Global Combine Algorithms for 2-D Meshes With Wormhole Routing *

M. Barnett † R. Littlefield ‡ D. G. Payne § R. van de Geijn ¶

Abstract

The problem of performing a global combine (summation) operation on distributed memory computers using a two-dimensional mesh interconnect with wormhole routing is considered. We present algorithms that are asymptotically optimal for short vectors ($O(\log(p))$ for p processing nodes) and for long vectors ($O(n)$ for n data elements per node), as well as hybrid algorithms that are superior for intermediate n. The algorithms are analyzed using detailed performance models that include the effects of link conflicts and other characteristics of the underlying communication system. The models are validated using experimental data from the Intel Touchstone DELTA computer. We show that no one algorithm is optimal for all vector lengths; rather, each of the presented algorithms is superior under some circumstances.

1 Introduction

A common situation on distributed memory multicomputers is that each node possesses a vector of data and all of the data across all of the nodes must be combined in some fashion. We consider a particular method for combining data: the *global combine*. Formally, the global combine can be stated as follows: Given p processing nodes, each of which owns a vector of data, x_i, of length n, a vector y is computed, where each element of y is the result of applying a commutative and associative operator, $\oplus$, pointwise to the elements of the vectors:

$$(\forall\, i \,:\, 0 \leq i < n \,:\, y[i] = (\oplus\, j \,:\, 0 \leq j < p \,:\, x_j[i]))\,.$$

In our definition of the operation, a copy of the result vector, y, is replicated on every node.

*Partial funding was provided by Intel Supercomputer Systems Division.

†Computer Science Department, University of Idaho, Moscow, Idaho 83843, mbarnett@cs.uidaho.edu

‡Pacific Northwest Laboratory, P.O. Box 999, Richland, Washington 99352, rj_littlefield@pnl.gov Pacific Northwest Laboratory is operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under contract DE-AC06-76RLO 1830.

§Supercomputer Systems Division, Intel Corporation, 15201 N.W. Greenbrier Pkwy. Beaverton, Oregon 97006, payne@ssd.intel.com

¶Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712–1188, rvdg@cs.utexas.edu

The global combine operation appears frequently in applications using replicated data storage to accumulate matrix or vector elements. The replicated storage technique is simple and effective in many application domains, such as electronic structure and molecular dynamics methods in computational chemistry [6, 10]. The global combine is also useful as a component of other general purpose algorithms, including both dense and sparse linear algebra. It occurs in algorithms as simple as distributed matrix-vector multiply and as complex as reducing a matrix to condensed form [5] and conjugate gradient methods [1].

Global combine is communication intensive, and must be properly implemented in order to be efficient. It is important not only to provide a highly optimized library routine, but also to expose the techniques used to implement the combine, since frequently a customization must be performed to suit the application.

The two-dimensional (2-D) mesh with wormhole routing is an attractive interconnection architecture for distributed memory multicomputers. Because links between nodes are physically localized and of fixed degree, a mesh can be scaled to arbitrarily large configurations while retaining high link bandwidth. This is an advantage over competing architectures such as the hypercube, whose links become physically longer and more numerous in larger configurations. However, the hypercube directly supports the communications patterns that are naturally used by many recursive divide-and-conquer algorithms. This results in conflict-free communications, which makes algorithms for the hypercube relatively simple to implement and analyze (though not necessarily to construct). The state of the art for performing the global combine on hypercubes is described in [7, 11, 12]. As we show in this paper, global communications on a mesh are more complicated: one must balance the use of long-distance connections, which minimize startup costs, against the use of local connections, which minimize network conflicts. In this paper we demonstrate algorithms for the global combine operation that are asymptotically optimal for both small and large amounts of data, having costs that are $O(\log p)$ and $O(n)$, respectively, for p processing nodes and n data elements per node. We also introduce a hybrid algorithm that is not asymptotically optimal, but in practice outperforms the others for wide ranges of n and p. In addition, we show that a different algorithm, optimized for a hypercube, is often the fastest method for meshes containing $p = 2^d$ nodes, if care is taken to order the communications to minimize network contention. We have previously summarized some of the early stages of this work in conference proceedings [3, 4].

2 System Model and Notation

Our target system is assumed to be a 2-D (r rows by c columns) mesh comprising $p = rc$ processing nodes, each having communication links to only its horizontal and vertical neighbors. The nodes are numbered 0 to $p-1$ in row-major order. We assume that the message startup cost and bandwidth does not depend on the distance between nodes. This property closely approximates the behavior of current systems using hardware "wormhole routing", in which the hardware routing delay is negligible compared to software overhead for message startup. The Intel DELTA and its successor Paragon systems are examples of such systems. We assume that the network links between nodes are bidirectional and that the network may have excess bandwidth, enabling multiple messages to traverse a link in the same direction without conflict.

parameters	*pattern*	*algorithm(s)*	*section*
α_{1way}, β_{1way}	exactly one send or one receive	fanin/fanout	3.2
α_{xchg}, β_{xchg}	two nodes exchange data	bidirectional exchange	3.3
		recursive halving	3.4
α_{shft}, β_{shft}	shift data around a ring of nodes	bucket	3.7

Table 1: Summary of Communication Cases and Cost Parameters.

We assume that each node can send at most one message and receive at most one message at the same time. A variable amount of overlap between sending and receiving can be modeled by our cost formulation, from complete overlap to none.

Simple models of communication assume that the time to send a message of L units is

$$\alpha + \beta L$$

where α represents the latency —the amount of time every message requires as overhead— and β represents the amount of time to send one unit. While useful as a rough measure, it is not detailed enough for our purposes. Various algorithms differ in the way they drive the underlying communication system. In some algorithms, each node either just sends or just receives in a single step; in others, each node both sends and receives. Some algorithms perform only two-node exchanges; others shift data around a ring of many nodes. On some computer systems, these differences can significantly affect startup and transfer costs, so that models that do not distinguish the various cases can be significantly wrong. Our models capture these effects by identifying the pattern of communication used by each algorithm, and allowing a separate α and β for each pattern: α_i and β_i with $i \in \{1way, xchg, shft\}$. The patterns and the algorithms that use them are summarized in Table 1.

Another source of inaccuracy arises from identifying the bandwidth each node possesses with the bandwidth of the communication network. In reality, the bandwidth of the communication network may be greater than the bandwidth from a node to the network. The latter is represented by the different parameters for each pattern; we use β_{net} for the former. A more accurate model for the per-unit cost of a message is

$$\max(\beta_i, k\beta_{net})$$

where k is the maximum number of messages contending for a single network communication link, and i is the communication pattern. In the absence of contention, this simplifies to β_i; since many algorithms on hypercubes do not induce contention, the simplified formula suffices. However, on a mesh-connected architecture, it may not be possible to avoid network contention and the more detailed model is necessary. Models which do not reflect the contention are unduly optimistic; contention can severely degrade the performance of an algorithm.

There is always a relationship between β_i and β_{net}: the ratio indicates the amount of excess bandwidth the communication network possesses. For simplicity, we assume there is a non-negative integer ν such that $2^\nu \beta_{net} = \beta_i$, i.e., the communication network can transfer elements 2^ν times faster than the link between a processing node and the communication network. On

parameter	*meaning*	*note*
α_i	startup cost	per step (depending on the pattern)
β_i	transfer cost with no link conflicts	per element (depending on the pattern)
β_{net}	transfer cost on a saturated link	per element
γ	combine cost	per element
n	length of vector	
p	total number of nodes	
r	number of rows in a 2-D mesh	
c	number of columns in a 2-D mesh	
m	the minimum of r and c	
d	$\log(p)$	

Table 2: Summary of Global Combine Parameters

the Intel Delta, the amount of excess bandwidth depends on the communication pattern that the nodes are engaged in. For a node that is either just sending a message or just receiving a message $\nu = 0$, i.e., there is no excess bandwidth. When nodes are trying to both send a message and receive a message, $\nu = 1$ is a reasonable approximation.

We use γ to represent the amount of time for a single combine operation on two elements. All of the parameters for the global combine are summarized in Table 2.

Each algorithm is modeled as a sequence of synchronous steps. Each step consists of sending, receiving, and combining L elements with an overall maximum of k messages requiring the same link. The time for each step is:

$$\alpha_i + L \max(\beta_i, k\beta_{net}) + L\gamma \, .$$

This model amounts to assuming that message startups can be done in parallel (across multiple nodes), but that conflicting messages share the bandwidth of the network link. It also assumes that communication and computation are not overlapped.

We summarize here the notation used in this paper. All logarithms are with respect to base two, and unless otherwise indicated, are to be understood as integer ceilings if the argument is not an integer power of two. Given the binary representation of a number, the bits are numbered from least-significant to most-significant. With r, c, and p defined as above, we denote the minimum of r and c as m. We also use d for $log(p)$. The length of the vector to be combined is always n. When p is a power of two, the mesh is called P2M (power-of-two), otherwise the mesh is called AM (arbitrary).

3 Algorithms & Performance Models

In this section we present the lower bound for the problem, and then describe six different methods of performing a global combine on a 2-D mesh. Our results apply naturally to 1-D meshes as well; in our formulas, they appear as a special case.

3.1 Lower Bounds

We start by giving some theorems that indicate a lower bound on the time required to perform a global combine.

Theorem 1 *Under our assumptions, a global combine involving p nodes requires that each node send and receive at least* $\log_2(p)$ *messages, yielding a communication startup term of* $k \log_2(p) \alpha_i$ *for some* α_i *and corresponding* k.

Proof: Since each node cannot send or receive more than one message at a time, the best possible schedule for communications organizes the nodes in a minimum spanning tree which has a depth of $\log_2(p)$. There are several algorithms that have the same depth tree but differing methods of driving the communication system. $\Box$

Theorem 2 *Under our assumptions, a global combine of vectors of length n requires at least time* $\frac{p-1}{p} n\gamma$. *Moreover, when we ignore communication startup, a strategy that yields this term for computation leads to a time complexity of at least*

$$2\frac{p-1}{p} n\beta_i + \frac{p-1}{p} n\gamma \, .$$

Proof: A single node can perform the global combine in time $(p-1)n\gamma$. If all nodes do an equal part of this computation, the time is reduced to $((p-1)n\gamma)/p$. For any approach that yields this perfect load balancing, $((p-1)n)/p$ elements must leave each processor, and $((p-1)n)/p$ of the result vector must arrive at each processor. Further, the elements leaving each processor must be received at other processors, and likewise the results arriving at each processor must be sent by others. Thus, the total number of elements sent and received by each processor is $4((p-1)n)/p$, and the corresponding time is $2(((p-1)n)/p)\beta_i$, where β_i is the time to send one element and receive another. $\Box$

Thus, the overall lower bound for the global combine (which ignores contention in the communication network) is

$$T_{opt}(n,p) \quad = \quad \log(p)\alpha_i + 2\frac{p-1}{p} \, n \, \beta_i + \frac{p-1}{p} \, n \, \gamma \, . \tag{1}$$

None of the algorithms discussed in this paper can achieve the entire lower bound for arbitrarily large meshes; when we say that an algorithm is asymptotically optimal, we mean that it achieves the optimal coefficient for either the startups or the transfer and combining times. Because the latency on message passing computers overwhelms the transfer time per element, when combining short vectors the first term dominates; an optimal algorithm uses the minimal number of steps to complete the global combine (since the number of startups is independent of n). It is only when n is large that the second and third terms dominate; an optimal algorithm in this case may use more steps, but achieve optimality with respect to the data transfer and computation time.

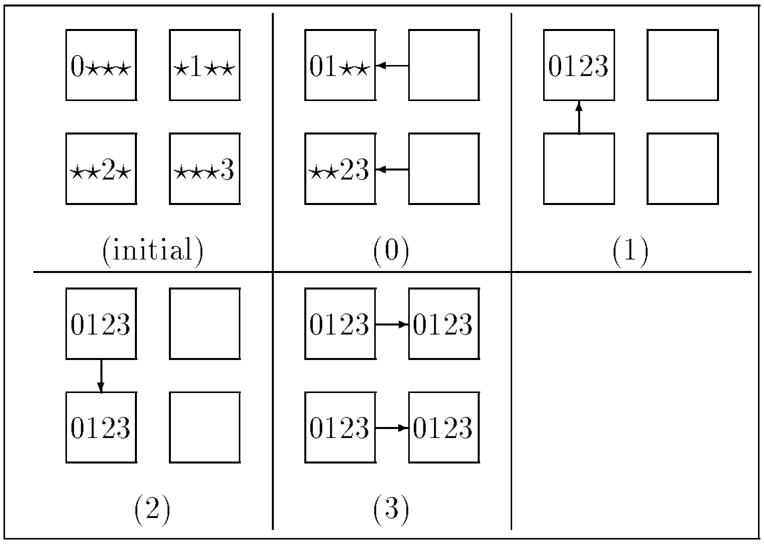

Figure 1: Fanin/Fanout for $r = 2$ and $c = 2$. Nodes that become inactive are drawn blank. For the first $\log(p)$ steps (here, steps 0 and 1), the receiving nodes combine the received vector with the resident vector. During the last $\log(p)$ steps (here, steps 2 and 3), the receiving nodes replace the resident vector with the received vector.

3.2 Version 1: Fanin/Fanout

A very simple first approach to the global combine embeds a minimum spanning tree in the 2-D mesh, combines to the root, and broadcasts the result to all other nodes, again utilizing the minimum spanning tree. Network conflicts are avoided by transferring data over short distances when many nodes are active and longer distances when fewer nodes are active [2].

The fanin stage of this algorithm has $d = \log(p)$ steps. View each node's index number in binary. Initially, all nodes are active. During step i, active nodes whose bit number i is 1 send their local data to the node whose index differs only in the ith bit, then become inactive waiting to receive a message during the fanout stage. Receiving nodes combine the incoming data into their local data, then proceed to the next step. Fanout is done by reversing the sequence of communication steps so as to broadcast the final result. An example for four nodes with $r = 2$ and $c = 2$ is shown in Figure 1. The numbers in the figure indicate the contribution to the vectors from the corresponding nodes. Thus the complete result vector, y, is denoted by "0123", while the initial conditions show each node possessing just its own vector.

Note that the algorithm operates as if the p nodes were in a one-dimensional linear array, i.e., it does not explicitly refer to the rows and columns of the mesh. It is possible to use an explicitly two dimensional formulation: first perform a fanin on each column to a single row, then a fanin/fanout on that row, followed by a fanout on each column. On a P2M, using any shortest-distance routing algorithm, both methods are conflict-free.

The algorithm can also be extended for an arbitrary mesh by having each node skip steps in which its nominal partner does not exist. Care must be taken, because with some routing

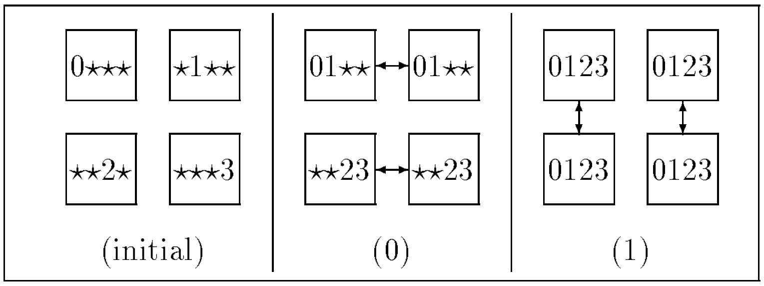

Figure 2: Bidirectional Exchange for $r = 2$ and $c = 2$. No nodes become inactive in this algorithm. During all of the steps, receiving nodes combine the received vector with the resident vector.

algorithms (not those on the DELTA), using the one-dimensional formulation on an arbitrary mesh may introduce link conflicts, if done improperly. The two-dimensional formulation avoids these conflicts, but it can have more steps than the one-dimensional case (since when p is not a power of two, $\lceil \log(r) \rceil + \lceil \log(c) \rceil \geq \lceil \log(rc) \rceil$). More steps means higher startup costs and since fanin/fanout is primarily useful for small n, when startup costs are dominant, the one-dimensional formulation is normally preferred. The details of such an explicit two-dimensional method are beyond the scope of this paper.

Assuming that there are no significant link conflicts, the cost (time complexity) of the fan-in, fan-out approach is

$$T_{fan}(n,p) \;=\; 2\log(p)\alpha_{1way} + 2\log(p)\; n\; \beta_{1way} + \log(p)\; n\; \gamma\,. \tag{2}$$

3.3 Version 2: Bidirectional Exchange

One problem with fanin/fanout is that it underutilizes both processors and communication links. The number of active nodes is halved at each step (during the fanin), and in each step, each active node either just sends or just receives. However, by replacing the one-way communications of Version 1 with bidirectional exchanges of the full vectors, every node is kept busy, and all nodes possess the result after only $\log(p)$ steps. As a result, some of the nodes now perform redundant computations, but more communication can be done in parallel and the algorithm requires fewer overall steps. The algorithm for four nodes with $r = 2$ and $c = 2$ is shown in Figure 2.

Bidirectional exchange works only on a P2M, and network conflicts cannot be avoided. A simple scheme for keeping conflicts to a reasonably low level is to have nodes exchange vectors only within their own column or row. The usual technique of choosing partners whose indices differ in one bit position accomplishes this, given row-major numbering of nodes. (However, this is not an optimal scheme for choosing partners, as we discuss in Section 3.6.)

Because the amount of data exchanged is constant, these exchanges can be performed in any order without affecting performance. However, to be consistent with recursive halving (described in Section 3.4), we choose an ordering that alternates between communicating within rows and within columns as long as possible, and that progressively increases the distance between partners in each case. Then the maximum number of messages contending for any link on step i, $0 \le i < d$, is

$$\chi(i) \quad = \quad 2^{\max(\lfloor i/2 \rfloor, i - \log(m))} \,. \qquad (3)$$

In fact, there may be as many as 2^{d-i-1} such links.

The cost of bidirectional exchange is the sum of all steps of the algorithm:

$$T_{P2Mbex}(n,p) \quad = \quad \sum_{i=0}^{d-1} [\alpha_{xchg} + n \max(\beta_{xchg}, \chi(i)\beta_{net}) + n\gamma] \,. \qquad (4)$$

For purposes of numerical modeling, an explicit summation like this is convenient and flexible. For example, with this approach it is easy to model the effects of using different communication protocols at various steps of the algorithm, an enhancement that can make a significant improvement on some machines [9].

However, to gain more insight, it is helpful to reduce the summation to a closed-form expression:

$$T_{P2Mbex}(n,p) \quad = \quad \log(p)\alpha_{xchg} + \mathcal{C}_\beta \; n \; \beta_{xchg} + \log(p) n\gamma \qquad (5)$$

where

$$\begin{array}{rcll} \mathcal{C}_\beta & = & \frac{r+c}{2^\nu} + 2\nu - 2 & \textbf{if } 0 \le \nu \le \log(m) \\ & = & \frac{\max(r,c)}{2^\nu} + \log(\min(r,c)) + \nu - 1 & \textbf{if } \log(m) \le \nu \le (d-1) - \log(m) \\ & = & \log(p) & \textbf{if } (d-1) - \log(m) \le \nu \,. \end{array}$$

The three cases in $\mathcal{C}_\beta$ arise from a consideration of the size of the mesh and the amount of the excess bandwidth. Remember from Section 2 that m is the minimum of r and c. The third case applies to meshes that are small enough for the communication network to provide contention-free communication by its excess bandwidth. The second case is for meshes that are long skinny rectangles, the extreme case being a linear array. For linear arrays—when either $r = 1$ or $c = 1$—it simplifies to approximately $\frac{p}{2}$. (The expression $(d-1) - \log(m)$ is equal to $\max(r,c)/2$.) The first case is the most interesting; it holds when the mesh is large relative to the available bandwidth. Note that if the mesh is a *large square mesh* (LSM), then $\mathcal{C}_\beta$ simplifies to approximately $\sqrt{p}$. Comparing the first and third cases shows how important it is to accurately model network contention.

When $\nu = 1$, as on the Delta, $\mathcal{C}_\beta$ simplifies to

$$\begin{array}{rcll} \mathcal{C}_\beta & = & \frac{r+c}{2} & \textbf{if } 2 \le \min(r,c) \\ & = & \frac{\max(r,c)}{2} + \log(\min(r,c)) & \textbf{if } \min(r,c) \le 2 \text{ and } 4 \le \max(r,c) \\ & = & \log(p) & \textbf{if } \max(r,c) \le 4 \,. \end{array}$$

3.4 Version 3: Recursive Halving

The third method for performing a global combine on a 2-D mesh reduces the redundant computation and communication incurred in bidirectional exchange. In recursive halving, nodes successively half the vector at each step. The pattern of communications is the same as that for bidirectional exchange, but the node pairs exchange and carry out the combine on opposite halves of the vector. Thus, at the end of $d = \log(p)$ steps, the global sum of the original vectors is distributed among the nodes, with n/p of the data on each node. Another $\log(p)$ steps of communications redistributes the result so that the entire vector sum resides on all of the nodes. Figure 3 shows the pattern of communication for four nodes with $r = 2$ and $c = 2$. The vectors can be considered as containing four (in general p) "pieces". Note that after the second (in general $\log(p)$) step, the result has been computed, with each node possessing one fourth (in general $1/p$-th) of the complete result.

Unlike bidirectional exchange, the order of exchanges matters with recursive halving. Contention is reduced by sending long messages over short distances, and by alternating between communicating within rows and within columns. Thus the number of conflicting messages is doubled only every two steps of the algorithm, while the message lengths are halved on each step.

Using Equation 3 for the contention, the cost of recursive halving using this ordering is

$$T_{P2Mrh}(n,p) = \tag{6}$$
$$\sum_{i=0}^{d-1}[\alpha_{xchg} + \frac{n}{2^{i+1}}\max(\beta_{xchg}, \chi(i)\beta_{net}) + \frac{n}{2^{i+1}}\gamma] \; + \sum_{i=0}^{d-1}[\alpha_{xchg} + \frac{n}{2^{i+1}}\max(\beta_{xchg}, \chi(i)\beta_{net})]$$

where the first summation models the $\log(p)$ combining steps, and the second summation models the subsequent $\log(p)$ collect steps.

The closed form is

$$T_{P2Mrh}(n,p) \;=\; 2\log(p)\alpha_{xchg} + \mathcal{C}_\beta \; n \; \beta_{xchg} + \frac{p-1}{p}n\gamma \tag{7}$$

where

$$\begin{aligned}
\mathcal{C}_\beta &= 2 + \tfrac{1}{2^{2\nu}} + \tfrac{d-2\log(m)-3}{m2^{\nu}} && \textbf{if } 0 \le \nu \le \log(m) \\
&= 2 + \tfrac{d-\log(m)-\nu-2}{2^{\nu}} && \textbf{if } \log(m) \le \nu \le (d-1)-\log(m) \\
&= 2\tfrac{p-1}{p} && \textbf{if } (d-1)-\log(m) \le \nu\,.
\end{aligned}$$

As in Section 3.3, the third case is for meshes small enough to avoid any contention, while the second case refers to skinny rectangles.

When $\nu = 1$, as on the Delta, $\mathcal{C}_\beta$ simplifies to

$$\begin{aligned}
\mathcal{C}_\beta &= 2 + \tfrac{1}{4} + \tfrac{d-2\log(m)-3}{2m} && \textbf{if } 2 \le \min(r,c) \\
&= \tfrac{d-\log(m)+1}{2} && \textbf{if } \min(r,c) \le 2 \text{ and } 4 \le \max(r,c) \\
&= 2\tfrac{p-1}{p} && \textbf{if } \max(r,c) \le 4\,.
\end{aligned}$$

In the absence of link conflicts, the coefficient corresponding to $\mathcal{C}_\beta$ is that of the third case: $2(p-1)/p$. Thus, on the mesh, in the worst case ($\nu = 0$), link conflicts result in no worse than

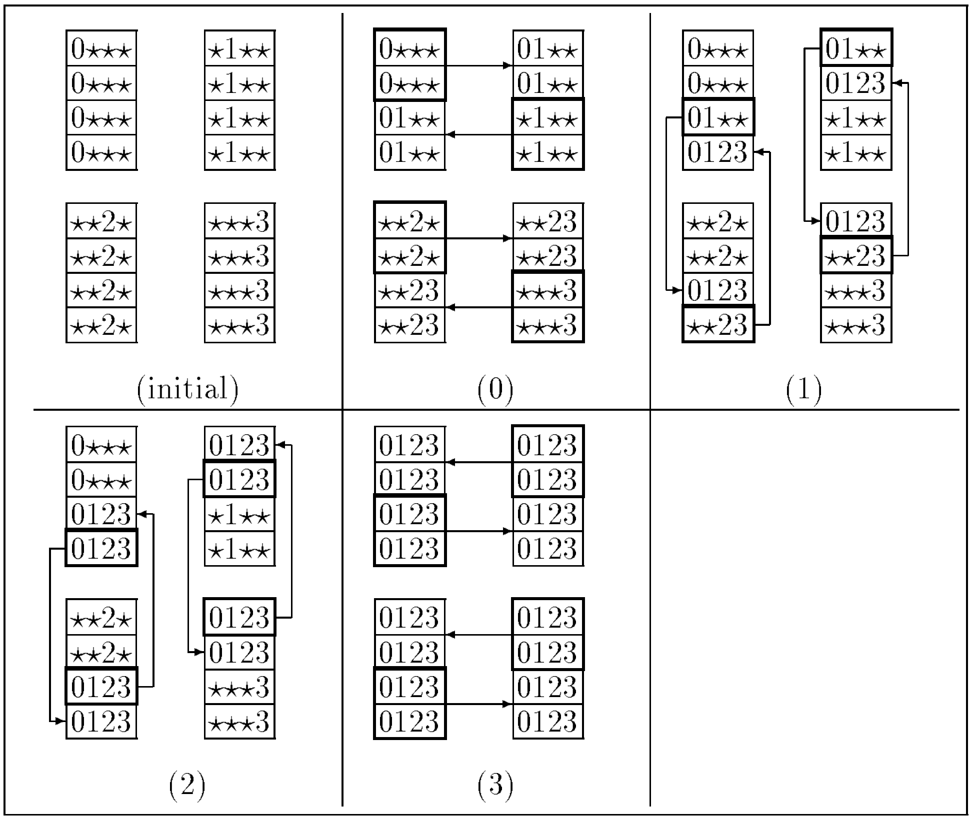

Figure 3: Recursive Halving for $r = 2$ and $c = 2$. The intial state shows the vectors split into 4 pieces. Each piece contains $n/4$ elements. In step 0, pairs of neighboring nodes exchange within each row. (The pieces that are sent in each step are outlined.) The convention is that nodes on the left send the top half, while nodes on the right send the bottom half. In step 1, pairs of neighbors exchange within each column. The convention is that nodes on the top send the top half; nodes on the bottom send the bottom half. After each of these steps, the received elements are combined into the appropriate part of the vector. After step 1, the entire vector is combined, but the results are distributed across all of the nodes. The last two steps redistribute the results in the reverse order of the combine steps. Thus, in step 2, node pairs in columns exchange, while in step 3, node pairs in rows exchange. The same pattern is used when the mesh is bigger: the difference is that the distance between the nodes in each pair increases during the combine phase and decreases during the redistribution phase.

a 50% cost penalty with this ordering and choice of partners for the exchanges. If the links have excess bandwidth ($\nu > 0$), the penalty is less. On the Delta, this cost model predicts that recursive halving is within 12.5% of optimal.

3.5 Version 4: Halving/Exchange Hybrid

There is a tradeoff between recursive halving and bidirectional exchange — recursive halving does no redundant computation and has less data transfer, but bidirectional exchange has fewer communication startups. Thus, recursive halving is faster for large n, while exchange is faster for small n.

These two algorithms can be combined in a natural way to yield a hybrid algorithm that switches from halving to exchange when the vector length has been reduced to a point where the extra startups become counterproductive. In general, the hybrid first does k combining steps of recursive halving, which partially combines the data, then $d-k$ steps of bidirectional exchange, which completes the combining and does part of the broadcast, and finally k broadcast steps of recursive halving to finish the broadcast.

Again, using Equation 3, the cost of this halving/exchange hybrid is

$$\begin{aligned} T^{k}_{P2Mheh}(n,p) = & \\ & \sum_{i=0}^{k-1}[\alpha_{xchg} + \frac{n}{2^{i+1}}\max(\beta_{xchg}, \chi(i)\beta_{net}) + \frac{n}{2^{i+1}}\gamma] \; + \\ & \sum_{i=k}^{d-1}[\alpha_{xchg} + \frac{n}{2^{k}}\max(\beta_{xchg}, \chi(i)\beta_{net}) + \frac{n}{2^{k}}\gamma] \; + \\ & \sum_{i=0}^{k-1}[\alpha_{xchg} + \frac{n}{2^{i+1}}\max(\beta_{xchg}, \chi(i)\beta_{net})] \end{aligned} \tag{8}$$

which is equivalent to

$$T_{P2Mrh}(2^k, n) + T_{P2Mbex}(p, \frac{n}{2^k}) - T_{P2Mrh}(2^k, \frac{n}{2^k}).$$

This hybrid can be optimized by choosing k to yield the smallest cost. Typically $k = 0$ for small amounts of data, $k = d$ for large amounts, and $0 < k < d$ for intermediate values. The resulting cost for the hybrid is the same as recursive halving and bidirectional exchange at the respective extremes, and lower for intermediate n.

On hypercubes, a formula for the optimal k has been derived [11, 12]. On the mesh, the many relationships between parameter values complicates the analytic determination of k. However, the optimal k for a specific n can easily be determined numerically due to the small number of possible values to be considered. An alternate approach, with some advantages in practice, is to precompute the ranges of n for which each k is optimal.

3.6 Better Exchange Patterns

The simple scheme for choosing a sequence of exchange partners, described above, has the deficiency that during each step, half of the communication links are unused — the exchanges are either all horizontal or all vertical. It is possible to utilize more of the links by doing some horizontal and some vertical exchanges in the same step. This requires more complex schemes of choosing partners.

One such scheme is to alternate between horizontal and vertical transfers using a checkerboard pattern, in which "white" nodes and "black" nodes communicate in different directions whenever possible.

The checkerboard scheme can be conflict-free through the first four exchange steps, whereas the simple column and row scheme starts getting conflicts after the first two steps. After four steps, the checkerboard scheme gets conflicts too, but the number of conflicting messages per link is only half that of the simple scheme.

The analysis and validation of the checkerboard and other improved schemes will be described in a future paper. All data shown in the current paper are for the simple scheme.

3.7 Version 5: Buckets

Another method for performing a global combine on a 2-D mesh embeds a set of uni-directional rings in the mesh. The formulation of the buckets algorithm is explicitly two-dimensional. The first stage of the algorithm is executed along one dimension, say within each column independently (as a sub-ring of r nodes). On each node, the local data is divided into r equal buckets. There are $r-1$ steps required to circulate the buckets around the ring, accumulating the result on the way and producing a result that is distributed among the nodes, with each node possessing one bucket that contains the sum of that bucket along the entire column. Next, stage one is repeated, but now along each row independently (as a sub-ring of c nodes) using the bucket that all of the nodes in that row share in common as the entire vector. At the end of stage 2, the vector has been combined, with the result being distributed among all p nodes. The process is then reversed, to communicate the full combined vector to all nodes. A simple example for one row of 4 nodes is shown in Figure 4.

This algorithm generates no network conflicts and naturally handles arbitrary meshes. Its cost is

$$\begin{aligned} T_{AMbkt}(n,p) = & \\ & \sum_{i=1}^{r-1}[\alpha_{shft} + \frac{n}{r}\beta_{shft} + \frac{n}{r}\gamma] + \sum_{i=1}^{c-1}[\alpha_{shft} + \frac{n}{p}\beta_{shft} + \frac{n}{p}\gamma] + \\ & \sum_{i=1}^{c-1}[\alpha_{shft} + \frac{n}{p}\beta_{shft}] + \sum_{i=1}^{r-1}[\alpha_{shft} + \frac{n}{r}\beta_{shft}] \end{aligned} \tag{9}$$

which has the closed form

$$T_{AMbkt}(n,p) \quad = \quad 2(r+c-2)\alpha_{shft} + 2\frac{p-1}{p}n\beta_{shft} + \frac{p-1}{p}n\gamma \,. \tag{10}$$

Note that there is optimal data transfer, but a relatively large number of message startups. This algorithm is the fastest approach for large n, but the slowest for small n.

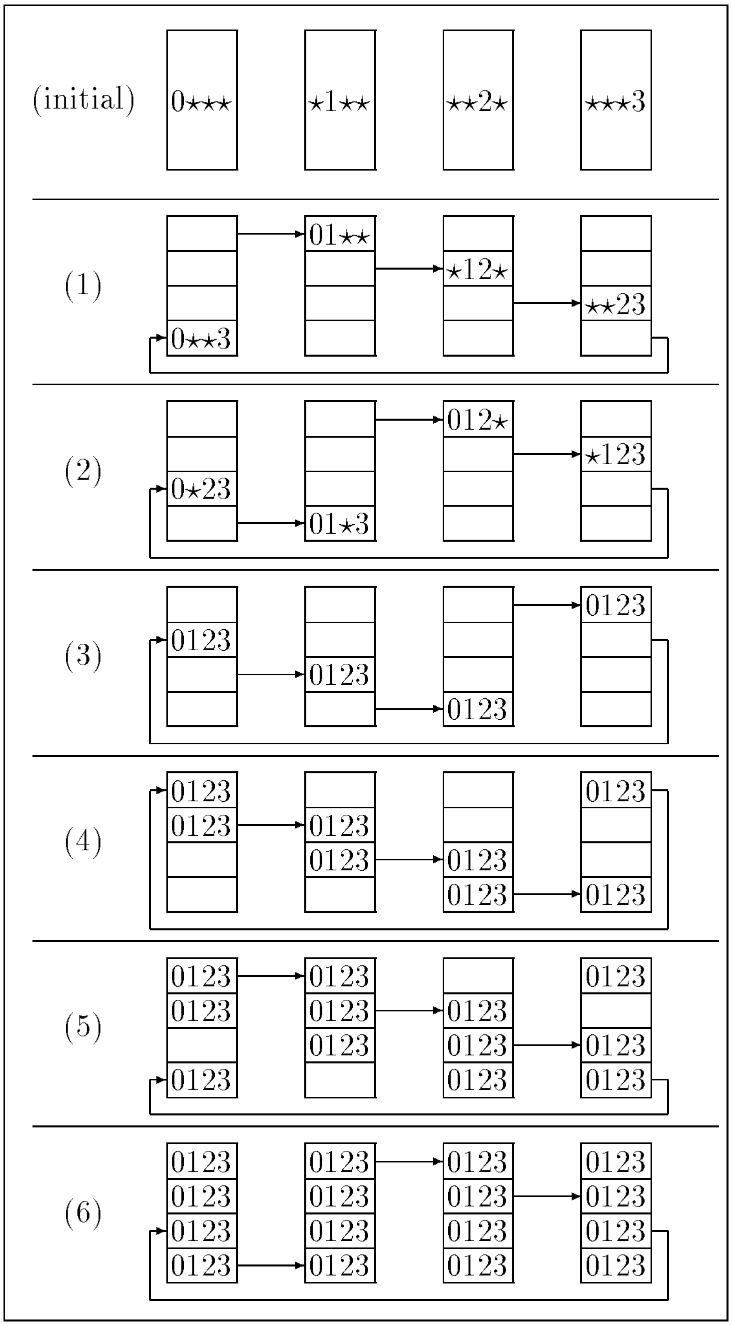

Figure 4: Buckets for $r = 1$ and $c = 4$. During the first 3 steps, nodes combine the received elements with the corresponding elements of the resident vector. After step 3, the result has been formed, but as in recursive halving, it is distributed across all of the nodes. If there were more than one row, the algorithm would then be performed with each column forming an independent ring. The nodes within a column would all have the same "bucket": the algorithm would be run completely (for $2(r-1)$ steps) resulting in the same picture as here. The last 3 steps distribute the buckets around the ring until all of the nodes possess the complete result.

3.8 Version 6: Buckets/Fan Hybrid

The 2-D buckets algorithm described above (version 5) can be viewed as a divide-and-conquer strategy. The first set of tasks (combining within each column) generates a second set of smaller tasks (combining within each row). In version 5, all tasks are completed in the same way, using a 1-D buckets algorithm. However, because the data length and node counts are different for the first set of tasks than for the second, different algorithms may be faster for the two cases. Exploiting this difference leads to various hybrid algorithms.

One useful hybrid is obtained by substituting the 1-D fanin/fanout algorithm in place of the buckets algorithm, to combine within a row. This hybrid buckets/fanin algorithm works with arbitrary mesh sizes. Its cost on an arbitrary mesh is

$$T_{AMbf}(n,p) = \tag{11}$$
$$2(r-1)\alpha_{shft} + 2\frac{r-1}{r}n\beta_{shft} + \frac{r-1}{r}n\gamma + 2\log(c)\alpha_{1way} + 2\log(c)\frac{n}{r}\beta_{1way} + \log(r)\frac{n}{r}\gamma\,.$$

Because fanin/fanout trades off higher transfer cost for a reduced number of startups, the substitution can be helpful in cases where the rows are long and the subvectors are relatively short. As will be illustrated in Section 4, this occurs over a wide range of n for large meshes on the DELTA. The substitution is particularly useful if $\alpha_{1way} < \alpha_{shft}$ or $\beta_{1way} < \beta_{shft}$, as on the DELTA.

Many other hybrid algorithms are possible and might be preferred under some circumstances, but are beyond the scope of this paper.

4 Experimental Results

In this section, we report experimental results gathered on the Intel Touchstone DELTA system for the various algorithms discussed in Section 3. We further compare those results with the performance model predictions presented in that section.

The Intel Touchstone DELTA computer system is a prototype 2-D mesh multicomputer that incorporates a 16×32 mesh of i860/XR processing nodes with a wormhole routing interconnection network [8]. Its communication characteristics are described in [9]. (The notation used in [9] is different from that used here. Our α_{1way} and β_{1way} are equivalent to what is described in [9] as a "loopback test"; our α_{xchg} and β_{xchg} are equivalent to a 2-processor "shift"; and our α_{shft} and β_{shft} are equivalent to a "shift" longer than 2 processors.)

Some of the DELTA's characteristics are particularly important to understand our experimental results. These are:

- The sender and the receiver of a message incur approximately equal startup costs, so that $\alpha_{xchg} \approx \alpha_{shft} \approx 2\alpha_{1way}$.
- When a node is just sending, it can essentially saturate a network link, so that $\beta_{1way} \approx \beta_{net}$. However, when a node is sending and receiving "simultaneously", its rate of injection of message packets into the network is approximately halved. This means that $\beta_{xchg} \approx$

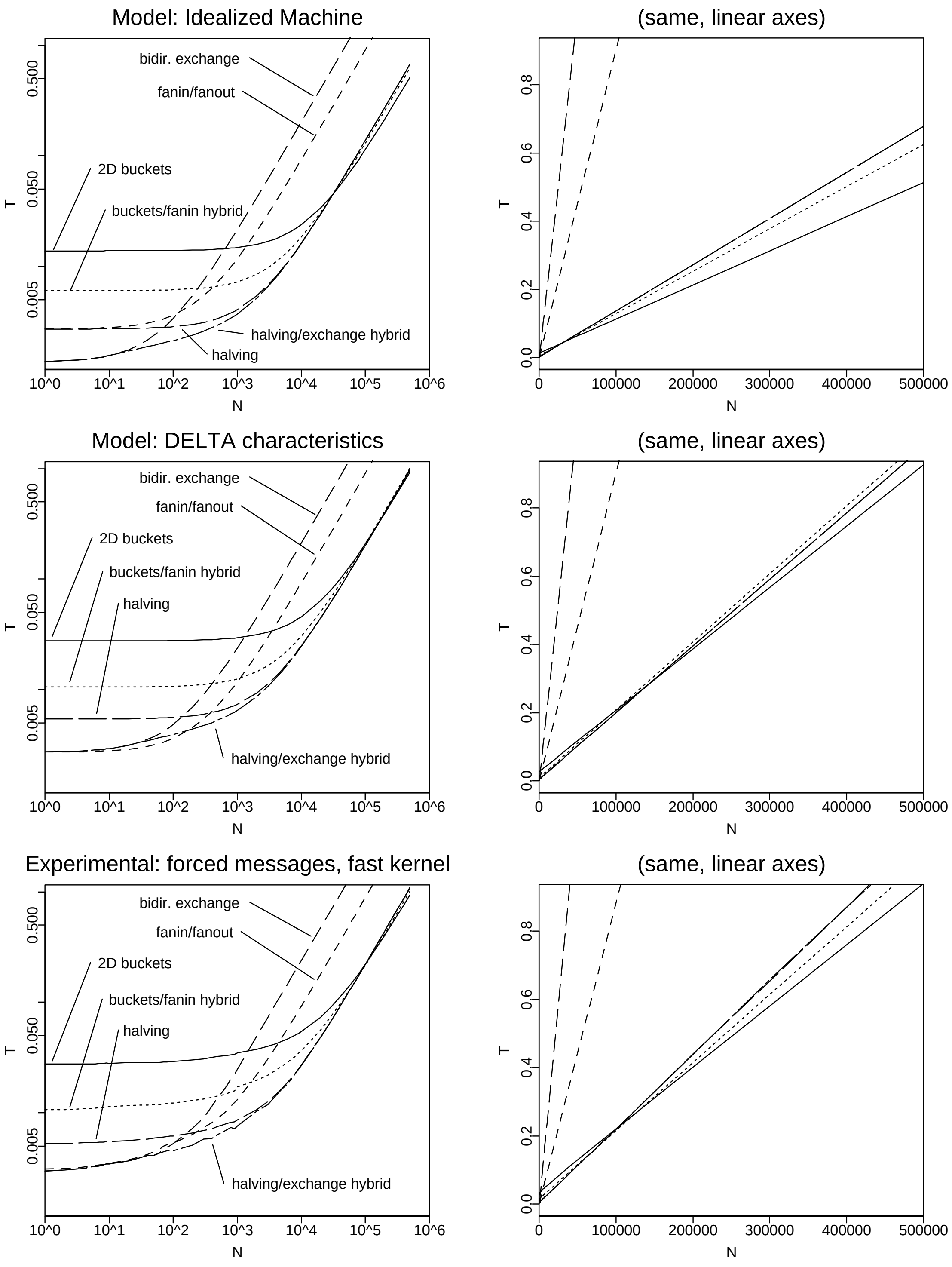

Figure 5: Performance of six algorithms, on a 16x32 mesh, as predicted by two sets of model assumptions, and as observed on the Intel DELTA system using "forced" messages and the "fast" kernel.

$\beta_{shft} \approx 2\beta_{net}$, i.e., $\nu = 1$ is a reasonable approximation for the algorithms described in this paper that can have link conflicts.

- All of our tests involved summation of double-precision (8 byte) floating point numbers, for which we measured $\gamma = 0.2\mu$sec.

- A variety of communication protocols are available on the DELTA, in common with other Intel systems such as the iPSC and Paragon. Except where noted, all experiments reported here were done with "forced" messages, which provide the lowest transfer cost (highest bandwidth), but also have higher effective startup cost due to the need for explicit synchronization messages. Typical values for forced messages are $\alpha_{1way} = 150\mu$sec and $\beta_{1way} = 0.4\mu$sec (per 8-byte element). Short messages are faster if "unforced", in which case typical values are $\alpha_{1way} = 75\mu$sec and $\beta_{1way} = 0.8\mu$sec. (Due to the extra flow control provided by unforced messages, they can also exhibit a significant difference between β_{xchg} and β_{shft}. This observation, while not illustrated in the current paper, is further justification for the appearance of β_{xchg} and β_{shft} as separate parameters.)

- Two operating system kernels are available on the DELTA [9]. In order to illustrate the performance of the basic DELTA technology, we used the "fast" kernel. Under the more robust production kernel, most timings would approximately double.

As shown in Figure 5, the first two characteristics have some effect on the relative performance of the various global combine algorithms. The top and middle rows of Figure 5 show the expected performance of all algorithms on a 16x32 mesh under two sets of model assumptions. The curves on the top assume an idealized machine in which message startups and data transfers are perfectly parallel and all transfers proceed at network bandwidth, i.e., $\alpha_{xchg} = \alpha_{shft} = \alpha_{1way}$, $\beta_{xchg} = \beta_{shft} = \beta_{1way}$, $\nu = 0$. The curves in the middle row assume characteristics that more accurately reflect the DELTA, i.e., $\alpha_{xchg} = \alpha_{shft} = 2\alpha_{1way}$, $\beta_{xchg} = \beta_{shft} = 2\beta_{1way}$, $\nu = 1$. (In both cases, the values for α_{1way}, β_{1way}, and γ are those of the DELTA.) Comparing the top and middle rows of Figure 5, note in particular the relative performance of fanin/fanout, bidirectional exchange, and halving.

On a power-of-two mesh (P2M), the halving/exchange hybrid (with optimum k) is seen to be usually the fastest method under both models. For very long vectors, the 2-D buckets algorithm is faster. The advantage of the buckets algorithm is diminished when there is excess network bandwidth, as in the middle and bottom graphs.

On a non-power-of-two mesh, however, fanin/fanout, buckets, or the buckets/fanin hybrid must be used because the other algorithms don't work. In that case, the DELTA's characteristics tend to favor fanin/fanout and the buckets/fanin hybrid over the pure buckets algorithm, because of the relatively smaller values of α_{1way} and β_{1way}.

The bottom row of Figure 5 shows experimental data from the DELTA that can be compared to the top and middle rows. The experimental data agree well with the DELTA model, but differ substantially from the idealized machine model. Two discrepencies even from the DELTA model must be noted. First, the DELTA model (middle row of graphs) predicts that recursive halving will slightly outperform the buckets/fanin hybrid for long vectors; in practice, the advantage is a few percent the other way. The reason is that the integer $\nu = 1$ is not completely adequate to describe the DELTA; a fractional value would be more accurate (and less tractable to analyze). Second, the DELTA model predicts that there should be a range of small n in which fanin/fanout should be fastest, due to link conflicts in the other algorithms. This is not observed in practice because of the compensating fact that the DELTA's α_{xchg} is actually slightly smaller than the model's $2\alpha_{1way}$.

Figure 6 illustrates a typical situation in choosing an algorithm for a non-power-of-two mesh. Here, any one of three algorithms (fanin, 2-D buckets, or buckets/fanin hybrid) can be best, depending on the data vector length. Whether messages should be forced or not also depends on the vector length, and in practice it may be useful to force some messages and not others within a single algorithm. A discussion of these issues is beyond the scope of this paper. The important point is that performance of the algorithms can be predicted accurately enough to choose the best algorithm for the circumstances.

5 Conclusions

In this paper, we have shown that it is possible to perform efficient global operations on mesh-connected architectures with wormhole routing. However, achieving good performance over a wide range of vector lengths and mesh sizes is more complex than on the hypercube architecture.

For meshes containing a power-of-two nodes, the hybrid algorithm combining recursive halving and bidirectional exchange is the fastest approach except for very long vectors, when the 2-D buckets algorithm is best. Excess network bandwidth allows recursive halving to be competitive even with long vectors.

For arbitrary mesh sizes, we have discussed one algorithm that is asymptotically optimal for short vectors (fanin/fanout, $O(\log(p))$) and another that is optimal for long vectors (buckets, $O(n)$). However, each algorithm performs badly in the other case. We have also introduced a hybrid algorithm (buckets/fanin) that outperforms both of the other algorithms over a wide range of mesh sizes and vector lengths.

Thus, to achieve the best possible performance over a full range of vector lengths and mesh sizes, it appears necessary to provide a family of global combine algorithms optimized for different circumstances. Fortunately, we have found that the performance of individual algorithms can be accurately predicted from the mesh size, vector length, and a set of easily measured parameters. This allows the complexity of choosing the appropriate algorithm to be hidden inside library routines with simple user interfaces.

We believe that the insights and analysis techniques presented in this paper have wide application in the design of fast algorithms for 2-D meshes with wormhole routing.

Acknowledgements

We thank the extremely helpful comments of the anonymous reviewers.

This research was performed in part using the Intel Touchstone Delta System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by the California Institute of Technology, Pacific Northwest Laboratory, and Intel Supercomputing Systems Division.

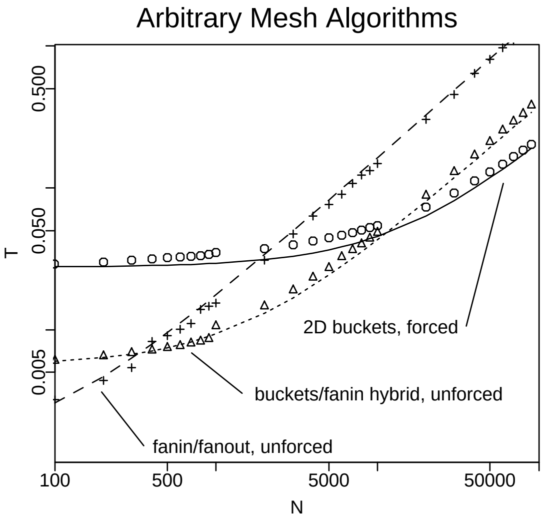

Figure 6: Observed and expected performance of three algorithms suitable for use on non-power-of-two meshes (measured on a 16x32 mesh). Where noted, unforced messages were used.

References

[1] D. Bailey, J. Barton, T. Lasinski, and H. Simon, The NAS Parallel Benchmarks, Report RNR-01-002, Jan. 1991.

[2] M. Barnett, D.G. Payne, and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. Computer Science report TR-91-38, Univ. of Texas, 1991.

[3] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn. Efficient Communication Primitives on Mesh Architectures with Hardware Routing, Sixth SIAM Conf. on Par. Proc. for Sci. Comp., Norfolk, Virginia, March 22-24, 1993.

[4] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn, Global Combine on Mesh Architectures with Wormhole Routing, to appear in the proceedings of the *7th International Parallel Processing Symposium*, Newport Beach, CA, April 13-16, 1993.

[5] J. Dongarra and R. van de Geijn, Reduction to Condensed Form on Distributed Memory Architectures, *Parallel Computing*, 18, pp. 973–982, 1992.

[6] M. Feyereisen and R. A. Kendall, An efficient implementation of the Direct-SCF Algorithm on Parallel Computer Architectures, Theoretica Chimica Acta, in press.

[7] C.-T. Ho and S. L. Johnsson, Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes, In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648, IEEE, 1986.

[8] S.L. Lillevik, The Touchstone 30 Gigaflop DELTA Prototype. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 671–677. IEEE Computer Society Press, 1991.

[9] R. Littlefield. Characterizing and Tuning Communications Performance on the Touchstone DELTA and iPSC/860. In *Proceedings of the 1992 Intel User's Group Meeting*, Dallas, TX, October 4-7.

[10] W. Smith, Molecular dynamics on hypercube parallel computers, Computer Physics Communications 62 (1991) 229-248.

[11] R.A. van de Geijn. Efficient global combine operations. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 291–294. IEEE Computer Society Press, 1991.

[12] R.A. van de Geijn, On Global Combine Operations, to appear in *Journal of Parallel and Distributed Computing.*