**Synchronous vs. Asynchronous (global perspective):**

*Synchronous operation means that communication procedure between sender and receiver processes can not be completed before both processes have started the communication operations and in particular receiver process has acknowledged a response. Namely, if synchronous operation is completed, it means that the second process have initiated a response to calling process and all resources involved in the operation can be reused by other processes.*

*Asynchronous operation on the contrary means that calling process does not have to wait for any response from the process it calls and can initiate its communication procedure without any coordination.*

**Blocking vs. Nonblocking (local perspective):**

*Blocking operation ensures that when control is returned to the calling process, all the system resources that were involved in the operation can be safely reused by other processes for other operations. For instance, buffers.*

*Non-blocking operation on the other hand allows the control to be returned to the calling process before all the side-effects related to the operation are completed and resources, occupied by the call may be reused. To be precise, non-blocking call only triggers the operation to start and does not guarantee readiness of the above-mentioned conditions on return. However, it may notify sender upon completion of communication.*

*(ref: http://adrianofesta.altervista.org/mpi\_slides.pdf)*

**Buffered vs. Unbuffered:**

*Buffered communication requires that user must explicitly take care of allocating big enough buffer required to transfer the data and also account for MPI overhead. In particular, user is required to allocate the buffer equal to the size of the data + overhead, attach it to the MPI and after completion take care of detaching etc.*

*? Unbuffered communication on the other hand does not require explicit allocation of the buffer and ensures that it happens automatically under the hood, meaning that user does not have to explicitly worry about the buffer, required to for instance transfer the data during communication.*

**One-sided vs. two-sided:**

*One-sided communication assumes that whenever calling process initiates the communication to some receiver process, no explicit notification acknowledgement from the receiver process is ever required. In other words, only sender process is explicitly involved in this type of communication procedure.*

*Two-sided communication always assumes that two parties involved into the communication process: sender and receiver. Receiver process has to somehow always acknowledge the caller that it has initiated the communication operation on its side.*

**What is the significance of the possibility of both properties of a pair applying to the same routine?**

*In all cases, both properties may be important, depending on the context and problem at hand.*

*For instance, sometimes it is important to initiate strictly synchronous way of cooperation and ensure that communication between processes have completed. For example, it could be the case when results of the computation depends on whether the receiver process have actually got the message, otherwise, some sort of invalidation of results (like data race in shared memory model) may occur and spoil the whole computational procedure. On the other hand, whenever there a lot of independent jobs that do not require to be completed in order, the probability of deadlock is low (or the deadlock in the particular process will not be fatal for the system) and system recourses are at scarce, it may make more sense to use the type of communication that do not require explicit answer from the receiver party and thus, save some systems resources.*

*Similarly, when the data size is large, it makes more sense to use buffered communication and ensure that all the required data transfers during communication between processes will occur without failures. On the other hand, if data size is small and system will surely be able to allocate required buffers itself, there is no point to create extra worry for programmer.*

Problem 2:

In code development it is advantageous to have the properties that simplify the process of development, testing and debugging of the program. Therefore asynchronous, non-blocking and two-sided communication routines are disadvantageous since their obvious conses is the fact that it is difficult to implement and debug code with the aforementioned properties. However, having something that does not causes deadlocks in testing might be more preferable than having something that deadlocks, since it will likely enable the execution of larger number of tests. Buffered is also more preferable since it is easier to test and therefore debug, but at the same time it creates need to explicitly care for the memory which is always not desirable as requires more code and effort.

In production, it is on one hand more desirable to have something that is more consistent and resilient. Therefore synchronous, blocking and two-sided are better choices as they are more prone to general safety than their counterparts. On the other hand, the fact that they can potentially cause deadlocks makes them less preferable as deadlock in production means that system is dead is someone loses a lot of money. Buffered vs unbuffered can be viewed from that perspective as well: buffered requires explicit work with raw memory, which naturally means that errors are more likely and in production this is highly undesirable. On the other hand, if by touching memory explicitly one can guarantee that system will never run out of memory on request and thus not break, it might be a potential solution.

Code optimization?