MPI Point-to-Point Communication II

Michaela Lechner, PDC Center for HPC

based on work by Michael Hanke

Summer School on High Performance Computing

Outline

What You Already Know

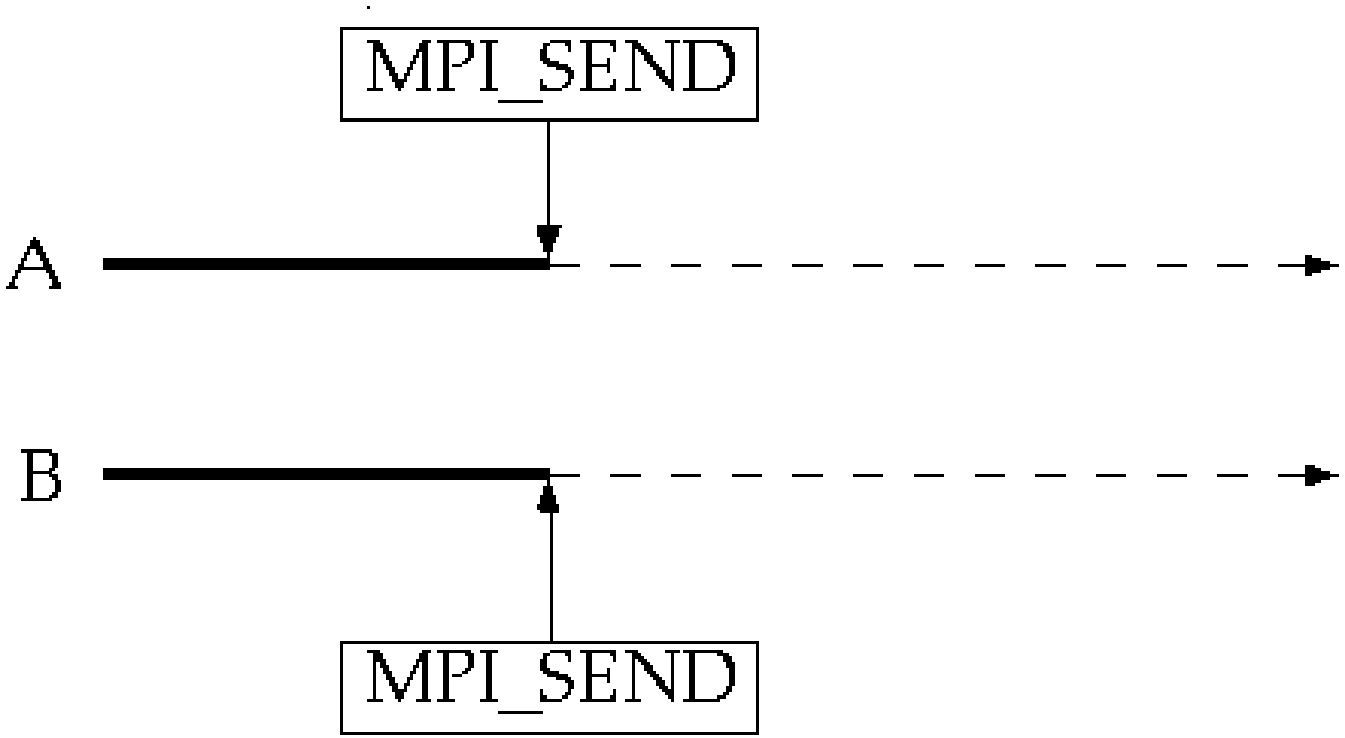
- Communications are blocking or nonblocking.
- Communication modes are selected by the sender.
- MPI defines four communication modes depending on synchronisation overhead and system buffer usage

synchronous	no system buffer, synchronisation overhead, portability,
ready	no system buffer, performant, **requires** that receives are ready for sends,
buffered	sender buffers, less synchronsiation overhead, additional system overhead
standard	best default compromise for non-blocking, buffering at receiver not garanteed.

What Is Covered in This Module?

- Deadlock
  - Causes
  - Avoidance
- Checking and acting on communications-"state"
  - status
  - wait, test and probe
- Parameters for special cases
  - Wildcards - MPI_ANY_SOURCE and MPI_ANY_TAG
  - Null Processes and Requests
- Programming recommendations

Basic Deadlock: Causes



- Deadlock most common with blocking communication.
- It occurs when synchronisation gets impossible:
  - blocking sends stall each other indefinitely
  - message size is greater than the threshold
  - insufficient system buffer space is available

Basic Deadlock: Principle Solutions

- Different ordering of calls between tasks (easiest)
- `MPI_Sendrecv` or `MPI_Sendrecv_replace` (elegant)
- Buffered communication mode (effectively decouples SEND from RECV)
- Non-blocking calls (best, but maybe more effort)
  - Post a non-blocking receive before any other communication.
  - Avoid buffer re-use between MPI_Irecv/ MPI_Isend and MPI_Wait.
  - Check state of MPI_Wait, MPI_Test, MPI_Probe.

MPI Calls: MPI_Sendrecv

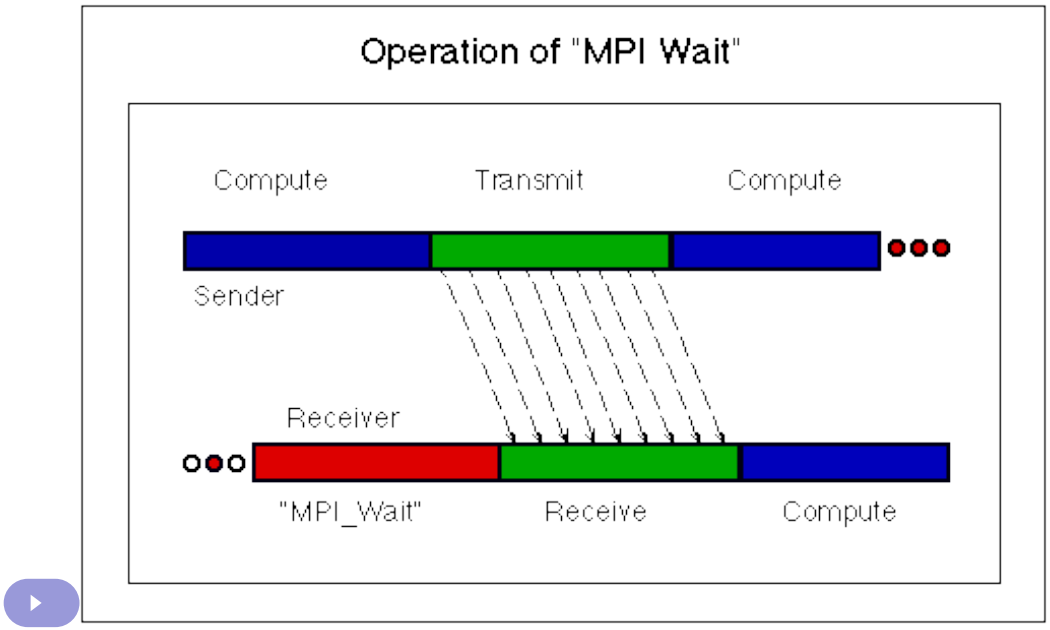
C

```
int MPI_Sendrecv (void *sendbuf,
                  int sendcount,
                  MPI_Datatype sendtype,
                  int dest,
                  int sendtag,
                  void *recvbuf,
                  int recvcount,
                  MPI_Datatype recvtype,
                  int source,
                  int recvtag,
                  MPI_Comm comm,
                  MPI_Status *status)
```

Informations About Transfers

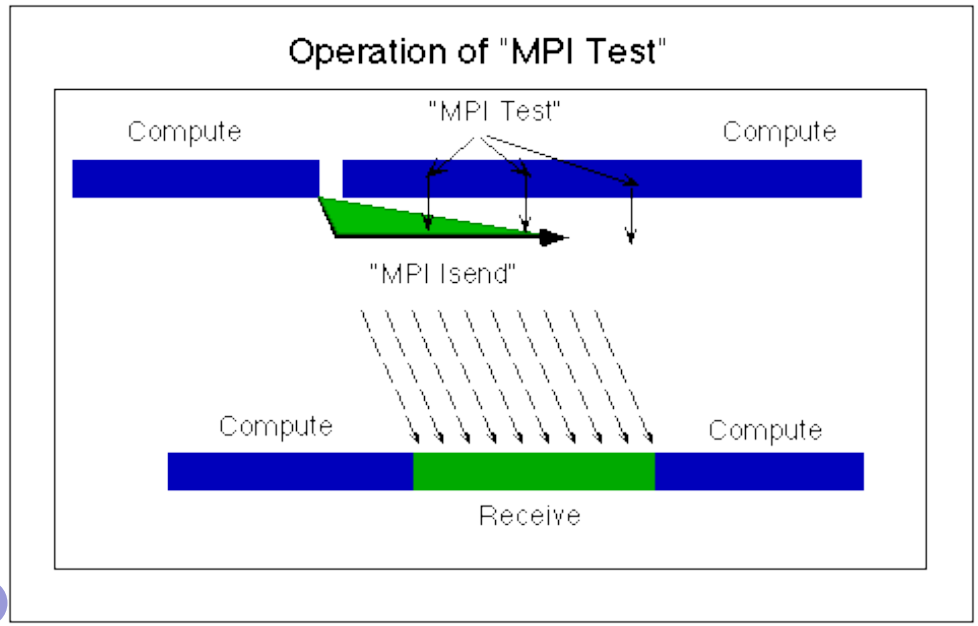
- `status` or MPI_Status returns/contains information on
  - source, tag and error.
- `MPI_Get_count` returns number of elements.
- Circumstances in which it makes sense to check the status:
  - blocking MPI_Recv or MPI_Wait, when `MPI_ANY_TAG` or `MPI_ANY_SOURCE` has been used
    - `MPI_ANY_TAG`, accept a message with any tag value
    - `MPI_ANY_SOURCE`, accept a message from any source
  - `MPI_Probe` or `MPI_Iprobe` to obtain information about incoming messages
    - `MPI_Probe` - Blocking test for a message
    - `MPI_Iprobe` - Nonblocking test for a message
  - `MPI_Test` to learn if the communication has completed

MPI_Wait



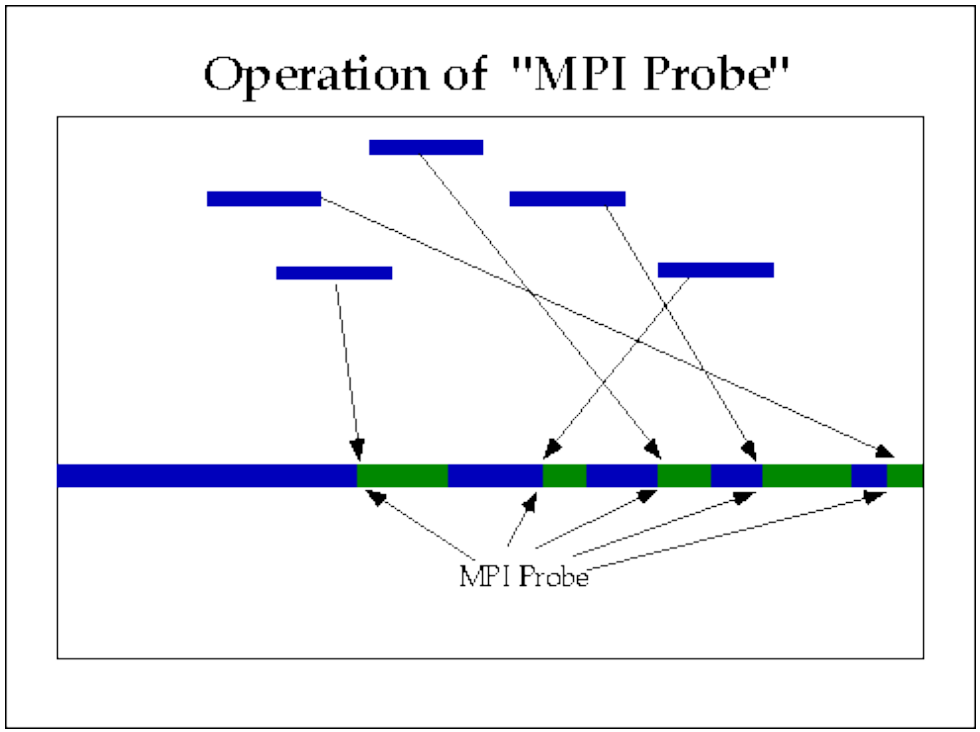
- Useful for both sender and receiver of *non-blocking* communications:
  - Receiving process blocks until message is received, under programmer control
  - Sending process blocks until send-operation completes, at which time the message-buffer is available for re-use

MPI_Test



- Used for both sender and receiver of a *non-blocking* communication.
  - Receiver checks to see if a specific sender has sent a message that is waiting to be delivered ... messages from all other senders are ignored.
  - Sender can find out if the message-buffer can be re-used ... have to wait until operation is complete before doing so.

MPI_Probe



- Receiver is notified when (i.e., this is a blocking call) messages from potentially any sender arrive and are ready to be processed.

Special Parameters

- Wildcards
  - `MPI_ANY_SOURCE` – the receiver is willing to accept messages from anyone.
  - `MPI_ANY_TAG` – the receiver is willing to accept any kind of message
- Null Processes and Requests
  - `MPI_PROC_NULL` and `MPI_REQUEST_NULL` act as valid parameters whose associated messages are ignored but do not generate errors.
  - Most useful for taking boundary-tests out of user code.

IBM MPI Implementation

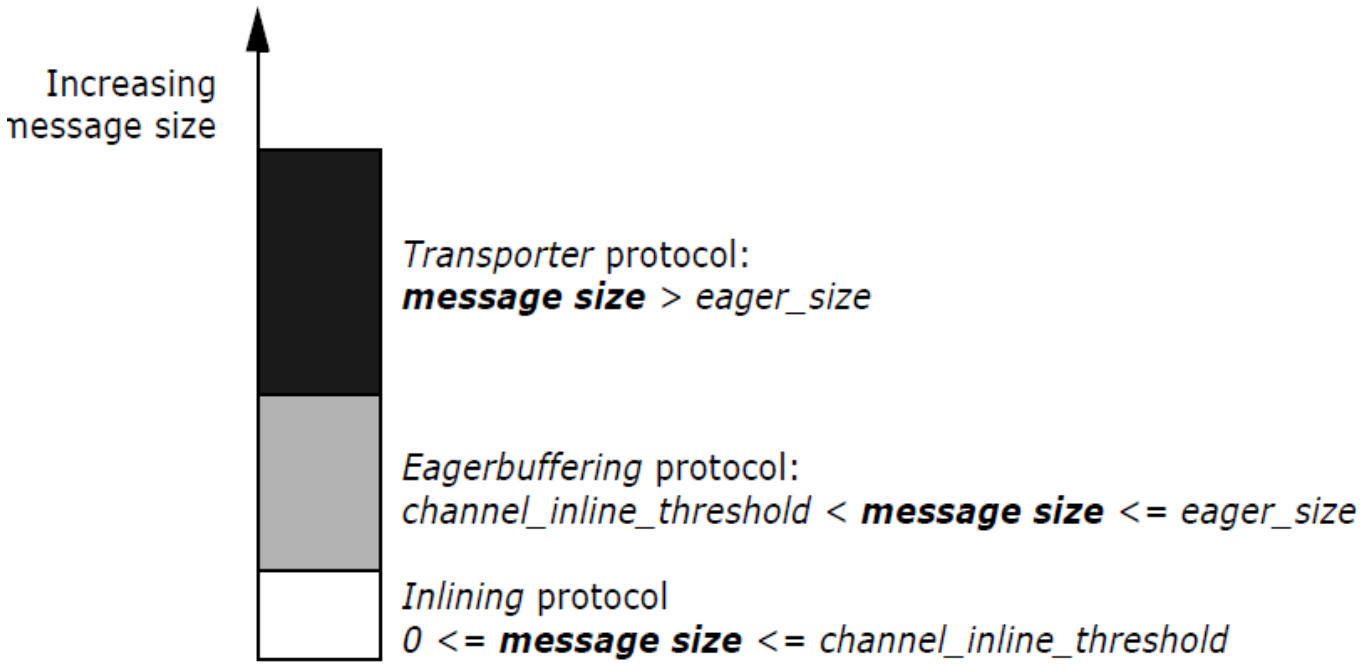
Implementing the four communication modes with two protocols:

1. "**rendezvous**" protocol used for synchronous mode and standard mode for message size > MP_EAGER_LIMIT.
2. "**eager**" protocol used for ready mode, buffered mode, and standard mode for message size <=MP_EAGER_LIMIT.
   - Ready mode: eager protocol and no buffer.
   - Standard mode for small messages: receive-side system buffer.
   - Buffered mode: send-side and receive-side system buffer

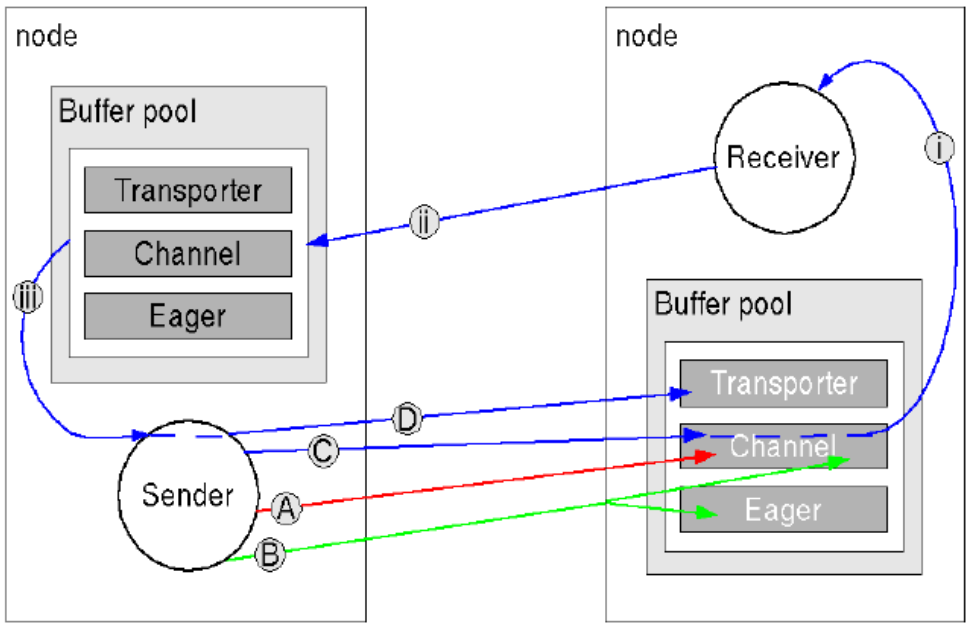
- Further optimizations for handling small messages.
- Environment variables to tweak message-passing behavior. I.e:
  - Message size threshold MP_EAGER_LIMIT,
  - Receive-side system buffer space MP_BUFFER_MEM.

Scali MPI Connect: ScaMPI

Four communication modes with three protocols, defined by two thresholds:



ScaMPI: Communication Protocols



A Inlining protocol

B Eagerbuffering protocol

C Transporter protocol

– Zerocopy protocol (similar to C, but by-passing the ringbuffer)

ScaMPI: Remarks

- Every node has one set of receiving buffers for every process it communicates with.
- The standard sizes of the buffers depend on the architecture and networks.
- The standard sizes assume an all-to-all communication pattern.
- The buffer sizes are completely configurable (but is usually not necessary).
- Consequences for programming:
  - Be careful when matching senders using `MPI_Probe`/`MPI_Recv`.
  - If communication and calculations do not overlap, non-blocking operations is usually performance ineffective.
  - Using buffered send usually degrades performance significantly in comparision with their unbuffered relatives.

Programming Recommendations

- Avoid deadlock by intelligent arrangement of sends/receives, or by posting non-blocking receives early.
- Use the appropriate "operation-status" call ("wait", "test", or "probe") to control the operation of non-blocking communications calls.
- Check the value of the "status" fields for problem reports.
- Intelligent use of wildcards can greatly simplify logic and code.

Summary

- Deadlock: Causes and avoidance
- Checking and acting on communications-"state"
- Parameters for special cases

A careful selection of the communication strategy is crucial for the efficiency of a parallel programme.

- What comes next?
  - Collective communications

MPI Status

C

```
typedef struct MPI_Status {
    int MPI_SOURCE;
    int MPI_TAG;
    int MPI_ERROR;
};
```

Fortran

```
integer status(MPI_STATUS_SIZE)
source = status(MPI_SOURCE)
tag = status(MPI_TAG)
error = status(MPI_ERROR)
```

MPI Wait and MPI Test

MPI_Wait suspends execution until an operation completes,
MPI_Test immediatly delivers true or false.

C

```
int MPI_Wait(MPI_Request *request,
    MPI_Status *status)
int MPI_Test(MPI_Request *request, int *flag,
    MPI_Status *status)
```

Fortran

```
MPI_WAIT(REQUEST, STATUS, IERROR)
MPI_TEST(REQUEST, FLAG, STATUS, IERROR)

LOGICAL FLAG
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR
```