CSE 4001

PARALLEL AND DISTRIBUTED COMPUTING

Lab Assessment – 4

L27+L28 | PLBG04
Dr. Narayanan Prasanth

FALL SEMESTER 2021-22

by

SHARADINDU ADHIKARI
19BCE2105

Q1. Develop an MPI program to compute the average of numbers using the following functions and compute the time required for the same.

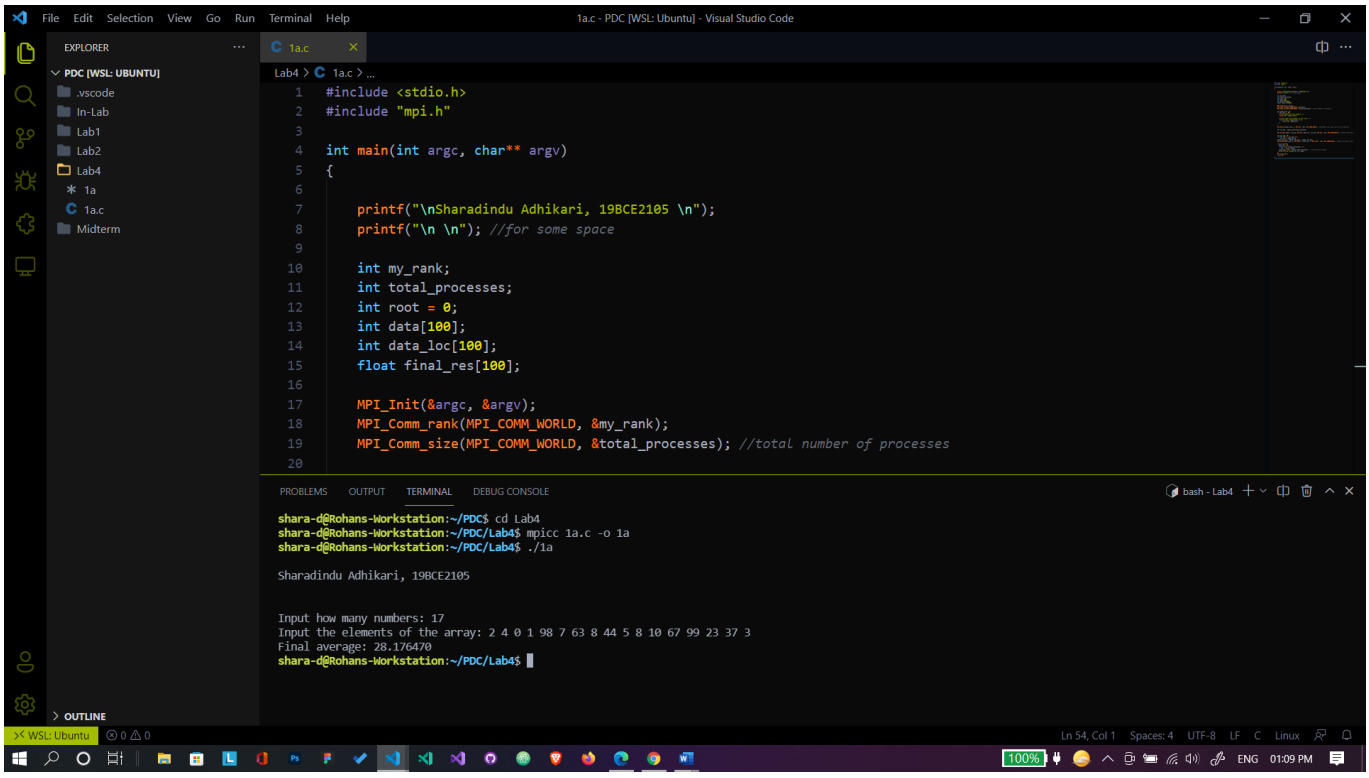
(a) MPI_Scatter and MPI_Gather

Code:

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

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

    printf("\nSharadindu Adhikari, 19BCE2105 \n");
    printf("\n \n"); //for some space

    int my_rank;
    int total_processes;
    int root = 0;
    int data[100];
    int data_loc[100];
    float final_res[100];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &total_processes); //total number of processes

    int input_size = 0;
    if (my_rank == 0){
       printf("Input how many numbers: ");
       scanf("%d", &input_size);

       printf("Input the elements of the array: ");
       for(int i=0; i<input_size; i++){
           scanf("%d", &data[i]);
       }
    }

    MPI_Bcast(&input_size, 1, MPI_INT, root, MPI_COMM_WORLD); //broadcast the
input size to all processes

    int loc_num = input_size/total_processes;

    MPI_Scatter(&data, loc_num, MPI_INT, data_loc, loc_num, MPI_INT, root,
MPI_COMM_WORLD); //scatter the data to all processes
```

```
    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;
    MPI_Gather(&loc_avg, 1, MPI_FLOAT, final_res, 1, MPI_FLOAT, root,
MPI_COMM_WORLD); //gather the data from all processes

    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;  //calculate the average
      printf("Final average: %f \n", avg);
    }
    MPI_Finalize();
    return 0;
}
```

Input & Output:



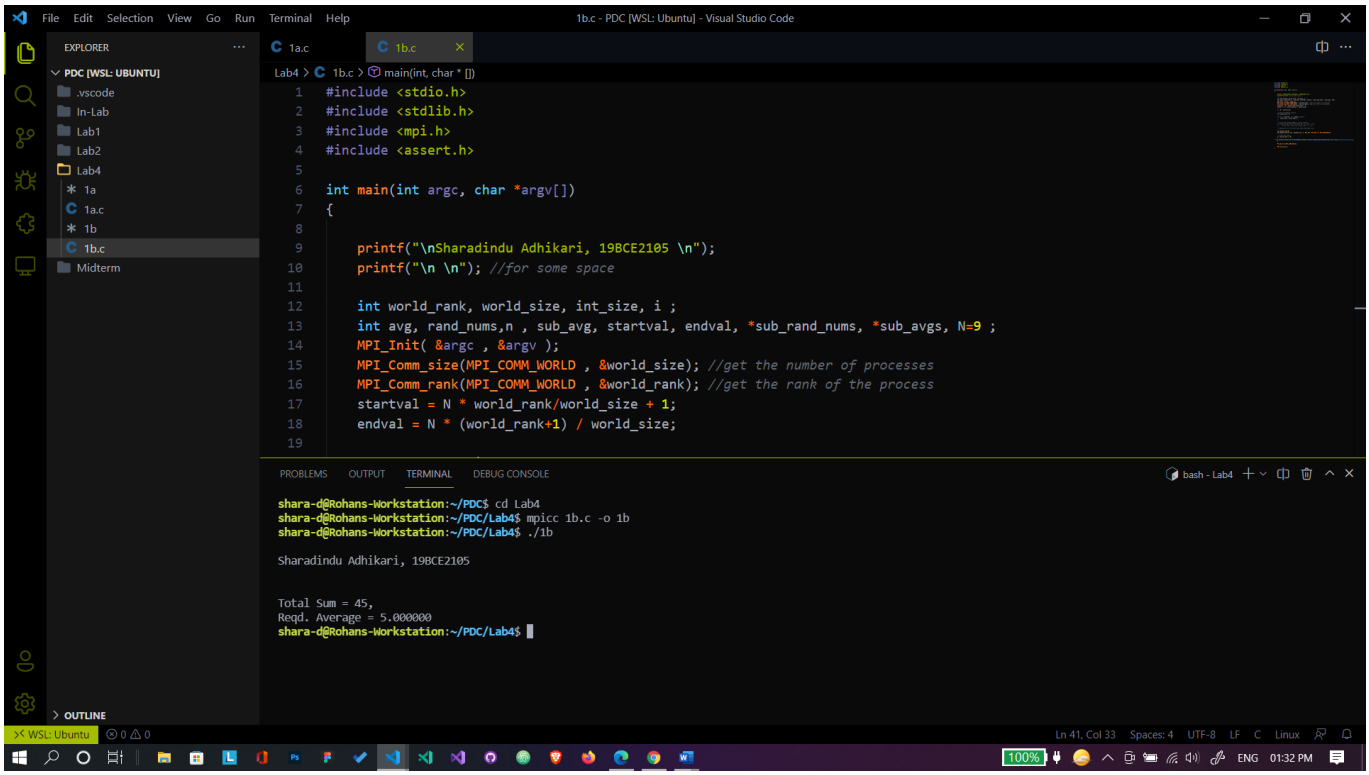
(b) MPI_Scatter and MPI_Reduce

Code & Input:

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

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

    printf("\nSharadindu Adhikari, 19BCE2105 \n");
    printf("\n \n"); //for some space

    int world_rank, world_size, int_size, i ;
    int avg, rand_nums,n , sub_avg, startval, endval, *sub_rand_nums, *sub_avgs,
N=9 ;
    MPI_Init( &argc , &argv );
    MPI_Comm_size(MPI_COMM_WORLD , &world_size); //get the number of processes
    MPI_Comm_rank(MPI_COMM_WORLD , &world_rank); //get the rank of the process
    startval = N * world_rank/world_size + 1;
    endval = N * (world_rank+1) / world_size;

    n = N / world_size;

    // Sum the numbers locally
    int local_sum = 0;

    for (i = startval; i <= endval; i++) {
        local_sum = local_sum+i;
    }


    int global_sum=0;
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the result
    if (world_rank == 0)
    {
       printf("Total Sum = %d,\nReqd. Average = %f\n", global_sum, global_sum *
1. /  N);
    }
```

```
    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
}
```

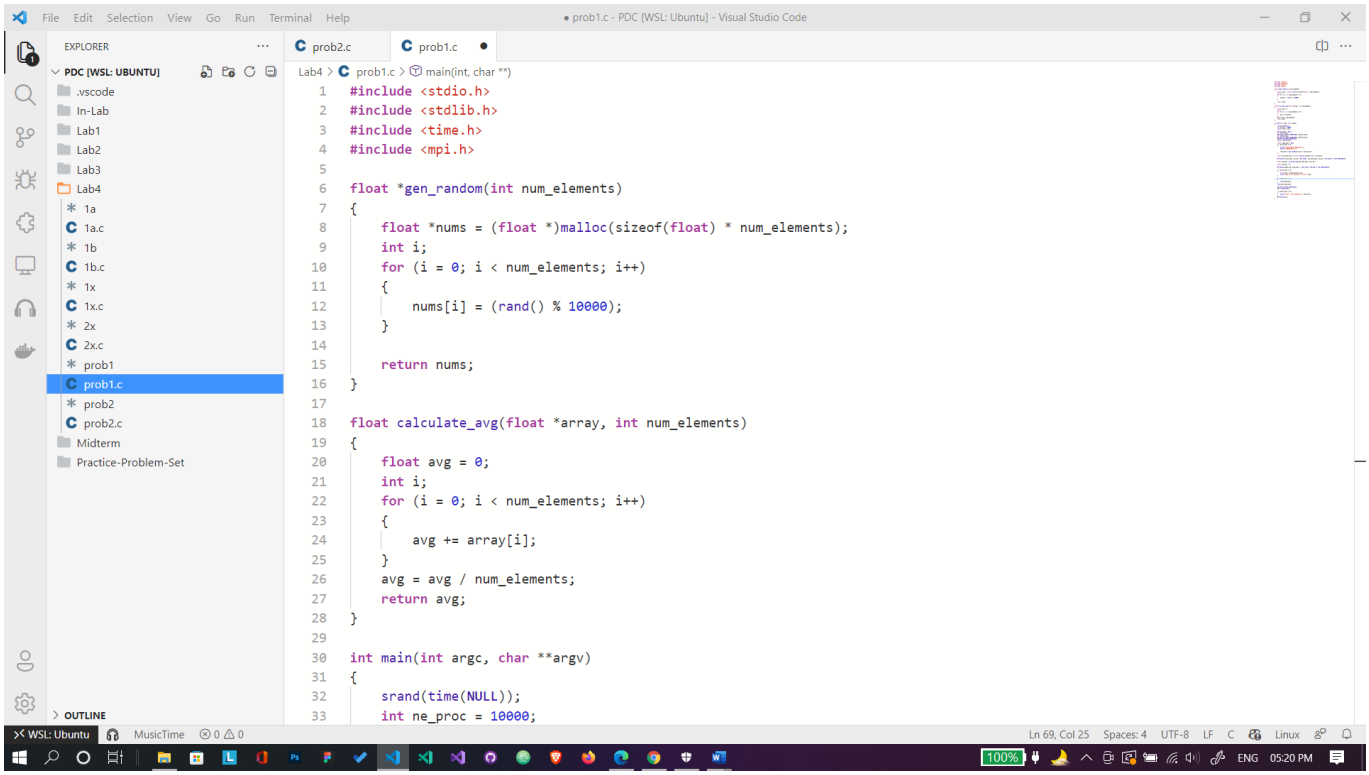
Output:



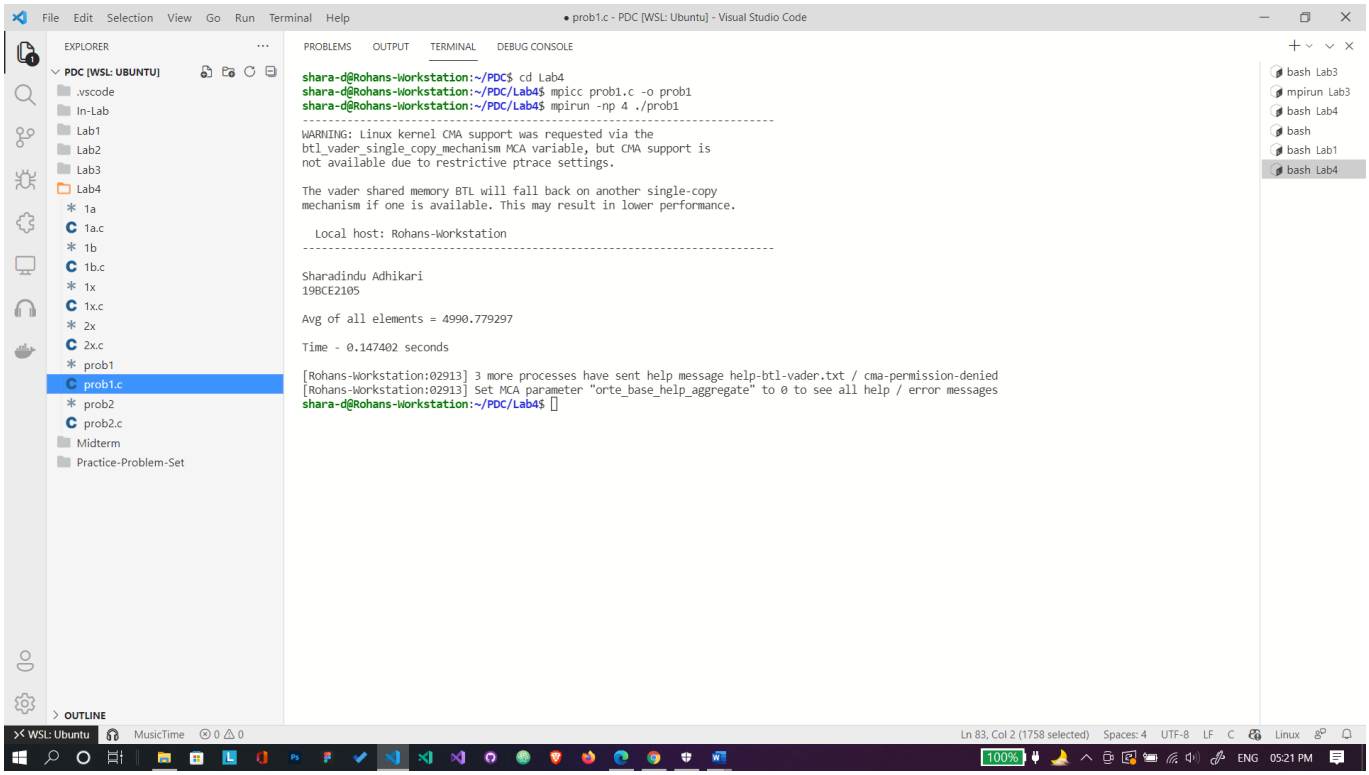
Alternative Solution to Q1:

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

float *gen_random(int num_elements)
{
    float *nums = (float *)malloc(sizeof(float) * num_elements);
    int i;
    for (i = 0; i < num_elements; i++)
    {
        nums[i] = (rand() % 10000);
    }

    return nums;
}

float calculate_avg(float *array, int num_elements)
{
    float avg = 0;
    int i;
    for (i = 0; i < num_elements; i++)
    {
        avg += array[i];
    }
    avg = avg / num_elements;
    return avg;
}

int main(int argc, char **argv)
{
    srand(time(NULL));
    int ne_proc = 10000;
    double start, end;

    MPI_Init(NULL, NULL);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Barrier(MPI_COMM_WORLD);
```

```
    start = MPI_Wtime();

    float *rand_nums = NULL;
    if (world_rank == 0)
    {
        printf("\nSharadindu Adhikari\n");
        printf("19BCE2105\n\n");

        rand_nums = gen_random(ne_proc * world_size);
    }

    float *sub_rand_nums = (float *)malloc(sizeof(float) * ne_proc);

    MPI_Scatter(rand_nums, ne_proc, MPI_FLOAT, sub_rand_nums, ne_proc,
MPI_FLOAT, 0, MPI_COMM_WORLD);

    float sub_avg = calculate_avg(sub_rand_nums, ne_proc);

    float sub_avgs = 0;

    MPI_Reduce(&sub_avg, &sub_avgs, 1, MPI_FLOAT, MPI_SUM, 0,
MPI_COMM_WORLD);

    if (world_rank == 0)
    {
        float avg = sub_avgs/world_size;
        printf("Avg of all elements = %f\n\n", avg);
    }

    if (world_rank == 0)
    {
        free(rand_nums);
    }
    free(sub_rand_nums);

    MPI_Barrier(MPI_COMM_WORLD);
    end = MPI_Wtime();

    if (world_rank == 0)
    {
        printf("Time - %lf seconds\n\n", end-start);
    }
    MPI_Finalize();
}
```


File Edit Selection View Go Run Terminal Help
• prob1.c - PDC [WSL: Ubuntu] - Visual Studio Code
EXPLORER
PDC [WSL: UBUNTU]
.vscode
In-Lab
Lab1
Lab2
Lab3
Lab4
1a
1a.c
1b
1b.c
1x
1x.c
2x
2x.c
prob1
prob1.c
prob2
prob2.c
Midterm
Practice-Problem-Set
OUTLINE
prob2.c
prob1.c
Lab4 > prob1.c > main(int, char **)
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
float *gen_random(int num_elements)
{
float *nums = (float *)malloc(sizeof(float) * num_elements);
int i;
for (i = 0; i < num_elements; i++)
{
nums[i] = (rand() % 10000);
}
return nums;
}
float calculate_avg(float *array, int num_elements)
{
float avg = 0;
int i;
for (i = 0; i < num_elements; i++)
{
avg += array[i];
}
avg = avg / num_elements;
return avg;
}
int main(int argc, char **argv)
{
srand(time(NULL));
int ne_proc = 10000;
WSL: Ubuntu MusicTime
Ln 69, Col 25 Spaces: 4 UTF-8 LF C Linux
100%
ENG 05:20 PM

OUTPUT:

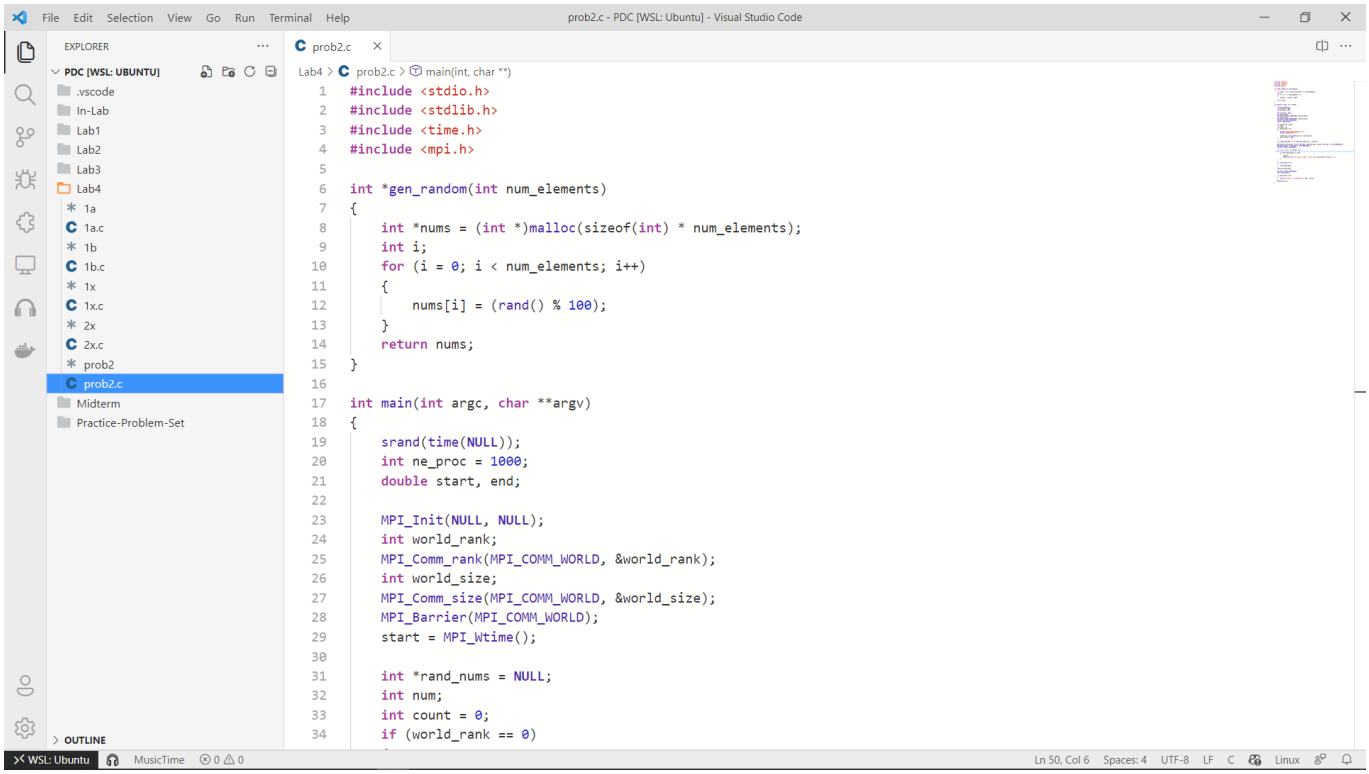

File Edit Selection View Go Run Terminal Help
• prob1.c - PDC [WSL: Ubuntu] - Visual Studio Code
EXPLORER
PDC [WSL: UBUNTU]
.vscode
In-Lab
Lab1
Lab2
Lab3
Lab4
1a
1a.c
1b
1b.c
1x
1x.c
2x
2x.c
prob1
prob1.c
prob2
prob2.c
Midterm
Practice-Problem-Set
OUTLINE
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
shara-d@Rohans-Workstation:~/PDC$ cd Lab4
shara-d@Rohans-Workstation:~/PDC/Lab4$ mpicc prob1.c -o prob1
shara-d@Rohans-Workstation:~/PDC/Lab4$ mpirun -np 4 ./prob1
WARNING: Linux kernel CMA support was requested via the
btl_vader_single_copy_mechanism MCA variable, but CMA support is
not available due to restrictive ptrace settings.
The vader shared memory BTL will fall back on another single-copy
mechanism if one is available. This may result in lower performance.
Local host: Rohans-Workstation
Sharadindu Adhikari
19BCE2105
Avg of all elements = 4990.779297
Time - 0.147402 seconds
[Rohans-Workstation:02913] 3 more processes have sent help message help-btl-vader.txt / cma-permission-denied
[Rohans-Workstation:02913] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
shara-d@Rohans-Workstation:~/PDC/Lab4$
bash Lab3
mpirun Lab3
bash Lab4
bash
bash Lab1
bash Lab4
WSL: Ubuntu MusicTime
Ln 83, Col 2 (1758 selected) Spaces: 4 UTF-8 LF C Linux
100%
ENG 05:21 PM

Q2.

2. Develop a MPI program to find the occurrence of a number in a given set (size not less than 20). Use collective communication to implement the same and print the time taken to complete the process. (4)

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

int *gen_random(int num_elements)
{
    int *nums = (int *)malloc(sizeof(int) * num_elements);
    int i;
    for (i = 0; i < num_elements; i++)
    {
        nums[i] = (rand() % 100);
    }
    return nums;
}

int main(int argc, char **argv)
{
    srand(time(NULL));
    int ne_proc = 1000;
    double start, end;

    MPI_Init(NULL, NULL);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();

    int *rand_nums = NULL;
    int num;
    int count = 0;
    if (world_rank == 0)
    {
        printf("\nSharadindu Adhikari\n");
```

```
        printf("19BCE2105\n\n");

        rand_nums = gen_random(ne_proc * world_size);
        num = rand() % 100;
    }

    int *sub_rand_nums = (int *)malloc(sizeof(int) * ne_proc);

    MPI_Scatter(rand_nums, ne_proc, MPI_INT, sub_rand_nums, ne_proc,
MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&num, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    for (int i = 0; i < ne_proc; i++)
    {
        if (sub_rand_nums[i] == num)
        {
            count++;
            printf("Element %d found at index : %d\n", num, world_rank *
ne_proc + i);
        }
    }

    if (world_rank == 0)
    {
        free(rand_nums);
    }
    free(sub_rand_nums);

    MPI_Barrier(MPI_COMM_WORLD);
    end = MPI_Wtime();

    if (world_rank == 0)
    {
        printf("\nTime - %lf seconds\n\n", end - start);
    }
    MPI_Finalize();
}
```



FINAL OUTPUT: