Overlapping Communication with Computation

Instructor

Dr B. Krishna Priya

Non-Blocking Communication Operations

- MPI provides a pair of functions for performing non-blocking send and receive operations. These functions are MPI_Isend and MPI_Irecv.
- MPI_Isend starts a send operation but does not complete, that is, it returns before the data is copied out of the buffer.
- MPI_Irecv starts a receive operation but returns before the data has been received and copied into the buffer.
- With the support of appropriate hardware, the transmission and reception of messages can proceed concurrently with the computations performed by the program upon the return.

- The calling sequences of MPI_Isend and MPI_Irecv are the following:
- int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
- int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

- a process that has started a non-blocking send or receive operation must make sure that this operation has completed before it proceeds with its computations.
- This is because a process that has started a non-blocking send operation may want to overwrite the buffer that stores the data that are being sent, or a process that has started a non-blocking receive operation may want to use the data it requested.

- MPI provides a pair of functions MPI_Test and MPI_Wait.
- int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
- int MPI_Wait(MPI_Request *request, MPI_Status *status)
- The request objectin MPI_Isend and MPI_Irecv is used as an argument in the MPI_Test and MPI_Wait.

- MPI_Test tests whether or not the non-blocking send or receive operation identified by its request has finished.
- It returns flag = {true} (non-zero value in C) if it completed, otherwise it returns {false} (a zero value in C).
- non-blocking operation has finished, the request object pointed to by request is deallocated and request is set to MPI_REQUEST_NULL.

- The MPI_Wait function blocks until the non-blocking operation identified by request completes.
- In that case it deal-locates the request object, sets it to MPI_REQUEST_NULL, and returns information about the completed operation in the status object.

- The programmer wants to explicitly deallocate a request object, MPI provides the following function.
- int MPI_Request_free(MPI_Request *request)
- Deallocation of the request object does not have any effect on the associated non□blocking send or receive operation.

- Avoiding Deadlocks By using non-blocking communication operations we can remove most of the deadlocks associated with their blocking counterparts.

```
 1    int a[10], b[10], myrank;
 2    MPI_Status status;
 3    ...
 4     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 5    if (myrank == 0) {
 6      MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
 7      MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
 8    }
 9    else if (myrank == 1) {
10      MPI_Recv(b, 10, MPI_INT, 0, 2, &status, MPI_COMM_WORLD);
11      MPI_Recv(a, 10, MPI_INT, 0, 1, &status, MPI_COMM_WORLD);
12    }
13    ...
```

However, if we replace either the send or receive operations with their non-blocking counterparts, then the code will be safe, and will correctly run on any MPI implementation.

```
 1    int a[10], b[10], myrank;
 2    MPI_Status status;
 3    MPI_Request requests[2];
 4    ...
 5     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 6    if (myrank == 0) {
 7      MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
 8      MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
 9    }
10    else if (myrank == 1) {
11      MPI_Irecv(b, 10, MPI_INT, 0, 2, &requests[0], MPI_COMM_WORLD);
12      MPI_Irecv(a, 10, MPI_INT, 0, 1, &requests[1], MPI_COMM_WORLD);
13    }
14    ...
```

This example also illustrates that the non-blocking operations started by any process can finish in any order depending on the transmission or reception of the corresponding messages. For example, the second receive operation will finish before the first does.