Maximizing MPI Point-to-Point Communication Performance on RDMA-enabled Clusters with Customized Protocols

Matthew Small Xin Yuan
Department of Computer Science, Florida State University
Tallahassee, FL 32303
{small, xyuan}@cs.fsu.edu

ABSTRACT

Message Passing Interface (MPI) point-to-point communications are usually realized with two protocols, the eager protocol for small messages and the rendezvous protocol for medium and large sized messages. Traditional sender-initiated rendezvous protocols are sub-optimal in many situations. In this work, we propose to refine the rendezvous protocol for medium and large messages on RDMA-enabled clusters with three protocols that are customized for different situations, a hybrid protocol for medium sized messages when the sender arrives early, a sender-initiated protocol for large messages when the sender arrives early, and a receiver-initiated protocol when the receiver arrives early. In comparison to traditional sender-initiated rendezvous protocols, the proposed scheme reduces unnecessary synchronizations, decreases the number of control messages that are in the critical path of communications, and improves the communication progress, which results in a significantly better communication-computation overlap capability. We present and analyze these protocols, and describe how these protocols and the eager protocol can be seamlessly integrated in one system without introducing an excessive number of control messages. We have implemented the proposed scheme for InfiniBand clusters. The experimental results demonstrate the effectiveness of the proposed technique.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

General Terms

Design, Performance

Keywords

MPI, RDMA, Rendezvous Protocol

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICS'09, June 8–12, 2009, Yorktown Heights, New York, USA.
Copyright 2009 ACM 978-1-60558-498-0/09/06 ...$5.00.

1. INTRODUCTION

Message Passing Interface (MPI) point-to-point communications are usually realized with two internal protocols, the *eager* protocol and the *rendezvous* protocol. The eager protocol, which is used to transfer small messages, pushes the entire message to the receiver side regardless of the receiver state. In the rendezvous protocol, which is used for medium and large messages, the sender and the receiver negotiate before the data are sent.

In traditional eager and rendezvous protocols, the sender always initiates a communication by sending the first message. The receiver is passive in that it does not communicate any information to the sender even when it arrives at the communication before the sender. We will refer to these protocols as *sender-initiated* protocols. The combination of sender-initiated eager and rendezvous protocols for MPI point-to-point communication was designed more than ten years ago. Such a combination has simple protocol operations and is suitable for older networking systems where the cost for transmitting and processing control messages is high. However, such a combination does not yield ideal performance in many situations. In particular, the sender-initiated rendezvous protocol introduces unnecessary synchronizations between the sender and the receiver, requires more rounds of messaging than needed in common situations, and has a limited capability for supporting communication and computation overlaps [2, 15, 16, 18]. These limitations may introduce significant performance penalties in MPI applications.

Modern networks such as InfiniBand [3] and Myrinet [13] have drastically reduced the communication cost over older generation networks: communications are performed in the user domain with operation system bypass; most communication processing is off-loaded to the network interface card; and one-sided Remote Direct Memory Access (RDMA) allows direct data transfer from the source buffer to the remote destination buffer. In such systems, the cost for transmitting and processing small control messages is relatively low. As a result, introducing more complex protocols or combinations of protocols can be beneficial when the improvement of the communication efficiency out-weights the penalty associated with the more complex protocols.

Due to the importance of point-to-point communications in MPI applications, there are many recent efforts trying to improve the rendezvous protocol. In particular, receiver-initiated rendezvous schemes have been proposed to improve the communication performance of the sender-initiated rendezvous protocol [2, 15, 16]. Although the receiver-initiated protocol can in principle overcome some problems in the sender-initiated protocol, the receiver-initiated protocol has its own limitations. Moreover, simultaneously supporting the sender-initiated eager/rendezvous protocols and the receiver-initiated rendezvous protocol without introducing excessive overheads posts significant challenges. Existing proposals [2, 15, 16] fall short of fully addressing these challenges. The Gravel library [2] provides mechanisms to support both the sender-initiated and receiver-initiated protocols. However, it requires the user (programmer or compiler) to select the protocol for a communication. It was proposed to use the receiver-initiated protocol to replace the sender-initiated rendezvous protocol [15]. This approach cannot support full MPI functionality such as MPI_ANY_SOURCE [15]. The scheme in [16] supports both sender-initiated and receiver-initiated rendezvous protocols simultaneously. However, among other issues, this scheme requires an acknowledgment message for each control message, which introduces significant overheads.

In this work, we extend the techniques in [2, 15, 16] and propose to refine the rendezvous protocol on RDMA-enabled clusters with three protocols that are customized for different situations, a hybrid protocol for medium sized messages when the sender arrives early, a sender-initiated rendezvous protocol for large messages when the sender arrives early, and a receiver-initiated rendezvous protocol when the receiver arrives early. In comparison to traditional sender-initiated rendezvous protocols, by using customized protocols for different situations, the proposed scheme reduces unnecessary synchronizations, decreases the number of control messages that are in the critical path of communications, and improves the communication progress, which results in a better communication-computation overlap capability. We present and analyze the proposed protocols, and describe how these three protocols and the eager protocol can be integrated without introducing excessive overheads. The resulting point-to-point communication system, which is a seamless integration of four protocols, achieves better performance than existing systems, and supports all MPI functionality including MPI_ANY_SOURCE and MPI_ANY_TAG. We have implemented the proposed scheme for InfiniBand clusters. The experimental results confirm the effectiveness of the proposed scheme.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we describe the existing eager and rendezvous protocols and discuss their weaknesses. In Section 4, we present the proposed customized protocols and discuss how they can be integrated. Section 5 presents the results of our experimental evaluation. Section 6 concludes the paper.

2. RELATED WORK

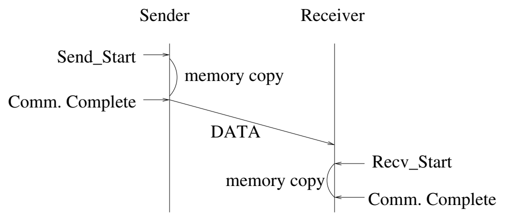
The limitations of the rendezvous protocol have been well documented (see e.g. [1, 5]) and various schemes that improve the communication progress and the communication-computation overlap capability have been proposed. Existing techniques can in general be classified into three types: using interrupts to improve communication progress [1, 17, 18], using asynchronous communication progress to improve communication-computation overlaps [6, 7, 8, 10, 19], and improving the protocol design [2, 15, 16, 18]. The interrupt driven message detection approach [1, 17, 18] allows each party (sender or receiver) to react to a message whenever the message arrives. The drawback is the non-negligible interrupt overhead, which is usually significant in synchronous communications. The asynchronous communication progress allows communications to be performed asynchronously with the main computation thread. This approach either needs a helper thread [8, 10, 19] or requires additional hardware support [6, 7]. This approach focuses on improving communication and computation overlaps and often introduces penalties for synchronous communications. As such, it may not be able to offer high performance for many applications. The third approach tries to improve the performance with better protocols, which can benefit both synchronous and asynchronous communications. It was shown that a sender-initiated RDMA read-based rendezvous protocol uses fewer control messages between the sender and the receiver than the RDMA write-based rendezvous protocol [18]. Pakin [15] showed that the receiver-initiated rendezvous protocol is simpler and can achieve higher performance in most cases than the sender-initiated protocol, and proposed to use the receiver-initiated protocol to replace the sender-initiated protocol for realizing subset MPI on Cell processors. However, the receiver-initiated rendezvous protocol alone cannot support some MPI features such as MPI_ANY_SOURCE. Rashti [16] proposed to combine the sender-initiated and receiver-initiated protocol in one communication system. However, this scheme introduces excessive control overheads, requiring each control message to be acknowledged. The Gravel library [2] supports many different protocols, but requires the user or the compiler to select the most appropriate protocol for each communication. Our work extends the results in [2, 15, 16]. First, we further refine the cases that can be handled by customized protocols and introduce a new hybrid protocol to handle medium sized messages when the receiver does not arrive early. Second, we show how all of the proposed protocols can be integrated in one system that supports full MPI functionality without excessive overheads.

3. EAGER AND RENDEZVOUS PROTOCOLS

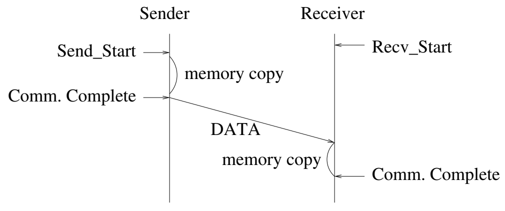
3.1 Eager protocol

A RDMA-based eager protocol is developed in MVAPICH [9]. In this scheme, the message data are first copied to the sender side buffer. Then, the data are RDMA written to a pre-allocated receiver side buffer associated with the sender side buffer (persistent buffer association [9]), and the sender side operation is complete after the RDMA write command is issued. The receiver polls the content of the pre-allocated receiver side buffer and copies the data to user buffer after the data arrive.

Two examples of the eager protocol are shown in Figure 1. Let *msize* be the message size to be transmitted; *MC(msize)* be the time for making a local copy of the message; *Tr(msize)* be the time to transfer a message of *msize* bytes, which is defined as the duration between the time when the sender issues the communication command to the time when the message is completely received by the receiver. Assuming issuing a communication command without waiting for its completion takes negligible time, the sender can complete the operation at time $MC(msize)$ after it starts the send operation. The receiver completes the operation at the time $MC(msize)$ after it starts the



(a) Receive posted after data arrived



(b) Receive posted before data arrived

Figure 1: The eager protocol

receive operation when the receiver starts the operation after the message arrives at the receiver side (Figure 1 (a)), or $Tr(msize) + 2MC(msize)$ after the sender starts the operation when the receiver starts the operation before the data message arrives. (Figure 1 (b)). When $msize$ is small, $MC(msize)$ is negligible, and the communication performance is close to optimal in both cases. The limitations of the protocol include the memory copy overheads and the buffer memory requirement at the receiving side where buffers must be pre-allocated for each connection ($O(N)$ memory requirement, where N is the number of nodes in the system [9]). These limitations prevent the eager protocol from being applied to medium and large messages.

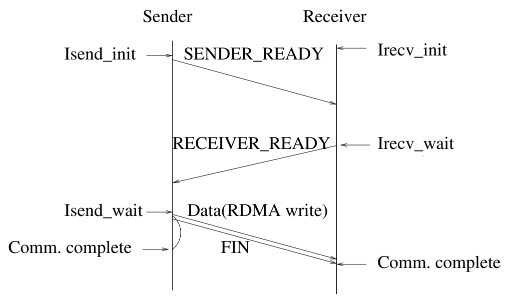
3.2 Rendezvous protocols

Rendezvous protocols do not require buffers and memory copies. However, using these protocols, the sender and the receiver must negotiate with each other before the data are transferred, which implies that both the sender and the receiver cannot complete the communication before the other party starts the operation. An example of the sender-initiated RDMA write-based rendezvous protocol [9] is shown in Figure 2 (a). In this protocol, the sender initiates the communication by sending a SENDER_READY packet, the receiver then responds with a RECEIVER_READY packet, after that the message data are RDMA written and a FIN packet is sent to indicate the completion of the communication. The sender must then wait for the communication of the data to complete before it can return from the operation.

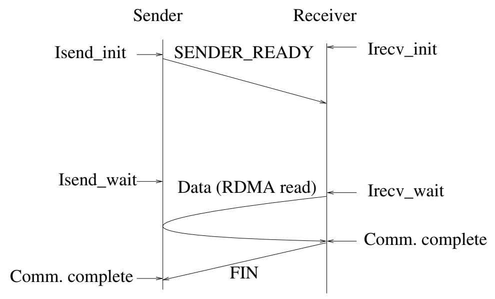
Sur [18] proposed a sender-initiated RDMA read-based rendezvous protocol, shown in Figure 2 (b). In this protocol, the receiver responds to the SENDER_READY packet with a RDMA read operation. After the RDMA read operation is completed, the receiver sends a FIN packet to the sender and completes the operation. The sender exits the operation after it receives the FIN packet. In comparison to the RDMA write-based protocol, this protocol eliminates RECEIVER_READY in the critical path, which may result in better communication progress [18]. However, the RDMA write-based protocol allows the sender to complete the operation after the receiver sends the RECEIVER_READY packet (before the receiver completes the operation), which can be done with one MPI routine call that invokes the communication progress engine, while in the RDMA read-based protocol, the sender completes the operation after the receiver receives the data and sends the FIN message, which usually takes two MPI routine calls, one to issue the RDMA read command and the other one to check the completion of the RDMA read operation and send the FIN packet. Hence, depending how the MPI routines are called, there is no clean-cut winner between these two protocols: they have different implicit synchronization structures.

Both the RDMA read-based protocol and the RMDA write-based protocol are not ideal in some situations. First, both protocols introduce (different) unnecessary synchronizations. Inherently, the only synchronization between the sender and the receiver is that the receiver completion of the operation depends on the sender starting the operation. However, using the RDMA write-based rendezvous protocol, the following sequence of events must happen: $SENDER\_READY \rightarrow RECEIVER\_READY \rightarrow RDMA\,write \rightarrow Recv.\,complete$. In the RDMA read-based protocol, the sender completion depends on the receiver completion, which is not inherently needed. The effects of the unnecessary synchronizations in the protocols is amplified when communication progress can only be made in MPI library calls. This is known as the communication progress issue in the rendezvous protocols. Second, both protocols are sub-optimal when the receiver arrives at the operation early. In this case, a more efficient receiver-initiated protocol can significantly improve performance [2, 15, 16].

When one considers possible situations in a point-to-point communication, there are cases when customized protocols can be used to produce near optimal performance by not introducing unnecessary synchronizations and by simplifying the protocol. We identify two such situations, (1) when the



(a) RDMA write based protocol



(b) RDMA read based protocol

Figure 2: Rendezvous Protocols

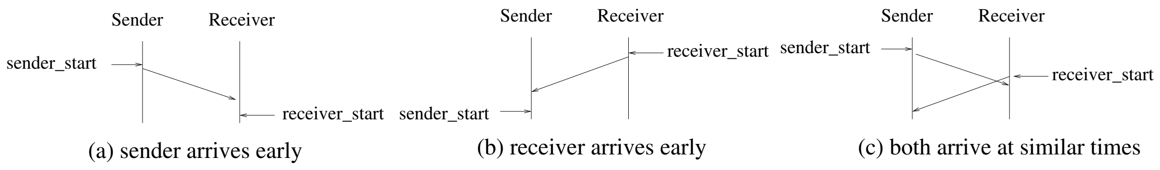


Figure 3: The timing when the sender and receiver arrive at a communication

receiver arrives at the communication early, and (2) when the message is medium sized. We propose to use customized protocols to handle different situations and to integrate these customized protocols in one system to achieve maximum communication efficiency.

4. THE PROPOSED SCHEME

In our proposed scheme, each party (sender or receiver) would try to notify the other party its readiness for the communication as soon as it arrives at the operation. The protocol for a given communication is selected not only based on the message size, but also based on the timing when the sender and the receiver arrive at the communication. There are three cases, shown in Figure 3, that characterize the timing when the parties arrive at the communication.

- *Sender arrives early* (Figure 3 (a)). In this case, after the sender arrives at the communication, it can send a control message to the receiver and the message will be delivered before the receiver arrives at the communication.
- *Receiver arrives early* (Figure 3 (b)). In this case, after the receiver arrives at the communication, it can send a control message to the sender and the message will be delivered before the sender arrives at the communication.
- *Both arrive at similar times* (Figure 3 (c)). This is the case when each party does not have time to deliver a control message before the other party arrives.

We refine the traditional rendezvous protocol with three protocols that are customized to different situations, the hybrid protocol ($HYBRID$), the sender-initiated rendezvous protocol ($SEND\_RNDV$), and the receiver-initiated rendezvous protocol ($RECV\_RNDV$). These protocols are then integrated with the eager protocol ($EAGER$) to form a complete point-to-point communication system. Hence, the full communication system consists of four protocols with two message size thresholds, $EAGER\_THRESHOLD$ and $HYBRID\_THRESHOLD$. Table 1 summarizes the protocols to be used in different situations.

For small messages ($msize \leq EAGER\_THRESHOLD$), the eager protocol shown in Figure 1 is near optimal for all situations as discussed in Section 3.

For medium sized messages ($EAGER\_THRESHOLD < msize \leq HYBRID\_THRESHOLD$), two protocols are used. When the sender arrives early, the sender first copies the message data to a local buffer. After that, it checks if the receiver has arrived (RECEIVER_READY message). If the receiver has arrived, a minor variation of the receiver-initiated protocol, shown in Figure 5 (a) is used. Since the data have been copied to the system buffer, the system buffer can be used in the RDMA write, which allows the sender to complete the operation without waiting for the RDMA write operation to complete. If the sender has not received the RECEIVER_READY message, the hybrid protocol shown in Figure 4 (a) is used. In this case, the sender issues a SENDER_READY message, which contains the address of the local buffer and other related information to facilitate the RDMA read from the receiver, to the receiver. The task in the sender side is completed; and the sender can exit the operation. When the receiver gets the SENDER_READY message, it performs a RDMA read to obtain the data from the sender buffer. Note that the sender side buffer must be released at some point. However, since this is a library buffer that the application will not access, the information for the receiver to notify the sender that the buffer can be released can be piggybacked in a later control message. HYBRID essentially behaves like the EAGER protocol. It does not introduce any unnecessary synchronizations. In comparison to EAGER, HYBRID does not require buffering at the receiving side (does not have the $O(N)$ memory requirement) and removes the memory copy at the receiving side. These enable HYBRID to efficiently transfer medium sized messages that are larger than those handled by EAGER.

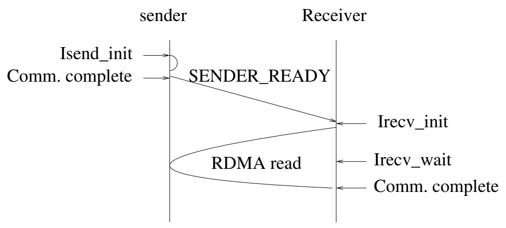
The hybrid protocol is also used when both parties arrive at similar times. This scenario is shown in Figure 4 (b). In this case, both SENDER_READY and RECEIVER_READY are sent. Once the sender sends the SENDER_READY message, it completes the operation. RECEIVER_READY will be ignored and dropped by the sender later. After the receiver sends RECEIVER_READY, it monitors both its data buffer and incoming control messages. If it sees a matching SENDER_READY, it performs the RDMA read operation to obtain the data.

When the receiver arrives at the communication early, it sends a RECEIVER_READY message to the sender, which carries the receiving buffer information. When the sender arrives, it can directly deposit the message into the receiver user space. Note that the RECEIVER_READY message is not in the critical path of the communication when the receiver arrives early. Hence, only the absolutely needed message transfer operation is in the critical path in this case: the protocol can deliver near optimal performance. The receiver-initiated protocol is depicted in Figure 5 (a).

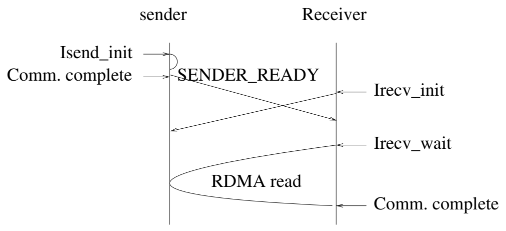
For large messages ($HYBRID\_THRESHOLD < msize$), we also separate two cases. When the receiver arrives early, the receiver-initiated protocol shown in Figure 5 (a), is used. When the sender arrives early, a sender-initiated protocol, either the RDMA write-based scheme in Figure 2 (a) or the

Message size	early party	protocol
$msize \leq eager\_thresh$ (small message)	sender	$EAGER$
	receiver	$EAGER$
	both	$EAGER$
$eager\_thresh < msize$ $msize \leq hybrid\_thresh$ (medium message)	sender	$HYBRID$
	receiver	$RECV\_RNDV$
	both	$HYBRID$
$hybrid\_thresh < msize$ (large message)	sender	$SEND\_RNDV$
	receiver	$RECV\_RNDV$
	both	$RECV\_RNDV$

Table 1: The proposed scheme

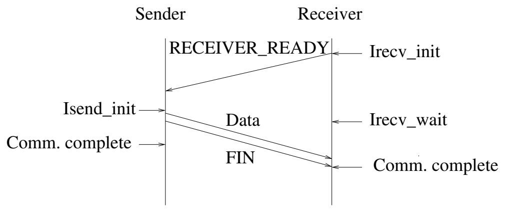


(a) hybrid protocol (sender early)

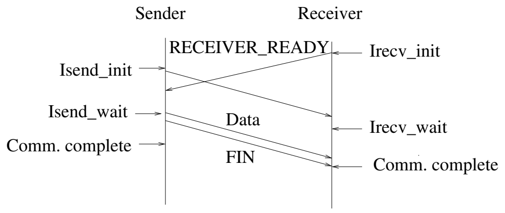


(b) hybrid protocol (both)

Figure 4: The hybrid protocol



(a) Receiver–initiated rendezvous(receiver early)



(b) Receiver–initiated rendezvous(both)

Figure 5: The receiver-initiated rendezvous protocol

RDMA read-based scheme in Figure 2 (b), can be used. As discussed earlier, there is no clean-cut winner between these two protocols: either one can provide a better performance in different situations. Our prototype system supports both protocols; either protocol can be pre-selected to handle the large messages when the sender arrives early. When both parties arrive at similar times, the receiver-initiated protocol is used: the SENDER_READY packet is ignored when the matching RECEIVER_READY packet has been sent. This scenario is depicted in Figure 5 (b). Notice that in this case, SENDER_READY is not in the critical path of the communication operation.

For a typical communication where the receiver is receiving from one sender, the receiver-initiated protocol can be used for all cases (sender never sends out SENDER_READY). We have discussed the two cases when the receiver arrives early and when both arrive at similar times. In the case when sender arrives early, it does not need to send the SENDER_READY message. Instead, it just needs to respond to the RECEIVER_READY message, and the behavior would be similar to the sender-initiated RDMA write-based protocol. However, the SENDER_READY message is essential for supporting MPI_ANY_SOURCE.

If one can integrate all these protocols into one communication system, the system will provide better communication performance than existing schemes that are based on the sender-initiated eager and rendezvous protocols. The advantages of the proposed integrated scheme are summarized in Table 2. For medium sized messages when the sender arrives early or both arrive at similar times, HYBRID behaves like EAGER and enjoys the advantages of EAGER over the rendezvous protocol. For medium sized messages and large messages when the receiver arrives early, the integrated system will use the receiver-initiated protocol that, in comparison to sender-initiated protocols, removes unnecessary synchronizations, reduces the number of rounds of control messages, and provides better communication progress. For large messages when both arrive at similar times, SENDER_READY is removed from the communication critical path, which results in a more efficient protocol.

4.1 Integrating the four protocols

We will now describe how the four protocols can be integrated without introducing excessive control overheads. Figure 6 gives the high level description of a system that simultaneously supports the four protocols. In the description, we assume that the receiver receives from one sender. The support of MPI_ANY_SOURCE will be discussed later. We also assume the buffers for eager messages and control messages are organized using the persistent buffer association technique [9] so that the sender always knows where to send an eager or control message and the receiver always knows where to poll the incoming eager and control messages. The sender-initiated protocol described in Figure 6 is the RDMA write-based rendezvous protocol. The RDMA read-based rendezvous protocol can also be used to replace the RDMA write-based protocol with minor modifications.

In Figure 6, we present the communication system by describing the tasks in four critical points during a communication. Isend represents the time when the send operation starts (e.g. MPI_Isend is called); Isend_Wait represents the time when the send operation needs to be completed (e.g. MPI_Wait is called on the MPI_Isend); Irecv represents the time when the receive operation starts (e.g. MPI_Irecv is called); Irecv_Wait represents the time when the receive operation needs to be finished (e.g. MPI_Wait is called on the MPI_Irecv). In the description, we omit some details. For example, the SENDER_READY message should contain the buffer address, message size, and other information so that the receiver can perform a RDMA read operation. The RECEIVER_READY message should contain the buffer address, and other information so that the sender can perform a RDMA write operation. We also assume that the preparations required for RDMA operations (e.g. memory registration) are done before the messages are sent out.

The communication system in Figure 6 seems to be somewhat straight-forward: the sender and the receiver perform protocol tasks based on the cases that we discussed. However, there are some subtleties in supporting the four protocols simultaneously. The subtleties include dealing with the mismatched protocols perceived by the sender and the receiver, and matching control messages. We will discuss the subtleties next.

Early party	Medium sized message	Large sized message
Sender	unnecessary synchronization removed, improved communication progress	NA
Receiver	simpler protocol, improved communication progress	simpler protocol, improved communication progress
Both	unnecessary synchronization removed; improved communication progress	number of protocol events in the critical path reduced

Table 2: Advantages of the integrated protocols over the traditional rendezvous protocol

```
Isend:
  (1)    If ($msize \leq EAGER\_THRESHOLD$) /* small msg, EAGER*/
  (2)      Copy data to eager buffer; post RDMA put; Done = 1; return
  (3)    Else if ($msize \leq HYBRID\_THRESHOLD$) /* medium msg */
  (4)      Check outstanding remote messages (drop older messages);
  (5)      If (find a matching RECEIVER_READY message) /* RECV_RNDV */
  (6)        Issue RDMA write to remote address, send FIN; return;
  (7)      Else
  (8)        Copy message to buffer; check remote message (drop older msgs);
  (9)        If (find a matching RECEIVER_READY) {Done = 1; goto (6);} /* RECV_RNDV */
  (10)       Else {send SENDER_READY; Done = 1; return;}/* HYBRID */
  (11)   Else /* large message */
  (12)     Check outstanding remote messages (drop older messages);
  (13)     If (find a matching RECEIVER_READY) goto (6) /* RECV_RNDV */
  (14)     Else {send SENDER_READY; return;} /* starting SEND_RNDV */
Isend_Wait:
  (1)    If (Done == 1) return; /* EAGER/HYBRID is done */
  (2)    Else if (RDMA write is outstanding for the communication) /* finishing RECV_ RNDV */
  (3)      Wait for the completion of the RDMA write message; return;
  (4)    Else /* finishing SEND_RNDV */
  (5)      Wait for RECEIVER_READY
  (6)      Issue RDMA write to remote address, send FIN;
  (7)      Wait for the RDMA write to complete; return;

Irecv:
  (1)    state = 0;
  (2)    If ($msize \leq EAGER\_THRESHOLD$) {return;}
  (3)    Check outstanding remote control message (drop older messages)
  (4)    If (find a matching SENDER_READY)
  (5)      If (message size $\leq HYBRID\_THRESHOLD$) /* HYBRID */
  (6)        Issue RDMA read to get the data; state = 1;return;
  (7)      Else /* SEND_RNDV */
  (8)        Send RECEIVER_READY; return;
  (9)    Send RECEIVER_READY; return;

Irecv_Wait:
  (1)    If (state == 1) {wait for the RDMA read to complete; return;}
  (2)    Poll control messages and eager messages
  (3)    If (receiving a matching eager message)
  (4)      Copy eager message to the user buffer; return;
  (5)    If (receiving a matching SENDER_READY message)
  (6)      If (message size is less than HYBRID_THRESHOLD)
  (7)        Issue RDMA read to get the data;
  (8)        wait for the read to complete; return
  (9)      Else drop the message /* useless message */
  (10)   If (receiving a matching FIN message) return;
  (11)   Goto (2); /* loop until one of the above happens */
```

Figure 6: High level description of the integrated scheme

• **Dealing with mismatched protocols perceived by the sender and the receiver**. MPI specification allows the message size posted by the sender to be different from that posted by the receiver. The only requirement is that the actual data size (posted by the sender) must be no larger than the size posted by the receiver. Hence, only the sender can decide the actual protocol. In our scheme, when the receiver posts a size larger than the EAGER_THRESHOLD, if the receiver does not receive SENDER_READY from the sender, it will send the RECEIVER_READY message, and then anticipate **all** possibilities by monitoring both the control messages and the eager messages (Line (2) in Irecv_Wait). It can easily be verified that all possible combinations of mismatched protocols (e.g. receiver RECV_RNDV and sender EAGER) will not cause problems in our system except that one extra useless RECEIVER_READY message may be sent. Note that even with no protocol mismatch, one extra useless control message may still be sent in our integrated scheme.

• **Matching control messages**. To support the four protocols in one framework, control messages for each communication must be matched. This is critical since our scheme allows great flexibility for the sender and the receiver to generate (sometimes useless) control messages. In the following discussion, we will assume that MPI_ANY_SOURCE and MPI_ANY_TAG are not used and each send/receiver operation is provided with a destination/source node and a tag. MPI_ANY_SOURCE and MPI_ANY_TAG will be discussed in the next bullet item. Our approach is to have each process keep track of the numbers of messages for each tag it sent to and received from other processes. Initially, the counters for all tags to/from another process are set to be the default value of 0. Every time a communication with a tag is posted, the corresponding counters are incremented. The header of each control message carries the counter to indicate its sequence. The receiver of a control message can compare the counter in the header with the counter that it maintains to decide whether the control message matches a given communication, is an old control message, or is for a future communication.

Consider an example where a receiver arrived early and posted two receives R1 and R2 with the same tag, where R1 is for a small message (using EAGER) and R2 is for a large message (using RECV_RNDV). R2 triggers a *RECEIVER_READY* message to be sent with a counter 1 (the second receive on the tag). Now, when sender arrives with the first send S1 of the tag, it will find a *RECEIVER_READY* message with a matching tag. However, since the local counter is 0 (to send the first message), the sender will decide that the *RECEIVER_READY* is for a future communication, and will perform the communication using EAGER without matching the *RECEIVER_READY* message. After this communication, the sender counter for the tag is incremented to be 1 (the first message for this tag has been sent). When the second send on the tag is posted, the *RECEIVER_READY* will be matched correctly.

Assuming that a constant number of tags are used in MPI programs, maintaining the counters for the tags/nodes introduces $O(N)$ memory overheads per node. Since the eager protocol also requires $O(N)$ eager buffer to be allocated in each node, the combined protocol is as scalable as EAGER. Note that an application may fail if it uses a large number of tags and the system runs out of the counter memory.

• **MPI_ANY_SOURCE and MPI_ANY_TAG**. Since our scheme supports simultaneously both sender-initiated and receiver-initiated protocols and since only the receiver can post a receive operation with MPI_ANY_SOURCE, we can support this feature by disabling the receiver-initiated messages for receiving operations with MPI_ANY_SOURCE. By having the receiver react to the sender with the sender-initiated scheme, the counters for the number of messages with each tag to each node can also be updated once the sender identity is resolved. MPI_ANY_TAG posts a similar problem for updating the counters, and can be handled in a similar fashion. Basically, when MPI_ANY_SOURCE and MPI_ANY_TAG are used, the receiver-initiated mechanism is suppressed and a sender-initiated protocol is used.

• **Eliminating the FIN messages in common cases**. In the receiver-initiated protocol, the last FIN message can be eliminated when the following two conditions are met: (1) the message size posted by the receiver matches the actual data size, and (2) receiving the last byte of a message indicates the arrival of the whole message. The first condition is common in MPI programs. The second condition is true in many contemporary systems including most InfiniBand clusters [9]. Receiver can put the message size that it anticipates and the value of the last byte in its buffer in the header of the RECEIVER_READY message. The sender can then check (1) whether the receiver anticipated size matches the actual data size and (2) whether the last byte value in the receiving end is not equal to the last byte value in its data to be sent. If both tests are true, the sender can just RDMA write the data without sending the FIN: the receiver will monitor the last byte in its data buffer (in addition to the control messages and eager messages) and detect the completion of the operation when the last byte value changes.

5. PERFORMANCE STUDY

We have implemented MPI point-to-point communication routines over InfiniBand using the Verbs API [4] based on the proposed technique. Five MPI point-to-point routines, MPI_Isend, MPI_Irecv, MPI_Send, MPI_Recv, and MPI_Wait, and other routines including MPI_Init(), MPI_Finalize(), MPI_Comm_rank(), and MPI_Comm_size(), are supported in our system. Our library can co-exist with MVAPICH [12]. When running an MPI application, we modify the five MPI routine calls in the application to invoke our routines, all other MPI routines that we do not support fall back to MVAPICH. The evaluation is done on an InfiniBand cluster, Draco, that has 16 compute nodes with a total of 128 cores. Each node is a Dell Poweredge 1950 with two 2.33Ghz Quad-core Xeon E5345's (8 cores per node) and 8GB memory. All nodes run Linux with the 2.6.9-42.ELsmp kernel. The compute nodes are connected by a 20Gbps InfiniBand DDR switch (CISCO SFS 7000D).

We compare the performance with MVAPICH2-1.2.rc1. Our system supports either sender-initiated RDMA write-based protocol or sender-initiated RDMA read-based protocol (for large messages when the sender arrives early). MVAPICH2-1.2.rc1 also supports both the RDMA write-based rendezvous protocol and the RDMA read-based rendezvous protocol. Unfortunately, the MVAPICH RDMA read-based rendezvous protocol is not stable on our experimental cluster: it failed to run most micro-benchmarks and all application benchmarks. Hence, we will focus on comparing our scheme with the default RDMA write-based rendezvous protocol in MVAPICH2-1.2.rc1. To obtain a fair comparison, our system also uses the RDMA write-based protocol when the sender arrives early. Since there is no clean winner between these two sender-initiated protocols, improving over either protocol demonstrates the effectiveness of the proposed integrated scheme. It must be noted that the experiments are designed to demonstrate that our proposed combined protocol is more efficient than the traditional rendezvous protocol used in MVAPICH, not to indicate that our system is better than MVAPICH. We use the default EAGER_THRESHOLD in MVAPICH, which is 12KB. In our library, the EAGER_THRESHOLD is also set to 12KB and the HYBRID_THRESHOLD is set to 40KB. These two threshold values are the optimal cut-off points for the pingpong benchmark in our platform.

5.1 Pingpong benchmark results

The pingpong benchmark tests the overall protocol efficiency when *there is no computation-communication overlap opportunities*. Figure 7 shows the results. Each result in the figure is the average of five runs. Our scheme consistently out-performs MVAPICH for this program. For medium sized messages (16KB and 32KB), the differences are more significant. For example, for the 16KB message, per round trip time using our scheme is 55 micro-seconds versus 67 micro-seconds with MVAPICH. This is due to the use of the hybrid protocol, which is more efficient for this size. For larger message sizes (64KB, 128KB, 256KB), the performance of our scheme is also better. For all of the sizes, our scheme is about 10 micro-seconds faster per round-trip. Two factors contribute to this performance gain. First, using our scheme, the SENDER_READY message is taken out of the critical path in the communication when the two processes arrive at a communication at similar times, and is eliminated when the receiver arrives early. Using monitoring counters in our system, we found that about 95% of the time, both parties arrive at the communications at similar times in this benchmark. Second, our scheme does not have the FIN message for this benchmark (eliminated by the technique discussed in Section 4.1). These results demonstrate that even without communication and computation overlap opportunities, the proposed integrated approach introduces less communication overheads and achieves higher performance than the traditional rendezvous protocol.

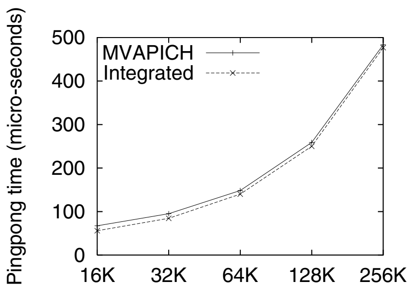


Figure 7: Pingpong performance

5.2 Progress benchmark results

In this subsection, we will use the benchmark shown in Figure 8, called progress benchmark, to evaluate the communication progress and communication-computation overlap capability of the proposed scheme. In this benchmark, the time for 1000 iterations of the loop is measured. Inside the loop, a barrier is first called to synchronize the sender and the receiver. After that, the sender performs some computation *comp*1, calls *MPI_Isend* to start the send operation, performs some more computation *comp*2, calls *MPI_Wait* to complete the send operation, and performs some more computation *comp*3. Similarly, the receiver also performs some computation *comp*4 after the barrier, calls *MPI_Irecv* to start the receive operation, performs some more computation *comp*5, calls *MPI_Wait* to complete the receive operation, and performs some more computation *comp*6. The message size and the computation in between the communication routines are parameters. We will use the notation configuration $(comp1, comp2, comp3, comp4, comp5, comp6)$ to represent configuration of the benchmark, where $compX$ represents the duration of the computation (in the unit of a basic loop). For each computation, the larger the number is, the longer the computation lasts. In the discussion, we will use notation $C(compX)$ for the time for $compX$ computations and $T(msize)$ for the time to transfer a message of $msize$ bytes. In the experiment, $C(X)+C(Y) \approx C(X+Y)$, $C(1) \approx 18\mu s$ and $T(100KB) \approx 90\mu s$.

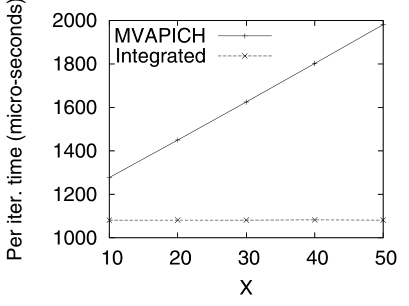
```
Process 1:                  Process 2:
  Loop:                       Loop:
    barrier()                   barrier()
    comp1                       comp4
    MPI_Isend()                 MPI_Irecv()
    comp2                       comp5
    MPI_Wait()                  MPI_Wait()
    comp3                       comp6
```

Figure 8: Progress micro-benchmark

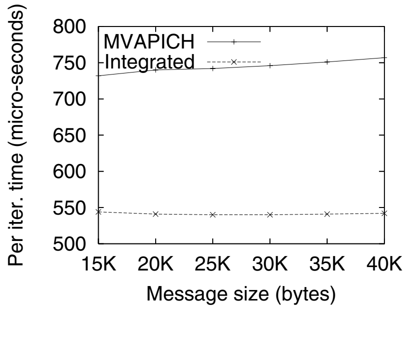
This benchmark can be configured to test the communication progress and the communication-computation overlap capability under various conditions. We designed four experiments to show the advantages of our integrated protocols, in particular, the advantages of the hybrid protocol and the receiver-initiated protocol. Since the performance when both sender and receiver arrive at similar times is implied in the pingpong results, we will focus on the cases when either the sender or the receiver arrives early.

Figure 9 shows the results for medium sized messages when the sender arrives early. Figure 9 (a) show the results for the configurations $(0, 0, 60, X = (10, 20, 30, 40, 50), 0, 0)$ and 30KB message size. In this configuration, the sender has a total of 60 units of computation while the receiver has a total of X units. As can be seen from the figure, the hybrid protocol completely eliminates the dependence from the sender to the receiver and as a result, the sender side computation can be completely overlapped with the communication and the receiver side computation, which results in a total time of roughly $C(60)$ for all different X's. Using the traditional rendezvous protocol, the sender cannot complete the communication until the receiver arrives at the operation. Hence, the sender side computation and receiver side computation are sequentialized in this configuration, resulting in a total per iteration time of roughly $C(60) + C(X) + T(msize)$, as shown in Figure 9 (a). Note that the penalty for unnecessary synchronizations depends on how the program is written and is independent of the time for transferring the data. Hence, the benefits for removing unnecessary synchronizations can potentially be very large. The hybrid protocol can also provide better communication progress at the receiver side. This is shown in Figure 9 (b) with the configuration $(0, 20, 0, 5, 5, 20)$. In this case, the receiver has a total 30 units of computation. When the hybrid protocol is used, the communication is overlapped with the 5 units of computation after the *MPI_Irecv* is called and the total per iteration time is roughly $C(30)$. Using the traditional rendezvous protocol, the data is moved at MPI_Wait at the sender side, hence the total per iteration time is $C(20) + C(20) + T(msize) \approx C(40) + T(msize)$. These results are captured in Figure 9 (b). By eliminating unnecessary synchronizations, HYBRID can significantly improve the communication progress both at the sender and the receiver in comparison to the traditional rendezvous protocol.

Figure 10 shows the progress benchmark results for large messages with receiver arriving early (so that RECV_RNDV is used in our scheme). Figure 10 (a) contains the results for configuration $(20, 20, 20, 0, 0, 0)$. This case tests the sender side computation-communication overlap capability. As can be seen from the figure, MVAPICH is not able to overlap computation with communication in this case. The total per iteration time is $C(60) + T(msize)$, which increases linearly with the message size. On the other hand, the integrated scheme uses RECV_RNDV for this communication and is able to overlap $comp2$ with the communication. The per iteration time is thus $C(40) + max\{C(20), T(msize)\}$, which increases only when $msize > 400KB$ and $T(msize) > C(20)$. Figure 10 (b) contains the results for configuration $(27, 0, 0, 20, 20, 20)$. This case tests the receiver side computation-communication overlap capability. As can be seen from the figure, the traditional rendezvous protocol is not able to overlap computation with communication in this case (per iteration time is still $C(60) + T(msize)$) while our scheme is able to overlap the communication and computation and has much lower total times for this program ($C(47) + max\{C(13), T(msize)\}$).

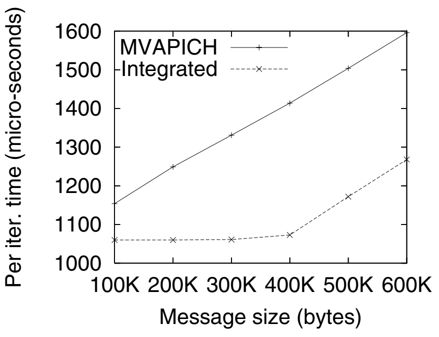


(a) Configuration $(0, 0, 60, X, 0, 0)$

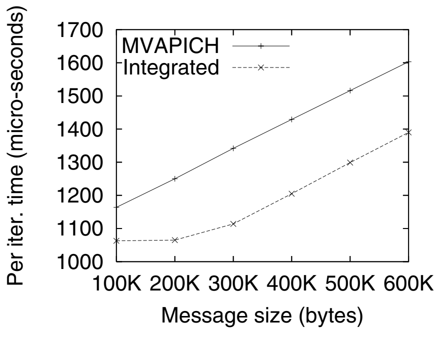


(b) Configuration $(0, 20, 0, 5, 5, 20)$

Figure 9: Advantage of HYBRID



(a) Configuration $(20, 20, 20, 0, 0, 0)$



(b) Configuration $(27, 0, 0, 20, 20, 20)$

Figure 10: Advantages of RECV_RNDV

These results demonstrate that for many common situations the integrated approach provides much better performance than the traditional rendezvous protocol. In particular, by using customized protocols, the proposed scheme reduces the unnecessary synchronization and improves communication progress at *both the sender and receiver sides*, which can result in significant performance gains.

5.3 Application benchmark results

We perform experiments using NAS applications benchmarks (NPB2.4) [14]. NPB2.4 has 6 point-to-point communication benchmarks: BT, CG, EP, LU, MG, and SP. EP does not have many communications. The communications in LU and MG are mainly performed by MPI_Irecv with MPI_ANY_SOURCE and our system uses the same protocol as MVAPICH. Hence, we will only discuss results for BT, CG, and SP.

Table 3 shows the total times, total communication times, and the communication improvement percentages using our scheme over MVAPICH for CLASS A, B, C of the three programs running on 8/9 and 16 processes. The communication time includes all Send, Isend, Recv, Irecv, and Wait times, which account for the majority of all communication times in these benchmarks. Since our system does not have optimizations for intra-node communication within a SMP node, we cannot fairly compare the protocols with intra-node communications. Hence, we run one process on each node so that all communications go through the InfiniBand interface. As can be seen from the table, our scheme obtains very significant improvement over MVAPICH for SP and BT (all classes) in terms of the communication time. This is because there are lots of communication and computation overlapping oppor-

Bench.	Class	N	MVAPICH		Integrated		Comm. improv.
			total	comm.	total	comm.	percentage
BT	A	9	32.95	1.41	32.12	0.60	135%
		16	18.40	1.29	17.38	0.57	126%
	B	9	137.21	2.99	136.00	1.81	65%
		16	79.12	4.14	76.84	1.63	154%
	C	9	565.39	7.49	563.37	5.44	38%
		16	321.76	10.11	315.51	3.95	156%
CG	A	8	0.71	0.14	0.70	0.12	17%
		16	0.43	0.13	0.43	0.13	0%
	B	8	25.43	2.05	25.17	1.86	10%
		16	14.34	1.89	14.30	1.82	4%
	C	8	64.92	3.73	64.50	3.32	12%
		16	35.66	3.41	35.46	3.25	5%
SP	A	9	16.48	1.08	16.00	0.62	74%
		16	9.46	0.96	9.14	0.64	50%
	B	9	76.38	2.58	75.18	1.40	84%
		16	41.51	2.83	39.95	1.33	128%
	C	9	325.25	6.99	322.35	3.97	76%
		16	180.57	6.53	177.22	3.02	116%

Table 3: Performance of application benchmark (all times are in seconds)

tunities in these two benchmarks. Our scheme is much better in exploring such opportunities, which was also demonstrated in the progressive benchmark results. Although communication does not account for a large percentage of the total application time in these benchmarks, the improvement in communication times transfers into improvement of the overall benchmarks time, as shown in the table. The CG benchmark does not have much communication-computation overlap opportunities. Our scheme still noticeably improves the communication performance in this program.

6. CONCLUSION

We develop an integrated point-to-point communication scheme that allows three customized protocols to be used for different situations that were handled by traditional sender-initiated rendezvous protocols. Compared to the traditional rendezvous protocol, the proposed scheme reduces unnecessary synchronizations, decreases the number of control messages that are in the critical path of the communications, and has a much better communication-computation overlap capability. We describe how the customized protocols and the eager protocol can be integrated in one system without introducing excessive control overheads. Our experiments with micro-benchmarks and application benchmarks confirm the effectiveness of the proposed scheme.

Acknowledgment

This work is supported in part by National Science Foundation (NSF) grant CCF-0541096 and through NSF TeraGrid resources provided by TACC.

7. REFERENCES

[1] G. Amerson and A. Apon, "Implementation and Design analysis of a Network Messaging Module using Virtual Interface Architecture," *International Conference on Cluster Computing*, 2004.

[2] A. Danalis, A. Brown, L. Pollock, M. Swany, and J. Cavazos, "Gravel: A Communication Library to Fast Path MPI," *Euro PVM/MPI*, September 2008.

[3] InfiniBand Trade Association, http://www.infinibandta.org.

[4] "InfiniBand Host Channel Adapter Verb Implementer's Guide", Intel Corp., 2003.

[5] J. Ke, M. Burtscher, and E. Speight, "Tolerating Message Latency through the Early Release of Blocked Receives," *Euro-Par 2005*, LNCS 2648, pp 19-29, 2005.

[6] C. Keppitiyagama and a. Wagner, "MPI-NP II: A Network Processor Based Message Manager for MPI," *International Conference on Communications in Computing*, 2000.

[7] C. Keppitiyagama and a. Wagner, "Asynchronous MPI Messaging on Myrinet," *IEEE International Parallel and Distributed Processing Symposium* (IPDPS), 2001.

[8] R. Kumar, A. Mamidala, M. Koop, G. Santhanaraman and D.K. Panda, "Lock-free Asynchronous Rendezvous Design for MPI Point-to-Point Communication," *EuroPVM/MPI*, Sept. 2008.

[9] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," the *17th International Conference on Supercomputing* (ICS), pages 295-304, June 2003.

[10] S. Majumder, S. Rixner, and V. S. Pai, "An Event-Driven Architecture for MPI Libraries," The *Los Alamos Computer Science Institute Symposium*, 2004.

[11] The MPI Forum. *MPI: A Message-Passing Interface Standard, Version 1.3*, May 2008. Available at http://www.mpi-forum.org/docs/mpi-1.3/mpi-report-1.3-2008-05-30.pdf.

[12] MVAPICH: MPI over InfiniBand and iWARP, http://mvapich.cse.ohio-state.edu.

[13] Myricom, http://www.myricom.com.

[14] NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/

[15] S. Pakin, "Receiver-initiated Message Passing over RDMA Networks," the *22nd IEEE International Parallel and Distributed Processing Symposium* (IPDPS), April 2008.

[16] M. J. Rashti and A. Afsahi, "Improving Communication Progress and Overlap in MPI Rendezvous Protocol over RDMA-enabled Interconnects," the *22nd High Performance Computing Symposium* (HPCS), June 2008.

[17] D. Sitsky and K. Hayashi, "An MPI Library Which Uses Polling, Interrupts, and Remote Copying for the Fujitsu AP1000+," *International Symposium on Parallel Architectures, Algorithms, and networks*, 1996.

[18] S. Sur, H. Jin, L. Chai, D. K. Panda, "RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits," *ACM PPoPP*, pages 32-39, 2006.

[19] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D.K. Panda, "Host-Assisted Zero-Copy Remote Memory Access Communication on InfiniBand," *IEEE International Parallel and Distributed Processing Symposium*, 2004.