J. Parallel Distrib. Comput. 69 (2009) 117–124

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Bandwidth optimal all-reduce algorithms for clusters of workstations

Pitch Patarasuk, Xin Yuan *

Department of Computer Science, Florida State University, Tallahassee, FL 32306, United States

ARTICLE INFO

Article history:
Received 22 May 2007
Received in revised form
2 August 2008
Accepted 23 September 2008
Available online 14 October 2008

Keywords:
All-reduce
Collective communication
Tree topology
Cluster of workstations

ABSTRACT

We consider an efficient realization of the all-reduce operation with large data sizes in cluster environments, under the assumption that the reduce operator is associative and commutative. We derive a tight lower bound of the amount of data that must be communicated in order to complete this operation and propose a ring-based algorithm that only requires tree connectivity to achieve bandwidth optimality. Unlike the widely used butterfly-like all-reduce algorithm that incurs network contention in SMP/multi-core clusters, the proposed algorithm can achieve contention-free communication in almost all contemporary clusters, including SMP/multi-core clusters and Ethernet switched clusters with multiple switches. We demonstrate that the proposed algorithm is more efficient than other algorithms on clusters with different nodal architectures and networking technologies when the data size is sufficiently large.

1. Introduction

The all-reduce operation combines values from all processes and distributes the results to all processes. It is commonly used in parallel computing. In the Message Passing Interface (MPI) standard [18], the routine for this operation is *MPI_Allreduce*.

We consider an efficient realization of the all-reduce operation with large data sizes in cluster environments, under the assumption that the reduce operator is associative and commutative. We derive a tight lower bound of the amount of data that must be communicated in order to complete the all-reduce operation, and use the lower bound to establish the minimum time required for this operation. We propose a ring-based algorithm for this operation that achieves bandwidth optimality on tree topologies in that (1) each node sends the minimum amount of data required to complete this operation, and (2) all communications are contention free.

Currently, the most widely used all-reduce scheme is the butterfly-like algorithm [22,23,27], where the all-reduce operation is realized with a recursive halving reduce-scatter followed by a recursive doubling all-gather. When the network can support the butterfly communication pattern without contention, this algorithm is optimal both in the latency term (using the minimum number of communication rounds needed) and in the bandwidth term (each node communicating the minimum amount of data required). The problem with the butterfly-like algorithm is that the butterfly communication pattern can cause network contention in many contemporary clusters, such as the widely deployed SMP/multi-core clusters. In contrast, our ring-based algorithm only requires a tree topology to be bandwidth optimal, and can achieve contention-free communication in almost all contemporary clusters including SMP/multi-core clusters and Ethernet switched clusters with multiple switches. The ring-based algorithm also requires less working memory and can be applied to clusters with non-power-of-two numbers of nodes. One limitation of the proposed algorithm is that it is only optimal in the bandwidth term, but not the latency term: the number of communication rounds is proportional to the number of processes. Another issue in the ring-based algorithm is that the reduction results are computed with different "bracketing", which may cause problems in the presence of rounding errors.

We evaluate the proposed algorithm on various clusters of workstations, including high-end SMP/multi-core clusters with Myrinet and InfiniBand interconnects and low-end Ethernet switched clusters. The results show that the proposed algorithm significantly outperforms other algorithms when the data size is sufficiently large, which demonstrates the effectiveness of the proposed algorithm.

The rest of the paper is organized as follows: Section 2 introduces the all-reduce operation and the communication model that we use. In Section 3, we derive the theoretical lower bound on the communication time required for this operation. Section 4 presents the proposed bandwidth optimal all-reduce algorithm. Section 5 reports the results of our experiments. The related work is discussed in Section 6. Section 7 concludes the paper.

* Corresponding author.
E-mail addresses: patarasu@cs.fsu.edu (P. Patarasuk), xyuan@cs.fsu.edu (X. Yuan).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.09.002

2. Background

2.1. All-reduce operation

We will use a generic operator $\oplus$ to denote the reduce operator in the all-reduce operation. *MPI_Allreduce* requires the reduce operator to be associative, that is, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$. Moreover, all built-in operations for *MPI_Allreduce* are also commutative, that is, $a \oplus b = b \oplus a$. We assume that the reduce operator is both associative and commutative in this paper.

In terms of operating results, an all-reduce operation is equivalent to a reduction operation that reduces the results to one process, followed by a broadcast operation that distributes the results to all processes. Specifically, let the N processes be denoted as $p_0, p_1, \ldots, p_{N-1}$. Before the all-reduce operation, each process $p_i, 0 \leq i \leq N-1$, has X data items $a_i^0, a_i^1, \ldots, a_i^{X-1}$. At the end of the operation, all processes have all X-item results $r^0, r^1, \ldots, r^{X-1}$, where $r^j = a_0^j \oplus a_1^j \oplus \cdots \oplus a_{N-1}^j, 0 \leq j \leq X-1$.

In the presence of rounding errors, MPI poses some quality requirements for this operation [23]. It is required that all processes must receive the same resulting data. In addition, the same reduction order and bracketing for all elements is not strictly required, but should be strived for.

2.2. Communication performance model

Since we consider operations with large data sizes, we will use a *linear model* to model the time for point-to-point communications: the time to transfer a message of size m, $T(m) = \beta m$, where β is a constant. Notice that the linear model ignores communication start-up overheads and the communication time is proportional to the message size. Let i be a constant, $T(i \times m) = i \times T(m)$ and $T(m_1 + m_2) = T(m_1) + T(m_2)$.

In analyzing all-reduce algorithms, we assume that all processes start the operation at the same time. The time for an all-reduce operation is defined as the time when all processes start the operation until the last process receives the last message (and computes the final result). Our analysis focuses on the communication time and ignore the computation time. In practice, both the communication start-up overheads and the computation in the operation, which are omitted in our analysis, can be significant.

3. The lower bounds

In a general-purpose all-reduce operation, data items are independent of each other. The reduction on different items are independent of one another. Hence, the amount of data that must be communicated in order to complete an all-reduce operation on X items is equal to X times the amount for the single-item all-reduce operation. To obtain the lower bound for a general all-reduce operation on X items, we only need to find the lower bound for the operation on a single item.

In a one-item all-reduce operation on N processes, each process $p_i, 0 \leq i \leq N-1$, has one *initial item* a_i. At the end of the all-reduce operation, all processes contain the *final result* $r = a_0 \oplus a_1 \oplus \cdots \oplus a_{N-1}$. Let $B = \{b_0, \ldots, b_i\}$ be a set of initial items. We will use the notation $\oplus(B) = \oplus(\{b_0, b_1, \ldots, b_i\}) = b_0 \oplus b_1 \oplus \cdots \oplus b_i$ to represent the reduction operation on the initial items in B. Let $ALL = \{a_0, a_1, \ldots, a_{N-1}\}$ be the set that contains all initial items, $r = \oplus(ALL)$. Let $B = \{b_0, b_1, \ldots, b_i\} \subseteq ALL$ be a subset of initial items, $\oplus(B) = b_0 \oplus b_1 \oplus \cdots \oplus b_i$ is a *partial result*. All initial items, $a_i, 0 \leq i \leq N-1$, as well as the final result r are partial results. Let ϕ be the empty set. Since $\phi \subseteq ALL$, for completeness, we introduce a null (empty) partial result, denoted as $\perp = \oplus(\phi)$. Intuitively, any computation (reduction) result that is not a partial result will not contribute to the calculation of the final result in any way and can be represented as $\perp$. Each partial result except $\perp$ has the same size as an initial item. Let $\oplus(B)$ be a partial result and $a \in B$, we say that the *effect* of item a is in the partial result $\oplus(B)$. Let $pr_1 = \oplus(A)$ and $pr_2 = \oplus(B)$ be two partial results. We extend the reduce operator to take partial results as parameters. Specifically,

$$pr_1 \oplus pr_2 = \begin{cases} \oplus(A \cup B), & \texttt{if } (A \cap B = \phi) \texttt{ and } (A \neq \phi) \texttt{ and } (B \neq \phi) \\ \perp, & \texttt{if } (A \cap B \neq \phi) \texttt{ or } (A = \phi) \texttt{ or } (B = \phi). \end{cases}$$

Basically, if two partial results cover disjoint sets of initial items, applying the reduce operation on these two partial results yields another partial result that covers the union of the initial items. When the items covered by these two partial results intersect, applying the reduce operation on the two partial results yields something that is not a partial result (some initial items appear multiple times), which is denoted by $\perp$ as discussed earlier.

In our proofs, we also use a $\ominus$ operator on partial results, which is defined as follows. Let $pr_1 = \oplus(A)$ and $pr_2 = \oplus(B)$ be two partial results,

$$pr_1 \ominus pr_2 = \oplus(A - B).$$

Basically, the $\ominus$ operator removes the effects of items in B from the partial result pr_1. For example, $(a \oplus b \oplus c) \ominus b = \oplus(\{a, b, c\} - \{b\}) = a \oplus c$; $a \ominus a = \oplus(\{a\} - \{a\}) = \perp$.

During the all-reduce operation, each process may send a partial result that it can compute to another process or receive a partial result from another process. The partial results that can be computed by a process in a given time depend on the initial item at the process as well as the set of partial results received by the process. We define the *state* of a process at a given time to be the set of partial results that include the initial item in the process and all partial results that the process received from other processes. We call the set of all partial results that can be computed at a given time in a process the *coverage set* of the process. Clearly, the coverage set is a function of the state. Let the state of a process be S, we denote the coverage set as $COVER(S)$:

$$COVER(S) = \{pr | \exists pr_1, pr_2, \ldots, pr_i \in S, pr = pr_1 \oplus pr_2 \oplus \cdots \oplus pr_i\}.$$

The partial results in a state are atomic items that cannot be separated even though they can be combined with other partial results. For example, if a process has a state $S = \{a, b \oplus c\}$, the coverage set is $COVER(S) = \{\perp, a, b \oplus c, a \oplus b \oplus c\}$, which does not include $a \oplus b$ or $a \oplus c$.

In the derivation of the lower bound, we focus on all-reduce schemes whose communications do no have race conditions such that all messages in an operation can be sequentialized. All-reduce algorithms with race conditions are usually incorrect and not considered in this paper. Without loss of generality, we will also assume that each message transfers one partial result between two processes: a message with multiple partial results are treated as multiple messages. A message is denoted as $(src \rightarrow dst, pr)$, where src is the source process, dst is the destination process, and pr is a partial result. Let a generic all-reduce scheme use n messages. Under the assumption that the messages do not have race conditions and can be sequentialized, the n messages can be ordered based on the timing when each message occurs. For concurrent messages that happen at the same time, the tie can be broken arbitrarily. Let messages $M_1, M_2, \ldots, M_n$ be the n messages in an all-reduce scheme ordered based on their timing. Let $M_j = (s_j \rightarrow d_j, pr_j)$ and $S_i^{M_j}$ be the state of process p_i before $M_j, 1 \leq j \leq n$. We will use the notation $S_i^{M_{n+1}}$ to denote the state of p_i after the last message (M_n). Clearly, we have $pr_j \in COVER(S_j^{M_j})$.

Since we are only interested in the number of partial results communicated, we will use the sequence of messages in an

algorithm to represent the algorithm. Let a message sequence be $M_1, M_2, \ldots, M_k$, where $M_j = (s_j \rightarrow d_j, pr_j)$. We say that the sequence of messages is *valid* when $pr_j \in COVER(S_j^{M_j})$, $1 \leq j \leq k$. Clearly, any all-reduce algorithm can be represented by a valid sequence of messages. Similarly, any valid sequence of messages can be realized by an algorithm.

Lemma 1. *The minimum number of partial results to be communicated to complete a one-item all-reduce operation on N processes over all algorithms is at most* $2 \times (N-1)$.

Proof. The proof is straight-forward by constructing an algorithm that communicates $2 \times (N-1)$ partial results to complete the operation. The all-reduce operation can be completed with a flat tree reduce algorithm that communicates $N-1$ partial results followed by a flat tree broadcast algorithm that communicates $N-1$ partial results. □

Lemma 2. *The number of partial results to be communicated to complete a one-item all-reduce operation on N processes using any algorithm is at least* $2 \times (N-1)$.

Proof. The lemma applies both when the messages are within the N processes and when additional processes are used to relay messages. We prove the lemma by induction on N.

Base case: $N = 2$. With one message, at most one process can send its data to the other process, and at most one process can compute $a_0 \oplus a_1$, which cannot complete the all-reduce operation. Hence, at least $1 + 1 = 2 = 2 \times (N-1)$ communications are needed.

Induction case: The induction hypothesis is that the minimum number of partial results to be communicated in order to complete a one-item all-reduce operation on N processes is at least $2 \times (N-1)$. Using this hypothesis, we will prove that the minimum number of messages required to complete the operation on $N+1$ processes is at least $2 \times ((N+1)-1) = 2N$.

Let the $N+1$ processes be $p_0, p_1, \ldots, p_N$ with p_i having initial item a_i before the operation. Let the minimum number of messages required to complete the operation on the $N+1$ processes be X+1, the ordered sequence of messages be $E_1, E_2, \ldots, E_X, E_{X+1}$, where $E_j = (s_j \rightarrow d_j, pr_j)$, $1 \leq j \leq X+1$, and the state of process p_i, $0 \leq i \leq N$, before message E_j be $S_i^{E_j}$. $S_i^{E_{X+2}}$ denotes the state after E_{X+1} (the last message). We have $r = a_0 \oplus a_1 \oplus \cdots \oplus a_N \in COVER(S_i^{E_{X+2}})$, $0 \leq i \leq N$.

We will prove $X + 1 \geq 2N$ by constructing a sequence of at most $X-1$ messages that allows the all-reduce operation on N processes to be completed, which requires at least $2N-2$ messages (induction hypothesis). Let us assume that p_N is the process that receives the last message (this assumption is valid since $\oplus$ is associative and commutative). The new sequence is constructed from the X-message sequence $E_1, E_2, \ldots, E_X$, in two steps.

In the first step, we construct a sequence F of X messages as follows. For each $E_j = (s_j \rightarrow d_j, pr_j)$, $1 \leq j \leq X$, there is a corresponding message $F_j = (s_j \rightarrow d_j, pr_j \ominus a_N)$. Message sequence F is exactly the same as the message sequence E except that the effect of a_N is removed. Note that some messages in sequence F may contain the empty partial result ($\perp$). Such empty messages will be removed in the next step. Let the state of process p_i, $0 \leq i \leq N$, before message F_j be $S_i^{F_j}$ (sequence F is applied from the initial condition of the operation). By induction on j, it is straightforward to show that

1. Message sequence F is valid, that is, $pr_j \ominus a_N \in COVER(S_{s_j}^{F_j})$, $1 \leq j \leq X$.
2. If $pr \in COVER(S_i^{E_j})$, $1 \leq j \leq X+1$ and $0 \leq i \leq N$, $pr \ominus a_N \in COVER(S_i^{F_j})$.

Using the message sequence E, before the last message (E_{X+1}), the final result $r = a_0 \oplus a_1 \oplus \cdots \oplus a_N$ must be in the coverage set of each of $p_0, \ldots, p_{N-1}$ since these processes do not receive the last message and their states do not change after the message. Hence, $r = a_0 \oplus a_1 \oplus \cdots \oplus a_N \in COVER(S_i^{E_{X+1}})$, $0 \leq i \leq N-1$, and $r \ominus a_N = a_0 \oplus a_1 \oplus \cdots \oplus a_{N-1} \in COVER(S_i^{F_{X+1}})$, $0 \leq i \leq N-1$: the reduction result of the N processes $(p_0, p_1, \ldots, p_{N-1})$ are in all of the N processes after the X messages in sequence F. Next, we will construct from sequence F a sequence of at most $X-1$ messages that allows the all-reduce operation on N processes $p_0, p_1, \ldots, p_{N-1}$ to complete. There are two cases.

Case (a): In message sequence F, if there is any message whose content is $\perp$ (empty message), this message can be removed and we obtain a sequence of at most $X-1$ messages that allows the all-reduce operation on N processes to complete.

Case (b): If there are no such messages in sequence F, each message in F contains a real partial result. Since $r = a_0 \oplus \cdots \oplus a_N$ is in the coverage set of all processes $p_0, p_1, \ldots, p_{N-1}$ after messages $E_1, E_2, \ldots, E_X$, process p_N must be the sender of at least one of these messages so that the effect of a_N can be distributed to other processes. Since we do not change senders and receivers from sequence E to sequence F, process p_N also must be the sender at least once in sequence F.

Let the first message in sequence F with p_N as the sender be $F_k = (s_k \rightarrow d_k, pr_k \ominus a_N)$, $s_k = p_N$. We create a new message sequence from sequence F as follows: (1) for every message whose sender and receiver are not p_N, keep the message intact; (2) if the receiver is p_N, change the receiver to d_k; (3) if the sender is p_N and the message is not F_k, change the sender to d_k (if the receiver is also d_k, then remove this message); (4) remove the F_k from the sequence. Basically, in the new sequence, d_k receives all the messages sent to p_N in sequence F, and sends out anything that p_N used to send out in sequence F. Since the effect of a_N has been removed in sequence F, this new sequence is valid. Each of the processes other than d_k and p_N receives exactly the same messages in the new sequence as in sequence F. Hence, the state of process p_i, $0 \leq i \leq N-1$ and $p_i \neq d_k$, is exactly the same after these two message sequences. Now consider process d_k. For the messages that it does not receive (from p_N) in the new sequence, it should be able to compute the partial results in the messages since d_k receives everything that p_N receives in sequence F: the state of d_k after the new sequence is a superset of its state after the sequence F. Since $a_0 \oplus \cdots \oplus a_{N-1}$ is in the coverage set of processes $p_0, p_1, \ldots, p_{N-1}$ after the sequence F, it should also be in the coverage set of all these processes after the new sequence which has at most $X-1$ messages (F_k is removed). Thus, $X-1$ messages are sufficient to complete the all-reduce operation on N processes. From the induction hypothesis, $X - 1 \geq 2N - 2$ and $X + 1 \geq 2N$. This concludes the induction case. □

Combining Lemmas 1 and 2, we obtain the lower bound result in the following lemma. Note that while we prove the lower bound for the case when the reduce operator is commutative, the bound also applies when the reduce operator is non-commutative since the set of all all-reduce algorithms with a non-commutative reduce operator is a subset of the set of all all-reduce algorithms with a commutative reduce operator.

Lemma 3. *The minimum number of partial results to be communicated to complete a one-item all-reduce operation on N processes over all algorithms is* $2 \times (N-1)$. □

Lemma 4. *Assume that* (1) *data are not compressed during the all-reduce operation; and* (2) *initial items are independent of one another. To perform an all-reduce operation on X items with itsize bytes item size on N processes, there exists at least one process that must communicate a total of at least* $\lceil 2 \times \frac{N-1}{N} \times X \rceil \times$ *itsize bytes data assuming the minimum unit for communication is an item.*

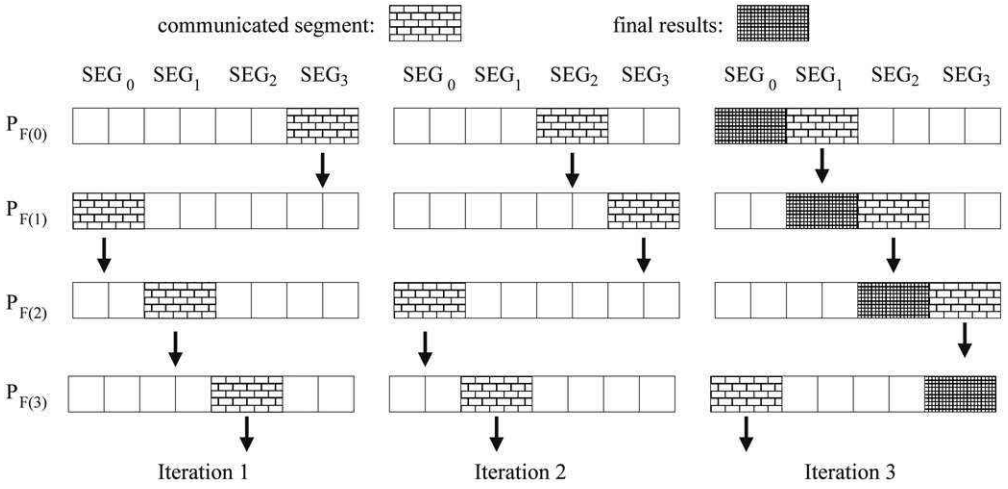


Fig. 1. Logical ring reduce-scatter algorithm.

Proof. Since the minimum number of items to be communicated in order to complete a one-item all-reduce operation is $2 \times (N-1)$ (Lemma 3), the minimum number of items to be communicated to complete an X item all-reduce operation is $X \times 2 \times (N-1)$. Since there are N processes that carry out the operation collectively, the communications can be distributed among the N processes. Hence, at least one process needs to communicate $\lceil \frac{2\times(N-1)\times X}{N} \rceil = \lceil 2 \times \frac{N-1}{N} \times X \rceil$ items and $\lceil 2 \times \frac{N-1}{N} \times X \rceil \times itsize$ byte data. □

Lemma 5. *Under the assumptions stated in Lemma 4, the minimum time for the all-reduce operation is at least* $T(\lceil 2 \times \frac{N-1}{N} \times X \rceil \times itsize)$. □

Let $msize = X \times itsize$ be the total data size in the operation. Using the linear communication model, $T(\lceil 2 \times \frac{N-1}{N} \times X \rceil \times itsize) \approx 2T(msize)$. This indicates that the best of any all-reduce algorithm can do is to achieve roughly 2 times the time to send the *msize*-byte data.

4. A ring-based bandwidth optimal algorithm

We will describe a ring-based bandwidth optimal all-reduce algorithm for tree topologies, and show how such an algorithm can be applied to high-end clusters. The tree topology by itself is not particularly interesting. We develop bandwidth optimal algorithms for the tree topology only because tree provides the minimum connectivity and most networks have tree embeddings: our bandwidth optimal all-reduce algorithm for trees can be applied to any topology with a tree embedding. Note that since our algorithm achieves bandwidth optimality, no algorithms can have a better theoretical performance on any topology, assuming that each node has one network port.

4.1. Bandwidth optimal algorithm for tree topologies

Our proposed bandwidth optimal all-reduce algorithm combines three existing ideas: (1) realizing the all-reduce operation by a reduce-scatter operation followed by an all-gather operation [22], (2) realizing both the reduce-scatter operation and the all-gather operation using logical ring based algorithms [7,25], and (3) constructing a contention-free logical ring on the tree topology [7]. Our contribution lies in combining these three components in an all-reduce algorithm and in showing that the resulting all-reduce algorithm is bandwidth optimal.

To achieve bandwidth optimality using the reduce-scatter followed by all-gather algorithm [22], both the reduce-scatter and the all-gather operations must be bandwidth optimal, which means that: (1) each process should communicate the minimum amount of data required to realize the operations; and (2) the network contention must be avoided. The optimality is achieved by using logical ring based reduce-scatter and all-gather algorithms, and by finding a contention-free logical ring on the tree topology. The logical ring all-gather algorithm is described in [25] and the technique to compute a contention-free logical ring on a tree topology can be found in [7]. Next, we will describe the logical ring reduce-scatter scheme.

Let the N processes be $p_0, p_1, \ldots, p_{N-1}$. Let $F : \{0, \ldots, N-1\} \rightarrow \{0, \ldots, N-1\}$ be a one-to-one mapping. Thus, $p_{F(0)}, p_{F(1)}, \ldots, p_{F(N-1)}$ is a permutation of $p_0, p_1, \ldots, p_{N-1}$. The logical ring pattern contains the following communications:

$$p_{F(0)} \rightarrow p_{F(1)} \rightarrow p_{F(2)} \rightarrow \cdots \rightarrow p_{F(N-1)} \rightarrow p_{F(0)}.$$

Using the logical ring pattern, the reduce-scatter operation is performed as follows. First, the *msize* source data in each process is partitioned into N segments, all segments having $\lceil \frac{msize}{N} \rceil$ bytes of data except the last segment, which has a segment size of $msize - (N-1)\lceil \frac{msize}{N} \rceil$. Let us number the segments by SEG_0, $SEG_1, \ldots, SEG_{N-1}$. The reduce-scatter operation is carried out by performing the logical ring pattern $N-1$ iterations. In the first iteration (iteration 1), process $p_{F(i)}$ sends segment $SEG_{(i-1) \bmod N}$ to $p_{F((i+1) \bmod N)}$. After each process receives the data, it performs a reduction operation on the received data segment with its corresponding data segment (the segment with the same segment index), and replaces its own data with the (partial) reduction results. For each remaining iteration $j : 2 \le j \le N-1$, each process $p_{F(i)}$ sends the newly computed $SEG_{(i-j) \bmod N}$ to $p_{F((i+1) \bmod N)}$. After receiving the data communicated in each iteration, each process performs the reduction operation on the data received with the corresponding segment in the local array and replaces the partial reduction results in the array. At the end of the $N-1$ iterations, $p_{F(i)}$ holds the reduction results in SEG_i, $0 \le i \le N-1$. Fig. 1 shows the logical ring implementation of reduce-scatter on four processes. As shown in the figure, in the first iteration, $n_{F(0)}$ sends SEG_3 to $n_{F(1)}$; $n_{F(1)}$ sends SEG_0 to $n_{F(2)}$; $n_{F(2)}$ sends SEG_1 to $n_{F(3)}$; and $n_{F(3)}$ sends SEG_2 to $n_{F(0)}$. After the communication, the reduction operation is performed: $n_{F(0)}$ on SEG_2, $n_{F(1)}$ on SEG_3, $n_{F(2)}$ on SEG_0, and $n_{F(0)}$ on SEG_1. After three iterations, $n_{F(0)}$ has SEG_0 results, $n_{F(1)}$ has SEG_1 results, $n_{F(2)}$ has SEG_2 results, and $n_{F(3)}$ has SEG_3 results.

Put it all together, the proposed algorithm first finds a contention-free logical ring on the tree topology. It then performs the all-reduce operation over the contention-free logical ring by using logical ring algorithms to carry out a reduce-scatter

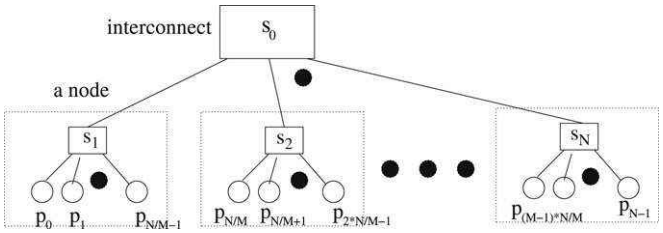


Fig. 2. The two-level tree model for SMP clusters.

operation followed by an all-gather operation. Both the ring based reduce-scatter and all-gather algorithms perform the communications in $N-1$ rounds with each node sending $\lceil \frac{msize}{N} \rceil$ bytes data in each round.

Theorem 1. *Consider an all-reduce operation on X items with the size of each item being itsize ($msize = X \times itsize$). When X is divisible by N, the proposed algorithm is bandwidth optimal.*

Proof. Both the reduce-scatter operation and the all-gather operation are performed in $N-1$ steps with each process sending and receiving $\frac{X \times itsize}{N}$ data in each step. Since there is no contention on the logical ring, both operations take $(N-1) \times T(\frac{X \times itsize}{N})$ time. Hence, the total communication time is $2 \times (N-1) \times T(\frac{X \times itsize}{N})$. Under the linear model, $2 \times (N-1) \times T(\frac{X \times itsize}{N}) = T(2 \times \frac{N-1}{N} \times X \times itsize)$, which is the theoretical optimal (Lemma 5). Hence, our all-reduce algorithm is bandwidth optimal. □

When X is not divisible by N, the total data size sent by each node is $2 \times (N-1) \times \lceil \frac{X}{N} \rceil \times itsize$, which may be much more than the theoretical optimal of $\lceil 2 \times \frac{N-1}{N} \times X \rceil \times itsize$ when N is large. In addition, in order for $2 \times (N-1) \times T(\frac{msize}{N})$ to approximate $T(2 \times \frac{N-1}{N} \times msize)$, $\frac{msize}{N}$ needs to be sufficiently large: if $\frac{msize}{N}$ is close to infinity, the performance of the proposed algorithm will be close to optimal. The exact threshold when the proposed algorithm is better than other algorithms is system dependent.

The proposed algorithm requires additional $\lceil \frac{X}{N} \rceil \times itsize$ working memory, which is better than the $\lceil \frac{X}{2} \rceil \times itsize$ working memory required by the butterfly-liked algorithm. The results in all processors will be the same after the ring-based algorithm. Hence, the algorithm meets the minimum MPI quality requirement [23]. However, the bracketing for computing the results is not the same for different data items.

4.2. Algorithm for high-end clusters

A high-end SMP/Multi-core cluster is typically formed by SMP and/or multi-core nodes connected by a high speed interconnect. The interconnect is usually a single cross-bar switch for small clusters and a fat-tree for large clusters, whose performance is close to a cross-bar switch. The communication between processors in different SMP nodes (inter-node communication) goes through the interconnect while the intra-node communication is performed within an SMP node, typically through memory operations. For the common case when one network interface card is equipped in each node, the system can be approximated by a two-level tree topology as shown in Fig. 2.

The algorithm described in the previous subsection can be applied when the tree topology is determined. In a high-end SMP cluster, even though the system can be approximated with a two-level tree, the exact tree topology is unknown until the SMP nodes are allocated. Hence, this algorithm cannot be directly used in a high-end SMP cluster since we need to develop the all-reduce routine before it can be used in programs running on a particular platform. However, by default, many SMP clusters assign MPI processes with consecutive ranks to processors (or cores) in each SMP node. For example, in a system with 4-core SMP nodes, n_0, n_1, n_2, and n_3 are assigned one SMP node (one process per core); n_4, n_5, n_6, and n_7 are assigned to another SMP node; and so on. For such clusters, using the 2-level tree representation, a contention-free logical ring for the default process assignment scheme can be built without knowing the exact topology as shown in the following lemma, which can be proved by examining the topology in Fig. 2.

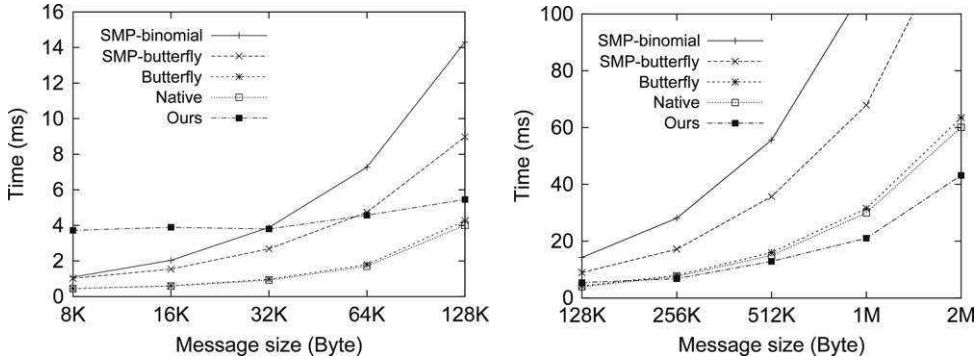
Lemma 6. *Under the assumption that MPI processes with consecutive ranks are assigned to processors (or cores) in each SMP node, logical ring pattern, $p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{N-1} \rightarrow p_0$, is contention free.* □

5. Experiments

Based on the algorithms described in the previous section, we implement all-reduce routines in two forms. For high-end SMP/multi-core clusters, we implement a stand-alone all-reduce routine based on Lemma 6. For clusters with a physical tree topology, we develop a routine generator that takes the tree topology information as input and automatically produces an all-reduce routine that uses the topology specific algorithm for the topology. The routine generator reads a topology description file that describes the connectivity among machines and switches, determines the order of the machines that realizes a contention-free ring, and produces the code that realizes the operations using the contention-free ring communication pattern. The generated routines are written in C and are based on MPICH point-to-point primitives.

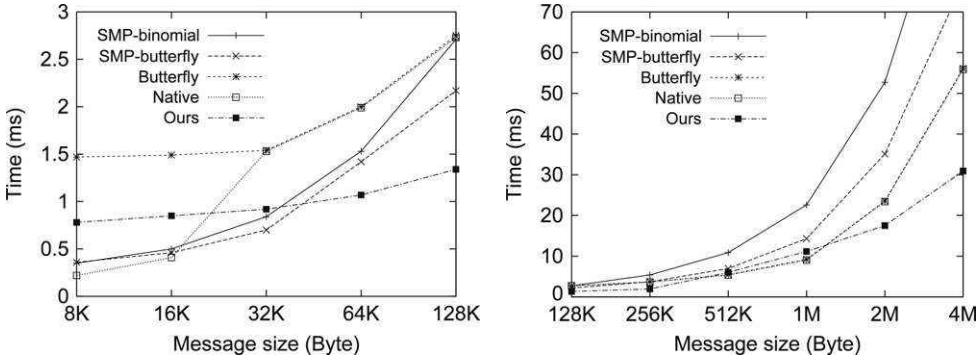
Three clusters are used in the experiments: the NCSA Teragrid IA-64 Linux cluster [19], the Draco cluster at the Department of Computer Science, Florida State University, and an Ethernet switched cluster. The NCSA Teragrid IA-64 Linux cluster is a Myrinet cluster with dual 1.5 GHz Intel Itantium 2 SMP nodes and 4 GB memory per node. The system runs the Linux 2.4.21–SMP operating system and uses the mpich-gm-1.2.6..14b library. The Draco cluster is an InfiniBand cluster with Dell PowerEdge 1950 nodes, each node having two dual-core Xeon E5345 processors (2.33 GHz, 4 cores per processor, 8 cores per node) and 8 GB memory. The nodes are connected with a 20 Gbps double data rate InfiniBand switch. The cluster runs the Linux 2.6.9–42.ELsmp kernel and uses the mvapich2.1.0.2p1 library. The Ethernet switched cluster consists of 32 compute nodes connected by Dell Powerconnect 2724 Gigabit Ethernet switches. The nodes are Dell Dimension 2400, each with a 2.8 GHz P4 processor and 640 MB memory. All nodes run Linux (Fedora) with the 2.6.5–358 kernel. The latest OPENMPI 1.2.5 [20] and MPICH 2.1.0.6p1 [10] are installed on this cluster.

The performance of the algorithms is measured using the Mpptest approach [11]. We compare the proposed algorithms with native MPI implementations on these clusters. In addition, the traditional butterfly-like algorithm [22,23,27], denote as *butterfly*, that is theoretically both latency and bandwidth optimal when the network contention is not a problem is also compared. For high-end SMP/multi-core clusters, we also compare the proposed scheme with two SMP specific implementations. One SMP specific implementation is based on algorithms recently developed for SMP clusters [26]. The implementation, denoted as *SMP-binomial*, has four logical phases: (1) an intra-node reduce operation for each SMP node using a binomial tree, (2) an inter-node reduce operation with a binomial tree to obtain the reduction results in one node, (3) an inter-node broadcast operation with a binomial tree to distribute the results to each SMP node, and (4) an intra-node broadcast to distribute the results to each processor. While this algorithm does not cause network contention, the inter-node



(a) Medium sized data. (b) Large sized data.

Fig. 3. Results on the NCSA Teragrid IA-64 cluster (128 processors, Myrinet).



(a) Medium sized data. (b) Large sized data.

Fig. 4. Results on the Draco cluster (128 cores, InfiniBand (20 Gbps)).

communication is not bandwidth optimal: there exists a node that communicates $O(\log(N) * msize)$ data. We enhance this algorithm by using the butterfly-like algorithm to perform inter-node all-reduce operation. This algorithm is denoted as *SMP-butterfly*. Notice that the main competitor of the proposed algorithm is *butterfly* whose main problem is that the communication pattern cannot be embedded in an SMP cluster without causing link contention. *SMP-butterfly* can be considered as an improvement over *butterfly*: eliminating the network contention by grouping intra-node communications together.

5.1. NCSA Teragrid IA-64 Linux cluster results

Fig. 3 shows the performance of different all-reduce algorithms on the NCSA Teragrid cluster with 128 processors (64 nodes). All programs are compiled with the 'mpicc -lm' command with no other flag (mpicc invokes the Intel compiler in the system). The native MPI library uses the *butterfly* algorithm and as a result, its performance is very close to that of our implementation of *butterfly*. As shown in the figure, *native* (*butterfly*) performs the best when the data size is small. However, as the data size increases, the network contention degrades the performance: the performance of *native* (*butterfly*) is not as good as the proposed algorithm when the data size is larger than 256 KB. *SMP-butterfly* eliminates network contention by performing the operation in three phases. However, using this algorithm, the total amount of inter-node and intra-node data communicated is still larger than what is needed. As a result, using this method to eliminate network contention is not effective in such a high-end cluster as shown in the figure. It is not surprising that *SMP-binomial* performs poorly when the data size is large since there exists one process that communicates $O(\log(N) * msize)$ data in this algorithm. The proposed algorithm outperforms *butterfly* when the data size is larger than 256 KB. Notice that $\frac{256\text{ KB}}{128} = 2$ KB: the threshold value of $\frac{msize}{N}$ for our bandwidth optimal algorithm to be more efficient is around 2 KB on this cluster. As the data size increases, the performance difference is more significant: the network contention problem is more severe as the data size increases. This experiment shows that the break-even point for the proposed ring-based algorithm is 256 KB when $N = 128$. When N is larger, however, the break-even point can be much larger since the number of communication rounds in the ring-based algorithm is $O(N)$.

5.2. Draco cluster results

Fig. 4 shows the results on the Draco cluster. The native MPI library also uses *butterfly* when the data size is larger than 32 KB. In this cluster, the trend for the relative performance of *SMP-binomial*, *SMP-butterfly*, and the proposed algorithm is similar to that in the NCSA cluster. The proposed algorithm performs better than *native* (*butterfly*) significantly for data sizes 64 KB, 128 KB, 256 KB, 2 MB and 4 MB. However, for data sizes 512 KB and 1 MB, the performance of the proposed algorithm is slightly worse. We believe this is mainly due to the interaction between memory references and network contention, which results in the abnormal performance results for the 512 KB and 1 MB data points. Nonetheless, the figure shows the trend of the relative performance of *native* (*butterfly*) and the proposed algorithm: the proposed algorithm is more effective as the data size becomes larger.

5.3. Ethernet switched cluster results

For the Ethernet switched cluster, we show the results on 32-node clusters with the topology shown in Fig. 5. We performed

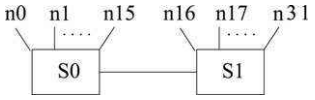


Fig. 5. Ethernet topology used in the experiments.

experiments on other topologies with multiple switches and the trend is similar. Fig. 6 show the results for the Ethernet cluster. On this cluster, the proposed algorithm is more efficient than both Open MPI and MPICH2 when the data size is larger than 256 KB. Hence, the threshold value of $\frac{msize}{N}$ for the proposed algorithm to be more efficient is $\frac{256\text{ KB}}{32} = 8$ KB, which is much larger than the 2 KB in the high-end NCSA cluster. This reflects the fact that on an Ethernet switched cluster with Gigabit switches, the communication start-up overhead is much larger than that in high-end clusters. Thus, even for reasonably large data sizes (e.g. 128 KB), it is still more important to reduce the communication start-up overheads. However, when the data size is larger, network contention and bandwidth efficiency become a problem and our bandwidth optimal algorithm is more efficient.

6. Related work

The all-reduce operation has been extensively studied. The one-item all-reduce operation has been studied under different names such as *census function* [2], *global combine* [3,5,6,27], and *gossip* [15]. The lower bound for the communication time under various communication models has been established [2,3,5,15]. In [15], it is shown that to complete a one-item all-reduce operation under the telephone model, at least $2N - 4$ connections must be established when $N > 4$. In [2,3,5], the lower bounds for the number of rounds of communications and for the number of data item to be communicated in sequence in various postal models are established. However, to the best of our knowledge, the lower bound on the total number of data items to be communicated to complete the operation has not been established.

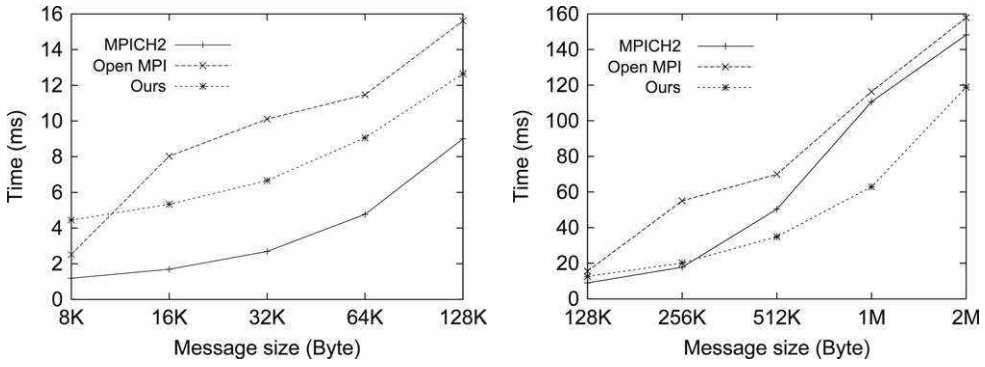
The all-reduce operation is one of the collective operations supported in the MPI standard [18], and thus, all MPI libraries support this operation. Many efficient platform independent algorithms for this operation have been proposed [22,23,25,27]. In [22], Rabenseifner proposed to realize the all-reduce operation by a reduce-scatter operation followed by an all-gather operation and gave various algorithms for the reduce-scatter and all-gather operations. The butterfly-like algorithm has been developed some time ago [22,27] and has been extended to handle non-power-of-two numbers of processes [23]. Various architecture specific all-reduce schemes have also been developed [1,4,12,17,26]. An all-reduce algorithm was designed for BlueGene/L systems in [1]. In [12], an all-reduce scheme that takes advantage of remote DMA (RDMA) capability was developed for VIA-based clusters. The work in [17] investigated an adaptive all-reduce algorithm in an InfiniBand cluster that deals with the situation when not all nodes arrive at the call site at the same time. A study on the all-reduce operation over WAN can be found in [4]. All-reduce algorithms were developed specifically for SMP clusters in [26]. Among all these algorithms, the butterfly-like algorithm [22,23,27] is widely used. The limitation of this algorithm is that it is difficult to realize the butterfly communication pattern without incurring network contention in contemporary SMP/multi-core clusters. Many all-gather [7,24,25] and reduce-scatter [13,24,25] algorithms, which are parts of the proposed all-reduce algorithm, have been developed. The technique for finding contention-free logical ring on tree topologies was developed in [7]. In this paper, we use the existing results in various papers (e.g. [7,22,25]) as the components of our proposed algorithm. Our contribution lies in showing that putting these existing components in the particular way yields a contention-free bandwidth optimal all-reduce algorithm for the tree topology. Like other architecture dependent collective algorithms [8,7,16,21] that work well in some situations, the proposed scheme can be used in advanced communication systems [9,14,28].

7. Conclusions

We investigate efficient implementations of the all-reduce operation with large data sizes under the assumption that the reduce operator is both associative and commutative. We derive a theoretical lower bound on the communication time of this operation and develop a bandwidth optimal all-reduce algorithm on tree topologies. This algorithm only requires tree connectivity to achieve bandwidth optimality and can be applied to contemporary clusters. We demonstrate the effectiveness of the proposed algorithm on various contemporary clusters, including high-end clusters with SMP and/or multi-core nodes connected by high-speed interconnects, and low-end Ethernet switched clusters. While our algorithm is contention-free and bandwidth optimal, it is not optimal in the latency term: the number of communication rounds is proportional to the number of processes.

Acknowledgments

This work is supported in part by National Science Foundation (NSF) grants: CCF-0342540, CCF-0541096, and CCF-0551555. Experiments are also performed on resources sponsored through an NSF Teragrid grant CCF-050010T.



(a) Medium sized message. (b) Large sized message.

Fig. 6. Results for the Ethernet switched cluster (32 single-core nodes).

References

[1] G. Almasi, et al., Optimization of MPI collective communication on BlueGene/L systems, in: International Conference on Supercomputing, ICS, 2005, pp. 253–262.
[2] A. Bar-Noy, S. Kipnis, B. Schieber, Optimal computation of census functions in the postal model, Discrete Applied Mathematics 58 (1995) 213–222.
[3] A. Bar-Noy, J. Bruck, C-T. Ho, S. Kipnis, B. Schieber, Computing global combine operstions in the multiport postal model, IEEE Transactions on Parallel and Distributed Systems 6 (8) (1995) 896–900.
[4] L. Bongo, O. Anshus, J. Bjorndalen, T. Larsen, Extending collective operations with application semantics for improving multi-cluster performance, in: Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar, 2004, pp. 320–327.
[5] J. Bruck, C.-T. Ho, Efficient global combine operations in multi-port message-passing systems, Parallel Processing Letters 3 (4) (1993) 335–346.
[6] J. Bruck, L.D. Coster, N. Dewulf, C.-T. Ho, R. Lauwereins, On the design and implementation of broadcast and global combine operations using the postal model, IEEE Transactions on Parallel and distributed Systems 7 (2) (1996) 256–265.
[7] A. Faraj, P. Patarasuk, X. Yuan, Bandwidth efficient all-to-all broadcast on switched clusters, International Journal of Parallel Programming 36 (4) (2008) 426–453.
[8] A. Faraj, X. Yuan, Pitch Patarasuk, A message scheduling scheme for all-to-all personalized communication on ethernet switched clusters, IEEE Transactions on Parallel and Distributed Systems 18 (2) (2007) 264–276.
[9] A. Faraj, P. Patarasuk, X. Yuan, A study of process arrival patterns for MPI collective operations, International Journal of Parallel Programming 36 (6) (2008) 543–570.
[10] W. Gropp, E.L. Lusk, N.E. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing 22 (6) (1996) 789–828.
[11] W. Gropp, E.L. Lusk, Reproducible measurements of MPI performance characteristics, in: Proceedings of PVMMPI, 1999, pp. 11–18.
[12] R. Gupta, P. Balaji, D.K. Panda, J. Nieplocha, Efficient collective operations using remote memory operations on VIA-based clusters, in: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IPDPS, April 2003, p. 46.
[13] G. Iannello, Efficient algorithms for the reduce-scatter operation in LogGP, IEEE Transactions on Parallel and Distributed Systems 8 (9) (1997) 970–982.
[14] A. Karwande, X. Yuan, D.K. Lowenthal, An MPI prototype for compiled communication on ethernet switched clusters, Journal of Parallel and Distributed Computing 65 (10) (2005) 1123–1133.
[15] W. Knodel, New gossips and telephones, Discrete Mathematics 3 (1) (1975) 95.
[16] R.G. Lane, S. Daniels, X. Yuan, An empirical study of reliable multicast protocols over ethernet-connected networks, Performance Evaluation Journal 64 (3) (2007) 210–228.
[17] A. Mamidala, J. Liu, D. Panda, Efficient barrier and allreduce on infiniband clusters using hardware multicast and adaptive algorithms, in: Proceedings of the 2006 IEEE International Conference on Cluster Computing, 2004, pp. 135–144.
[18] 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.
[19] NCSA Teragrid IA-64 Linux Cluster. http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/TGIA64LinuxCluster.
[20] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/.
[21] P. Patarasuk, X. Yuan, A. Faraj, Techniques for pipelined broadcast on ethernet switched clusters, Journal of Parallel and Distributed Computing 68 (6) (2008) 809–824.
[22] R. Rabenseifner, Optimization of collective reduction operations, in: International Conference on Computational Science, in: LNCS, vol. 3036, 2004, pp. 1–9.
[23] R. Rabenseifner, J.L. Traff, More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems, in: EuroPVM/MPI, in: LNCS, vol. 3241, 2004, pp. 36–46.
[24] W.B. Tan, P. Strazdins, The analysis and optimization of collective communications on a beowulf cluster, in: Proc. of the Ninth International Conference on Parallel and Distributed Systems, ICPADS'02, 2002, p. 659.
[25] R. Thakur, R. Rabenseifner, W. Gropp, Optimizing of collective communication operations in MPICH, International Journal of High Performance Computing Applications 19 (1) (2005) 49–66.
[26] V. Tipparaju, J. Nieplocha, D. Panda, Fast collective operation using shared and remote memory access protocols on clusters, in: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IPDPS, 2003, p. 84.
[27] R. van de Geijn, On global combine operations, Journal of Parallel and Distributed Computing 22 (2) (1994) 324–328.
[28] X. Yuan, R. Melhem, R. Gupta, Algorithms for supporting compiled communication, IEEE Transactions on Parallel and Distributed Systems 14 (2) (2003) 107–118.

Pitch Patarasuk received the B.S. degree in civil engineering from Chulalongkorn University, Thailand, in 1999, and the M.S. degree in computer science from Florida State University in 2004. He is currently a Ph.D. student in the Computer Science Department at Florida State University. His research interests include distributed systems, cluster computing, and communication optimizations.

Xin Yuan received his B.S. and M.S. degrees in Computer Science from Shanghai Jiaotong University in 1989 and 1992, respectively. He obtained his Ph.D. degree in Computer Science from the University of Pittsburgh in 1998. He is currently an associate professor at the Department of Computer Science, Florida State University. His research interests include parallel and distributed systems, compilers, and networking. His email address is: xyuan@cs.fsu.edu.