Lab Activity

Course: Parallel and Distributed Computing

Course Code: CSE 3009

Faculty: Dr. Amrita Parashar

Submitted By:

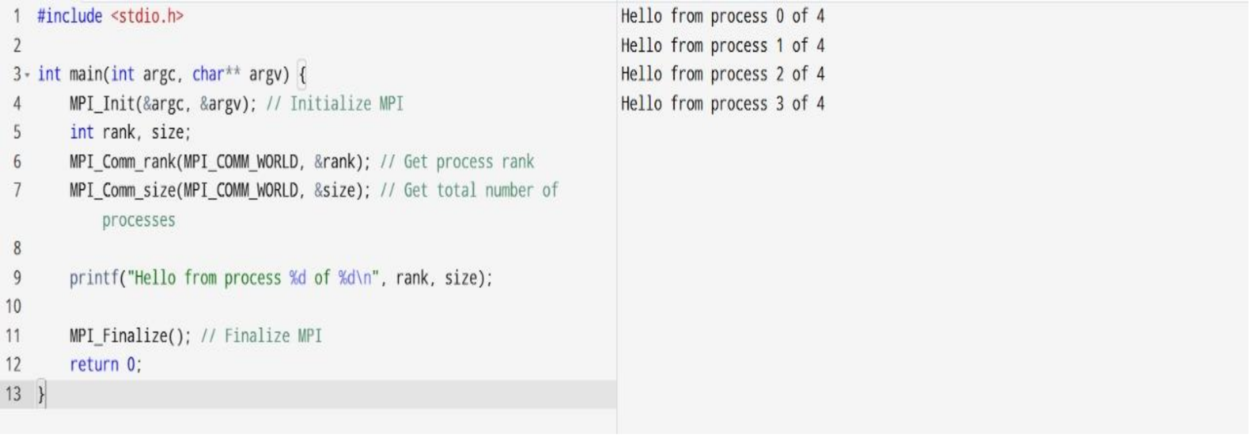
Name: Navneet Tiwari
Reg. No. : 22BCE10311

Submitted On:

04 April 2025

Implement the following problems

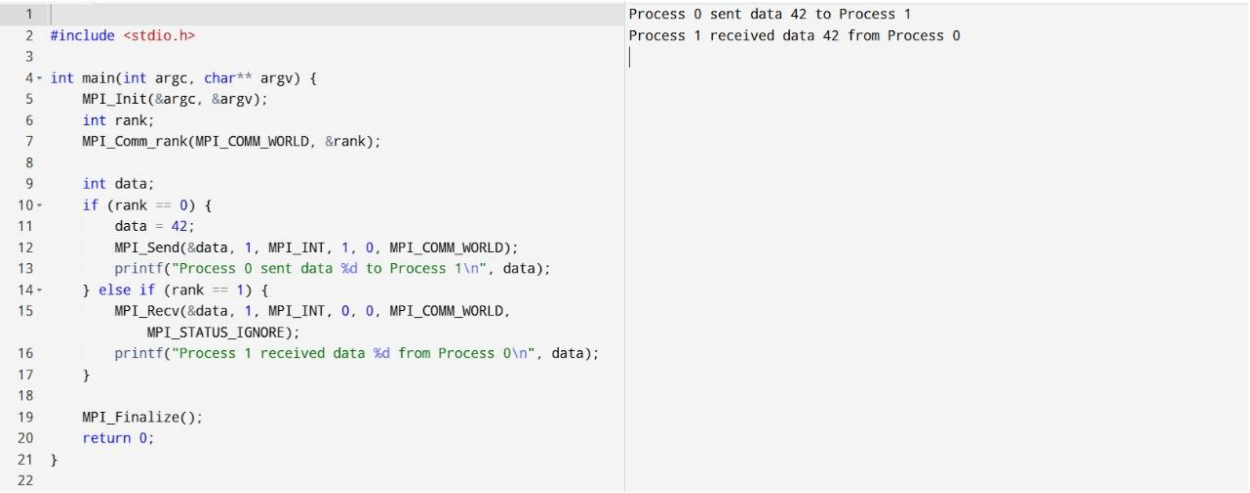
1. MPI – Basics of MPI



```c
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv); // Initialize MPI
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Get process rank
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Get total number of processes

    printf("Hello from process %d of %d\n", rank, size);

    MPI_Finalize(); // Finalize MPI
    return 0;
}
```

```
Hello from process 0 of 4
Hello from process 1 of 4
Hello from process 2 of 4
Hello from process 3 of 4
```

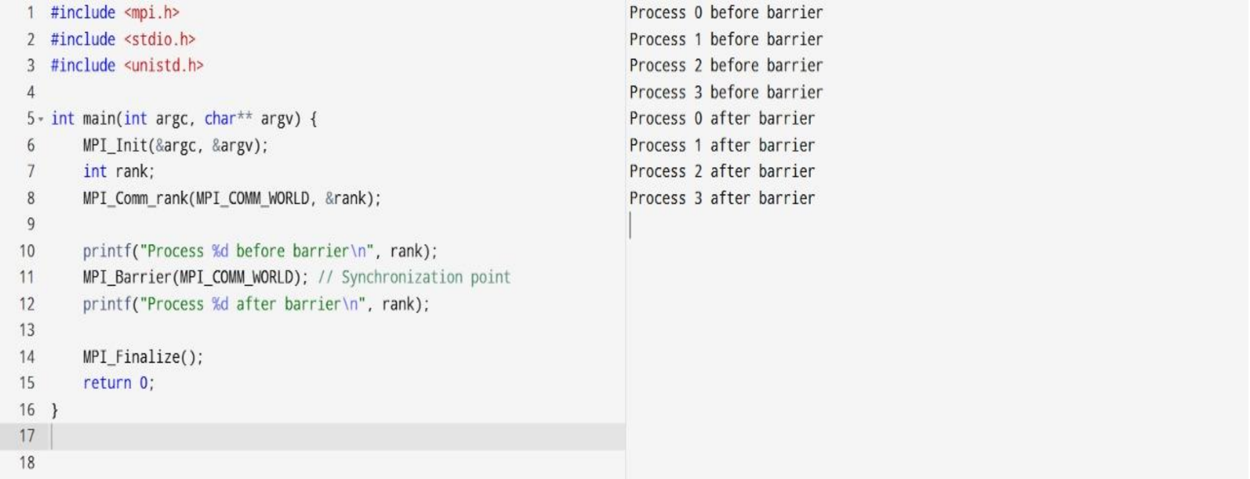
2. MPI – Communication between MPI process



```c

#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int data;
    if (rank == 0) {
        data = 42;
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent data %d to Process 1\n", data);
    } else if (rank == 1) {
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received data %d from Process 0\n", data);
    }

    MPI_Finalize();
    return 0;
}

```

```
Process 0 sent data 42 to Process 1
Process 1 received data 42 from Process 0
```

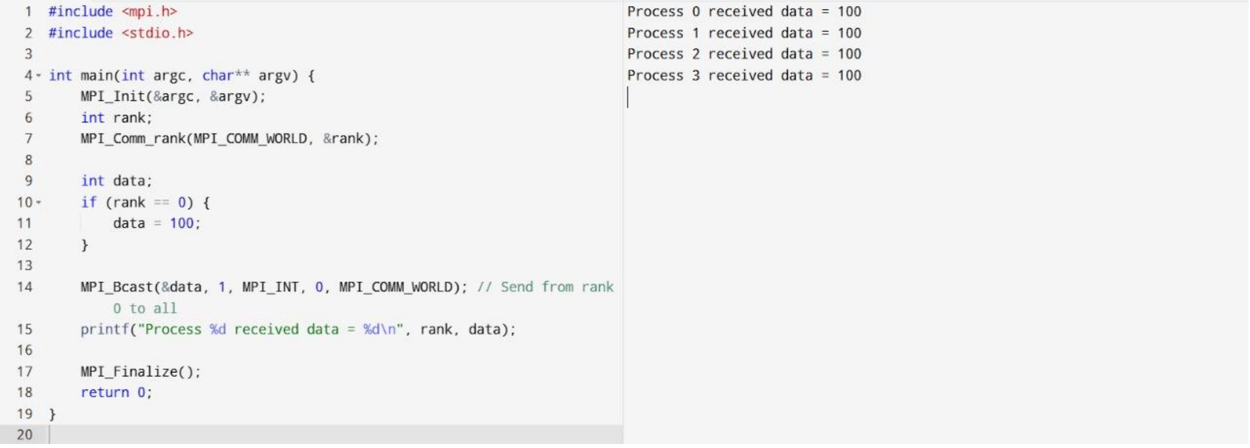
3. MPI – Collective operation with "synchronization"



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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Process %d before barrier\n", rank);
    MPI_Barrier(MPI_COMM_WORLD); // Synchronization point
    printf("Process %d after barrier\n", rank);

    MPI_Finalize();
    return 0;
}

```

```
Process 0 before barrier
Process 1 before barrier
Process 2 before barrier
Process 3 before barrier
Process 0 after barrier
Process 1 after barrier
Process 2 after barrier
Process 3 after barrier
```

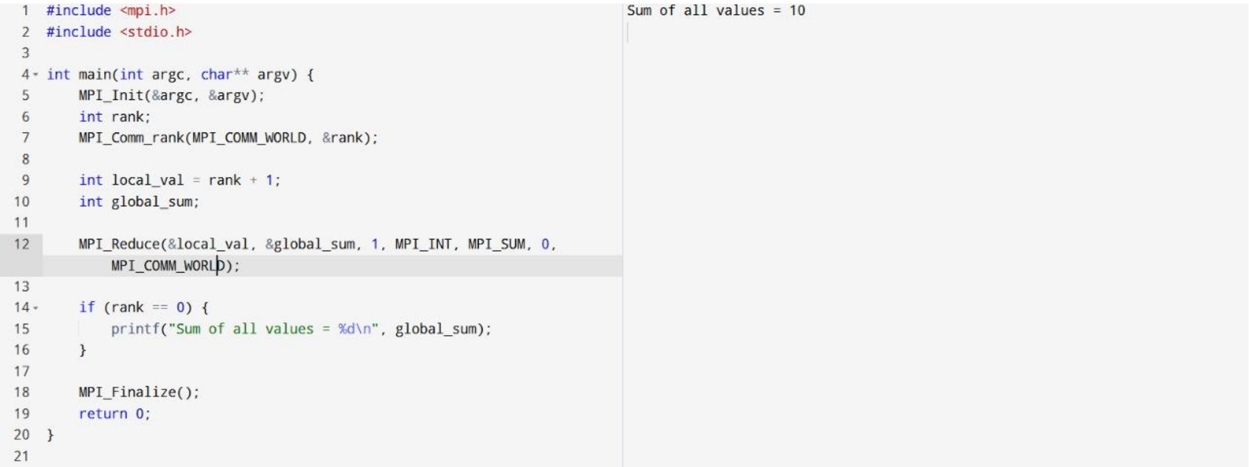
4. MPI – Collective operation with "data movement"



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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int data;
    if (rank == 0) {
        data = 100;
    }

    MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD); // Send from rank 0 to all
    printf("Process %d received data = %d\n", rank, data);

    MPI_Finalize();
    return 0;
}
```

```
Process 0 received data = 100
Process 1 received data = 100
Process 2 received data = 100
Process 3 received data = 100
```

5. MPI – Collective operation with "collective computation"



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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int local_val = rank + 1;
    int global_sum;

    MPI_Reduce(&local_val, &global_sum, 1, MPI_INT, MPI_SUM, 0,
        MPI_COMM_WORLD);

    if (rank == 0) {
        printf("Sum of all values = %d\n", global_sum);
    }

    MPI_Finalize();
    return 0;
}
```

```
Sum of all values = 10
```