

Lab 9- CSE 4001

Name - Ayush Sharma
Reg no - 15BCE1335
Faculty: Prof Gayatri R.

1) Send and Receive program of MPI

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // We are assuming at least 2 processes for this task
    if (world_size < 2) {
        fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number;
    if (world_rank == 0) {
        // If we are rank 0, set the number to -1 and send it to process 1
        number = -1;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (world_rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number);
    }
    MPI_Finalize();
}
```

```
Terminal
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $ ./run.py send_
mpirun -n 2 ./mpi-send-and-receive/code/send_recv
Process 1 received number -1 from process 0
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $
```

2) Ping Pong Problem

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

```
int main(int argc, char** argv) {
    const int PING_PONG_LIMIT = 10;

    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```

// We are assuming at least 2 processes for this task
if (world_size != 2) {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
            world_rank, ping_pong_count, partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
            world_rank, ping_pong_count, partner_rank);
    }
}
MPI_Finalize();
}

```

```

Terminal
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $ ./run.py ping_pong
mpirun -n 2 ./mpi-send-and-receive/code/ping_pong
0 sent and incremented ping_pong_count 1 to 1
0 received ping_pong_count 2 from 1
0 sent and incremented ping_pong_count 3 to 1
0 received ping_pong_count 4 from 1
0 sent and incremented ping_pong_count 5 to 1
0 received ping_pong_count 6 from 1
0 sent and incremented ping_pong_count 7 to 1
0 received ping_pong_count 8 from 1
0 sent and incremented ping_pong_count 9 to 1
0 received ping_pong_count 10 from 1
1 received ping_pong_count 1 from 0
1 sent and incremented ping_pong_count 2 to 0
1 received ping_pong_count 3 from 0
1 sent and incremented ping_pong_count 4 to 0
1 received ping_pong_count 5 from 0
1 sent and incremented ping_pong_count 6 to 0
1 received ping_pong_count 7 from 0
1 sent and incremented ping_pong_count 8 to 0
1 received ping_pong_count 9 from 0
1 sent and incremented ping_pong_count 10 to 0
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $

```

3) Ring

```

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

```

```

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int token;
    // Receive from the lower process and send to the higher process. Take care
    // of the special case when you are the first process to prevent deadlock.
    if (world_rank != 0) {
        MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        printf("Process %d received token %d from process %d\n", world_rank, token,
                world_rank - 1);
    }
}

```

```

} else {
    // Set the token's value if you are process 0
    token = -1;
}
MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
        MPI_COMM_WORLD);
// Now process 0 can receive from the last process. This makes sure that at
// least one MPI_Send is initialized before all MPI_Recv (again, to prevent
// deadlock)
if (world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n", world_rank, token,
            world_size - 1);
}
MPI_Finalize();
}

```

```

Terminal
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $ ./run.py ring
mpirun -n 5 ./mpi-send-and-receive/code/ring

Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 4 received token -1 from process 3
Process 0 received token -1 from process 4
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $ 
rom process 1
rom process 2
rom process 3
rom process 4

ends a value of negative one to process one.
g until it gets back to process zero.

```