MPI - Point to Point Communication

Blocking-Nonblocking Communication

Özden AKINCI
UHeM

14 Jun 2012, @ ITU SDKM
Outline

• Blocking Communication
  • Blocking Send
    • Message Envelope
  • Blocking Receive
  • Synchronous Send
  • Ready Send
  • Buffered Send
  • Buffered Blocking Send Communication
  • Standard Send
• Deadlock Activities
Outline

- Non-Blocking Communication
- Non-Blocking Post-Send
- Non-Blocking Post-Receive
- Request Objects
- Posting Send Operation
- Posting Receive Operation
- Completion: Waiting and Testing
- Querying for a Message's Arrival
- Point-to-point Bandwidth Comparisons
VPN Connection

VPN Group Name: uybhm-calistay
Group Pass: 1j12iHMz
Host: 160.75.120.250
Username: duXYZ
Password: password of duXYZ

XYZ=001-150

6/13/12
Point to Point Communication

• The communication mode is selected with the send routine.
• Four blocking send routines,
• Four non-blocking send routines.
• The receive routine does not specify communication mode.
• The receive routine is simply blocking or non-blocking.
## Point to Point Communication

<table>
<thead>
<tr>
<th>Communication Mode</th>
<th>Blocking Routines</th>
<th>Non-Blocking Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>MPI_Ssend</td>
<td>MPI_Issend</td>
</tr>
<tr>
<td>Ready</td>
<td>MPI_Rsend</td>
<td>MPI_Irsend</td>
</tr>
<tr>
<td>Buffered</td>
<td>MPI_Bsend</td>
<td>MPI_Ibsend</td>
</tr>
<tr>
<td>Standard</td>
<td>MPI_Send</td>
<td>MPI_Isend</td>
</tr>
<tr>
<td></td>
<td><strong>MPI_Recv</strong></td>
<td><strong>MPI_Irecv</strong></td>
</tr>
<tr>
<td></td>
<td>MPI_Sendrecv</td>
<td>MPI_Sendrecv_replace</td>
</tr>
</tbody>
</table>

The receive routine is simply blocking (MPI_Recv) or non-blocking (MPI_Irecv).
Blocking Communication

Blocking Send

• **MPI_Send** performs a standard-mode, blocking send.

• The send buffer specied by **MPI_Send** consists of count successive entries of the type indicated by **datatype**, starting with the entry at address **buf**.

• Note that we specify the message length in terms of number of entries, not number of bytes.
Blocking Communication

Message Envelope

• Messages carry
  – information + data

• Information = message envelope

• Information fields:
  – **Source**,  
  – **Destination**,  
  – **Tag (>0, integer)**  
  – **Communicator**.
Blocking Communication

Blocking Receive

MPI_Recv performs a standard-mode, blocking receive.

• The arguments to MPI_Recv are described as
  
  – **Receive Buffer:**
    
    • The length of the received message must be less than or equal to the length of the receive buffer.
  
  – **Message Selection** (source, tag, comm)
    
    • MPI_ANY_SOURCE,
    • MPI_ANY_COMM
  
  – **Return Status** (MPI_Status)
    
    • MPI_SOURCE,
    • MPI_TAG,
    • MPI_ERROR
Blocking Communication

Synchronous Send

1- The **sending** task:
   Sends to the receiving task a "ready to send" message.

2- The **receiver** task:
   When executing the receive call, it sends a "ready to receive" message back to sending task.

3- The data are then transferred.

4- When a **synchronous mode send operation** is completed, the **sending process** may assume “the destination process has begun receiving the message”.

6/13/12
Blocking Communication

Synchronous Send

1. "Ready to send"
2. "Ready to receive"
3. Data
Blocking Communication

Ready Send

• It **requires** that the "**ready to receive**" notification has arrived, indicating that the receiving task has posted the receive.

• If the "ready to receive" **message hasn't arrived**, the **ready mode send** will incur an **error**.

• This mode should not be used unless the user is certain that the corresponding receive has been posted.

• **Overhead**: Ready mode aims to minimize system overhead and synchronization overhead incurred by the sending task.
Blocking Communication

Ready Send

1 “Ready to receive”
2 Data
Blocking Communication

Buffered Send

- The blocking buffered send `MPI_Bsend` copies the data from the message buffer to a user-supplied buffer, and then returns.

- The sending task can then proceed with calculations that modify the original message buffer, knowing that these modifications will not be reflected in the data actually sent.

- The data will be copied from the user-supplied buffer over the network once the "ready to receive" notification has arrived.
Blocking Communication

Buffered Send

1. Copies the data to the system buffer
2. “Ready to receive”
3. Data is transferred from system buffer.
Buffered Blocking Send Communication

**MPI_Buffer_attach**

- Provides to MPI a buffer in the user memory to be used for buffering outgoing messages.
- For using buffered send routine, there must be a user defined buffer.
- **User defined buffer must be attached with MPI_Buffer_attach.**

```c
int MPI_Buffer_attach(void *buf,
                      int size)
```

- Initial buffer address
- Buffer size, in bytes
Buffered Blocking Send Communication

MPI_Buffer_detach

- Detach the buffer currently associated with MPI. The call returns the address and the size of the detached buffer.
- This operation will block until all messages currently in the buffer have been transmitted.

```c
int MPI_Buffer_detach(void *buf, int *size)
```

- **Initial buffer address**
- **Buffer size, in bytes**
Blocking Communication

Buffered Send

- **Overheads:**
  - Buffered mode incurs extra system overhead, because of the additional copy from the message buffer to the user-supplied buffer.
  - Synchronization overhead is eliminated on the sending task -- the timing of the receive is now irrelevant to the sender.
Blocking Communication

Standard Send

• There are two scenarios, depending on whether the message size is greater or smaller than a threshold value, called the cutoff limit.

• **Scenario 1:** Message size less than or equal to cutoff limit.
  – The blocking standard send `MPI_Send` copies the message over the network into a system buffer on the receiving node.
  – The standard send then returns, and the sending task can continue computation.
  – There is one system buffer per task that will hold multiple messages.
  – The message will be copied from the system buffer to the receiving task when the receive call is executed.
Blocking Communication

Standard Send

• **Scenario 1:** Message size less than or equal to cutoff limit.

Message will be copied from the system buffer to the receiving task when the receive call is executed.
Standard Send

- **Scenario 2**: Message size greater than cutoff limit.
  - The behavior of the blocking standard send MPI_Send is essentially the same as for synchronous mode.

![Diagram of blocking communication]

Data transmission completed.

6/13/12
Activity: Deadlock (01deadlock)

- Deadlock occurs when 2 (or more) processes are blocked and each is waiting for the other to make progress.

- Neither process makes progress because each depends on the other to make progress first.

- The program shown below is an example -- it fails to run to completion because processes 0 and 1 deadlock.
Activity: Deadlock (01deadlock)

```c
if( myrank == 0 ) {
    /* Receive, then send a message */
    MPI_Recv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
    MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
}
else if( myrank == 1 ) {
    /* Receive, then send a message */
    MPI_Recv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
    MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
}
```
Activity: Deadlock (02safeExchange)

• The following program shown is similar to the program in the preceding section.

• Its communication is better organized and the program does not deadlock.

• Once again, process 0 attempts to exchange messages with process 1.

• Process 0 receives, then sends; process 1 sends, then receives. The protocol is safe.
if( myrank == 0 ) {
    /* Send, then receive a message */
    MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
    MPI_Recv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
}
else if( myrank == 1 ) {
    /* Receive, then send a message */
    MPI_Recv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
    MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
}
Activity: Avoiding Deadlock

• Barring system failures, this program always runs to completion.

• Note that increasing array dimensions and message sizes have no effect on the safety of the protocol.

• The program still runs to completion.

• This is a useful property for application programs -- when the problem size is increased, the program still runs to completion.
Activity: Avoiding Deadlock  
(Sometimes but Not Always)  
(03dependBuffer)

- The following program is similar to preceding examples.
- Again, process 0 attempts to exchange messages with process 1.
- This time, both processes send first, then receive.
- Success depends on the availability of buffering in MPI.
- There must be enough MPI internal buffer available to hold at least one of the messages in its entirety.
Activity: Avoiding Deadlock
(Sometimes but Not Always)
(03dependBuffer)

if( myrank == 0 ) {
    /* Send a message, then receive one */
    MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
    MPI_Recv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
}
else if( myrank == 1 ) {
    /* Send a message, then receive one */
    MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
    MPI_Recv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
}
Activity: Avoiding Deadlock
(Sometimes but Not Always)
(03dependBuffer)

• After compile the code (03dependBuffer) run as following:

  mpirun -np 2 ./dependBuffer.x -c 10000
  mpirun -np 2 ./dependBuffer.x -c 100000

• Did you get the deadlock?
• Try

  mpirun -np 2 ./dependBuffer.x -c 32768
  mpirun -np 2 ./dependBuffer.x -c 32769

• For Intel MPI I_MPI_EAGER_THRESHOLD= 262144 bytes

6/13/12
Activity: Avoiding Deadlock
(Sometimes but Not Always)

• Under most MPI implementations, the program shown will run to completion.

• However, if the message sizes are increased, sooner or later the program will deadlock.

• This behavior is sometimes seen in computational codes -- a code will run to completion when given a small problem, but deadlock when given a large problem.

• In general, depending on MPI internal buffer to avoid deadlock makes a program less portable and less scalable.
Non-Blocking Communication

• The non-blocking interface to send and receive requires two calls per communication operation:
  – One call to initiate the operation,
  – Second call to complete it.
• Initiating a send operation is called “posting a send”.
• Initiating a receive operation is called “posting a receive”.
Non-Blocking Communication

• A send or receive operation has been posted, MPI provides two distinct ways of completing it.

1. A process can **test** to see if the operation has completed, without blocking on the completion.

2. Alternately, a process can **wait** for the operation to complete.
Non-Blocking Communication

Non-Blocking Post-Send

• A non-blocking post-send initiates a send operation, but does not complete it.

• The post-send will return before the message is copied out of the send buffer.

• A separate complete-send call is needed to complete the communication, that is, to verify that the data has been copied out of the send buffer.
Non-Blocking Communication

Non-Blocking Post-Send

- With suitable hardware, the transfer of data out of the sender memory may proceed concurrently with computations done at the sender after the send was initiated and before it completed.
Non-Blocking Communication

Non-Blocking Post-Receive

- Similarly, a non-blocking post-receive initiates a receive operation, but does not complete it.

- The call will return before a message is stored into the receive buffer.

- A separate complete-receive is needed to complete the receive operation and verify that the data has been received into the receive buffer.
Non-Blocking Communication

Request Objects

• Request objects are used to identify communication operations and link the posting operation with the completion operation.

• The request object is used to identify various properties of a communication operation:
  – The communication buffer that is associated with it,
  – Store information about the status of the pending communication operation.

• MPI_REQUEST_NULL is used to indicate an invalid request handle.
Non-Blocking Communication

Posting Send Operation

• A nonblocking post-send call indicates that the system may start copying data out of the send buffer.

• The sender must not access any part of the send buffer (neither for loads nor for stores) after a nonblocking send operation is posted, until the complete-send returns.
Non-Blocking Communication

Posting Send Operation

• A process calls the routine MPI_Isend to post (initiate) a send without blocking on completion of the send operation.
• MPI_Isend needs additional output argument, a request handle.
• The request handle identifies the send operation that was posted.
• The request handle can be used to check the status of the posted send or to wait for its completion.
Non-Blocking Communication

Posting Send Operation

```c
int MPI_Isend(void *buf,
               int count,
               MPI_Datatype datatype,
               int dest,
               int tag,
               MPI_Comm comm,
               MPI_Request *request);
```
Non-Blocking Communication

Posting Send Operation

• When this routine returns, a send has been posted (but not yet completed).

• Another call to MPI is required to complete the send operation posted by this routine.

• None of the arguments passed to MPI_Isend should be read or written until the send operation is completed.
Non-Blocking Communication

Posting Receive Operation

• A process calls the routine `MPI_Irecv` to post (initiate) a receive without blocking on its completion.
• The status argument of `MPI_Irecv` is replaced by a request handle; both are output arguments.
• The request handle identifies the receive operation that was posted and can be used to check the status of the posted receive or to wait for its completion.
• None of the arguments passed to `MPI_Irecv` should be read or written until the receive operation it invokes is completed.
Non-Blocking Communication

Posting Receive Operation

```c
int MPI_Irecv(void *buf,
              int count,
              MPI_Datatype datatype,
              int source,
              int tag,
              MPI_Comm comm,
              MPI_Request *request);
```
Non-Blocking Communication

Posting Receive Operation

• The sending and receiving processes must agree on the datatype; if they disagree, it is an error.
• When this routine returns, the receive has been posted (initiated) but not yet completed.
• Another call to MPI is required to complete the receive operation posted by this routine.
• None of the arguments passed to MPI_Irecv should be read or written until the receive operation is completed.
Non-Blocking Communication

Completion: Waiting and Testing

- Posted sends and receives must be completed.
- If a send or receive is posted by a non-blocking routine, then its completion status can be checked by calling one of a family of completion routines.
- The completion of a send (receive) indicates that the sender (receiver) is now free to access the send (receive) buffer.
Non-Blocking Communication

Completion: Waiting and Testing

- MPI provides both blocking and nonblocking completion routines.
- The blocking routines are `MPI_Wait` and its variants.
- The nonblocking routines are `MPI_Test` and its variants.
Non-Blocking Communication

Completion: Waiting

• A process that has posted a send or receive by calling a non-blocking routine can subsequently wait for the posted operation to complete by calling `MPI_Wait`.

```c
int MPI_Wait( MPI_Request *request,
              MPI_Status *status );
```
Non-Blocking Communication

Completion: Waiting

• The request argument is expected to identify a previously posted send or receive.

• MPI_Wait returns when the send or receive identified by the request argument is complete.
Non-Blocking Communication

Completion: Testing

• A process that has posted a send or receive by calling a non-blocking routine can subsequently wait for the posted operation to complete by calling MPI_Test.

• MPI_Test returns immediately.

```
int MPI_Test( MPI_Request *request,
              int *flag,
              MPI_Status *status );
```

true if the operation completed
Querying for a Message's Arrival

MPI_Probe/MPI_Iprobe

- The **MPI_Probe** and **MPI_Iprobe** operations allow checking of incoming messages, without actual receipt of them.

- The user can then decide how to receive them, based on the information returned by the probe in the status variable.

- For example, the user may allocate memory for the receive buffer, according to the length of the probed message.
Querying for a Message's Arrival

**MPI_Probe**

```c
int MPI_Probe(int source,
              int tag,
              MPI_Comm comm,
              MPI_Status *status)
```
MPI_Iprobe

int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

Returns flag=true if there is a message that can be received and that matches the pattern specified by the arguments source, tag, and comm.
Non-Blocking Communication

Non Blocking Send/Receive Example

```c
if( myrank == 0 ) {
    MPI_Irecv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &request );
    MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
    MPI_Wait( &request, &status );
}
else if( myrank == 1 ) {
    MPI_Irecv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &request );
    MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
    MPI_Wait( &request, &status );
}
```
Non-Blocking Communication

Non-Blocking Point to Point

• The sending task posts the non-blocking standard send when the message buffer contents are ready to be transmitted.

• It returns immediately without waiting for the copy to the remote system buffer to complete.

• The programmer does not know at this point whether the data to be sent have been copied out of the send buffer, or whether the data to be received have arrived.
Non-Blocking Communication

Non-Blocking Point to Point

• The receiving task calls a non-blocking receive as soon as a message buffer is available to hold the message.

• The non-blocking receive returns without waiting for the message to arrive.
Point-to-point Bandwidth Comparisons

Point to Point Comparisons: Small Messages

- Bandwidth (MB/sec) vs Message Size (Bytes)
- Various lines represent different send and recv operations.
Point-to-point Bandwidth Comparisons

Point to Point Comparisons: Medium Messages

Bandwidth (MB/sec) vs Message Size (Bytes)

- Send withRecv
- Send with irecv
- Send with Recv
- Send with recv
- Send with Recv
- Send with irecv
- Send with Recv
- Send with irecv
- Send with Recv

6/13/12
Point-to-point Bandwidth Comparisons
References


