

MPI Point-to-Point Communications

-
-

Copyright (c) 2012 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

Immediate: there is *no performance requirement* on MPI_Isend.

An immediate send **must return 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.

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**.

Blocking Send Modes

Blocking

MPI_Send

Standard

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

Buffered

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

Synchronous

will **not return until** matching receive posted

MPI_Rsend

Ready

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

Immediate Send Modes

Immediate

MPI_Isend

Nonblocking send. But **not necessarily asynchronous**.

You can **NOT use** the send buffer **until** either a **successful, wait/test** or you **KNOW** that the message has been **received** (see MPI_Request_free).

Standard

MPI_IbSEND

buffered nonblocking

Buffered

MPI_ISSend

Synchronous nonblocking.

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

Synchronous

MPI_IrSEND

As with MPI_Rsend, but **nonblocking**.

Ready

Blocking

does **not return until the send buffer can be re-used.**

After the data in the send buffer is

- a) actually transferred to the **receive buffer**
- b) 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.

Message buffering can be **expensive**

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.
will return before the buffer can be safely re-used.

A separate send (receive) complete call
is needed to complete the communication,
to verify that the data has been transferred

With a special hardware

The use of nonblocking receives may also avoid system buffering and memory-to-memory copying,

NonBlocking Communication (2)

Standard, Buffered, Synchronous Non-blocking Send
can be **started whether or not** a matching receive has been **posted** ;

Ready Non-blocking Send
can be **started only if** a matching receive is **posted**.

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

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

NonBlocking Communication (3)

Standard Non-blocking Send

If message buffer is used

the **send-complete** call may return before a matching receive occurred

Otherwise

the **send-complete** call may not return until a matching receive occurred, and the message was copied into the receive buffer.

Buffered Non-blocking Send

the message *must be buffered* if there is no pending receive.

the **send-complete** call is **local**,

and must succeed regardless of the status of a matching receive.

Synchronous Non-blocking Send

the send can complete only if a matching receive has started.

→ a receive has been **posted**, and has been **matched** with the send.

the **send-complete** call is **non-local**.

The **send-complete** call returns, 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.)

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>