MPI Point-to-Point Communications

Young Won Lim
09/01/2012

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Communicator

Communicators and Groups defines
collection of processes that may communicate with each other.

Need to specify a communicator as an argument.

MPI_COMM_WORLD - predefined communicator
that includes all of your MPI processes.

Within a communicator, every process has its own unique, integer identifier,
called rank or “task ID”.

Used to specify the source and destination.
Also can be used in conditional statements.

Communication Parameters

Point to point communication

Simple latency / bandwidth model

Not good for ping-pong benchmark data

MPI message transfer is complex

Message Envelope

supplementary information such as

length, sender, tag, etc

Eager Protocol

Rendezvous Protocol

Communication Modes

Blocking

- **Standard**
- **Buffered**
- **Synchronous**
- **Ready**

Immediate

- **Standard**
- **Buffered**
- **Synchronous**
- **Ready**

Blocking

does not return until
the message data and envelope have been safely stored away
so that the sender is free to access and overwrite the send buffer.

The message might be copied directly into the matching receive buffer,
or it might be copied into a temporary system buffer.

Message buffering decouples the send and receive operations.

A blocking send can complete
as soon as the message was buffered,
even if no matching receive has been executed by the receiver.

On the other hand, message buffering can be expensive,
as it entails additional memory-to-memory copying,
and it requires the allocation of memory for buffering.

MPI offers the choice of several communication modes that allow one to control the choice of the communication protocol.

Standard Communication Mode

It is up to MPI to decide whether outgoing messages will be buffered.

1) MPI may buffer outgoing messages.
→ the send call may complete before a matching receive is invoked.

2) Buffer space may be unavailable, or
MPI may choose not to buffer outgoing messages, for performance reasons.
→ the send call will not complete until a matching receive has been posted, and the data has been moved to the receiver.

Thus, a send in standard mode can be started whether or not a matching receive has been posted.
It may complete before a matching receive is posted.

The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Buffered Communication Mode

A buffered mode send operation can be started
whether or not a matching receive has been posted.

It may complete before a matching receive is posted.

However, unlike the standard send, this operation is **local**,
and its completion does not depend on the occurrence of a matching receive.

Thus, if a send is executed and no matching receive is posted,
then MPI must buffer the outgoing message,
so as to allow the send call to complete.

An error will occur if there is insufficient buffer space.
The amount of available buffer space is controlled by the user.

Buffer allocation by the user may be required
for the buffered mode to be effective.

Synchronous Communication Mode

A send that uses the synchronous mode can be started
whether or not a matching receive was posted.

However, the send will complete successfully
only if a matching receive is posted,
and the receive operation has started
to receive the message sent by the synchronous send.

Thus, the completion of a synchronous send
not only indicates that the send buffer can be reused,
but also indicates that the receiver has reached a certain point in its execution,
namely that it has started executing the matching receive.

If both sends and receives are blocking operations
then the use of the synchronous mode provides
synchronous communication semantics:
a communication does not complete at either end
before both processes rendezvous at the communication.

A send executed in this mode is non-local.

Ready Communication Mode

A send that uses the ready communication mode may be started only if the matching receive is already posted.
Otherwise, the operation is erroneous and its outcome is undefined.

On some systems, this allows the removal of a hand-shake operation that is otherwise required and results in improved performance.

The completion of the send operation does not depend on the status of a matching receive, and merely indicates that the send buffer can be reused.

A send operation that uses the ready mode has the same semantics as a standard send operation, or a synchronous send operation;

it is merely that the sender provides additional information to the system (namely that a matching receive is already posted), that can save some overhead.

In a correct program, therefore, a ready send could be replaced by a standard send with no effect on the behavior of the program other than performance.

NonBlocking Communication (1)

overlapping communication and computation
light-weight threads vs nonblocking communication.

A nonblocking send (receive) start call initiates the send (receive) operation,
but does not complete it.

The send (receive) start call will return
before the message was copied out of (into) the send (receiver) buffer.

A separate send (receive) complete call
is needed to complete the communication, i.e., to verify that the data
has been copied out of the send buffer (received into the receive buffer) .

With suitable hardware, the transfer of data out of the sender (receiver) memory
may proceed concurrently with computations done at the sender (receiver) after
the send (receive) was initiated and before it completed.

The use of nonblocking receives may also avoid
system buffering and memory-to-memory copying,
as information is provided early on the location of the receive buffer.

NonBlocking Communication (2)

Nonblocking send start calls can use the same four modes as blocking sends: standard, buffered, synchronous and ready.

Sends of all modes, ready excepted, can be started whether or not a matching receive has been posted ;
a nonblocking ready send can be started only if a matching receive is posted.

In all cases, the send start call is local:
it returns immediately, irrespective of the status of other processes.

If the call causes some system resource to be exhausted, then it will fail and return an error code. Quality implementations of MPI should ensure that this happens only in ``pathological'' cases. That is, an MPI implementation should be able to support a large number of pending nonblocking operations.

The send-complete call returns when data has been copied out of the send buffer. It may carry additional meaning, depending on the send mode.

NonBlocking Communication (3)

If the send mode is **synchronous**, then the send can complete only if a matching receive has started. That is, a receive has been posted, and has been matched with the send. In this case, the send-complete call is non-local. Note that a synchronous, nonblocking send may complete, if matched by a nonblocking receive, before the receive complete call occurs. (It can complete as soon as the sender ``knows'' the transfer will complete, but before the receiver ``knows'' the transfer will complete.)

If the send mode is **buffered** then the message must be buffered if there is no pending receive. In this case, the send-complete call is local, and must succeed irrespective of the status of a matching receive.

If the send mode is **standard** then the send-complete call may return before a matching receive occurred, if the message is buffered. On the other hand, the send-complete may not complete until a matching receive occurred, and the message was copied into the receive buffer.

Nonblocking sends can be matched with blocking receives, and vice-versa.

Send Modes (1)

MPI_Send
MPI_Send will not return until you can use the send buffer.
It may or may not block
(it is allowed to buffer, either on the sender or receiver side,
or to wait for the matching receive).

MPI_Bsend
May buffer;
returns immediately and you can use the send buffer.
A late add-on to the MPI specification.
Should be used only when absolutely necessary.

MPI_Ssend
will not return until matching receive posted

MPI_Rsend
May be used ONLY if matching receive already posted.
User responsible for writing a correct program.

Send Modes (2)

MPI_Isend

Nonblocking send. But not necessarily asynchronous.
You can NOT reuse the send buffer until either a successful, wait/test or you KNOW that the message has been received (see MPI_Request_free).
Note also that while the I refers to immediate, there is no performance requirement on MPI_Isend. An immediate send must return to the user without requiring a matching receive at the destination. An implementation is free to send the data to the destination before returning, as long as the send call does not block waiting for a matching receive. Different strategies of when to send the data offer different performance advantages and disadvantages that will depend on the application.

MPI_Ibsend

buffered nonblocking

MPI_Issend

Synchronous nonblocking. Note that a Wait/Test will complete only when the matching receive is posted.

MPI_Irsend

As with MPI_Rsend, but nonblocking.

Message Aggregation

References

[1] http://en.wikipedia.org/
[2] http://static.msi.umn.edu/tutorial/scicomp/general/MPI/mpi_coll_new.html
[3] https://computing.llnl.gov/tutorials/mpi/
[4] https://computing.llnl.gov/tutorials/mpi/
[5] Hager & Wellein, Introduction to High Performance Computing for Scientists and Engineers
[6] http://www.mpi-forum.org/docs/mpi-11-html