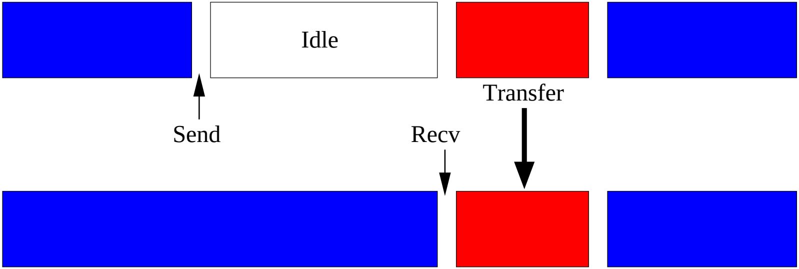
Sending modes in MPI

- We have claimed our MPI functions so far have been blocking
- That is they wait until the operation has been completed before exiting and allowing the program to continue



- Not exactly true. There are several modes for executing Sends and Recvs depending on buffering and synchronization wishes.

Sending modes in MPI

Synchronous MPI_Ssend Returns only when the matching recieve occurs.

Buffered MPI_Bsend Returns immediately whether or not the transfer has occured.

Ready MPI_Rsend Give an error if called before the matching receive occurs.

Standard MPI_Send Returns when you can safely use the original data.

All are matched by the same MPI_Recv call. No modes for Recv.

Synchronous Send

- MPI_Ssend(...)
- Full syncronization but potentially high overhead.
- Send may start whenever it wants.
- Recv must start before Send returns.
- Safest form of Sending. Maybe useful for porting code.

Buffered Send

- MPI_Bsend(...)
- Needs explicit buffer management by the user.
- MPI_Buffer_attach(void *buf, int size)
- Only one buffer allowed per MPI task at a time.
- Need to ensure buffer is big enough for all the data.
- MPI_Buffer_detach(void *buf, int) to detach the buffer.

Ready Send

- MPI_Rsend(...)
- Recieve must have started before Rsend is called.
- Useful on some specialized networks that can support handshakeless communications.
- Normally do not use this function!

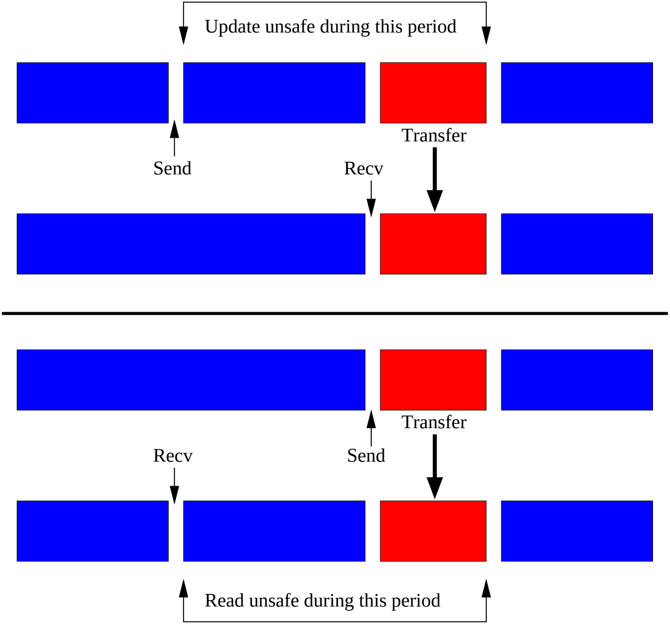
Blocking Message Passing

- Unless the MPI_Send(. . .) and MPI_Recv(. . .) occur around the same time one of the processes will end up idling
- As mentioned previously these functions can also end up deadlocking if the ordering of the functions on each process is wrong — two MPI_Send(. . .) calls posted with no MPI_Recv(. . .)
- MPI_Sendrecv(. . .) can solve some of these problems but idling will still occur
- Solution is to use non-blocking operations

Non Blocking Message Passing

- In non blocking MPI, the functions return immediately after being called
- The data may not yet have been transferred
- Process is free to continue computation that does not depend on the data involved in the transfer
- When the data is required, perform a check to see if the non blocking call has been completed
- If so then free to continue computation
- Otherwise wait until the transfer is complete

Non Blocking Message Passing



Non Blocking Message Passing

- MPI_Isend(data, count, type, dest, tag, comm, request);
- MPI_Irecv(data, count, type, src, tag, comm, request);
- New structure MPI_Request contains the information about the state of the transfer
- Use MPI_Test(request, flag, status) or MPI_Wait(request, status) to find out if the transfer is completed
- If feeling brave (foolhardy) use MPI_Request_free(request) to abandon the checking

Non Blocking Message Passing

- Using MPI_Isend and MPI_Wait directly one after the other is equivalent to just using MPI_Send
- Doesn't add much, if any, overhead
- Normally just place the MPI_Wait just before the next call the needs to update the data
- We can use MPI_Test to build an event driven programme. This is often used on the Recv'ing side

Non Blocking Message Passing

- Usual rules about src, dest and tag apply — can use MPI_ANY_SOURCE and MPI_ANY_TAG in MPI_Irecv(. . .)
- MPI_Isend(. . .) can be received by any form of MPI_Recv
- MPI_Irecv(. . .) can get data from any form of MPI_Send

Multiple messages

- Sometimes we want to set up a group of non-blocking communications
- Create arrays of request and status objects
- Three options for waiting on multiple communications
- MPI_Waitall(count, requests, statuses)
- MPI_Waitany(count, requests, index, statuses)
- MPI_Waitsome(count, requests, compcount, indices, statuses)
- Similar functions for Test

Probing

- MPI also provides functions for you to see if there is a communication heading your direction
- MPI_Probe(src, tag, comm, stat)
- Allows us to allocate appropriate sized recv buffers.
- More efficient than having a super-sized catch all buffer.
- A non-blocking version MPI_Iprobe(src, tag, comm, flag, stat)
- Mixed views on how useful Probe is.

Cancelling

- Sometimes a non-blocking call will never be matched.
- Good code should clean up after itself.
- Use MPI_Cancel(req) to start the process.
- Then have to complete the request using wait/test as usual
- Finally use MPI_Test_cancelled(stat, flag) to see if the cancel was successful.

Golden Rule of Non Blocking Comms

- After MPI_Isend(...) do not change the data in the message until MPI_Test(...) or MPI_Wait(...) have indicated that it is safe to do so
- After MPI_Irecv(...) do not use the data in the message until MPI_Test(...) or MPI_Wait(...) have indicated that it is safe to do so

Persistent Communication

- If the same Send/Recv pair is called repeatedly can set up a persistent communication to avoid repeated message setup and teardown
- Saves on creation and destruction of MPI objects in the library
- Often useful in halo exchanges in data decomposition
- Three step process
  1. Create request objects
  2. Send and receive the messages using MPI_Start
  3. Delete the objects

Persistent Communication

```c
MPI_Request send_req, recv_req;
MPI_Status stat;

MPI_Send_init(sendbuf, count, type, dest, tag, comm, &send_req);
MPI_Recv_init(recvbuf, count, type, src, tag, comm, &recv_req);

for(i=0;i<N;i++) {
    /* Halo exchange */
    MPI_Start(&recv_req);
    MPI_Start(&send_req);

    MPI_Wait(&send_req, stat);
    MPI_Wait(&recv_req, stat);

    /* Update values */
    do_stuff();
}

MPI_Request_free(send_req);
MPI_Request_free(recv_req);
```