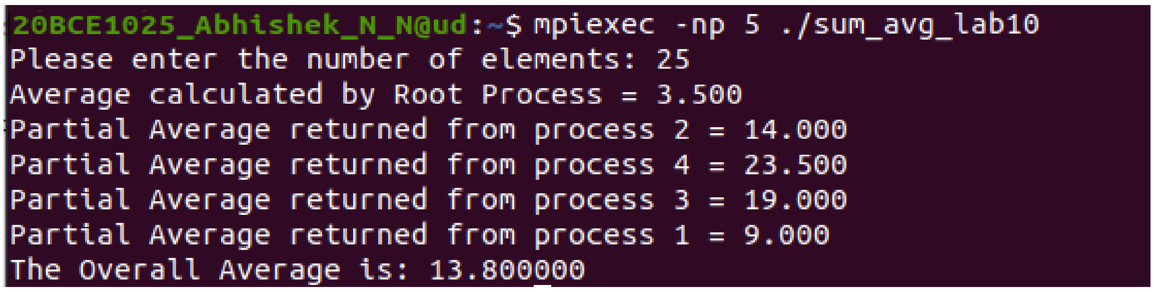
```c
ierr = MPI_Init(&argc, &argv);
root_process = 0;
ierr = MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
ierr = MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

if (my_id == root_process) {
     printf("Please enter the number of elements: ");
     scanf("%d", &n);

     if (n > max_n) {
          printf("Too many numbers.\n");
          exit(1);
     }

     /* Initialize an array */

     for (i = 0; i < n; i++) {
          array[i] = (float)(rand() % 100) / 7;
     }

     k = n / num_procs;
     /* Distribute a portion of the vector to each child
     process */

     for (an_id = 1; an_id < num_procs; an_id++) {
          start_n = an_id * k + 1;
          end_n = (an_id + 1) * k;
          if ((n - end_n) < k)
          end_n = n - 1;

          num_rows_to_send = end_n - start_n + 1;

          ierr = MPI_Send(&num_rows_to_send, 1, MPI_INT,
          an_id, send_data_tag,
          MPI_COMM_WORLD);

          ierr = MPI_Send(&array[start_n],
          num_rows_to_send, MPI_INT, an_id, send_data_tag,
          MPI_COMM_WORLD);
     }

     /* Calculate the sum and average of the values in the
     segment assigned to the root
     process */

     sum = 0;
```

```
        for (i = 0; i < k + 1; i++) {
             sum += array[i];
        }
        avg = sum / (k + 1);
        printf("Average calculated by Root Process = %.3f\n",
        avg);
        /* and, finally, I collect the partial sums from the
        slave processes, print them, and add them to the grand
        sum, and print it */

        for (an_id = 1; an_id < num_procs; an_id++) {
             ierr = MPI_Recv(&partial_avg, 1, MPI_FLOAT,
             MPI_ANY_SOURCE, return_data_tag, MPI_COMM_WORLD,
             &status);
             sender = status.MPI_SOURCE;
             printf("Partial Average returned from process %i
             = %.3f\n", sender,
             partial_avg);
        }
        avg += partial_avg;
        avg = avg / num_procs;
        printf("The Overall Average is: %f\n", avg);
}

else {
        /* I must be a slave process, so I must receive my
        array segment, storing it in a "local" array, array1.
        */

        ierr = MPI_Recv(&num_rows_to_receive, 1, MPI_INT,
        root_process, send_data_tag, MPI_COMM_WORLD, &status);
        ierr = MPI_Recv(&array2, num_rows_to_receive, MPI_INT,
        root_process, send_data_tag, MPI_COMM_WORLD, &status);
        num_rows_received = num_rows_to_receive;

        /* Calculate the sum and avergae of my portion of the
        array */ partial_sum = 0;
        for (i = 0; i < num_rows_received; i++) {
             partial_sum += array2[i];
        }
        partial_avg = (float)partial_sum / num_rows_received;
        /* and finally, send my partial sum to the root
        process */
        ierr = MPI_Send(&partial_avg, 1, MPI_FLOAT,
        root_process, return_data_tag, MPI_COMM_WORLD);
}
```

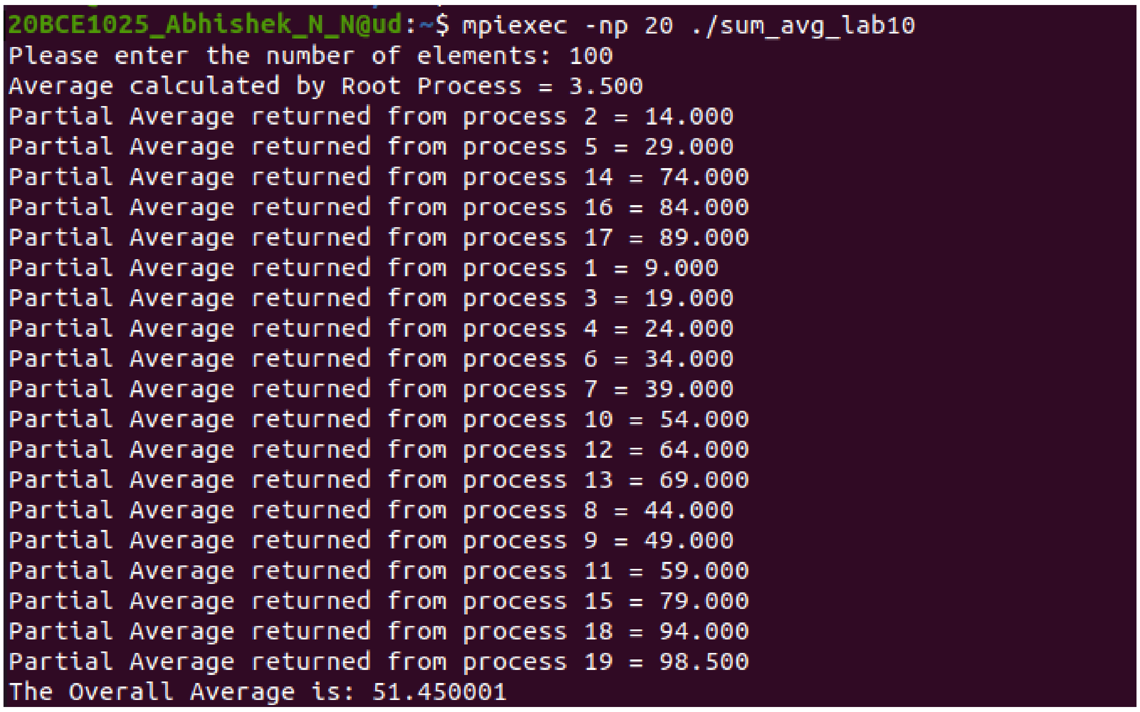
```c
        ierr = MPI_Finalize();
        return 0;
}
```



```
20BCE1025_Abhishek_N_N@ud:~$ mpicc sum_avg_lab10.c -o sum_avg_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 4 ./sum_avg_lab10
Please enter the number of elements: 8
Average calculated by Root Process = 2.000
Partial Average returned from process 1 = 4.500
Partial Average returned from process 2 = 6.500
Partial Average returned from process 3 = 8.000
The Overall Average is: 5.250000
```



```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 5 ./sum_avg_lab10
Please enter the number of elements: 25
Average calculated by Root Process = 3.500
Partial Average returned from process 2 = 14.000
Partial Average returned from process 4 = 23.500
Partial Average returned from process 3 = 19.000
Partial Average returned from process 1 = 9.000
The Overall Average is: 13.800000
```

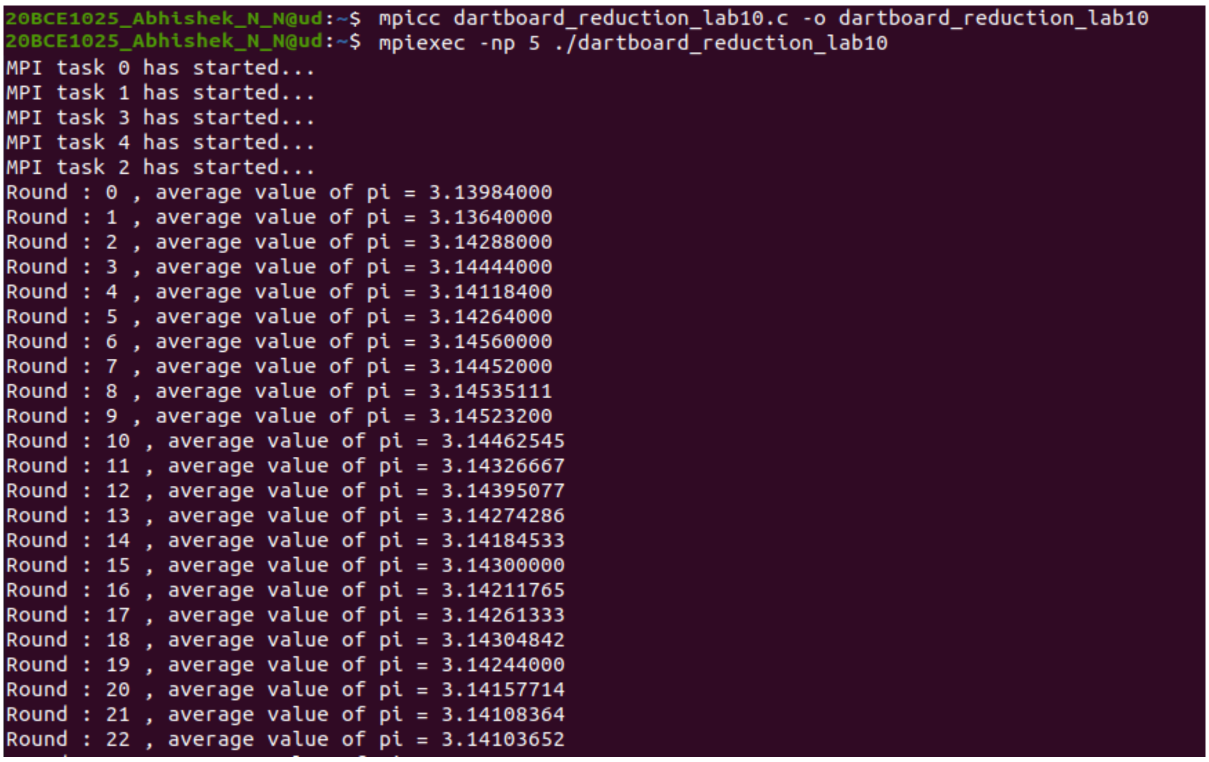


```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 20 ./sum_avg_lab10
Please enter the number of elements: 100
Average calculated by Root Process = 3.500
Partial Average returned from process 2 = 14.000
Partial Average returned from process 5 = 29.000
Partial Average returned from process 14 = 74.000
Partial Average returned from process 16 = 84.000
Partial Average returned from process 17 = 89.000
Partial Average returned from process 1 = 9.000
Partial Average returned from process 3 = 19.000
Partial Average returned from process 4 = 24.000
Partial Average returned from process 6 = 34.000
Partial Average returned from process 7 = 39.000
Partial Average returned from process 10 = 54.000
Partial Average returned from process 12 = 64.000
Partial Average returned from process 13 = 69.000
Partial Average returned from process 8 = 44.000
Partial Average returned from process 9 = 49.000
Partial Average returned from process 11 = 59.000
Partial Average returned from process 15 = 79.000
Partial Average returned from process 18 = 94.000
Partial Average returned from process 19 = 98.500
The Overall Average is: 51.450001
```

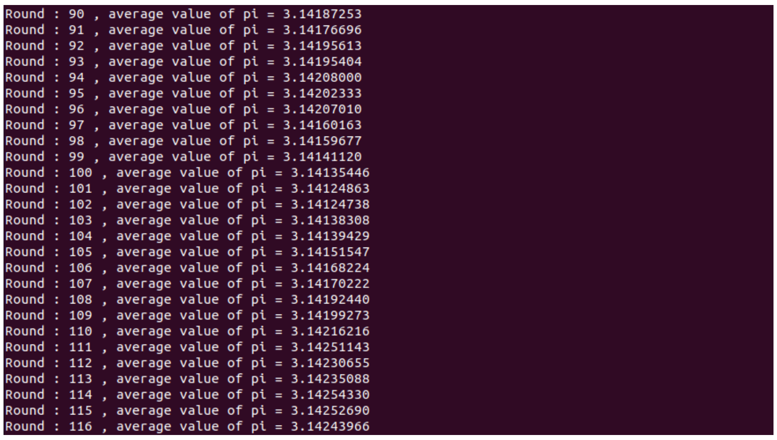
2. Write a MPI program to compute PI using "dartboard" technique for 1000 rounds by using reduction collective computation.

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

#include "mpi.h"
void srandom(unsigned seed);
double pi_board(int darts);
#define NUM_DARTS 5000
#define NUM_ROUNDS 1000
#define MASTER 0
int main(int arc, char *argv[]) {
    double homepi, pisum, pi, avepi;
    int t_id, num_tasks, rc, i;
    MPI_Status status;
    MPI_Init(&argc, &argv); // Initializing MPI
    MPI_Comm_size(MPI_COMM_WORLD,&num_tasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&t_id);
    printf("MPI task %d has started...\n", t_id);
    srandom(t_id);
    avepi = 0;
    for (i = 0; i < NUM_ROUNDS; i++) {
        homepi = pi_board(NUM_DARTS);
        rc = MPI_Reduce(&homepi, &pisum, 1, MPI_DOUBLE, MPI_SUM,
        MASTER, MPI_COMM_WORLD); // MPI_Reduce is used to sum
        the values of pi across all tasks if (rc != MPI_SUCCESS)
        printf("%d: mpc_reduce failed ! \n", t_id);
        if (t_id == MASTER) {
            pi = pisum / num_tasks;
            avepi = ((avepi * i) + pi) / (i + 1);
            printf("Round : %d , average value of pi = %10.8f\
            n", i, avepi);
        }
    }
    if (t_id == MASTER)
    printf("\nReal value of PI: 3.1415926535897 \n");
    MPI_Finalize();
    return 0;
}

double pi_board(int d) // Function for cal PI
```

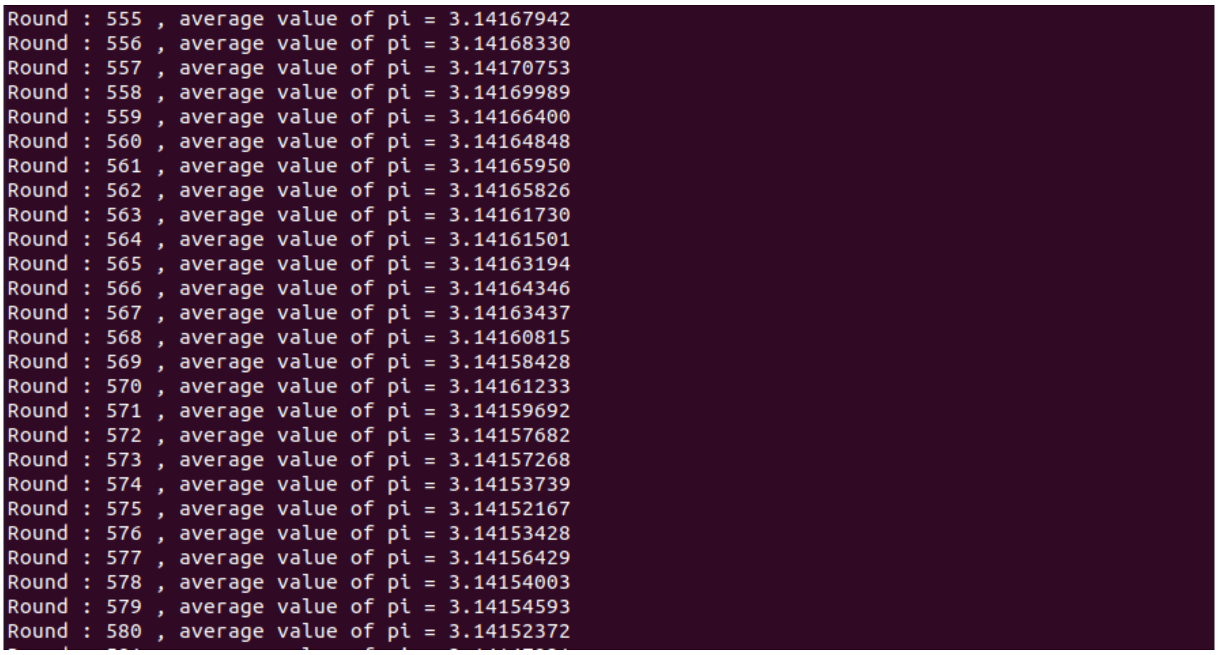
```
{
        #define sqr(x) ((x) * (x)) long random(void);
        double x_coord, y_coord, pi, r;
        int score, n;
        unsigned int cconst;
        cconst = 2 << (31 - 1);
        score = 0;
        for (n = 1; n <= d; n++) {
                r = (double)random() / cconst;
                x_coord = (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)d; // Calculating PI
        return(pi);
        return (pi);
}
```



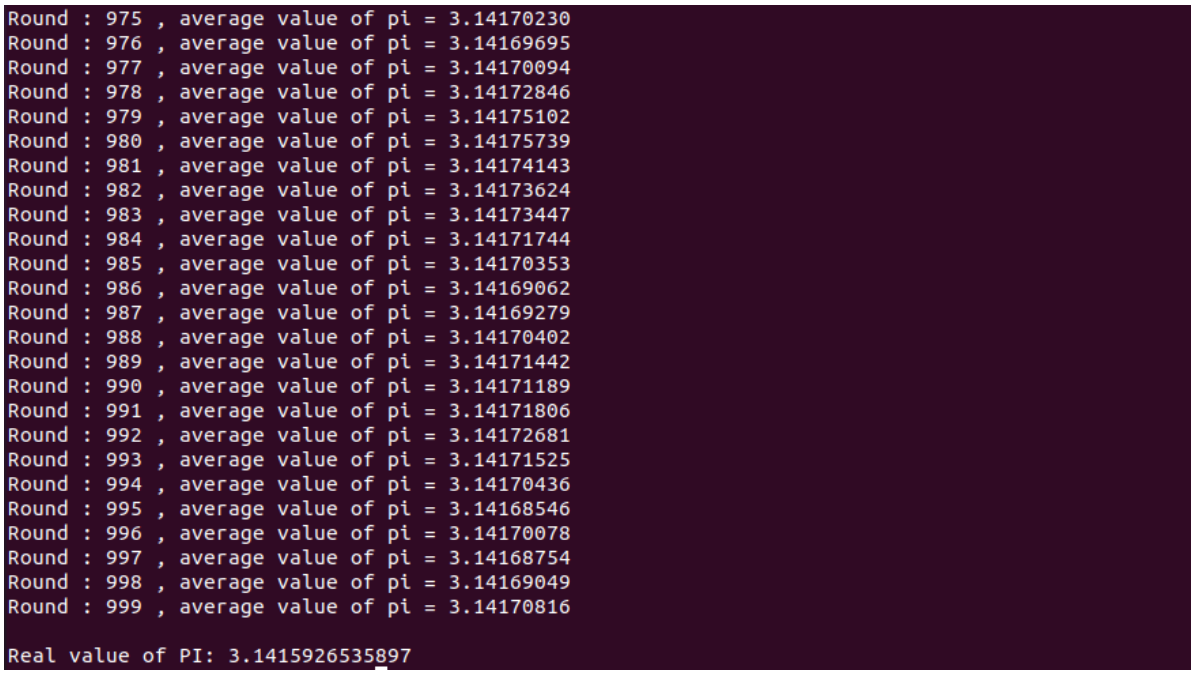
```
20BCE1025_Abhishek_N_N@ud:~$ mpicc dartboard_reduction_lab10.c -o dartboard_reduction_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 5 ./dartboard_reduction_lab10
MPI task 0 has started...
MPI task 1 has started...
MPI task 3 has started...
MPI task 4 has started...
MPI task 2 has started...
Round : 0 , average value of pi = 3.13984000
Round : 1 , average value of pi = 3.13640000
Round : 2 , average value of pi = 3.14288000
Round : 3 , average value of pi = 3.14444000
Round : 4 , average value of pi = 3.14118400
Round : 5 , average value of pi = 3.14264000
Round : 6 , average value of pi = 3.14560000
Round : 7 , average value of pi = 3.14452000
Round : 8 , average value of pi = 3.14535111
Round : 9 , average value of pi = 3.14523200
Round : 10 , average value of pi = 3.14462545
Round : 11 , average value of pi = 3.14326667
Round : 12 , average value of pi = 3.14395077
Round : 13 , average value of pi = 3.14274286
Round : 14 , average value of pi = 3.14184533
Round : 15 , average value of pi = 3.14300000
Round : 16 , average value of pi = 3.14211765
Round : 17 , average value of pi = 3.14261333
Round : 18 , average value of pi = 3.14304842
Round : 19 , average value of pi = 3.14244000
Round : 20 , average value of pi = 3.14157714
Round : 21 , average value of pi = 3.14108364
Round : 22 , average value of pi = 3.14103652
```



```
Round : 90 , average value of pi = 3.14187253
Round : 91 , average value of pi = 3.14176696
Round : 92 , average value of pi = 3.14195613
Round : 93 , average value of pi = 3.14195404
Round : 94 , average value of pi = 3.14208000
Round : 95 , average value of pi = 3.14202333
Round : 96 , average value of pi = 3.14207010
Round : 97 , average value of pi = 3.14160163
Round : 98 , average value of pi = 3.14159677
Round : 99 , average value of pi = 3.14141120
Round : 100 , average value of pi = 3.14135446
Round : 101 , average value of pi = 3.14124863
Round : 102 , average value of pi = 3.14124738
Round : 103 , average value of pi = 3.14138308
Round : 104 , average value of pi = 3.14139429
Round : 105 , average value of pi = 3.14151547
Round : 106 , average value of pi = 3.14168224
Round : 107 , average value of pi = 3.14170222
Round : 108 , average value of pi = 3.14192440
Round : 109 , average value of pi = 3.14199273
Round : 110 , average value of pi = 3.14216216
Round : 111 , average value of pi = 3.14251143
Round : 112 , average value of pi = 3.14230655
Round : 113 , average value of pi = 3.14235088
Round : 114 , average value of pi = 3.14254330
Round : 115 , average value of pi = 3.14252690
Round : 116 , average value of pi = 3.14243966
```



```
Round : 555 , average value of pi = 3.14167942
Round : 556 , average value of pi = 3.14168330
Round : 557 , average value of pi = 3.14170753
Round : 558 , average value of pi = 3.14169989
Round : 559 , average value of pi = 3.14166400
Round : 560 , average value of pi = 3.14164848
Round : 561 , average value of pi = 3.14165950
Round : 562 , average value of pi = 3.14165826
Round : 563 , average value of pi = 3.14161730
Round : 564 , average value of pi = 3.14161501
Round : 565 , average value of pi = 3.14163194
Round : 566 , average value of pi = 3.14164346
Round : 567 , average value of pi = 3.14163437
Round : 568 , average value of pi = 3.14160815
Round : 569 , average value of pi = 3.14158428
Round : 570 , average value of pi = 3.14161233
Round : 571 , average value of pi = 3.14159692
Round : 572 , average value of pi = 3.14157682
Round : 573 , average value of pi = 3.14157268
Round : 574 , average value of pi = 3.14153739
Round : 575 , average value of pi = 3.14152167
Round : 576 , average value of pi = 3.14153428
Round : 577 , average value of pi = 3.14156429
Round : 578 , average value of pi = 3.14154003
Round : 579 , average value of pi = 3.14154593
Round : 580 , average value of pi = 3.14152372
```

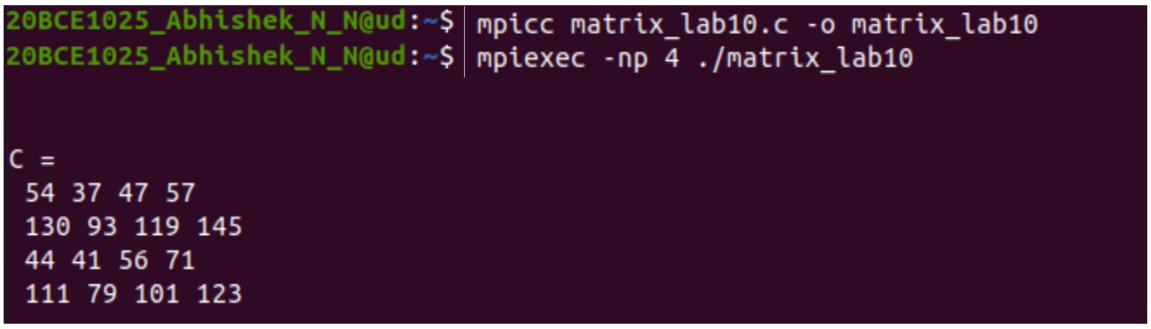


```
Round : 975 , average value of pi = 3.14170230
Round : 976 , average value of pi = 3.14169695
Round : 977 , average value of pi = 3.14170094
Round : 978 , average value of pi = 3.14172846
Round : 979 , average value of pi = 3.14175102
Round : 980 , average value of pi = 3.14175739
Round : 981 , average value of pi = 3.14174143
Round : 982 , average value of pi = 3.14173624
Round : 983 , average value of pi = 3.14173447
Round : 984 , average value of pi = 3.14171744
Round : 985 , average value of pi = 3.14170353
Round : 986 , average value of pi = 3.14169062
Round : 987 , average value of pi = 3.14169279
Round : 988 , average value of pi = 3.14170402
Round : 989 , average value of pi = 3.14171442
Round : 990 , average value of pi = 3.14171189
Round : 991 , average value of pi = 3.14171806
Round : 992 , average value of pi = 3.14172681
Round : 993 , average value of pi = 3.14171525
Round : 994 , average value of pi = 3.14170436
Round : 995 , average value of pi = 3.14168546
Round : 996 , average value of pi = 3.14170078
Round : 997 , average value of pi = 3.14168754
Round : 998 , average value of pi = 3.14169049
Round : 999 , average value of pi = 3.14170816

Real value of PI: 3.1415926535897
```

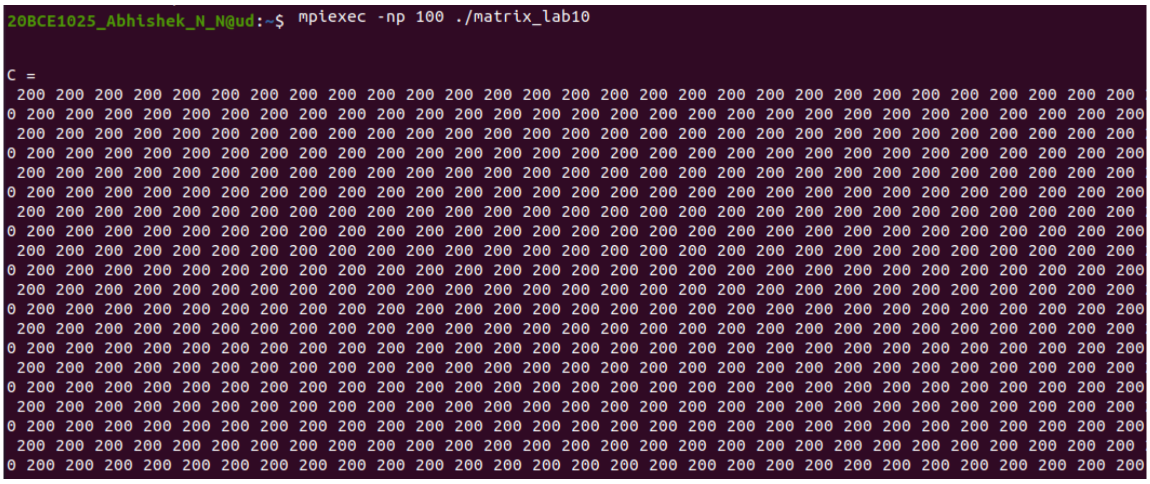
3. Write a MPI program to perform matrix multiplication (1000x1000) using scatter and gather routines.

```
#define N 200
#include <math.h>
#include <mpi.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.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, 2, 3, 4}, {5, 6, 7, 8}, {9, 1, 2, 3},
     {4, 5, 6, 7}};

     int b[N][N]={{1,2,3,4},{5,6,7,8},{9,1,2,3},{4,5,6,7}};
     int c[N][N];
     int a[N][N], b[N][N];
     int aa[N], CC[N];
     for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
               a[i][j] = 1; b[i[j]=2;
          }
     }
     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];
     }
     cc[i] = sum;
```

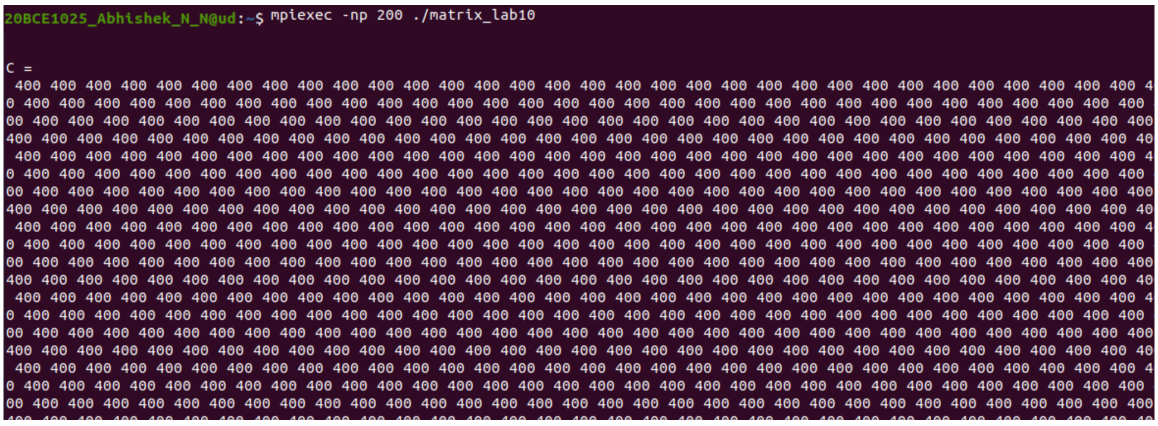
```c
        sum = 0;
        }

        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)
        print_results("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");
}
```



```
20BCE1025_Abhishek_N_N@ud:~$ mpicc matrix_lab10.c -o matrix_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 4 ./matrix_lab10


C =
 54 37 47 57
 130 93 119 145
 44 41 56 71
 111 79 101 123
```



```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 100 ./matrix_lab10


C =
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
```



```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 200 ./matrix_lab10


C =
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
```