On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model

Jehoshua Bruck, *Senior Member, IEEE*, Luc De Coster, Natalie Dewulf, Ching-Tien Ho, *Member, IEEE Computer Society*, and Rudy Lauwereins

Abstract—There are a number of models that were proposed in recent years for message passing parallel systems. Examples are the postal model and its generalization the LogP model. In the postal model a parameter λ is used to model the communication latency of the message-passing system. Each node during each round can send a fixed-size message and, simultaneously, receive a message of the same size. Furthermore, a message sent out during round r will incur a latency of λ and will arrive at the receiving node at round $r + \lambda - 1$.

Our goal in this paper is to bridge the gap between the theoretical modeling and the practical implementation. In particular, we investigate a number of practical issues related to the design and implementation of two collective communication operations, namely, the broadcast operation and the global combine operation. Those practical issues include, for example, 1) techniques for measurement of the value of λ on a given machine, 2) creating efficient broadcast algorithms that get the latency λ and the number of nodes n as parameters and 3) creating efficient global combine algorithms for parallel machines with λ which is not an integer. We propose solutions that address those practical issues and present results of an experimental study of the new algorithms on the Intel Delta machine. Our main conclusion is that the postal model can help in performance prediction and tuning, for example, a properly tuned broadcast improves the known implementation by more than 20%.

Index Terms—Broadcast, global combine, postal model, complete graph, collective communication.

1 Introduction

This paper explores various theoretical and practical issues in designing and implementing efficient algorithms for broadcast and global combine operations for message passing parallel systems using the postal model. The *postal model* was introduced by Bar-Noy and Kipnis [7]. In this model, the system consists of n nodes (processors with local memory), each node can simultaneously use a single input port and a single output port and the communication latency is λ. Namely, each node during each communication *round* can send a fixed-size message and, simultaneously, receive a message of the same size. Furthermore, a message sent out during round r will incur a latency of λ and will arrive at the receiving node at round $r + \lambda - 1$. More specifically, if the elapse time of the sender is t_0, then the elapse time of the receiver, starting from the time the sender issues the send command until the time the receiver receives the message, is λt_0. Note that the frequently used one-port model [23] is a special case of the postal model in which $\lambda = 1$. The recently proposed LogP model [15] is a generalization of the postal model.

- *J. Bruck is with the California Institute of Technology, Mail Code 116-81, Pasadena, CA 91125. E-mail: bruck@systems.caltech.edu.*
- *L. De Coster, N. Dewulf, and R. Lauwereins are with K.U. Leuven–ESAT, Kard. Mercierlaan 94, B-3001 Heverlee, Belgium. E-mail: {decoster, lauwereins}@esat.kuleuven.ac.be.*
- *C.-T. Ho is with the IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. E-mail: ho@almaden.ibm.com.*

Manuscript received Dec. 12, 1994; revised Apr. 1, 1995.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number D95088.

Both the broadcast and global combine operations are frequently used in many applications for message-passing systems (see [18]). Several collective communication libraries, such as Express [17] by Parasoft and the External User Interface (EUI) [1], [2] of the Scalable POWERparallel System (SP-1) by IBM, provide primitives for broadcast and global combine. These operations have also been included as part of the collective communication routines in the Message-Passing Interface (MPI) standard proposal [25].

Regarding the topology of the message-passing system, as in the postal model we assume that the communication between any two nodes has the same characteristics. Namely, we assume that the communication network can be modeled as a complete graph. However, we also address the channel contention issue in Section 2.6 and show that a congestion-free algorithm, taken into account a mesh topology with wormhole and XY routing, performs better than a general algorithm assuming a complete graph by about 10% only. This complete-graph approach for modeling the network has several advantages: 1) it allows designing of portable algorithms and 2) it gives simple and, yet, reasonable communication model for multistage interconnection networks and many high-bandwidth point-to-point networks (such as the hypercube) with virtual cut-through-like routing algorithms. In fact, this model has been widely used by several researchers (see [21]) and has been adopted by numerous communication libraries, such as Express, PARMACS [22], PICL [20], Zipcode [27], Venus [3], and CCL [2].

We assume that the n nodes are labeled from 0 through $n-1$. In the broadcast operation, node 0 initially holds a data item, denoted d, and the goal is to make this data item also known to all other $n-1$ nodes. Many efficient algorithms have been known for the broadcast operation for various communication models and network topologies (see, for example, [5], [16], [19], [21], [23], [26], [28]). For the fully-connected model, efficient broadcast algorithms for the one-port model were given in [9], [12] and for the postal model were given in [7], [8]. Theoretically, optimal broadcast algorithm of one data item in the postal model can be easily derived by constructing a spanning tree based on a simple greedy algorithm [7]. However, in implementing the optimal broadcast algorithm on a real machine, we came across many interesting and practical issues that need to be resolved in order to have an efficient and feasible implementation. For example, how do we measure the value of λ on a given machine? Given the latency λ and the number of nodes n, how do we construct the algorithm efficiently in time and in space? In Section 2, we propose solutions that address those practical issues and present results of an experimental study of the new algorithms on the Intel Delta machine. Our main conclusion is that the postal model can help in performance prediction and tuning, for example, a properly tuned broadcast improves the known implementation by more than 20%.

In the global combine operation, each node i initially holds a data item d_i. Given a commutative and associative combining operator "$\oplus$," the goal is to compute

$$D = d_0 \oplus d_1 \oplus \cdots \oplus d_{n-1}$$

and to place the result D in all the n nodes. Examples of such combining operators are MAX, MIN, (logical or bitwise) AND, OR, XOR, integer addition, and integer multiplication. Many efficient algorithms have been known for the global combine operation for various communication models and network topologies (see, for example, [5], [29]). For the fully-connected model, efficient global combine algorithms for the one-port or multi-port model were given in [10], [13] and for the postal model were given in [11], [6], [24]. For the global combine algorithms in the postal model [11], [6], it is assumed that λ is an integer. In a real machine, λ typically is not an integer. A natural solution is to conceptually force each receive operation to idle for a period of $\lceil\lambda\rceil - \lambda$ time step, in order to synchronize with the send operation. In Section 3 we show that, depending on the value of λ relative to $\lceil\lambda\rceil$, it may be advantageous to force each send operation to idle for a period of $\lambda / \lfloor\lambda\rfloor - 1$ time step, in order to synchronize with the receive operation. The formula for the break-even points of λ versus $\lfloor\lambda\rfloor$ is explicitly derived.

2 Efficient Broadcast in the Postal Model

2.1 Preliminaries

We first review a few known broadcast algorithms. In the one-port model and when the data item to be broadcast is one, the naive algorithm based on recursive splitting is in fact an optimal one. Assume that S is the set of nodes involved in the broadcast operation, $|S| = n$, and the source node $s \in S$. Specifically, one can partition S into two subsets, denoted S' and S'', of sizes $\lceil n/2\rceil$ and $\lfloor n/2\rfloor$, respectively. Assume, without loss of generality, that $s \in S'$. Pick any node s' in S'' as the leader. Then, in one step node s can send the data item to node s', and the original broadcast problem is reduced to two broadcast subproblems of sizes $\lceil n/2\rceil$ and $\lfloor n/2\rfloor$, respectively, which can be solved concurrently and recursively. Clearly, such broadcast algorithm finishes in $\lceil \log_2 n\rceil$ steps. Note that when n is a power of two the derived spanning tree is a binomial tree, which corresponds to the well-known recursive-doubling broadcast algorithm on a hypercube [23]. We refer to such broadcast algorithm as the *binomial-tree broadcast*. Fig. 1a shows an example of the binomial-tree broadcast for $n = 8$ in the one-port model. Fig. 1b shows the same binomial-tree broadcast but in the postal model with $\lambda = 2$. In the figure, the labels on the edges represent the time step during which the message is sent, while the labels on the nodes represent the time step during which the message is received.

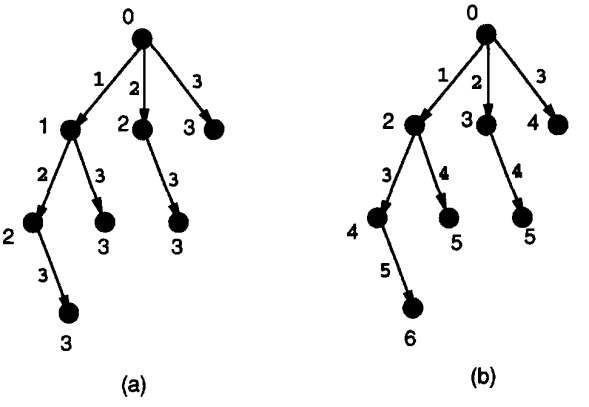


Fig. 1. The binomial-tree broadcast for $n = 8$ in the one-port model (a), and in the postal model with $\lambda = 2$ (b).

In the postal model, an optimal spanning tree can be easily constructed based on the following recursion [7]. Let $N_\lambda(t)$ be the maximum number of nodes that can be reached (including the source node) in time t in the postal model. Then,

$$N_\lambda(t) = \begin{cases} N_\lambda(t-1) + N_\lambda(t-\lambda), & \text{if } t \geq \lambda, \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

Note that when $\lambda = 1$, $N_\lambda(t) = 2^t$. When $\lambda = 2$, $N_\lambda(*) = 1, 1, 2, 3, 5, 8, 13, \cdots$, etc., is the Fibonacci sequence. For convenience, define the inverse function

$$T_\lambda(n) = \min_t \{N_\lambda(t) \geq n\}.$$

Clearly, broadcast within an n-node set can be finished in time $T_\lambda(n)$ in the postal model. Fig. 2 shows an example of the optimal spanning tree for $\lambda = 2$ and $n = 8$ in the postal model. We refer to such broadcast algorithm as the *λ-tree broadcast*, to be described in details in Section 2.4. From the figure, the λ-tree broadcast finishes in five time steps as compared to six time steps required by the binomial-tree broadcast in Fig. 1b.

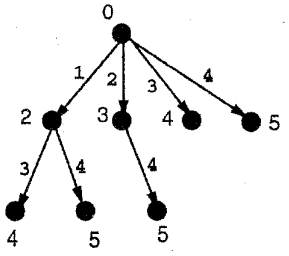


Fig. 2. The λ -tree broadcast for $n = 8$ and $\lambda = 2$.

2.2 Measurement of the Latency λ

In this subsection, we discuss issues related to the measurement of λ. In order to measure the parameter λ on a given machine, we use two simple experiments described below. (The main reason for choosing two separate experiments is to increase the confidence of the implementation correctness by double checking on the derived λ values.) For convenience, denote a source node by P_0 and denote k randomly chosen distinct nodes (different from the source node) by $P_1, P_2, \cdots, P_k$.

In the first experiment, P_0 sends successively a message of a given size to $P_1, P_2, \cdots, P_k$. When P_k receives the message, it sends back a message of the same size to P_0. The total communication time observed at P_0 can be modeled, based on the postal model, as $T = t_0(k - 1 + 2\lambda)$, where t_0 is the time for the sender to send out a message of the given size. By measuring the total times T for different values of k, the parameters λ and t_0 can be extracted.

In the second experiment, P_0 sends successively a message of a given size to $P_1, P_2, \cdots, P_k$, as before. Then, as soon as P_k receives the message from P_0, it sends successively a message of the same size to $P_{k-1}, P_{k-2}, \cdots, P_0$. The total communication time observed at P_0 can be modeled as

$$T = 2t_0(k - 1 + \lambda).$$

As in the first experiment, the parameters λ and t_0 can be extracted using the total times T for different values of k.

From our experiments, we learn that the parameter 1 on the Delta system mainly depends on the message size. Fig. 3 shows the extracted values of λ, as a function of the message size, on the Delta using the two experiments. There are essentially two kinds of behavior. For message sizes smaller than 512 bytes, which is the packet size on the Delta,[1] λ increases as a function of the message size and reaches a maximum of approximately 1.8 at 512 bytes. For message sizes larger than 512 bytes λ decreases hyperbolically to 1.

The explanation for this behavior follows from Fig. 4. In this figure, the upper curve shows the times observed at the receiver side (λt_0), while the lower curve shows that at the sender side (t_0), both as a function of message sizes. For message sizes smaller than 512 bytes the receive-time increases faster than the send-time, so the ratio of both times (which is λ) also increases. On the other hand, for message sizes larger than 512 bytes both curves are increasing along the message sizes, but the difference between them remains the same; thus, λ decreases to 1 hyperbolically.

1. Actually, 32 bytes header information will be added to the user message before passing to the underlying communication subsystem.

There are other less critical factors affecting the value of λ, such as whether *blocking sends* or *nonblocking sends* are used and the congestion behavior which will be described later. Note that a *blocking send* returns only after the message has been copied out of the user's buffer, while a *nonblocking send* returns immediately possibly before the message being copied out of the user's buffer. Fig. 5 shows the measured λ on the Delta for nonblocking sends (called **isend** on the Delta) compared to blocking sends (called **csend** on the Delta). For message sizes larger than 512 bytes, using nonblocking sends yields larger λ's because of the possible overlap between successive communications at the sender. On the other hand, for message sizes smaller than 512 bytes, the measured λ's using nonblocking sends are larger, possibly because of the additional overhead in creating the message identifier structure etc. at the sender. Since we will focus on broadcasting a message of size 512 bytes in the following (to explore the extreme behavior of the postal model on the Delta), all sends in subsequent algorithms are implemented as blocking sends.

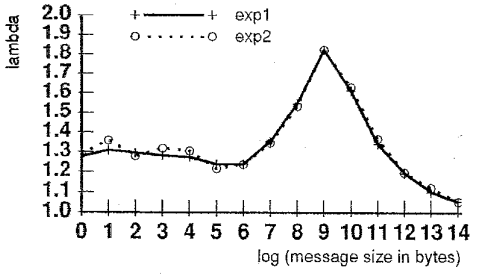


Fig. 3. The measured values of λ on the Delta as a function of message sizes.

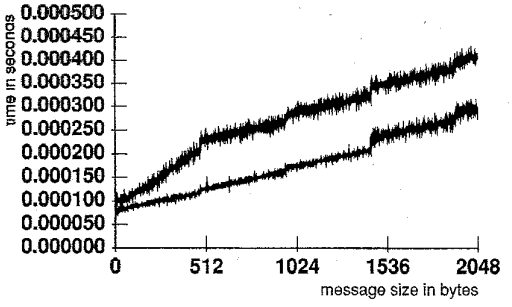


Fig. 4. The measured times observed at the receiver (upper curve) and the sender (lower curve) as a function of message sizes.

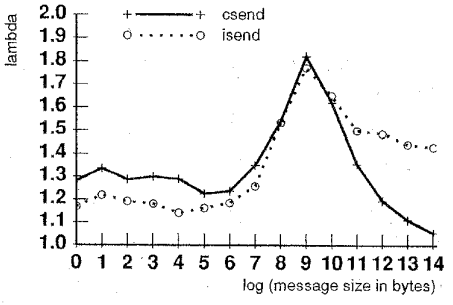


Fig. 5. The measured values of λ on the Delta for blocking sends and nonblocking sends.

2.3 Explicit Construction of the λ-Tree

Now that we have measured the value of λ for a given message size on a given machine, we like to find an efficient implementation of the λ-tree broadcast algorithm. One possible approach is to construct the λ tree explicitly either on the fly or in advance. We now consider the time complexity of constructing the λ tree, given n and λ.

First, the λ tree can be derived using a simple top-down greedy algorithm of time complexity $O(n)$. The algorithm maintains two queues: an *old queue* for existing nodes and a *new queue* for newly spanned nodes. Each entry of the queue contains, among others, a node id and a time stamp at which time the node is ready to send the next message. Initially, there is only one entry with node id 0 (the root) and time stamp 0 at the new queue; the old queue is empty; and the current tree size is $s = 1$. The following iteration is repeated until the current tree size s reaches n.

- Dequeue the entry, say with node id i, which has a smaller time stamp, say t, between the first entries of the two queues.
- Enqueue a new entry with node id s and time stamp $t + \lambda$ into the new queue.
- Increment the current tree size s by 1.
- Update the existing entry (with node id i) with a new time stamp $t + 1$ and enqueue it into the old queue.

The iteration corresponds to that node i sends a message to node s (with the value of s before increment) at time t; node i will be ready to send the next message again at time $t + 1$ while node s will receive the message and ready to propagate it at time $t + \lambda$. It is easy to see that the entries in each queue remain sorted according to the time stamp in the ascendingly order. Clearly, the algorithm is of time complexity $O(n)$. Note that if such an algorithm is executed in advance and a look-up table is used in run time, an order of $O(n)$ in space is needed for the look-up table for a given λ.

When λ is an integer constant, it is possible to construct the λ tree in $O(\log n)$ time as follows. The construction follows the recursion in (1) in constructing λ trees of sizes $N_\lambda(1)$, $N_\lambda(2)$, $\cdots$, etc., in a bottom-up manner using previously constructed smaller λ trees. For instance, if $\lambda = 2$ and $n = 16$, we will construct λ trees of sizes $N_\lambda(*) = 1, 2, 3, 5, 8$, 13 and stop at size 21. Since the size of the tree grows exponentially along t, it takes $O(\log n)$ time steps. When λ is not an integer but a rational number, it is still possible to construct the λ tree in time $O(f \log n)$ where $u = \text{lcm}(1, \lambda)$ is the least common multiple of 1 and λ, and $f = u/\lambda$ is a scaling factor. (See detailed discussion in Section 2.5 later.) Note that f can be very large, depending on λ.

2.4 Efficient Implementation of λ-Tree Broadcast

All the approaches described in the previous subsection are not practical for efficient implementation. This is because for each broadcast of a different message size (which may implies a different value of λ), either the λ tree of size n has to be constructed on the fly (which is of time $O(n)$ or $O(f \log n)$) or a look-up table of size $O(n)$ has to be stored in advance with respect to the given λ. Even with a given λ, different values of n will imply different λ trees. We now describe a simple and efficient implementation of the λ tree broadcast without explicitly constructing the λ tree.

Let **bcast** (S, n, s, m, α) be an algorithm which broadcasts a message m in an n-node set S from a source node $s \in S$. In practice, the λ-tree broadcast is no different from the binomial-tree broadcast if both algorithms are written in the recursive divide-and-conquer manner as follows.

- If $n = 1$, then **bcast** returns.
- Otherwise, we perform the following steps.
  1) Partition S into two subsets S' and S'' of sizes $n' = \min(round(\alpha n), n - 1)$ and $n'' = n - n'$, respectively, such that $s \in S'$. (Note that *round* is the round-off function.)
  2) Select a leader $s' \in S''$.
  3) Send the message m from s to s'.
  4) Perform **bcast** (S', n', s, m, α) and **bcast** $(S'', n'', s', m, \alpha)$ concurrently and recursively.

Clearly, one likes to choose α from the range $0.5 \le \alpha < 1$. In the binomial-tree broadcast, α is set to 0.5. However, in the λ-tree broadcast, α is chosen as a function of λ and possibly as a function of n. For a given λ and n, there is a range of optimal α which can be applied to the partitioning in step 1 above such that a λ tree can be derived. There are mainly two reasons for having a range of α, as opposed to having a fixed value of α.

First, there is a degree of freedom regarding the optimal partitioning for many values of n. Recall the definitions of $N_\lambda(t)$ and $T_\lambda(n)$ from Section 2.1 and assume that $T_\lambda(n) = t$. From (1), we have the recursion of $N_\lambda(t) = N_\lambda(t - 1) + N_\lambda(t - \lambda)$. Thus, any pair of (n', n'') such that $n' + n'' = n$, $n' \le N_\lambda(t - 1)$ and $n'' \le N_\lambda(t - \lambda)$ defines an optimal partitioning. More specifically, assume $N_\lambda(t) - n = \Delta \ge 0$. That is, Δ more nodes can be added into a λ tree of n nodes without increasing the overall broadcast time. In this case, any pair of (n', n'') in the set

$$\{(N_\lambda(t-1) - \Delta,\ N_\lambda(t-\lambda)),\ (N_\lambda(t-1) - \Delta + 1, N_\lambda(t-\lambda) - 1),\ \cdots,\ (N_\lambda(t-1),\ N_\lambda(t-\lambda) - \Delta)\}$$

defines an optimal partitioning and the cardinality of this set is $\Delta + 1$. For example, if $\lambda = 2$ and $n = 13$, then $\Delta = 0$ and $(n', n'') = (8, 5)$ is the only optimal partitioning. As another example, if $\lambda = 2$ and $n = 14$, then $\Delta = 21 - 14 = 7$ and any (n', n'') in $\{(6, 8), (7, 7), \cdots, (13, 1)\}$ is an optimal partitioning.

The second reason for having a range of α is that, the rounding of αn gives additional flexibility in the choice of α. For instance, consider the earlier case where $(n', n'') = (8, 5)$ is the only optimal partitioning of $n = 13$. Any values of α satisfying $round\ (13\alpha) = 8$, i.e., $\frac{7.5}{13} \le \alpha < \frac{8.5}{13}$, can be used to generate an optimal partitioning.

Up until now, we assume the optimal range of α, for a given λ, is further dependent on the value n. If the intersection of the optimal α-ranges belonging to each n is not an empty range, then it is possible to use a fixed real number α throughout the λ-tree broadcast algorithm. Fig. 6 shows the optimal α-ranges for n from 2 up to 250 and with $\lambda = 2$. It is possible to draw a straight line in-between the left and the right curve of the plot. In this case, a fixed value of α can be used in the λ-tree broadcast algorithm for up to $n = 250$ (and, in fact, this can be shown to hold for any n with

$\lambda = 2$). Fig. 7 gives another example of optimal α-ranges for $\lambda = 1.95$, an arbitrary chosen value. As can be seen from the figure, there is no longer a fixed α that can be applied to all values of n shown in the Fig. 8 shows the convergence behavior of the optimal α-ranges for a much larger granularity on the number of nodes (along the Y axis). In this figure, each data point represents a triangle top. In this case, one can use a look-up table. Note, however, that one does not need to store α for all values of n. Instead, only the tops of the triangles on the figure give rise to constraints, so the size of the look-up table is very small in practice. For instance, Fig. 9 shows the required look-up table sizes as a function of n and with $\lambda = 1.293$, an arbitrary chosen value.

2.5 Theory for the Legal Range of α

In this subsection, we formalize the discussion regarding the legal range of α as a function of n and λ. Note that λ is not necessarily an integer, but is assumed to be a rational number. We will refer to the greatest common divider (gcd) of 1 and λ as the largest rational number which divides both 1 and λ. Also, we will refer to the least common multiple (lcm) of 1 and λ as the smallest integer which is a multiple of λ. For instance, if $\lambda = 1.8$ then $\gcd(1, \lambda) = 0.2$ and $\mathrm{lcm}(1, \lambda) = 9$. In this subsection, we use $\lfloor n \rfloor$ to denote $\max_{n'}\{T_\lambda(n') < T_\lambda(n)\}$. For convenience, let $T_\lambda(n) = t$. For a given n and λ, we like to derive the minimum and maximum sizes of the subset S', denoted n_{min} and n_{max} respectively, for a λ-tree broadcast. There are two cases. In the first case $T_\lambda(n+1) > T_\lambda(n)$. In this case, $n = N_\lambda(t)$ for some t and $n_{min} = n_{max} = N_\lambda(t-1)$. As an example, if $n = 13$ and $\lambda = 2$, then $n_{min} = n_{max} = 8$ and $(|S'|, |S''|) = (8, 5)$ is the only optimal partitioning. In the second case $T_\lambda(n+1) = T_\lambda(n)$. In this case, there is a degree of freedom. It is easy to derive that $n_{max} = N_\lambda(t-1)$ (when the subset S' is fully loaded) and $n_{min} = n - N_\lambda(t-\lambda)$ (when the subset S'' is fully loaded). This explains the shape of the Christmas tree in Figs. 6 and 7. Specifically, each triangle in the Christmas tree is defined by the following three (x, y) coordinates, ignoring the round-off effect:

$$\text{top} = \left(\frac{N_\lambda(t-1)}{N_\lambda(t)}, N_\lambda(t)\right),$$

$$\text{left bottom} = \left(\frac{\lfloor N_\lambda(t)\rfloor - N_\lambda(t-\lambda)+1}{\lfloor N_\lambda(t)\rfloor + 1}, \lfloor N_\lambda(t)\rfloor + 1\right),$$

$$\text{right bottom} = \left(\frac{N_\lambda(t-1)}{\lfloor N_\lambda(t)\rfloor + 1}, \lfloor N_\lambda(t)\rfloor + 1\right).$$

Note that the triangle degenerates to one point if $\lfloor N_\lambda(t)\rfloor + 1 = N_\lambda(t)$.

Let $u = \mathrm{lcm}(1, \lambda)$ and let $f = u/\lambda$. Clearly, f is an integer. For example, if $\lambda = 1.8$, then $u = \mathrm{lcm}(1, 1.8) = 9$ and $f = u/\lambda = 9/1.8 = 5$. The function $T_\lambda(n)$ can only increase at times which are a linear combination of 1 and λ, because the path to the last node is a combination of send-time and receive-time. So the function $T_\lambda(n)$ can only change at multiples of $1/f$, which is the greatest common divider of 1 and λ. This gives rise to a scaling of (2) with a factor f. Define $t' = t * f$ and $N_\lambda(t) = N'(t')$, then,

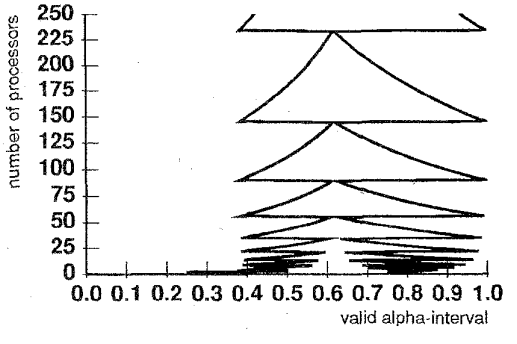


Fig. 6. The optimal α-ranges for $2 \le n \le 250$ and $\lambda = 2$.

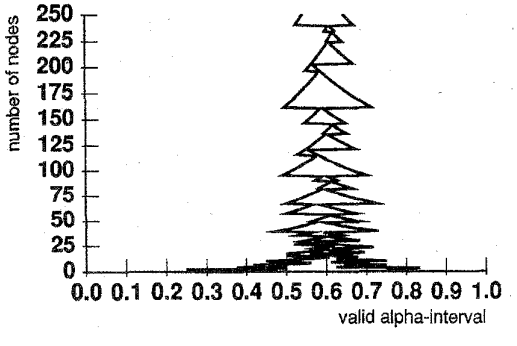


Fig. 7. The optimal α-ranges for $2 \le n \le 250$ and $\lambda = 1.95$.

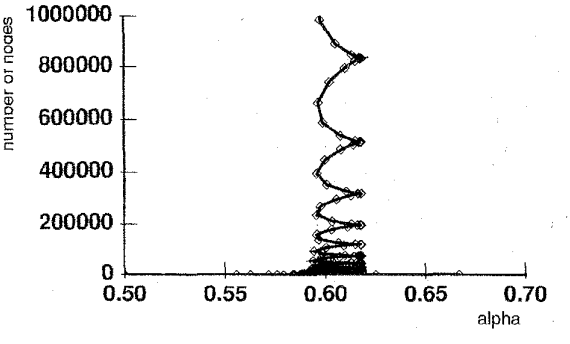


Fig. 8. The convergence of α for $\lambda = 1.95$.

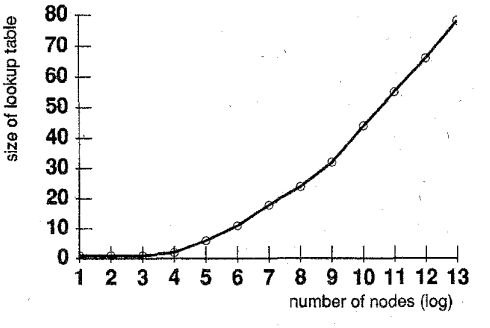


Fig. 9. The observed look-up table sizes as a function of n and with $\lambda = 1.293$.

$$N'(t') = \begin{cases} N'(t'-f) + N'(t'-\lambda * f), & \text{if } t' \geq u, \\ 1, & \text{otherwise.} \end{cases} \quad (2)$$

Note that f and $\lambda * f$ are integers, so $N'(t')$ only changes at integer times t', which allows the function to be derived in a bottom-up manner. For example, for the same example $\lambda = 1.8$, we have $N_\lambda(t) = N'(5t)$ and $N'(t')=N'(t'-5) + N'(t'-9)$ for $t' \geq 9$.

From the recursion in (2), the asymptotic position of the triangles can be derived, through some exercise, as:

$$\text{top} = (\beta^f, N'(t')),$$

$$\text{left bottom} = (1-\beta^{u-1}, N'(t'-1)+1),$$

$$\text{right bottom} = (\beta^{u-1}, N'(t'-1)+1),$$

where

$$\beta = \lim_{t'\to\infty} \frac{N'(t')}{N'(t'+1)}.$$

When λ is an integer, the asymptotic position of the triangle is defined by

$$\text{top} = (\beta, N(t)),$$

$$\text{left bottom} = (1-\beta^{\lambda-1}, N_\lambda(t'-1)+1),$$

$$\text{right bottom} = (\beta^{\lambda-1}, N_\lambda(t-1)+1),$$

where

$$\beta = \lim_{t\to\infty} \frac{N_\lambda(t)}{N_\lambda(t+1)}.$$

2.6 Broadcast Experiments on the Delta

After measuring the latency parameter λ as a function of the message size and searching for a fitting α according to this λ, the question arises if we can decrease the total broadcast time by implementing a λ tree broadcast. For simplicity, we use only one fixed α throughout the λ -tree broadcast knowing that this is only a nearly optimal solution. The algorithm can be improved by using a look-up table, which gives α as a function of n, but then also the algorithm becomes more complicated.

Figs. 10 and 11 show the measured broadcast time, in an 8×8 mesh, as a function of α for message sizes of 16 Kbytes and 512 bytes, respectively. In both experiments, we randomly permute the 64 nodes in an 8×8 mesh to generate the ordered list of pids and pick the first node as the source node. For the 16 Kbyte messages, the minimum time occurs around $\alpha = 0.5$ as expected, because λ is measured about 1.05 at 16 Kbytes (see Fig. 3). In this case, the α-broadcast algorithm degenerates to the binomial broadcast algorithm. On the other hand, the minimum time for the 512-byte messages occurs around $\alpha = 0.6$.

Fig. 12 shows the expected behavior of the broadcast time for a 512 byte message as a function of α. The time is normalized to time steps where one step is the time observed by the sender to send a 512 byte message. As can be seen, the expected minimum broadcast time occurs when α is in the range of 0.56 to 0.59. The measured time is given in Fig. 13 and it mostly agrees with the predicted one. The measured minimum time occurs at $\alpha = 0.58$, which improves the measured binomial-tree broadcast (i.e., with $\alpha = 0.5$) by about 21%. Note that all measured times for $0.56 \leq \alpha \leq 0.66$ (with an increment of 0.05) are within 6% of the minimum time (which occurs at $\alpha = 0.58$).

So far, we have designed the binomial-tree and λ-tree

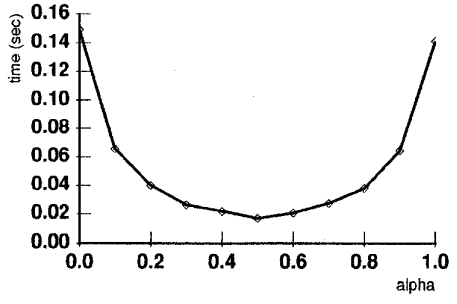


Fig. 10. The measured broadcast time as a function of α for a 16 Kbyte message in an 8×8 submesh on the Delta.

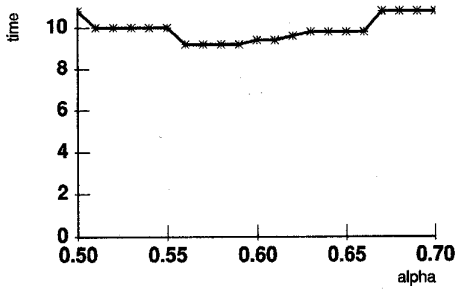


Fig. 12. Predicted time to broadcast a 512 byte message as a function of α.

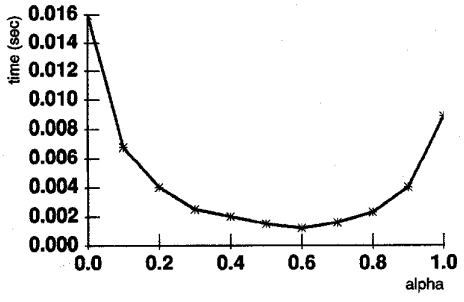


Fig. 11. The measured broadcast time as a function of α for a 512 byte message in an 8×8 submesh on the Delta.

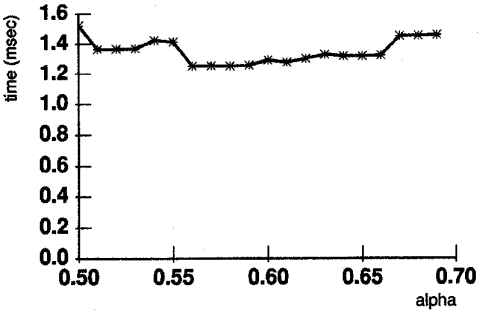


Fig. 13. Measured time to broadcast a 512 byte message as a function of α.

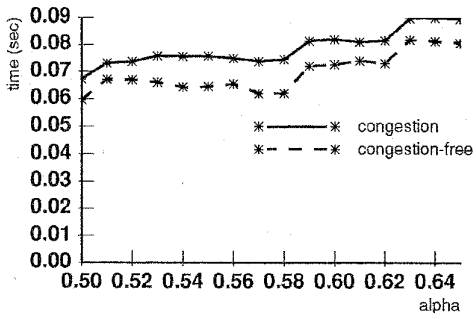


Fig. 14. Comparison of congestion vs. congestion-free algorithms for broadcasting a 64 Kbyte message.

broadcasts assuming a complete-graph model. However, the implementation is done on the Delta system which has a mesh topology. Naturally, there may be congestion in running the broadcast algorithms on the mesh (as we also purposely permute the list of pids). To measure the effect of congestion, we note that if we order the list of pids in the row-major order starting from the source node in a cyclic manner, then it can be shown that the λ -tree broadcast is congestion-free [26]. Thus, we also implemented the congestion-free λ-tree broadcast. The result is shown in Fig. 14 where the message size is 64 Kbytes. Note that by implementing the congestion-free algorithm a reduction of about 10% is obtained.

3 Efficient Global Combine in the Postal Model

3.1 Preliminaries

In the global combine operation, each node i initially holds a data item d_i. Given a commutative and associative combining operator "$\oplus$," the goal is to compute

$$D = d_0 \oplus d_1 \oplus \cdots \oplus d_{n-1}$$

and to place the result D in all the n nodes. There are a number of optimal algorithms for computing the global combine operations assuming the postal model for the message passing system [6], [11], [13]. Those algorithms have the following properties: 1) the parameter λ is typically assumed to be an integer, 2) each node needs to both send and receive messages simultaneously, and 3) the message sent from a node typically depends on the message received during the previous step.

However, on a real machine the measured value of λ is typically not an integer (e.g., see Section 2). The question is what is the practical approach in implementing a global combine operation on a machine with λ not an integer? Our main contribution in this section is providing a solution to this question.

To get a better understanding of the issue, we first give a brief description of the global combine algorithm for the postal model given in [11] where λ is assumed to be an integer. For convenience, the index on node id and the subscripts on S and d are assumed to be modulo n for the rest of this subsection. Recall the recursion of $N_\lambda(t)$ in (1). Suppose that $n = N_\lambda(t)$, for some $t \geq \lambda$. The following scheme computes any associative and commutative combine function on n inputs in t rounds. Each node i initializes a local variable S_i to the value of its piece of data d_i. In round r, for $1 \leq r \leq t-\lambda+1$, each node i sends the value of S_i to node $i + N_\lambda(r+\lambda-2)$. Consequently, in round r, for $\lambda \leq r \leq t$, node i receives the value of $S_{i-N_\lambda(r-1)}$ that was sent at round $r-\lambda+1$ by node $i - N_\lambda(r-1)$. Then, node i computes $S_{i-N_\lambda(r-1)} \oplus S_i$ and places the result in its variable S_i. One may verify by induction that after round r, each node i holds the reduction result of

$$d_{i-N_\lambda(r)+1} \oplus d_{i-N_\lambda(r)+2} \oplus \ldots \oplus d_i.$$

For an arbitrary value of n, assuming $N_\lambda(t-1) < n \leq N_\lambda(t)$, one can introduce a sequence of deficiency parameters $\varepsilon = (\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_{t-\lambda})$, in which $\varepsilon_j \in \{0, 1\}$, for the new recursion in the following.

$$N_{\lambda,\varepsilon}(t) = \begin{cases} N_{\lambda,\varepsilon}(t-1) + N_{\lambda,\varepsilon}(t-\lambda) - \varepsilon_{t-\lambda}, & \text{if } t \geq \lambda, \\ 1, & \text{otherwise.} \end{cases}$$

Note that it is possible to derive the values of ε so that $n = N_{\lambda,\varepsilon}(t)$. Based on this recursion, one can modify the above algorithm for $n = N_\lambda(t)$ to one for $n = N_{\lambda,\varepsilon}(t)$. The basic idea is to keep two local variables, say S_i and T_i. The assertion is that after round r, S_i contains the reduction result of $N_{\lambda,\varepsilon}(r) - 1$ consecutive inputs starting from d_{i+1}, while T_i contains the reduction result of $N_{\lambda,\varepsilon}(r)$ consecutive inputs starting from d_i. To maintain this assertion from round r to round $r+1$, each node i sends its value of S_i or T_i, depending on $\varepsilon_{r+1-\lambda} = 0$ or 1, to an appropriate node and receives a value, which was sent out $\lambda - 1$ rounds earlier, from another appropriate node. See [6], [11] for more details.

In order to make the global combine algorithm work for an arbitrary λ (i.e., λ is not necessarily an integer) one typically treats the latency λ as $\lceil\lambda\rceil$ by conceptually forcing the receive operation to idle for $\lceil\lambda\rceil - \lambda$ time (in order to wait for a corresponding send operation to complete). We refer to it as the *delay-receive* approach. Fig. 15 illustrates the delay-receive approach for $\lambda = 2.5$ in which each receive is delayed to take $\lceil\lambda\rceil = 3$ time steps. In this section, we provide an alternative, called *delay-send* approach by conceptually "forcing" each send operation to idle for $\lambda/\lfloor\lambda\rfloor - 1$ time. Fig. 16 illustrates the delay-send approach for $\lambda = 2.5$ in which each send is delayed to take $\lambda/\lfloor\lambda\rfloor = 1.25$ time steps.

We show that there exists a trade-off between the delay-receive approach and the delay-send approach, depending on the value of λ versus $\lfloor\lambda\rfloor$. For instance, when $1 < \lambda < 1.44$, the delay-send approach performs better than the delay-receive approach. Thus, we provide a way to tune the performance of the global combine operation for a continuous range of values of λ.

3.2 The Delay-Receive Approach for Global Combine

Recall the recursive definition of $N_\lambda(t)$ defined in (1):

$$N_\lambda(t) = \begin{cases} N_\lambda(t-1) + N_\lambda(t-\lambda) & \text{if } t \geq \lambda, \\ 1 & \text{otherwise.} \end{cases}$$

Here, $N_\lambda(t)$ gives, as a recursion, the maximum number of nodes that can be reached in the broadcast problem in t

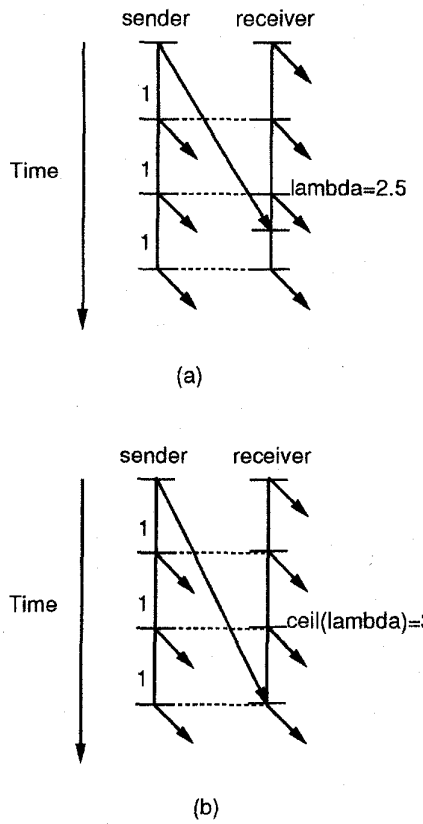


Fig. 15. The delay-receive approach for $\lambda = 2.5$.

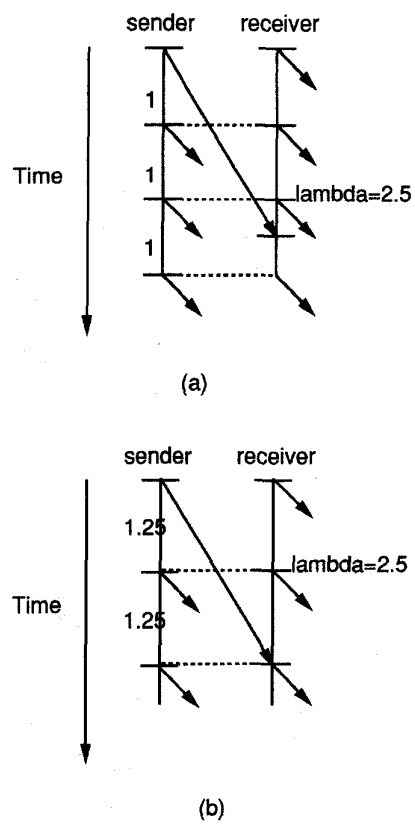


Fig. 16. The delay-send approach for $\lambda = 2.5$.

time steps in the postal model. Note that λ can be any real number and the recursion is still valid. However, in the case of the global combine algorithm in the postal model, the above recursion is only valid when λ is an integer. Let $N'_\lambda(t)$ be the maximum number of nodes for which the global combine algorithm in [11] finishes in time t based on the delay-receive approach (when λ is not an integer).

Similarly, define $T'_\lambda(n)$ as the inverse function of $N'_\lambda(n)$. For convenience, also define

$$\gamma(\lambda) = \lim_{t\to\infty} \frac{N_\lambda(t+1)}{N_\lambda(t)}.$$

That is, when λ is an integer, $\gamma(\lambda)$ is the asymptotic growing ratio of the sequence $N_\lambda(*)$, i.e., $N_\lambda(t) \approx \gamma^t(\lambda)$ for a sufficiently large t.

Then, one can easily derive the recursion:

$$N'_\lambda(t) = \begin{cases} N'_\lambda(t-1) + N'_\lambda(t-\lceil\lambda\rceil) & \text{if } t \geq \lambda, \\ 1 & \text{otherwise.} \end{cases}$$

Clearly, $N'_\lambda(t) \approx \gamma^t\lceil\lambda\rceil$ asymptotically. For a given n, we have $T'_\lambda(n) \approx \log_{\gamma(\lceil\lambda\rceil)} n$ asymptotically.

3.3 The Delay-Send Approach for Global Combine

Let $N''_\lambda(t)$ be the maximum number of nodes that can be "covered" in time t using the global combine algorithm in [11] with the delay-send approach. Similarly, define $T''_\lambda(n)$ as the inverse function of $N''_\lambda(t)$. In the delay-send approach, each send operation is forced to idle $\lambda/\lfloor\lambda\rfloor - 1$ time steps. Thus, every $\lfloor\lambda\rfloor$ consecutive send operations take a total of λ time steps (including the send-idle time).

One can easily derive the recursion:

$$N''_\lambda(t) = N''_\lambda(t - \lambda/\lfloor\lambda\rfloor) + N''_\lambda(t-\lambda).$$

By scaling down all parameters by a factor of $\lambda/\lfloor\lambda\rfloor$, the recursion becomes

$$N_{\lfloor\lambda\rfloor}(t') = N_{\lfloor\lambda\rfloor}(t'-1) + N_{\lfloor\lambda\rfloor}(t' - \lfloor\lambda\rfloor),$$

where $t' = t\lfloor\lambda\rfloor/\lambda$ and $N''_\lambda(t) = N_{\lfloor\lambda\rfloor}(t')$. Note that $N''_\lambda(t)$ can be derived as a function of γ and λ as follows:

$$N''_\lambda(t) = N_{\lfloor\lambda\rfloor}(t') \approx \gamma^{t'}(\lfloor\lambda\rfloor) = \gamma^{t\lfloor\lambda\rfloor/\lambda}(\lfloor\lambda\rfloor).$$

Thus, $t' = \log_{\gamma(\lfloor\lambda\rfloor)} n$ and

$$T''_\lambda(n) = t = t'\lambda/\lfloor\lambda\rfloor = (\lambda/\lfloor\lambda\rfloor)\log_{\gamma(\lfloor\lambda\rfloor)} n.$$

To derive the break-even point for the two approaches, let $T'_\lambda(n) = T''_\lambda(n)$ which yields $\lambda/\lfloor\lambda\rfloor = \log\ \gamma(\lfloor\lambda\rfloor)/\ \log\ \gamma(\lceil\lambda\rceil)$. Table 1 gives the values of the asymptotic growing rate, γ, for $\lambda = 1$ through 10. Table 2 gives the break-even points of λ for $\lfloor\lambda\rfloor = 1$ through 9. For instance, when $1 < \lambda < 2$, the breakeven point between the two approaches is $\lambda \approx \log 2/\log 1.618 \approx 1.44$. Figs. 17 and 18 show the corresponding figures for the two tables. For convenience, Fig. 18 shows the break-even value of $\lambda - \lfloor\lambda\rfloor$ as a function of $\lfloor\lambda\rfloor$.

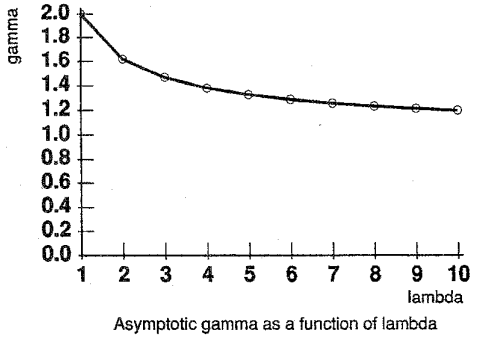


Fig. 17. The asymptotic growing rate $\gamma(\lambda)$ as a function of λ.

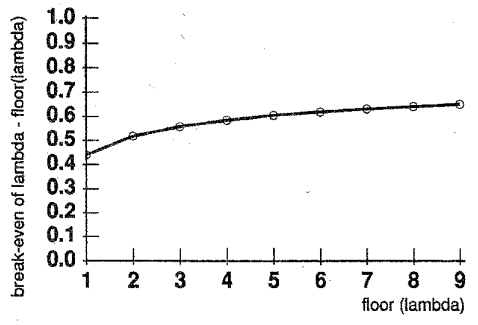


Fig. 18. The break-even value of λ minus $\lfloor\lambda\rfloor$ as a function of $\lfloor\lambda\rfloor$.

TABLE 1
THE VALUE OF $\gamma(\lambda)$ AS A FUNCTION OF λ

λ	1	2	3	4	5	6	7	8	9	10
$\gamma(\lambda)$	2.000	1.618	1.466	1.380	1.325	1.285	1.255	1.232	1.213	1.197

TABLE 2
THE BREAK-EVEN VALUE OF λ AS A FUNCTION OF $\lfloor\lambda\rfloor$

$\lfloor\lambda\rfloor$	1	2	3	4	5	6	7	8	9
break-even λ	1.440	2.518	3.558	4.584	5.604	6.618	7.630	8.640	9.649

4 Conclusion

We studied a number of practical issues related to the design and implementation of two collective communication operations, namely, the broadcast operation and the global combine operation using the postal model. We have proposed techniques to estimate the value of the parameter λ in the postal model for a given machine. For the broadcast operation we have proposed efficient algorithms that get the latency λ and the number of nodes n as parameters. Our main conclusion is that the postal model can help in performance prediction and tuning, for example, our experimental study on the Intel Delta machine showed that a properly tuned broadcast improves the known implementation by more than 20%. Recently, Culler et al. [14] experimented with a number of sorting algorithms on the CM-5 and concluded that the LogP model is helpful in the development of the fast parallel sorting algorithms. For the global combine operation we proposed efficient algorithms for parallel machines with λ which is not an integer. We showed that there exists a trade-off between the delay-receive approach and the delay-send approach, depending on the value of λ versus $\lfloor\lambda\rfloor$. For instance, when $1 < \lambda < 1.44$, the delay-send approach performs better than the delay-receive approach. Thus, we provide a way to tune the performance of the global combine operation for a continuous range of values of λ.

Acknowledgment

The work on the Intel Delta parallel system was facilitated by a joint study with the Caltech Concurrent Supercomputing Facility. We thank Paul Messina, Director of the CCSF, for his support and help. We also thank Bob Cypher, Alex Ho, and Eric Leu for various helpful discussions.

Jehoshua Bruck was supported in part by the National Science Foundation Young Investigator Award CCR-9457811; by the Sloan Research Fellowship; by a grant from the IBM Almaden Research Center, San Jose, California; and by a grant from the AT&T Foundation. Luc De Coster was supported by the Belgian National Fund for Scientific Research.

References

[1] V. Bala, J. Bruck, R. Bryant, R. Cypher, P. deJong, P. Elustondo, D. Frye, A. Ho, C.T. Ho, G. Irwin, S. Kipnis, R. Lawrence, and M. Snir, "The IBM external user interface for scalable parallel systems," Parallel Computing, vol. 20, no. 4, pp. 445–462, Apr. 1994.

[2] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.T. Ho, S. Kipnis, and M. Snir, "CCL: A portable and tunable collective communication library for scalable parallel computers," IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 2, pp. 154–164, Feb. 1995.

[3] V. Bala and S. Kipnis, "Process groups: A mechanism for the coordination of and communication among processes in the Venus collective communication library," Proc. Seventh Int'l Parallel Processing Symp., IEEE, Apr. 1993.

[4] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. van de Geijn, and J. Watts, "Interprocessor collective communication library (InterCom)," Scalable High-Performance Computing Conf., IEEE, pp. 357–364, May 1994.

[5] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn, "Global combine on mesh architectures with wormhole routing," Seventh Int'l Parallel Processing Symp., IEEE, Newport Beach, Calif., Apr. 1993.

[6] A. Bar-Noy, J. Bruck, C.T. Ho, S. Kipnis, and B. Schieber, "Computing global combine operations in the multi-port postal model," Fifth IEEE Symp. Parallel and Distributed Processing, pp. 336–343, Dec. 1993.

[7] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the postal model for message-passing systems," Math. Systems Theory, vol. 27, no. 5, pp. 431–452, 1994.

[8] A. Bar-Noy and S. Kipnis, "Multiple message broadcasting in the postal model," Seventh Int'l Parallel Processing Symp., IEEE, Apr. 1993.

[9] A. Bar-Noy and S. Kipnis, "Broadcasting multiple messages in simultaneous send/receive systems," Fifth Symp. Parallel and Distributed Processing, IEEE, pp. 344–347, Dec. 1993.

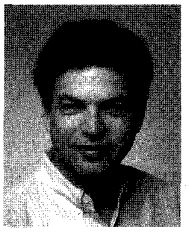
[10] A. Bar-Noy, S. Kipnis, and B. Schieber, "An optimal algorithm for computing census functions in message-passing systems," Parallel Processing Letters, vol. 3, no. 1, Mar. pp. 19–23, 1993.

[11] A. Bar-Noy, S. Kipnis, and B. Schieber, "Optimal computation of census functions in the postal model," to appear in Discrete Applied Math.

[12] J. Bruck, R. Cypher, and C.T. Ho, "Multiple message broadcasting with generalized Fibonacci trees," Fourth Symp. Parallel and Distributed Processing, IEEE, Dec. 1992.

[13] J. Bruck and C.T. Ho, "Efficient global combine operations in multi-port message-passing systems," Parallel Processing Letters, vol. 3, no. 4, pp. 335–346, Dec. 1993.

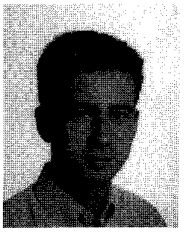
[14] D. Culler, A.C. Dusseau, R.P. Martin, and K.E. Schauser, "Fast Parallel Sorting under LogP: From theory to practice," Proc. Workshop on Portability and Performance for Parallel Processing, Southampton, England, 1993.
[15] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," Fourth SIGPLAN Symp. Principles and Practices of Parallel Programming, ACM, May 1993.
[16] F. Desprez, A. Ferreira, and B. Tourancheau, "Efficient communication operations in reconfigurable parallel computers," Technical Report CS-93-209, Univ. of Tennessee, Aug. 1993.
[17] Express 3.0 Introductory Guide. Parasoft Corporation, 1990.
[18] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, vol. I: General Techniques and Regular Problems. Englewood Cliffs, N.J.: Prentice-Hall, 1988.
[19] P. Fraigniaud and E. Lazard. "Methods and problems of communication in usual networks," Technical Report 91-33, IMAG, Ecole Normale Supérieure de Lyon, France, Oct. 1991.
[20] G.A. Geist, M.T. Heath, B.W. Peyton, and P.H. Worley, "A user's guide to PICL: A portable instrumented communication library," ORNL Technical Report, ORNL/TM-11616, Oct. 1990.
[21] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A survey of gossiping and broadcasting in communication networks," Networks, vol. 18, no. 4, pp. 319–349, 1988.
[22] R. Hempel, "The ANL/GMD macros (PARMACS) in FORTRAN for portable parallel programming using the message passing programming model, user's guide and reference manual," Technical Memorandum, Gesellschaft für Mathematik und Datenverabeitung mbH, West Germany.
[23] S.L. Johnsson and C.T. Ho, "Spanning graphs for optimum broadcasting and personalized communication in hypercubes," IEEE Trans. Computers, vol. 38, no. 9, pp. 1,249–1,268, Sept. 1989.
[24] R. Karp, A. Sahay, E. Santos, and K. Schauser, "Optimal broadcast and summation in the LogP model," Fifth Symp. Parallel Algorithms and Architectures, June 1993.
[25] Message Passing Interface Forum, "Document for a standard message-passing interface," Univ. of Tennessee, Technical Report No. CS-93-214, Nov. 1993.
[26] P.K. McKinley, H. Xu, A. Esfahanian, and L. Ni, "Unicast-based multicast communication in wormhole-routed networks, Proc. 1992 Int'l Conf. Parallel Processing, vol. II, pp. 10–19, Aug. 1992.
[27] A. Skjellum and A.P. Leung, "Zipcode: A portable multicomputer communication library atop the Reactive Kernel," Proc. Fifth Distributed Memory Computing Conf., IEEE, pp. 328–337, Apr. 1990.
[28] Q.F. Stout and B. Wagar, "Intensive hypercube communication: prearranged communication in link-bound machines," J. Parallel and Distributed Computing, vol. 10, pp. 167–181, 1990.
[29] R.A. van de Geijn, "Efficient global combine operations," Sixth Distributed Memory Computing Conf., IEEE, Apr. 1991.



Jehoshua Bruck (S'86–M'89–SM'93) received the BSc and MSc degrees in electrical engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively, and the PhD degree in electrical engineering from Stanford University in 1989.

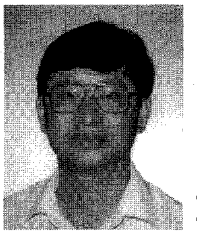
He is an associate professor of computation and neural systems and electrical engineering at the California Institute of Technology. His research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, computation theory and neural systems. Dr. Bruck has an extensive industrial experience, including, serving as manager of the Foundations of Massively Parallel Computing group at the IBM Almaden Research Center from 1990 to 1994, a research staff member at the IBM Almaden Research Center from 1989 to 1990, and a researcher at the IBM Haifa Science center from 1982 to 1985.

Dr. Bruck is the recipient of a 1995 Sloan Research Fellowship, a 1994 National Science Foundation Young Investigator Award, a 1992 IBM Outstanding Innovation Award for his work on "Harmonic Analysis of Neural Networks," and a 1994 IBM Outstanding Technical Achievement Award for his contributions to the design and implementation of the SP-1, the first IBM scalable parallel computer. He received five IBM Plateau Invention Achievement Awards, and he holds 18 patents. He is a senior member of the IEEE.



Luc De Coster received his MSc degree in electrical engineering from the K.U.Leuven, Belgium, in 1994. In 1993, he was a visiting researcher at the IBM Almaden Research Center, San Jose, Calif. with respect to his Master thesis. He is currently a research assistant of the National Fund for Scientific Research (Belgium) and PhD student of the ESAT/ACCA Laboratory of the K.U.Leuven, Belgium. His main research interests are in computer architectures, parallel processing, and digital signal processing.

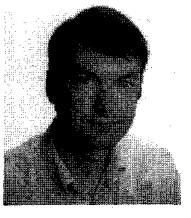
Natalie Dewulf received her MSc in electrical engineering from the K.U.Leuven, Belgium, in 1994. In 1993, she was a visiting researcher at the IBM Almaden Research Center, San Jose, Calif. with respect to her Master thesis. She is currently working with a design team for broadband telecommunication (ATM) at Alcatel Bell, Belgium. Her main research interests are in computer architectures, parallel processing, and telecommunication.



Ching-Tien Ho received a BS degree in electrical engineering from the National Taiwan University in 1979 and MS, M.Phil, and PhD degrees in computer science from Yale University in 1985, 1986, and 1990, respectively.

Dr. Ho joined IBM Almaden Research Center as a research staff member in 1989. He became project leader in 1992 and manager of the Foundations of the Massively Parallel Computing group in 1994, where he leads the development of collective communication, as part of IBM MPL and MPI, for IBM SP-1 and SP-2 parallel systems. His primary research interests include communication issues for interconnection networks, algorithms for collective communications, graph embeddings, fault tolerance, and parallel algorithms and architectures. He has published over 60 journal or conference papers in these areas.

Dr. Ho is a co-recipient of the 1986 "Outstanding Paper Award" of the International Conference on Parallel Processing. He has received an IBM Outstanding Innovation Award, an IBM Outstanding Technical Achievement Award, and three IBM Plateau Invention Achievement Awards. He has six patents granted or pending. He is an Editor of *IEEE Transactions on Parallel and Distributed Systems*. He has served on program committees of the IEEE Symposium on Parallel and Distributed Processing, and International Conference on Parallel and Distributed Systems. He is a member of the IEEE Computer Society and the ACM.



Rudy Lauwereins received his MSc and PhD degrees in electrical engineering from the K.U.Leuven, Belgium, in 1983 and 1989, respectively. In 1991, he was a visiting researcher at the IBM Almaden Research Center, San Jose, Calif., on a postdoctoral NFWO research fellowship. He is currently a senior research associate of the National Fund for Scientific Research (Belgium) and an associate professor of the ESAT/ACCA Laboratory of the K.U.Leuven, Belgium. His main research interests are in computer architectures, parallel processing, digital signal processing, and fault-tolerant computing. In these fields, he has authored and co-authored more than 120 international publications in journals and conference proceedings.