1. Introduction:

This report discusses the implementation of a parallelized MPI program for a set of tasks communicating in a ring topology. The primary goal is to demonstrate the use of MPI in a parallel programming environment while managing communication between tasks to avoid deadlocks. The C program, ring.c, is developed to achieve the required steps, including a left circular shift of data and ordered output.

1. The Left Circular Shift

Circular shift refers to a technique where elements of an array (or a data structure) are rearranged such that they move by a specific number of positions, either to the left or right, while maintaining their order. When an element reaches the boundary (the beginning or the end of the array), it "circles around" to the opposite side and continues shifting.

In the ring topology implementation, each task is connected to its immediate left and right neighbors. Tasks at the boundary are connected in such a way that the left neighbor of the first task is the last task, while the right neighbor of the last task is the first task, forming a circular structure. For example, consider four tasks from 0 to 3. The ring topology is represented by the following diagram:

Diagram

Description automatically generated

Figure 1: a ring topology with 4 tasks

When designing the left circular shift for parallel programming, care must be taken since the shift will be executed simultaneously by all tasks. Without careful design, it can lead to a deadlock situation. Deadlock occurs when each task in the system is waiting for some other tasks to release a resource or complete a particular operation, forming a cycle of dependencies. The pseudocode for a single left circular shift is described as:

Send m to the left

Receive m from the right

Because each task initializes its m value independently, the above pseudocode can guarantee that no deadlock occurs. Next, we can move on to implement it in the C program. It is essential that a task only communicates with its immediate neighbors. We can define the left and right neighbors using the rank of the task and the total number of initialized tasks:

|  |
| --- |
| int world\_rank, world\_size;  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);  int right = (world\_rank + 1) % world\_size;  int left = (world\_rank - 1 + world\_size) % world\_size; |

The modulo operation is used to ensure that it will "circle around." The pseudocode can then be implemented (in order) as:

|  |
| --- |
| MPI\_Send(&m, 1, MPI\_INT, left, 0, MPI\_COMM\_WORLD);  MPI\_Recv(&m, 1, MPI\_INT, right, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE); |

For performing a left circular shift three places, we use a for-loop and perform the left circular shift three times for each task:

|  |
| --- |
| for (int i = 0; i < 3; i++)  {  MPI\_Send(&m, 1, MPI\_INT, left, 0, MPI\_COMM\_WORLD);      MPI\_Recv(&m, 1, MPI\_INT, right, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  } |

1. Strategy for Achieving Ordered Output

Each task must output its m-value after performing the left circular shift three times. Moreover, the outputs must be ordered based on the rank of the task, and only point-to-point communication is allowed. To differentiate between the messages sent during the circular shift and those sent for ordered output, we use different message tags. This approach helps to avoid confusion and ensure proper communication between tasks.

Each task can output the m-value on its own. However, the order of the output is unpredictable. This is because the MPI tasks are competing for access to the shared To address this issue, we can designate a single task to handle the m-value output so that we can order these outputs based on the rank. This task will be the master, while the others would be workers. In the implementation, task 0 is chosen as the master task. The master task is responsible for collecting the m-value from the worker tasks, organizing them based on rank, and then printing the m-values in order.

To address this issue, we can designate a single task to handle the m-value output so that we can order these outputs based on the rank. This task will be the master, while the others would be workers. In the implementation, task 0 is chosen as the master task. The master task is responsible for collecting the m-value from the worker tasks, organizing them based on rank, and then printing the m-values in order

During the left circular shift, we use a message tag of 0 for the communication between tasks. When collecting the m-values for ordered output, we use a message tag of 1. By using different message tags, we ensure that the master task can correctly identify and process the messages related to ordered output

This can be achieved by having all the worker tasks send their m-values as messages using MPI\_Send() with a message tag of 1 to the master. Then, using a for-loop, the master will collect these messages and print them in order. We use the if-else clause to partition the work between the master and the workers. The code can be implemented as:

|  |
| --- |
| if (world\_rank == 0)  {  for (int i = 0; i < world\_size; i++) {      if (i == 0) {              printf("Task %d: %d\n", i, m);          } else {  int m\_received;              MPI\_Recv(&m\_received, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);              printf("Task %d: %d\n", i, m\_received);          }      }  } else {  MPI\_Send(&m, 1, MPI\_INT, master, 1, MPI\_COMM\_WORLD);  } |

1. Record of communication between tasks 0 and 3

The communication record between tasks 0 and 3 demonstrates the left circular shift operation within the ring topology. Task 0 sends its m-value to task 3 (its left neighbor), and task 3 receives the m-value from task 0 (its right neighbor). This pattern aligns with the left circular shift operation, where each task sends its m-value to its left neighbor and receives an m-value from its right neighbor

This record is a part of the overall communication that takes place during the left circular shift operation, with all tasks in the ring simultaneously performing send and receive steps. The record confirms that the implementation works as intended, properly shifting m-values across the ring topology

1. Orderd output of m-values

The ordered output of m-values shows the result of the left circular shift operation performed by the tasks within the ring topology. After executing the shift three places to the left, the final m-values for each task are presented in an ordered manner based on their task ranks. The output indicates that the program successfully completed the left circular shift and achieved the desired ordering of the m-values.