MPI communications modes
From a post in stack overflow by ggulgulia, slightly revised by Luca Formaggia

Terminology:

- send-receive buffer. The buffer used to transfer data by a send-receive command. Is given as argument of the relative command.
- system buffer. A possible buffer used internally by the MPI system to store temporalily messages communicated between processes.

First one needs to know, **send has four modes** of communication : **Standard, Buffered, Synchronous** and **Ready** and each of these can be ***blocking*** and ***non-blocking***

Unlike in send, **receive has only one mode** and can be **blocking** or **non-blocking** .

Before proceeding further, one must also be clear that I explicitly mention which one is **MPI_Send\ Recv buffer** and which one is **system buffer** (which is a local buffer in each processor owned by the MPI Library used to move data around among ranks of a communication group)

BLOCKING COMMUNICATION

Blocking doesn't mean that the message was delivered to the receiver/destination. It simply means that the (send or receive) command returns only when the (send or receive) buffer is available for reuse. To reuse the buffer, it's sufficient that the send/receive buffer is copied by MPI to the system buffer (buffered communication) and then, say for e.g, MPI_Send can return.
MPI standard makes it very clear to decouple the message buffering from send and receive operations. A blocking send can complete as soon as the message was buffered, *even though no matching receive has been posted*. But in some cases message buffering can be expensive and hence direct copying from send buffer to receive buffer might be efficient. Hence, *MPI Standard provides four different send modes* to give the user some freedom in selecting the appropriate send mode for her application. Lets take a look at what happens in each mode of communication :

1. Standard Mode
In the **standard** mode, (the une of the usuale MPI_Send function) i*t is up to the MPI Library, whether or not to buffer the outgoing message.* In the case where the library decides to buffer the outgoing message, the send can complete even before the matching receive has been invoked. In the case where the library decides not to buffer (for performance reasons, or due to unavailability of buffer space), the send will not return until a matching receive has been posted and the data in send buffer has been moved to the receive buffer.
Thus MPI_Send in standard mode is non-local in the sense that send in standard mode can be started whether or not a matching receive has been posted and its successful completion may depend on the occurrence of a matching receive (due to the fact it is implementation dependent if the message will be buffered or not) .
The syntax for standard send is below :

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,  int dest, int tag, MPI_Comm comm)
```

However, since the used cannot know whether the send is buffered or not, it is safe to assume that is not buffered and that a MPI_Send() returns only when the matching MPI receive has been called.

2. Buffered Mode

Like in the standard mode, the send in buffered mode can be started irrespective of the fact that a matching receive has been posted and the send may complete before a matching receive has been posted. However the main difference arises out of the fact that if the send is started and no matching receive is posted the outgoing message **must** be buffered.

Syntax for buffer send :

```
int MPI_Bsend(const void *buf, int count, MPI_Datatype datatype,int dest, int tag, MPI_Comm comm)
```

3. Synchronous Mode

In synchronous send mode, send can be started whether or not a matching receive was posted. However **the send will complete successfully only if a matching receive was posted and the receiver has started to receive the message.** The completion of synchronous send not only indicates that the buffer in the send can be reused, but also the fact that receiving process has started to receive the data. If both send and receive are blocking then the communication does not complete at either end before the communicating processor rendezvous.

Syntax for synchronous send :

```
int MPI_Ssend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

4. Ready Mode

Unlike the previous three mode, a send in ready mode **can be started only if the matching receive has already been posted.** Completion of the send doesn't indicate anything about the matching receive and merely tells that the send buffer can be reused. A send that uses ready mode has the same semantics as standard mode or a synchronous mode with the additional information about a matching receive. A correct program with a ready mode of communication can be replaced with synchronous send or a standard send with no effect to the outcome apart from performance difference.

Syntax for ready send :

```
int MPI_Rsend(const void *buf, int count, MPI_Datatype datatype, int dest,  int tag, MPI_Comm comm)
```

A Summary:

Having gone through all the 4 blocking-send, they might seem in principal different but depending on implementation the semantics of one mode may be similar to another.

For example MPI_Send in general is a blocking mode but depending on implementation, if the message size is not too big, MPI_Send will copy the outgoing message from send buffer to system buffer ('which mostly is the case in modern system) and return immediately. This means that an incorrect syncronization between Send and Recv may cause deadlock in certain situations (buffering activted) and not in others. In general is safer to assume that a MPI_Send is not-

buffered. If you want to be sure that your program does not fall into a dedlock you should use **MPI_Ssend,** which has the true semantics of a blocking communication. If it works, you can then change it back to MPI_Send(), which may be more efficient.

Of course, you may also decide to use non-blocking communications, see later.

What about receive:
Fortunately MPI decided to keep things easer for the users in terms of receive and **there is only one receive in Blocking communication : MPI_Recv**, and can be used with any of the four send modes described above. For MPI_Recv, **blocking means** that receive returns only after it contains the data in its buffer. This implies that receive can complete only after a matching send has started but doesn't imply whether or not it can complete before the matching send completes.
What happens during such blocking calls is that the computations are halted until the blocked buffer is freed. This usually leads to wastage of computational resources as Send/Recv is usually copying data from one memory location to another memory location, while the registers in cpu remain idle.

NON-BLOCKING COMMUNICATION :

For Non-Blocking Communication, the application creates a **request for communication** for send and / or receive and gets back a handle and then terminates. That's all that is needed to guarantee that the process is executed. I.e the MPI library is notified that the operation has to be executed.
For the sender side, this allows overlapping computation with communication.

For the receiver side, this allows overlapping a part of the communication overhead , i.e copying the message directly into the address space of the receiving side in the application.

You should use MPI_Irecv() and MPI_Isend() for non-blocking receive/send. In general commands starting with an I are non-blocking (I stands for *immediate return*)