PPL Week-3

Rachit Takate – 230962294 – B54

I. Write a MPI program to read N values in the root process. Root process sends one value to each process. Every process receives it and finds the factorial of that number and returns to the root process. Root process gathers the factorial and finds sum of it. Use N number of processes.

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

int main(int argc, char* argv[]) {
    int rank, size, *arr, val, res = 1;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Status stat;


    if(!rank) {
        arr = calloc(size, sizeof(int));
        for(int i = 0; i < size; i++)
            scanf("%d", &arr[i]);
    }

    MPI_Scatter(arr, 1, MPI_INT, &val, 1, MPI_INT, 0, MPI_COMM_WORLD);

    for(int i = 2; i <= val; i++)
        res *= i;

    MPI_Reduce(&res, &val, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if(!rank) {
        printf("Final val: %d\n", val);
        printf("Rachit 230962294\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output

```
^C(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpicc Q1.c -o Q1
(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpirun -np 4 ./Q1
2 3 4 5
Final val: 152
Rachit 230962294
```

II. Write a MPI program to read an integer value M and NxM elements into an 1D array in the root process, where N is the number of processes. Root process sends M elements to each process. Each process finds average of M elements it received and sends these average values to root. Root process collects all the values and finds the total average. Use collective communication routines.

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

int main(int argc, char* argv[]) {
    int rank, size, m;
    float *mat, *arr, res;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Status stat;

    if(!rank) {
        scanf("%d", &m);
        mat = calloc(size * m, sizeof(float));

        for(int i = 0; i < size * m; i++)
            scanf("%f", &mat[i]);
    }

    MPI_Bcast(&m, 1, MPI_INT, 0, MPI_COMM_WORLD);
    arr = calloc(m, sizeof(float));
    MPI_Scatter(mat, m, MPI_FLOAT, arr, m, MPI_FLOAT, 0, MPI_COMM_WORLD);

    for(int i = 1; i < m; i++)
        arr[0] += arr[i];
    arr[0] /= m;
    MPI_Reduce(&arr[0], &res, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

    if(!rank) {
        res /= size;
        printf("Final val: %f\n", res);
        printf("Rachit 230962294\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output

```
(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpicc Q2.c -o Q2
(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpirun -np 3 ./Q2
3
1 2 3
2 3 4
3 4 5
Final val: 3.000000
Rachit 230962294
```

III. Write a MPI program to read a string. Using N processes (string length is evenly divisible by N), find the number of non-vowels in the string. In the root process print number of non-vowels found by each process and print the total number of non-vowels.

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

int main(int argc, char* argv[]) {
    int rank, size, len, *buff, acc = 0;
```

```
    char inp[32], *subs;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Status stat;

    if(!rank) {
        gets(inp);
        len = strlen(inp);
        buff = calloc(size, sizeof(int));
    }

    MPI_Bcast(&len, 1, MPI_INT, 0, MPI_COMM_WORLD);
    len /= size;

    subs = calloc(len, sizeof(char));
    MPI_Scatter(inp, len, MPI_CHAR, subs, len, MPI_CHAR, 0, MPI_COMM_WORLD);

    for(int i = 0; i < len; i++)
        switch (subs[i]) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            break;

        default:
            acc++;
            break;
        }

    MPI_Gather(&acc, 1, MPI_INT, buff, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if(!rank) {
        acc = 0;
        for(int i = 0; i < size; i++) {
            printf("P%d count: %d\n", i, buff[i]);
            acc += buff[i];
        }

        printf("Final val: %d\n", acc);
        printf("Rachit 230962294\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output

```
(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpirun -np 3 ./Q3
length
P0 count: 1
P1 count: 2
P2 count: 2
Final val: 5
Rachit 230962294
```

IV. Write a MPI Program to read two strings S1 and S2 of same length in the root process. Using N processes including the root (string length is evenly divisible by N), produce the

resultant string as shown below. Display the resultant string in the root process. Use Collective communication routines.

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

int main(int argc, char* argv[]) {
    int rank, size, len;
    char inp1[32], inp2[32], *subs, *final;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Status stat;

    if(!rank) {
        gets(inp1);
        gets(inp2);

        len = strlen(inp1);
        final = calloc(2 * len, sizeof(char));
    }

    MPI_Bcast(&len, 1, MPI_INT, 0, MPI_COMM_WORLD);
    len /= size;

    subs = calloc(2 * len, sizeof(char));
    MPI_Scatter(inp1, len, MPI_CHAR, subs, len, MPI_CHAR, 0, MPI_COMM_WORLD);
    MPI_Scatter(inp2, len, MPI_CHAR, subs + len, len, MPI_CHAR, 0,
MPI_COMM_WORLD);

    MPI_Gather(subs, len * 2, MPI_CHAR, final, len * 2, MPI_CHAR, 0,
MPI_COMM_WORLD);

    if(!rank) {
        printf("Str: %s\n", final);
        printf("Rachit 230962294\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output

```
(dse) mca@computinglab26-26:~/230962294/PP Lab/Week3$ mpirun -np 3 ./Q4
string
length
Str: stleringngth
Rachit 230962294
```