Collective Communication in Wormhole-Routed Massively Parallel Computers

Philip K. McKinley
Yih-jia Tsai
Michigan State University

David F. Robinson
Quincy University

Most MPC networks use wormhole routing to reduce the effect of path length on communication time. Researchers have exploited this by designing ingenious algorithms to speed collective communication.

The supercomputer market is now dominated by parallel architectures. Among these, massively parallel computers (MPCs) are an important class of systems. The memory of an MPC is physically distributed among an ensemble of computing nodes that communicate by sending data through a network. Communication operations can be either *point-to-point*, with one source and one destination, or *collective*, with more than two participating processes. Collective operations are invoked by nodes to distribute, gather, and exchange data; to perform global computation operations on distributed data; and to synchronize with one another at specific points in program flow.

The design of collective communication operations depends on the MPC's underlying network architecture. While there has been little consensus on some aspects of communication architectures, such as network topology, a good deal of agreement exists regarding the most efficient way to switch messages through the network. Most MPCs use wormhole routing, in which each message is divided into small pieces that are pipelined through the network (see "Wormhole-routed architectures" sidebar). Compared with the store-and-forward switching method used in early multicomputers, wormhole routing reduces the effect of path length on communication time. However, in situations where multiple messages exist in the network concurrently, wormhole routing can exacerbate channel contention, which occurs when blocked messages hold some communication channels while waiting for others. Invoking a collective operation, which can involve many messages, poses this situation.

In recent years, many projects have addressed the design of efficient collective communication algorithms for wormhole-routed systems. By exploiting the relative distance-insensitivity of wormhole routing, these new algorithms often differ fundamentally from their store-and-forward counterparts. In this article, we examine software and hardware approaches to implementing collective communication operations. Although we emphasize methods in which the underlying architecture is a *direct* network—such as a hypercube or mesh, as opposed to an *indirect* switch-based network—several approaches apply to systems of either type. We illustrate several issues arising in this research area and describe the major classes of algorithms proposed to solve these problems.

0018-9162/95/$4.00 © 1995 IEEE

COLLECTIVE COMMUNICATION OPERATIONS

A collective operation is usually defined in terms of a group of processes. The operation is executed when all processes in the group call the communication routine with matching parameters. We classify collective operations into three types according to their purpose: data movement, global computation, and process control. Figure 1 on page 42 depicts seven collective operations among N processes. For each operation, the left side of figure elements (a) through (g) shows the processes before the operation, and the right side shows them afterward.

Data-movement operations come in several flavors. In a *broadcast*, one process sends the same message to every group member, whereas in a *scatter*, one process sends a different message to each member. *Gather* is the dual operation of scatter, in that one process receives a message from each group member. These basic operations can be combined to form more complex operations. In *all-to-all broadcast*, every process sends a message to every other group member. In *complete exchange*, also referred to as *all-to-all scatter-gather*, every group member sends a different message to every other group member. Permutation operations, such as shift and transpose, are also collective data-movement operations.

Global computation operations include both *reduction* and *scan*. In reduction, an associative and commutative operation is applied across data items supplied by each group member. Examples include sum, max, min, and bitwise operations. In an $N/1$ reduction operation, also referred to as *global combine*, the resultant data resides at a single process, called the *root*. In an N/N reduction operation, every process involved in the operation obtains a copy of the reduced data. In scan operations, given processes $P_1, P_2, \ldots, P_N$ holding data items $d_1, d_2, \ldots, d_N$, respectively, an associative and commutative operator $\oplus$

Wormhole-routed architectures

An MPC network's switching strategy determines how a message is removed from one channel and placed on another along the path to its destination. In store-and-forward switching, each intermediate node receives the entire message (or packet) before forwarding it. To reduce this load on local processors, each node can be equipped with a separate router to perform this task.

Wormhole routing further improves communication performance. This switching technique divides each message into small pieces called *flits*. The message header, comprising one or more flits, contains routing information. When a channel has been acquired by a message, it is reserved (or held) by this message until the last flit has been transmitted on the channel, and then it is released. As the header advances along the specified route, the remaining flits follow in pipeline fashion. If the header flit encounters a busy channel, it is blocked in the network until the channel becomes available. While store-and-forward switching does not require a router at each node, the fine-grained nature of wormhole routing usually necessitates this additional hardware.

In the absence of channel contention, the time to send a unicast message in a wormhole-routed network, t_{wh}, is approximated by $t_s + t_h (d - 1) + lt$, where t_s is the combined start-up latency at the source and destination, t_h is the delay at the intermediate routers encountered on the path, d is the number of channels traversed, l is the length of the message, and t is the per-flit transmission time. The value of $t_h (d - 1)$ is often small compared with that of lt. As a result, the term $t_h (d - 1)$ is often ignored in performance evaluation of communication algorithms. This "distance-insensitivity" of wormhole routing has been demonstrated in experiments on commercial systems.

Several other architectural parameters affect the design of collective communication algorithms. Many wormhole-routed MPCs use direct network topologies, in which nodes are connected by point-to-point communication channels. Figure A depicts three such topologies. Research and commercial systems that use wormhole routing include the Ncube-2 (hypercube), the Intel Touchstone Delta and Paragon (2D mesh), the MIT J machine (3D mesh), and the Cray T3D (3D torus). Other parallel systems, such as the TMC CM-5, Meiko CS-2, and IBM SP1, also use wormhole routing, though their networks are switch based.

Mesh-like topologies are popular partly because they lend themselves to simple routing algorithms.

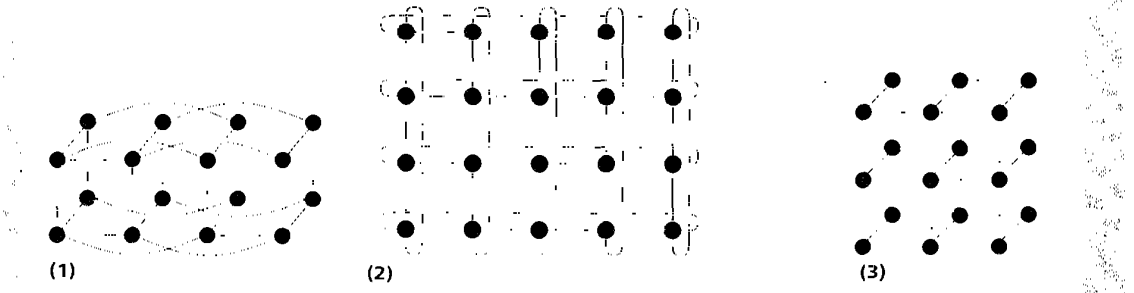

Figure A. MPC direct network topologies. (1) 4-cube; (2) 5 × 4 2D torus; and (3) 3 × 3 × 2 3D mesh.

is applied to the data. The result at each process depends on the process's rank. In a *prefix* scan, the result at process P_i is $d_1 \oplus d_2 \oplus \ldots \oplus d_i$. In a *postfix* scan, the result at process P_i is $d_i \oplus d_{i+1} \oplus \ldots \oplus d_N$.

The most common process control operation is a *barrier*, which defines a logical synchronization point in member process execution. All group members must arrive at the barrier before any may proceed.

The remainder of this article describes several techniques for collective communication design in wormhole-routed systems. Although we focus primarily on implementations of broadcast operations, many of these methods may be applied to other collective operations.

BROADCAST AND MULTICAST TREES

Many algorithms for collective operations are based on a tree of point-to-point, or *unicast*, messages. A *broadcast tree* involves all nodes in the network; a *multicast tree* involves a subset of these nodes. Multicast trees are needed if the application is allocated only a subset of nodes or the operation involves only a subset of the application processes. A multicast tree should involve solely those processors at source and destination nodes, since interrupting other processors wastes computing resources. Although a broadcast operation can be implemented with a multicast tree algorithm, this special case is often treated separately, since the participation of all nodes leads to simpler and more regular message-passing patterns.

Hamiltonian path

Because first-generation MPCs used store-and-forward switching, many early tree-based algorithms used only nearest-neighbor communication. Perhaps the simplest broadcast "tree" is a Hamiltonian path. Figure 2a illustrates such a path in a four-dimensional hypercube, or 4-cube. The path, which starts at node 0000 and ends at node 1000,

Routing can be either deterministic (the path between a given source and destination is fixed) or adaptive (the path is influenced by current network traffic). Although research in adaptive wormhole routing is promising, most routing algorithms presently used in commercial systems are deterministic. *Dimension-ordered routing*, the most commonly used method, forwards a message through dimensions of the topology in strictly ascending (alternatively, descending) order. This method prevents deadlock by imposing an ordering on channel acquisition. Moreover, its predictable nature offers designers of collective operations more control over the constituent messages than does adaptive routing.

The *port model* of the system defines the number of internal channels connecting each local processor to its router. These ports can share one physical link or be implemented with separate links. In a one-port system, a node must transmit (or receive) messages sequentially. Architectures with multiple ports alleviate this bottleneck. Figure B1 shows a one-port router in a 2D mesh. Figure B2 shows an *all*-port router in a 2D mesh, in which every external channel has a corresponding port. A system can also possess a *k*-port architecture, where the number of ports is greater than one but less than the number of external channels. The number of input ports can also differ from the number of output ports. The port model is important in designing collective operations, which usually require nodes to send and/or receive multiple messages.

Finally, many wormhole-routed systems multiplex multiple *virtual channels* onto each physical communication channel.[3] Each virtual channel has its own flit buffer and control lines. Virtual channels can be used to eliminate deadlock, but how they are used in a particular system determines potential message contention, thereby affecting the design of collective communication operations.

References

1. W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *J. Distributed Computing*, Vol. 1, No. 3, 1986, pp. 187-196.
2. L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, Vol. 26, No. 2, Feb. 1993, pp. 62-76.
3. W.J. Dally, "Virtual Channel Flow Control," *IEEE Trans. Computers*, Vol. 3, No. 3, Mar. 1992, pp. 194-205.

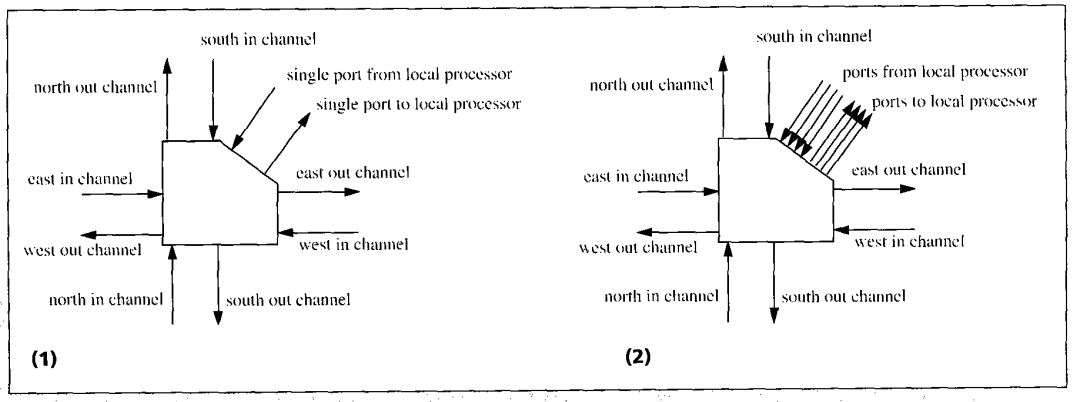

Figure B. Port models for a 2D mesh: (1) one-port router and (2) all-port router.

is based on a binary-reflected Gray code. In either a store-and-forward or a wormhole-routed network, the time needed to complete a broadcast operation in this manner is $(N-1) \times t_u$, where N is the number of network nodes and t_u is the time required to send a unicast message.

Spanning binomial tree

An approach that makes better use of the dense hypercube topology is the well-known *spanning binomial tree* (SBT) algorithm, which is illustrated for a 3-cube in Figure 2b. The message-passing steps appear in brackets. In the first step, the source node sends the message to the neighbor whose address differs from its own in the lowest (alternatively highest) bit position. In the second step, these two nodes send the message to their respective neighbors in the next dimension. This *recursive doubling* process continues until, in the last step, half of the nodes in the hypercube forward the message to the other half through the highest dimension. This algorithm requires n message-passing steps to reach all nodes in an n-cube.

Message-partitioning

This technique is often used to reduce the execution time of collective operations. An early example is the edge-disjoint spanning tree (EDST) broadcast algorithm for hypercubes.[1] In this algorithm, the message is partitioned into n segments, each transmitted along a different spanning tree, as illustrated in Figure 2c. By spreading message segments among more channels than the SBT algorithm does, the EDST algorithm broadcasts a message of length l in time $O((l/n) \log_2 N) = O(l)$.

Other collective operations

Broadcast and multicast trees can be used to support other collective operations. For example, Figure 3a depicts a scatter operation using an SBT in a one-port, eight-node system; the numbers in brackets represent message-passing steps. Each node stores its own message and forwards the others to its children. The gather operation is implemented by reversing the direction of message transmission, as shown in Figure 3b. The same structure can be used to implement $N/1$ reduction, in which the reduction operation is performed pairwise on data items as they proceed toward the root. An N/N reduction operation, and the special case of barrier synchronization, can be implemented by distributing that result to all nodes using the same SBT.

EXPLOITING DISTANCE-INSENSITIVITY IN MESHES

Many new-generation wormhole-routed MPCs use low-dimensional mesh and torus topologies, which are more

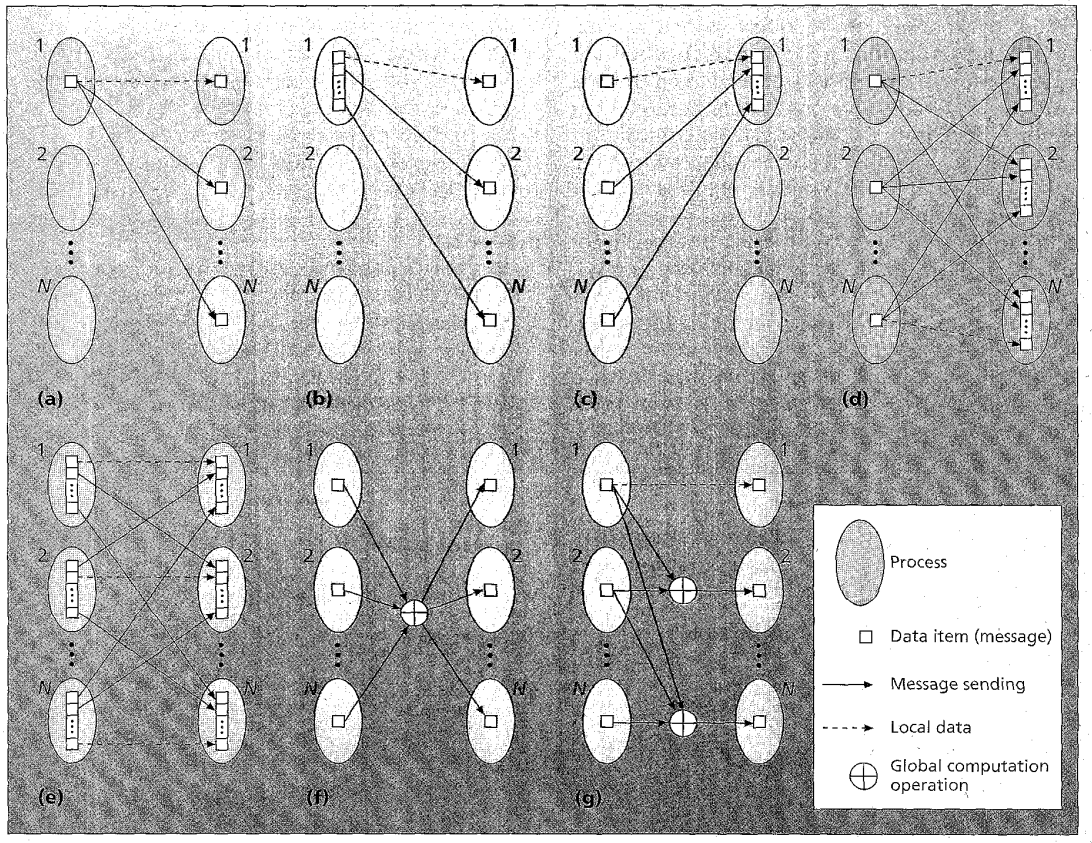

Figure 1. Semantics of various collective operations: (a) broadcast from process 1; (b) scatter from process 1; (c) gather at process 1; (d) all-to-all broadcast; (e) complete exchange; (f) *N/N* reduction; and (g) prefix scan.

easily constructed than hypercubes. Although these topologies exhibit larger internode distances than hypercubes, the relative distance-insensitivity of wormhole routing largely obviates this problem for unicast communication. In collective operations, using multiple-hop message paths increases parallelism and decreases contention among constituent messages.

Dimensional broadcast trees

Many algorithms for broadcast communication in wormhole-routed mesh networks are based on algorithms designed for store-and-forward hypercubes. An algorithm described by Barnett et al.[2] uses a recursive doubling process within each mesh dimension. Figure 4 illustrates this algorithm's operation in an 8×8 2D mesh. In each message-passing step, each node holding a copy of the message is responsible for a part of a row or column. The node divides its part in half and sends a copy of the message to the node in the other half that occupies the same relative position. The algorithm thus takes advantage of the pipelining effect of wormhole routing while avoiding channel contention.

In general, this algorithm requires

$$\sum_{i=0}^{d-1} \lceil \log w_i \rceil$$

message-passing steps in a d-dimensional mesh in which the width of dimension i is w_i. This result is optimal in some cases, such as when the width of every dimension is a power of 2, but not in all cases. For example, in a 10×10 mesh, the dimensional approach requires eight steps, whereas the lower bound is $\lceil \log_2 100 \rceil = 7$ steps. The lower bound on the number of steps

$$\left\lceil \log_2 \left(\prod_{i=0}^{d} w_i \right) \right\rceil = \lceil \log_2 (N) \rceil$$

can be achieved by constructing an SBT containing N vertices, placing the source node at the tree's root, and randomly populating the rest of the tree with the remaining nodes. Of course, channel contention among messages can produce delays that are not reflected by simply counting the number of steps.

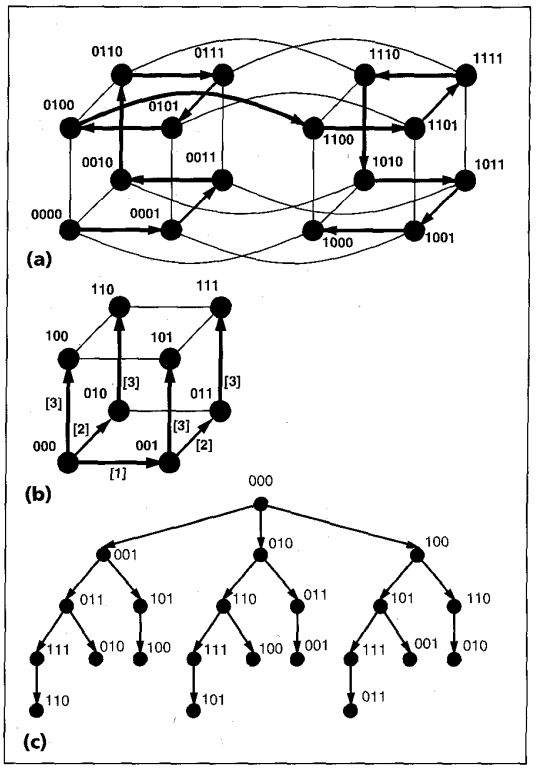

Figure 2. Hypercube broadcast trees based on nearest-neighbor communication: (a) Hamiltonian path; (b) spanning binomial tree; and (c) edge-disjoint spanning trees.

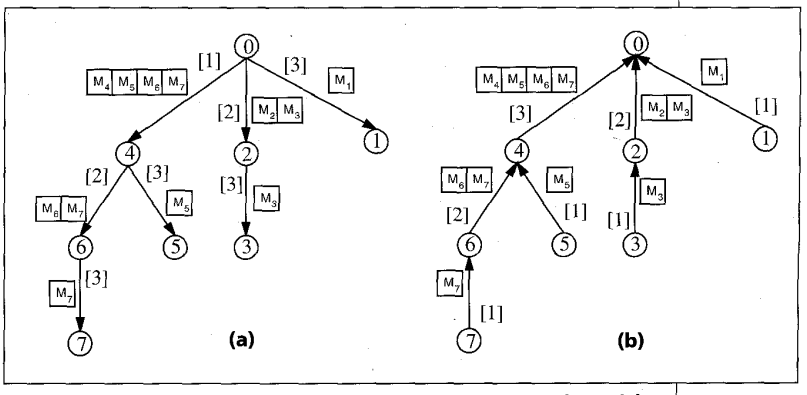

Figure 3. Using broadcast trees in other collective operations: (a) scatter and (b) gather/reduction.

Contention-free multicast trees

In studying the problem of multicast

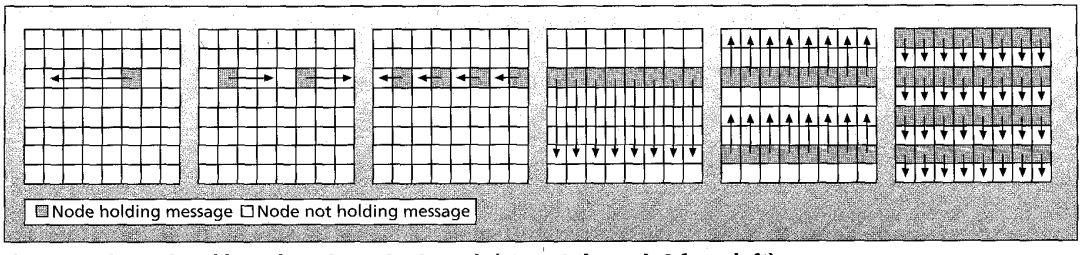

Figure 4. Dimensional broadcast in an 8×8 mesh (steps 1 through 6 from left).

communication in wormhole-routed systems, McKinley et al.[3] proposed a method that achieves the lower bound while avoiding channel contention. Like the broadcast algorithms described above, this approach uses recursive doubling. The U-mesh algorithm for n-dimensional mesh topologies first sorts the source and destination addresses lexicographically into a list, denoted as Φ. The source node successively divides Φ in half. If the source itself is in the lower half, it sends a copy of the message to the first node in the upper half. That node delivers the message to all other nodes in the upper half by invoking the same algorithm recursively. If the source is in the upper half, it sends a copy of the message to the last node in the lower half. The source continues this procedure until Φ contains only its own address. Figure 5 illustrates the U-mesh algorithm's operation in a 6×6 2D mesh.

It has been shown that ordering the source and destinations in this way results in contention-free multicast operations.[3] Variations of this algorithm have been proposed for the hypercube and torus as well. All these algorithms require $\lceil \log_2 (m+1) \rceil$ message-passing steps to reach m destinations, which is the lower bound for any multicast operation, including the special case of broadcast.

Broadcast with message-partitioning

Because the dimensional broadcast and U-mesh algorithms do not require message-partitioning, they result in relatively simple code and few communication start-ups. Message-partitioning, however, can improve performance for long messages. An example of this approach to broadcast in meshes is the *scatter-collect* algorithm,[2] whose operation in an 8×8 2D mesh is illustrated in Figure 6. The source node partitions the message and performs a scatter operation across the row of the source node, with each node receiving a particular segment of the message. This operation could be implemented using the approach in Figure 3a, which requires three steps to reach eight nodes. Next, each node holding a part of the message performs a scatter in its column. In the third step, the nodes in each row form a logical ring and circulate segments around the ring until all nodes in that row hold all segments. Finally, a similar circulation in each column results in a copy of the entire message at every node. This algorithm avoids channel contention, and pipelining message segments offers good performance for broadcasting long messages.[2] Similar message-passing patterns can be used to implement both $N/1$ and N/N reduction operations.

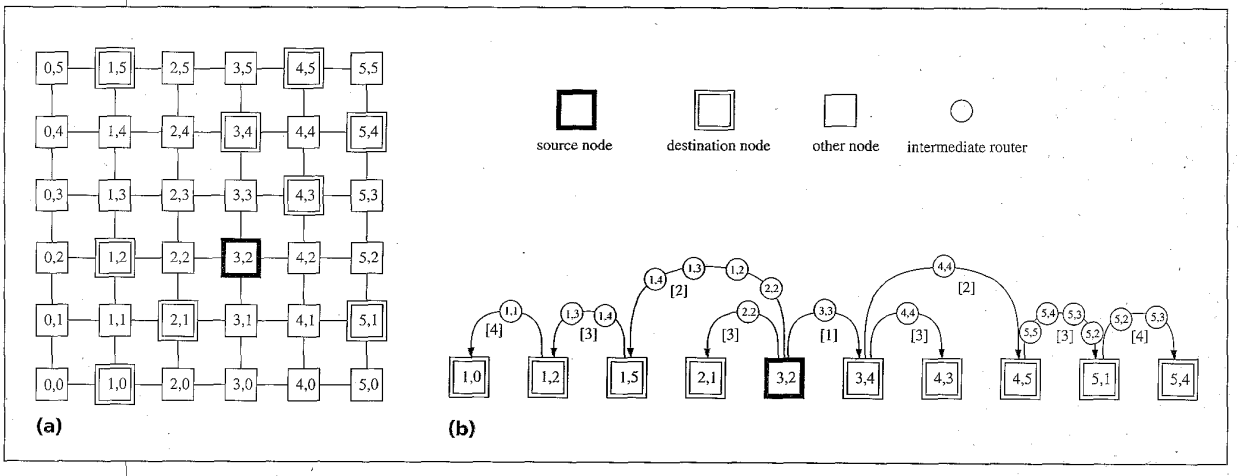

Figure 5. U-mesh multicast solution in a 6×6 mesh: (a) multicast example and (b) multicast tree.

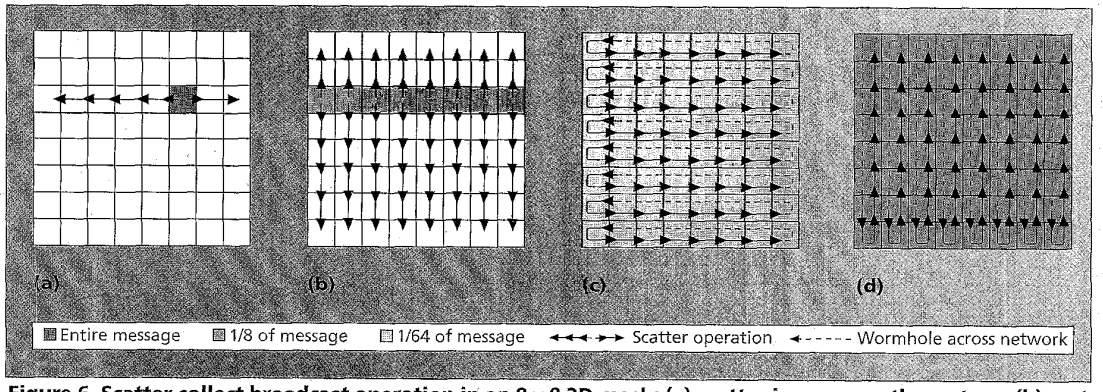

Figure 6. Scatter-collect broadcast operation in an 8×8 2D mesh: (a) scatter in one row, three steps; (b) scatter in columns, three steps; (c) collect in rows, eight steps; and (d) collect in columns, eight steps.

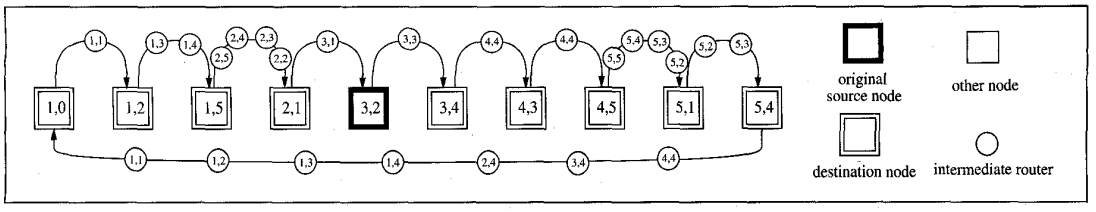

Figure 7. Collect phase of scatter-collect multicast (example from Figure 5).

Multicast with message-partitioning

U-mesh[3] is optimal when the message is small enough so that start-up latency dominates unicast latency. A scatter-collect extension of this algorithm can provide better performance for longer messages. The source node uses the U-mesh tree to *scatter* message segments to the destinations. These nodes then form a logical ring and circulate the message segments until all destinations have received the entire message. When the ring is constructed according to lexicographic ordering, this circulation process is contention-free.

Suppose that the U-mesh tree depicted in Figure 5b were used to scatter segments of a message to the nine destinations. The corresponding logical ring is shown in Figure 7. Let us consider a message of length l flits divided into 10 segments. (See previous sidebar for definitions). Ignoring start-up latency, if nine segments are scattered using the tree in Figure 5b, nodes (1, 0) and (5, 4) receive their segments at time $10(l/10)\ t_f = lt_f$, where t_f is the channel's flit transmission time. The nine-step circulation of segments requires time $9(l/10)\ lt_f$, bringing the total to $1.9\ lt_f$. By comparison, using a simple U-mesh tree for the multicast operation would require time $\lceil log_2 10 \rceil lt_f = 4\ lt_f$.

Another approach comprising a family of pipelined multicast algorithms was recently proposed by De Coster et al.[4] The algorithm begins with the source node partitioning the message into p packets. The source sends all packets in succession to its first child in the tree. When the source has sent the last packet to the first child, it begins sending the packets to the second child. A child stores each packet it receives and also forwards a copy to its own children. Although the tree is based on a lexicographic ordering of the source and destinations, its particular structure can be tailored to the number of nodes m and the number of packets p. For example, if $p = 4$, the tree shown in Figure 8a is optimal for the multicast example in Figure 5a (intermediate routers are not shown). On the other hand, if $p = 10$, the optimal "tree" is simply a path, as shown in Figure 8b.

Returning to our timing comparisons, if the message is divided into four packets ($p = 4$) and start-up latency is negligible, multicasting a message of length l flits using the tree in Figure 8a requires nine steps of sending packets of size $(l/4)$, for a total time of $2.25\ lt_f$. However, if the message is partitioned into 10 packets, the total time using the tree in Figure 8b is $1.8\ lt_f$, an improvement over the earlier scatter-collect multicast algorithm. Further, if $p = 100$, the time is only $1.08\ lt_f$. This algorithm is contention-free, and experimental results have demonstrated its effectiveness.[4]

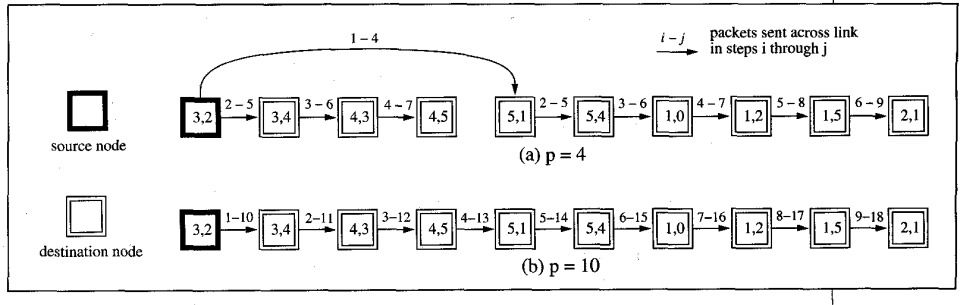

Figure 8. Alternate multicast trees for example in Figure 5; (a) $p = 4$ and (b) $p = 10$.

TAKING ADVANTAGE OF MULTIPORT INTERFACES

The system's port model, which determines how many messages a node can send or receive concurrently, is also important in designing collective operations. Collective algorithms that exploit multiple ports have been designed for both meshes and hypercubes.

Broadcast trees in meshes

The Extended Dominating Node (EDN) model[5] supports the systematic construction of collective operations for multiport wormhole-routed systems. The model is based on the concept of dominating sets from graph theory. A dominating set D of a graph G is a set of vertices in G in that every vertex in G is either in D or is adjacent to at least one vertex in D. The EDN model broadens the concept of node domination to include nodes reachable in a single communication step under a given routing algorithm (for example, *XY* routing in 2D mesh networks). Multiple levels of EDNs define the message-passing steps comprising a collective operation. The key in applying the EDN method to the development of collective operations lies in finding EDN sets that form regular and recursive patterns.

Figure 9 illustrates how the EDN method can be used to implement broadcast in a 16×16 2D mesh network under *XY* routing. In the first two message-passing steps, called start-up steps, the source node delivers the message to the four highest level EDNs, that is, level-three EDNs. These EDNs proceed to "dominate" the 12 level-two EDNs by delivering the message to them in step 3. By the end of step 4, every level-one EDN has received the message from either a level-two or -three EDN. In step 5 (not shown), the messages are delivered to all remaining nodes when

the EDNs send to appropriate neighboring nodes, since every level-zero node (unshaded) is adjacent to at least one EDN (shaded). In general, the EDN broadcast algorithm requires at most $(k + 2)$ steps in a mesh of size $2^k \times 2^k$. The EDN model has also been applied to other collective operations, including reduction and permutations.[5]

Broadcast trees in hypercubes

Ho and Kao[6] developed a broadcast algorithm for all-port wormhole-routed hypercubes. The algorithm uses the concept of a dimension-simple path to recursively divide the network into subcubes of nearly equal size. A path $P = v_0, v_1, \ldots, v_d$ in an n-cube is called *dimension-simple* if there exists a sequence $i_1, i_2, \ldots, i_d$ of distinct cube dimensions such that for all $v_j, j \geq 1$, v_j is obtained from v_{j-1} by complementing the bit at dimension i_j. The path P is called *ascending* if $i_1 < i_2 < \ldots < i_d$. For example, an ascending dimension-simple path from node 0000000 in a 7-cube is

$$0000000 \rightarrow 0000001 \rightarrow 0000011 \rightarrow 0000111 \rightarrow 0001111 \rightarrow 0011111 \rightarrow 0111111 \rightarrow 1111111.$$

In an all-port wormhole-routed hypercube in which dimension-ordered routing is performed by resolving

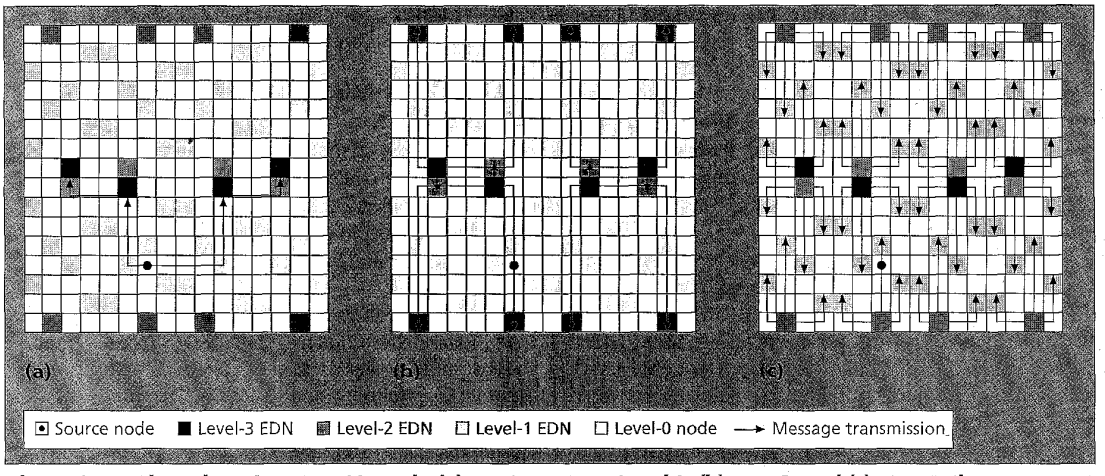

Figure 9. EDN broadcast in a 16 × 16 mesh: (a) start-up steps 1 and 2; (b) step 3; and (c) step 4, the penultimate step.

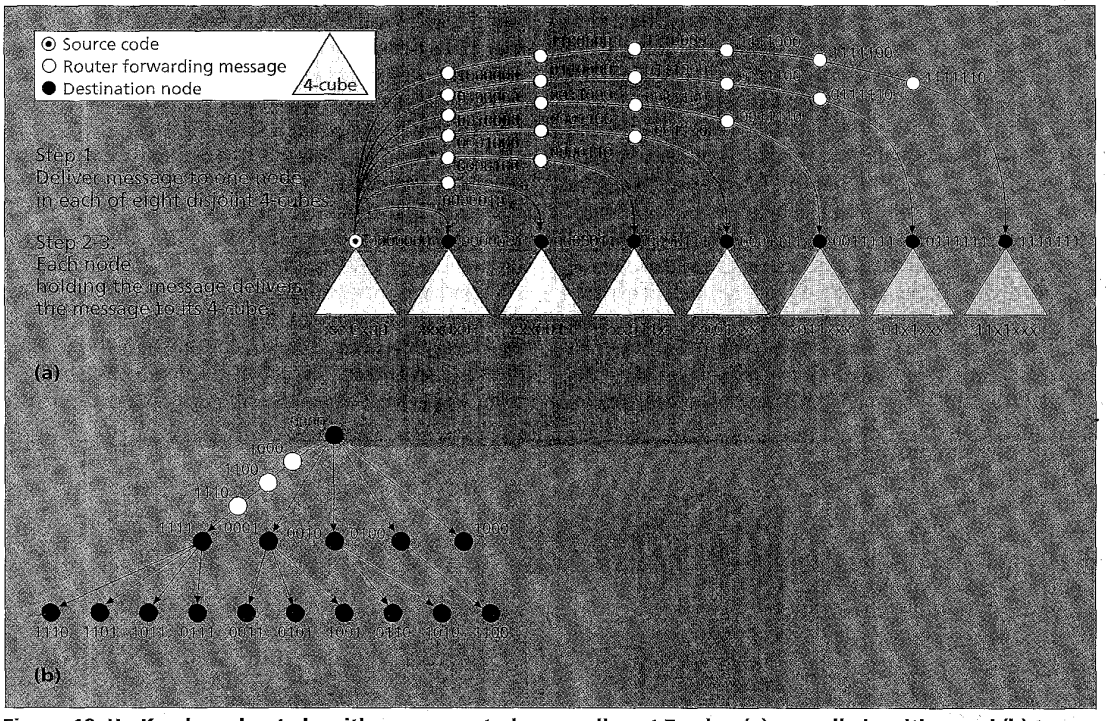

Figure 10. Ho-Kao broadcast algorithm as executed on an all-port 7-cube: (a) overall algorithm and (b) two-step broadcast in a 4-cube.

addresses from top to bottom, one node can send a message to all nodes along such a path simultaneously (some staggering may occur due to message start-up latency). In this manner, an n-cube can be partitioned into $n + 1$ subcubes such that each subcube contains one node on the dimension-simple path. Figure 10 depicts the operation of the Ho-Kao broadcast algorithm in a 7-cube. The number of steps required by this broadcast algorithm is optimal to within a multiplicative constant.[6]

TOPOLOGY-INDEPENDENT APPROACHES

Although many collective communication algorithms have been designed for specific topologies, several researchers have taken a fundamentally different approach. Their idea is to design families of parameterized algorithms that can be tuned to perform well on different architectures under various system conditions. Examples include the postal model[7] and the more general LogP model.[8] These approaches typically assume a "fully connected" model of communication in which the communication costs are equal between all node pairs. This model is consistent with systems whose nodes are interconnected by a switch, and it approximates the behavior of systems using a wormhole-routed direct network.

The postal model

One of the main parameters of these models is the communication delay. The postal model[7] is based on a parameter $\lambda = t_u/t_{snd}$, where t_{snd} is the time it takes for an originator to send a message, or sending latency, and t_u is the total time that passes until the destination receives the message. This model accommodates situations in which a node can transmit multiple messages before the first destination receives a message; the analogy is sending letters through the postal system. Let us assume that λ is an integer and time is measured in *rounds*, with the duration of a round equal to the sending latency. In broadcast, the value of λ affects which type of tree achieves the best performance. Figure 11 shows two different broadcast trees, each involving eight nodes, when $\lambda = 2$. Next to each node is the number of the round in which it receives the message; a node sends the message to its children in left-to-right order. The binomial tree in Figure 11a, although optimal when $\lambda = 1$, requires six rounds to complete the broadcast operation, while the tree in Figure 11b requires only five.[7]

The binomial tree is based on the recursive doubling or—from the perspective of the entire group of nodes—recursive *halving*. Broadcasting in the postal model is based on the observation that, depending on the value of λ, a partitioning method other than halving may result in better performance. An optimal *λ-tree* for m nodes and a specific value of λ is constructed by using a generalized Fibonacci function:[7]

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

$N_\lambda(t)$ represents the maximum number of nodes that can be reached in time t on a one-port architecture exhibiting λ. If $t < \lambda$, only the source itself can hold the message at time t. If $t \geq \lambda$, the number of nodes reached is the sum of two terms. The first term, $N_\lambda(t-1)$, denotes the number of nodes that have been reached by the penultimate round. The second term, $N_\lambda(t-\lambda)$, denotes the maximum number of nodes that can be reached in the last round. As with other algorithms, performance can be further improved by partitioning large messages into multiple packets and pipelining the packets on the branches of the tree.[7]

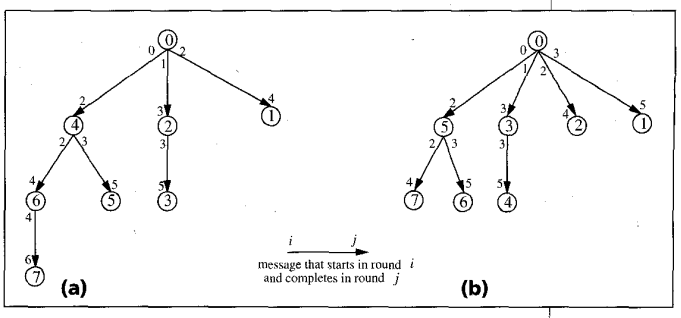

Figure 11. Comparison of broadcast trees with $\lambda = 2$: (a) binomial tree and (b) lambda tree.[7]

Systems issues

Additional factors must be considered when applying such models to actual systems. For example, the value of λ typically depends on the message size and is usually not an integer. Even for a fixed value of λ, the optimal structure of the broadcast tree depends on the number of nodes m. Bruck et al.[9] proposed maintaining a relatively small lookup table indicating how to partition destination sets for specific ranges of m. In addition, although channel contention is ignored under a fully connected model, experimental results showed that ordering the destinations lexicographically, as in the U-mesh algorithm, can improve performance.[9]

PATH-BASED ROUTING

Thus far, we have described several issues in designing collective operations for systems that provide only unicast communication in hardware. To improve performance, some collective operations may be supported directly by the system architecture. In one approach, used in the TMC CM-5 and the Cray T3D, collective operations are implemented in a special network separate from the primary data network. In another approach, the data network is enhanced with "generic" functionality that can support several collective operations. We concentrate on the latter approach, whose resultant algorithms are more closely related to those methods described in the rest of this article.

One such technique is *intermediate reception*, which enables a router to copy the flits of an incoming message to the local processor's memory while simultaneously forwarding them toward another destination. In this way, a message can be routed as a single worm through several destination nodes, depositing a copy of the message at each. This approach is sometimes called path-based routing,[10] while the constituent messages are termed multidestination worms.[11] Intermediate reception has been used in the University of Michigan HARTS (Hexagonal Architecture for Real-Time Systems) machine, which has a wraparound

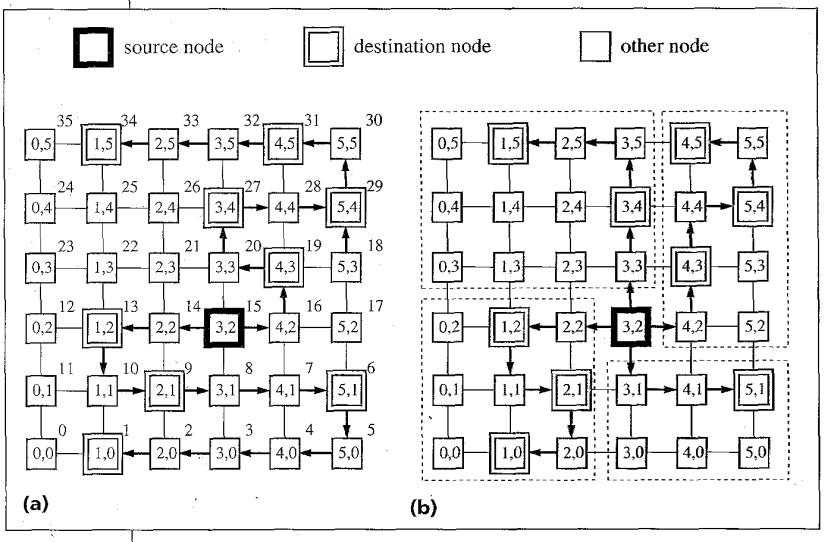

Figure 12. Examples of path-based routing: (a) dual-path and (b) multipath.

hexagonal mesh topology, and in a version of the Caltech mesh-routing chips.

Multidestination worms can be prefixed with a list of destination node addresses or offsets between destinations. Once the message header reaches the first destination node in the list, the router at that node strips its own address from the head of the list and forwards the remainder of the message worm to the next destination. The routing rules should be liberal enough to allow many intermediate destination nodes to be visited by one message without incurring deadlock. One solution is to base all routing decisions on a Hamiltonian path (HP) in the network, which prevents deadlock by imposing a total ordering on channel usage. Lin et al.[10] used this method to develop a family of path-based multicast routing algorithms for 2D meshes. In the most basic of these algorithms, termed dual-path (and illustrated in Figure 12a), the source node of a multicast generates at most two multidestination worms. One worm visits all destination nodes that appear after the source on the HP, and the other visits destinations appearing before the source. To reduce path lengths, the algorithm lets messages "skip over" some nodes on the HP, provided that the nodes visited by any message worm appear in the same relative order as defined by the HP. This algorithm can be extended to generate up to four concurrent multidestination message worms in a 2D mesh, as shown in Figure 12b, by partitioning the mesh into quadrants.

Panda et al.[11] and Boppana et al.[12] showed that the port model of the system is critical in preventing deadlock in systems supporting path-based routing. Boppana et al. showed that equipping each node with two input ports and restricting their use (one for "forward" and one for "backward" messages) can prevent deadlock in path-based systems. Panda et al. showed that when multidestination worms conform to dimension-ordered routing, $n + 1$ input ports per node are sufficient to prevent deadlock in a k-ary n-cube. Finally, multiple multidestination worms may be combined to form a single collective operation. Panda et al.[11] used this concept to design multicast and broadcast algorithms for various topologies under different base routing algorithms, including deterministic and adaptive types. In this approach, an operation is implemented as a sequence of *phases*, with each phase comprising one or more multidestination worms. The multiphase method avoids delays due to excessive path length, which may arise if one worm is used to reach many nodes in a large mesh. Given the relatively minor modifications to routers needed to implement multidestination worms, such multiphase methods are likely to be used to improve the performance of many other collective operations.[11]

WORMHOLE ROUTING HAS PLAYED A MAJOR ROLE in reducing the cost of unicast communication among nodes in MPCs. Achieving similar improvements in the performance of collective operations, however, has required redesigning these algorithms to exploit the distance-insensitivity of this switching technique. While our discussion has primarily addressed broadcast and multicast operations, similar techniques have been used in designing other collective operations, such as all-to-all broadcast, reduction, and scan. These topics, as well as adaptive routing, fault tolerance, and switch-based interconnection networks, are important in their own right, and surveys of research and commercial projects in these areas would likely be of great interest to the community. ■

Acknowledgments

The authors are sincerely grateful to the anonymous reviewers and to C.-T. Ho of IBM for corrections to and suggestions for improving this article. We also thank the Michigan State University faculty and students whose work has either directly or indirectly contributed to this article: Lionel M. Ni, Abdol-Hossein Esfahanian, Betty H.C. Cheng, Xiaola Lin, Hong Xu, Christian Trefftz, Chengchang Huang, Dan Judd, and Yih Huang. This work was supported in part by NSF Grants MIP-9204066 and CCR-9503838, and by DOE Grant DE-FG02-93ER25167.

References

1. S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computers*, Vol. C-38, No. 9, Sept. 1989, pp. 1,249-1,268.
2. M. Barnett et al., "Broadcasting on Meshes with Worm-Hole Routing," Tech. Rep. TR-93-24, Dept. Computer Science, Univ. Texas at Austin, 1993.
3. P.K. McKinley et al., "Unicast-Based Multicast Communication in Wormhole-Routed Direct Networks," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 12, Dec. 1994, pp. 1,252-1,265.
4. L. De Coster, N. Dewulf, and C.-T. Ho, "Efficient Multi-Packet Multicast Algorithms on Meshes with Wormhole and Dimension-Ordered Routing," *Proc. 1995 Int'l Conf. Parallel Processing*, Vol. III, IEEE CS Press, Los Alamitos, Calif., Order No. RS-00027, 1995, pp. 137-141.
5. Y.-J. Tsai and P.K. McKinley, "An Extended Dominating Node

Collective communication libraries

Collective communication appears in all three major methods of programming distributed-memory computers. First, collective operations are inherent in message-passing algorithms such as sorting and searching, as well as in matrix-related algorithms such as linear system solvers, eigenvalue solvers, and transform operations.[1] Second, many constructs in data-parallel languages can be translated into collective operations during compilation.[2] Third, collective operations are used to support synchronization, shared-data invalidation, and updating in implementations of distributed shared memory.[3] As a result, many commercial and publicly available communication libraries offer primitives for collective operations. In addition, the recently proposed Message Passing Interface (MPI) standard[4] contains an entire chapter on collective operations. Of course, while the interfaces and semantics are well-defined, implementations of these operations usually depend on the architecture.

To improve flexibility, scalability, and portability, several research groups are developing unifying models and frameworks for designing and implementing collective communication libraries. The Basic Linear Algebra Communication Subprograms (BLACS) library[5] supports ScaLapack, a distributed-memory version of the Lapack numerical linear algebra package. BLACS assumes a logical 2D grid node configuration and provides global and row/column broadcast and reduction operations.

In developing the InterCom collective communication library for multidimensional meshes and hypercubes, Barnett et al.[6] focused on data size. Operations are implemented as hybrid algorithms that work well with both short and long vectors, and accommodate operations among arbitrary subsets of nodes. The InterCom library has been implemented on an Intel Paragon.

Extensive work on collective communication libraries is being conducted by IBM researchers. Bala et al.[7] incorporated several novel features into CCL, a collective communication library for the IBM SP1 system. The concept of *process groups* lets applications define, partition, and invoke collective operations across groups of processes dynamically without using explicit member lists. Semantics in send, receive, and barrier operations are defined formally to guarantee correctness. Since CCL was designed under the assumption of a fully connected topology, many of the algorithms can be tuned for better performance in specific environments. The operations are parameterized to account for message size, startup latency, and port model.[7]

References

1. V. Kumar et al., *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, Redwood City, Calif., 1994.
2. J. Li and M. Chen, "Compiling Communication-Efficient Programs for Massively Parallel Machines," *IEEE Trans. Parallel and Distributed Systems*, Vol. 2, July 1991, pp. 361-375.
3. B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms," *Computer*, Vol. 24, No. 8, Aug. 1991, pp. 52-60.
4. Message-Passing Interface Forum, "MPI: A Message-Passing Interface Standard," Version 1.0, Univ. Tennessee, Knoxville, 1994, ftp://ftp.mcs.anl.gov/pub/mpi/mpi-report.ps.Z.
5. J. Dongarra, R. van de Geijn, and R. Whaley, "Two-Dimensional Basic Linear Algebra Communication Subprograms," in *Proc. Sixth SIAM Conf. Parallel Processing*, SIAM, Philadelphia, 1993, pp. 347-352.
6. M. Barnett et al., "Interprocessor Collective Communication Library (InterCom)," in *Proc. Scalable High-Performance Computing Conf.*, IEEE SCP, 1994, pp. 357-364.
7. V. Bala et al., "CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computers," *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, Feb. 1995, pp. 154-164.

Approach to Collective Communication in Wormhole-Routed 2D Meshes," *Proc. Scalable High-Performance Computing Conf.*, IEEE CS Press, Los Alamitos, Calif., Order No. PRO5680, 1994, pp. 199-206.

6. C.-T. Ho and M. Kao, "Optimal Broadcast in All-Port Wormhole-Routed Hypercubes," *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 2, Feb. 1995, pp. 200-204.
7. A. Bar-Noy and S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems," *Proc. 1992 Symp. Parallel Algorithms and Architectures*, ACM, New York, 1992, pp. 13-22.
8. D. Culler et al., "Towards a Realistic Model of Parallel Computation," *Proc. Fourth ACM SIGPlan Symp. Principles and Practice of Parallel Programming (PPOPP)*, ACM, New York, 1993, pp. 1-12.
9. J. Bruck et al., "On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model," *Proc. 1994 Symp. Parallel and Distributed Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. PRO6427, 1994, pp. 594-602.
10. X. Lin, P.K. McKinley, and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2D Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 8, Aug. 1994, pp. 793-804.
11. D.K. Panda, S. Singal, and P. Prabhakaran, "Multidestination Message-Passing Mechanism Conforming to Base Wormhole Routing Scheme," *Proc. First Int'l Parallel Computer Routing and Communication Workshop*, Springer-Verlag, 1994, pp. 131-145.
12. R.V. Boppana, S. Chalasani, and C.S. Raghavendra, "On Multicast Wormhole Routing in Multicomputer Networks," *Proc. 1994 Symp. Parallel and Distributed Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. PRO6427, 1994, pp. 722-729.

Philip K. McKinley *has been an assistant professor in the Department of Computer Science at Michigan State University since 1990. He was a member of the technical staff at Bell Laboratories in Naperville, Illinois, from 1982 to 1990 (on leave of absence from 1985 to 1989). His current research interests include scalable architectures and software, communications libraries for parallel and distributed computing, multicast communication, high-speed network architectures and protocols, and parallel numerical algorithms.*

McKinley received a BS degree in mathematics and computer science from Iowa State University in 1982, an MS degree from Purdue University in 1983, and a PhD degree from the University of Illinois at Urbana-Champaign in 1989, the latter both in computer science. He is a member of the IEEE and ACM.

Yih-jia Tsai *is a PhD candidate in the Department of Computer Science at Michigan State University. His research interests include communication issues in parallel processing, parallel algorithms, network interfaces, and device-driver designs for high-speed networks.*

Tsai received a BS degree in mechanical engineering from National Taiwan University and an MS degree in computer science from Michigan State University. He is a member of SIAM.

David F. Robinson *has been an assistant professor in the Department of Computer Science at Quincy University since 1994. He was a member of the technical staff at ROLM Corporation in Austin, Texas, from 1983 to 1986. His current research interests include parallel and distributed computing, communications protocols for high-performance computing, scalable software, monitoring and performance evaluation of parallel systems, and computer science education.*

Robinson received a BS degree in computer science from the University of Michigan-Flint in 1983 and MS and PhD degrees in computer science from Michigan State University in 1989 and 1994, respectively. He is a member of the IEEE and ACM.

Readers can contact Philip McKinley and Yih-jia Tsai at the Department of Computer Science, Michigan State University, A714 Wells Hall, East Lansing, MI 48824-1027; e-mail {mckinley,tsaiyi}@cps.msu.edu. David Robinson can be contacted at the Department of Computer Science, Quincy University, 1800 College Ave., Quincy, IL 62301-2670; e-mail robinda@quincy.edu. Readers can visit a number of related papers and technical reports of the Communications Research Group at Michigan State University at ftp://ftp.cps.msu.edu/pub/crg and http://www.cps.msu.edu/~mckinley/crgweb.

Howard Rubin, Computer*'s software metrics area editor, coordinated the review of this article and recommended it for publication. His e-mail address is 71031.377@compuserve.com.*

Further reading

This article is not intended to be an exhaustive survey of the literature on collective communication in the field of wormhole-routed networks, which is growing rapidly. Descriptions of additional contributions and their references can be found in McKinley, Tsai, and Robinson,[1] which is a longer form of this article. Readers may also be interested in two earlier surveys[2,3] that describe work in related research areas.

References

1. P.K. McKinley, Y.-J. Tsai, and D. Robinson, "A Survey of Collective Communication on Wormhole-Routed Massively Parallel Computers," Tech. Rep. MSU-CPS-94-35, Michigan State Univ. Dept. Computer Science, 1994 (revised Feb. 1995). Also available on the WWW at http://www.cps.msu.edu/~mckinley.
2. S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, "A Survey of Gossiping and Broadcasting in Communication Networks," *Networks*, Vol. 18, 1988, pp. 319-349.
3. P. Fraigniaud and E. Lazard, "Methods and Problems of Communication in Usual Networks," Tech. Report 91-38, Laboratoire de l'Informatique Superieure De Lyon, France, 1991.