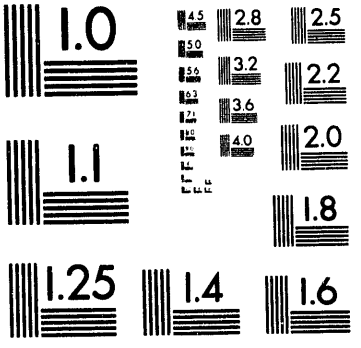
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

1 of 1

Conf-9304200--1

PNL-SA-22002

GLOBAL COMBINE ON MESH ARCHITECTURES WITH WORMHOLE ROUTING

M. Barnett
R. Littlefield
D. G. Payne
R. van de Geijn

April 1993

Presented at the
7th International Parallel
Processing Symposium
April 13-16, 1993
Newport Beach, California

Prepared for
the U.S. Department of Energy
Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

Global Combine on Mesh Architectures with Wormhole Routing *

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

Abstract

Several algorithms are discussed for implementing global combine (summation) on distributed memory computers using a two-dimensional mesh interconnect with wormhole routing. These include 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. Performance models are developed 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. Each of the combine algorithms is shown to be superior under some circumstances.

1 Introduction

The two-dimensional (2-D) mesh with wormhole routing is an attractive interconnection architecture for distributed memory multicomputers. The mesh is more scalable in some ways than competing architectures such as the hypercube. The challenge in doing global communication on a mesh is to 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. The state of the art for performing the global combine on hypercubes is described in [5]. Earlier work on 2-D mesh combining is reported in [2].

The global combine operation can be stated as follows: Given p processing nodes, each of which owns a vector of data, x_i, of length n, a global combine forms $y = \oplus_{i=0}^{p-1}(x_i)$, where $\oplus$ is a commutative and associative operator defined on the elements of the vectors. In this paper we choose to have a copy of the resulting y end up on every node.

2 System Model

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. Messages are sent with startup cost (latency) and bandwidth that do not depend on the distance between nodes (the wormhole routing property), through communication links that are bidirectional. The network may have excess bandwidth, enabling multiple messages to traverse a link in the same direction without conflict.

Our performance models assume that each algorithm operates in a sequence of synchronous steps that are parameterized as follows:

startup cost
α_{alg} per step (depending on the algorithm)
transfer cost with no link conflicts
β_{alg} per element (depending on the algorithm)
transfer cost on a saturated link
β_{net} per element
combine cost
γ per element

*Pacific Northwest Laboratory is operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under contract DE-AC06-76RLO 1830. Additional 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

§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

More precisely, suppose that a particular step of some algorithm *alg* requires transferring and combining a (sub)vector containing L elements, and that there are k messages contending for a saturated link. Then the time for that step is modeled as

$$T_{alg}(L,k) = \alpha_{alg} + L\max(\beta_{alg}, k\beta_{net}) + L\gamma$$

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.

To simplify some of the cost expressions, we assume that there exists some positive integer ν such that $2^{\nu}\beta_{net} = \beta_{alg}$. On the Intel Delta, in circumstances where it matters, $\nu = 1$ is a good approximation for all algorithms discussed in this paper.

Note that we do not assume that the time to send a message is $T(L) = \alpha + \beta L$ where α and β are independent of which algorithm is being executed. The reason for this is that various algorithms differ in the way they use the underlying communication system. In some algorithms, each node transfers one message per step; in others, each node transfers two. 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 the startup and transfer costs, so that models that do not distinguish the various cases can be significantly wrong.

3 Algorithms & Performance Models

In this section we describe six different methods of performing a global combine on a 2-D mesh.

3.1 Notation

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. Define r, c, and p as above, $m = \min(r,c)$, $d = log(p)$, and let n be the vector length per node to be combined. If p is restricted to $p = 2^d$, the mesh is called P2M (power-of-two), otherwise the mesh is called AM (arbitrary).

3.2 Version 1: Fanin/Fanout

The 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 transfering data over short distances when many nodes are active and longer distances when fewer nodes are active.

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.

On a P2M, using any shortest-distance routing algorithm, this fanin/fanout algorithm is conflict-free. Fanin/fanout can also be extended to handle an arbitrary mesh, simply by having each node skip steps in which its nominal partner does not exist. With some routing algorithms (not those on the DELTA), using this "1-D" formulation with non-power-of-two nodes can introduce link conflicts. These conflicts could be avoided by using an explicitly 2-D formulation: fanin on each column to a single row, fanin on the row, fanout on the row, and fanout on each column. However, the 2-D formulation can have more steps than the 1-D when p is not a power of two (since $\lceil\log(r)\rceil + \lceil\log(c)\rceil \geq \lceil\log(rc)\rceil$), and thus can have higher startup costs. Since fanin/fanout is primarily useful for small n, when startup costs are dominant, the 1-D formulation is normally preferred.

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

$$T_{fan} = 2\log(p)\alpha_{fan} + 2\log(p)n\beta_{fan} + \log(p)n\gamma$$

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 steps.

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 later), we choose an ordering so as to alternate between communicating within rows and within columns as long as possible, and so as to progressively increase the distance between partners in each case. Then the maximum number of messages contending for any link on step $i = 0..d-1$ is

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

The cost of bidirectional exchange can be obtained by summing over all steps of the algorithm:

$$T_{P2Mbex} = \sum_{i=0}^{d-1} [\alpha_{bex} + n \max(\beta_{bex}, \chi(i)\beta_{net}) + n\gamma]$$

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 [4].

However, to gain more insight, it is helpful to reduce the summation to a closed-form expression. To work in this direction, we use the behavior of max(...) in $\chi(i)$ to split the summation into two pieces:

$$T_{P2Mbex} =$$
$$\sum_{i=0}^{2\log(m)-1} [\alpha_{bex} + n \max(\beta_{bex}, 2^{\lfloor i/2 \rfloor}\beta_{net}) + n\gamma] +$$
$$\sum_{i=2\log(m)}^{d-1} [\alpha_{bex} + n \max(\beta_{bex}, 2^{i-\log(m)}\beta_{net}) + n\gamma]$$

One particularly interesting case is for a large square mesh (LSM). For a large square mesh, $2\log(m) = d$, so the second summation disappears. By using the assumption that $\beta_{bex} = 2^{\nu}\beta_{net}$, considering the behavior of max(...), and assuming that $\log(p) > 2\nu$, we can split the remaining summation into two pieces to yield:

$$T_{LSMbex} =$$
$$\sum_{i=0}^{2\nu-1} [\alpha_{bex} + n\beta_{bex} + n\gamma] +$$
$$\sum_{i=2\nu}^{\log(p)-1} [\alpha_{bex} + n2^{\lfloor i/2 \rfloor}(\beta_{bex}/2^{\nu}) + n\gamma]$$

which can be reduced to

$$T_{LSMbex} =$$
$$\log(p)\alpha_{bex} + (\tfrac{\sqrt{p}}{2^{\nu-1}} + 2\nu - 2)n\beta_{bex} + \log(p)n\gamma$$

With $\nu = 1$, as on the Delta with a large square mesh (DLSM), this further reduces to

$$T_{DLSMbex} = \log(p)\alpha_{bex} + \sqrt{p}\, n\beta_{bex} + \log(p)n\gamma$$

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, only half the vector is exchanged between the node pairs. The nodes exchange and carry out the combine on opposite halves of the vector. The pattern of communications is the same as that for bidirectional exchange except that the length of the vector is reduced by half at every step. 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.

Unlike bidirectional exchange, the order of exchanges matters with recursive halving. Contention is minimized 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.

The cost of recursive halving using this ordering is

$$T_{P2Mrh} =$$
$$\sum_{i=0}^{d-1} [\alpha_{rh} + \tfrac{n}{2^{i+1}} \max(\beta_{rh}, \chi(i)\beta_{net}) + \tfrac{n}{2^{i+1}}\gamma] +$$
$$\sum_{i=d-1}^{0} [\alpha_{rh} + \tfrac{n}{2^{i+1}} \max(\beta_{rh}, \chi(i)\beta_{net})]$$

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

Again, an interesting special case is for large square meshes, for which the cost is:

$$T_{LSMrh} = 2\log(p)\alpha_{rh} + f(\nu)n\beta_{rh} + \frac{p-1}{p}n\gamma$$

where

$$f(\nu) = (2 + \frac{1}{2^{2\nu}} - \frac{3}{\sqrt{p}\,2^{\nu}})$$

For $\nu = 1$, as on the Delta, the total cost on a large square mesh reduces to:

$$T_{DLSMrh} = 2\log(p)\alpha_{rh} + (\frac{9}{4} - \frac{3}{2\sqrt{p}})n\beta_{rh} + \frac{p-1}{p}n\gamma$$

In the absence of link conflicts, the coefficient corresponding to $f(\nu)$ would be $2(p-1)/p$. Thus, on the mesh, in the worst case ($\nu = 0$), link conflicts result in no worse than 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 work and has less data transfer, but bidirectional exchange has fewer communication startups. Thus, 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.

The cost of this halving/exchange hybrid is

$$T_{P2Mheh}(k) =$$
$$\sum_{i=0}^{k-1}[\alpha_{rh} + \tfrac{n}{2^{i+1}}\max(\beta_{rh}, \chi(i)\beta_{net}) + \tfrac{n}{2^{i+1}}\gamma] +$$
$$\sum_{i=k}^{d-1}[\alpha_{bex} + \tfrac{n}{2^{k}}\max(\beta_{bex}, \chi(i)\beta_{net}) + \tfrac{n}{2^{k}}\gamma] +$$
$$\sum_{i=k-1}^{0}[\alpha_{rh} + \tfrac{n}{2^{i+1}}\max(\beta_{rh}, \chi(i)\beta_{net})]$$

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.

Analytically determining k in the general case is complicated due to the many possible relationships between parameter values. 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.

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

$$T_{AMbkt} = 2(r+c-2)\alpha_{bkt} + 2\frac{p-1}{p}n\beta_{bkt} + \frac{p-1}{p}n\gamma$$

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.

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

$$T_{AMbf} = 2(r-1)\alpha_{bkt} + 2\frac{r-1}{r}n\beta_{bkt} + \frac{r-1}{r}n\gamma + 2\log(c)\alpha_{fan} + 2\log(c)\frac{n}{r}\beta_{fan} + \log(r)\frac{n}{r}\gamma$$

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 $\alpha_{fan} < \alpha_{bkt}$ or $\beta_{fan} < \beta_{bkt}$, 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 [3]. Its communication characteristics are described in [4].

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 $\alpha_{bex} \approx \alpha_{rh} \approx \alpha_{bkt} \approx 2\alpha_{fan}$.
- When a node is just sending, it can essentially saturate a network link, so that $\beta_{fan} \approx \beta_{net}$. 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 $\beta_{bex} \approx \beta_{rh} \approx \beta_{bkt} \approx 2\beta_{net}$, i.e., $\nu = 1$ for algorithms where link conflicts can occur.
- All of our tests involved summation of double-precision (8 byte) floating point numbers, for which we measured $\gamma = 0.2$.
- 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 startup. Typical values for forced messages are $\alpha_{fan} = 150\mu$sec and $\beta_{net} = 0.4\mu$sec (per 8-byte element). Short messages are faster if "unforced", in which case typical values are $\alpha_{fan} = 75\mu$sec and $\beta_{net} = 0.8\mu$sec.
- Two operating system kernels are available on the DELTA [4]. 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 1, the first two characteristics have some effect on the relative performance of the various global combine algorithms. Figure 1 shows

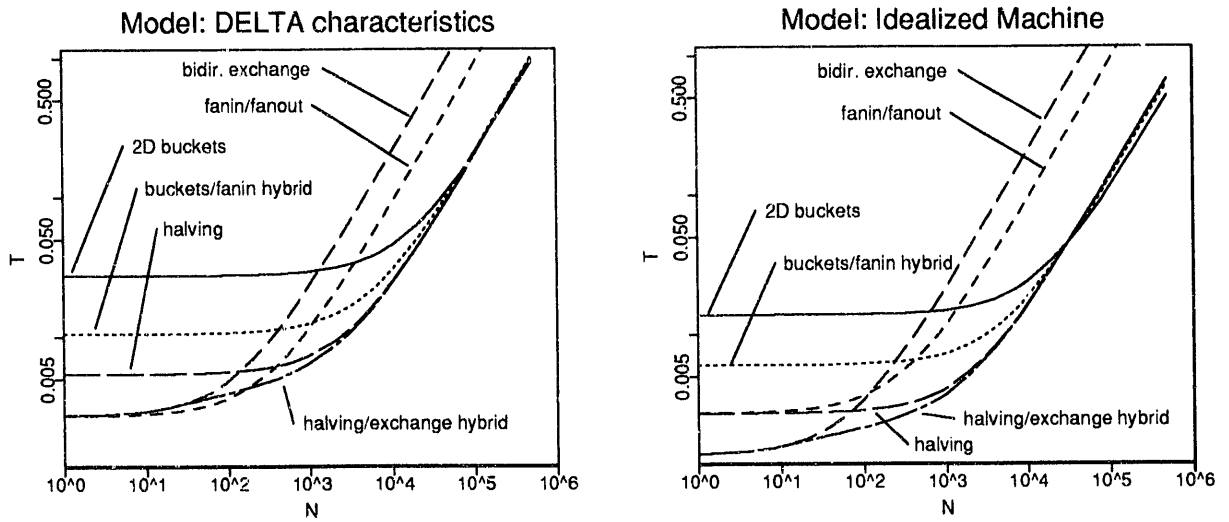

Figure 1: Expected performance of six algorithms, on a 16x32 mesh, under two sets of model assumptions.

the expected performance of all algorithms on a 16x32 mesh under two sets of model assumptions. The curves on the left assume the characteristics of the DELTA, as described above. The curves on the right assume an idealized machine in which message startups and data transfers are perfectly parallel and all transfers proceed at network bandwidth, i.e., $\alpha_{fan} = \alpha_{bex} = \alpha_{rh} = \alpha_{bkt}$, $\beta_{fan} = \beta_{bex} = \beta_{rh} = \beta_{bkt}$, $\nu = 0$. (The values are those of α_{fan}, β_{fan}, and γ from the left-hand curves.) Note in particular the relative performance of fanin/fanout, bidirectional exchange, and halving.

On a 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 left-hand graph. (Under the DELTA model, there is also a range of small n in which fanin/fanout is fastest, due to link conflicts in the other algorithms. This is not observed in practice because of the compensating fact that α_{bex} is actually slightly smaller than the model's $2\alpha_{fan}$.)

For 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_{fan} and β_{fan}.

Figure 2 shows experimental data from the DELTA that can be compared to Figure 1. The experimental data is in close agreement with the DELTA model and differs substantially from the idealized machine model.

Figure 3 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

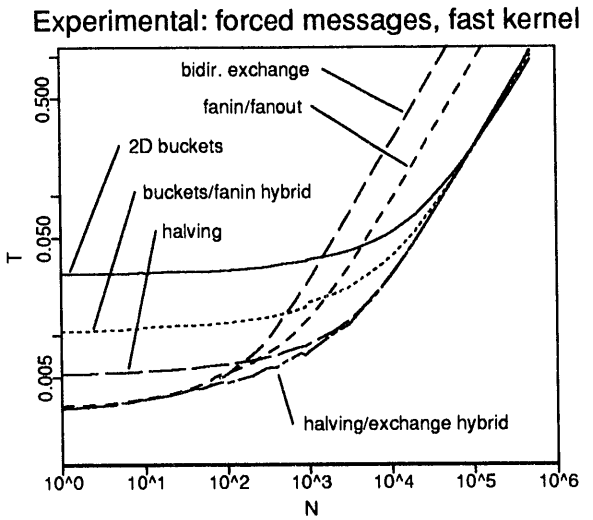

Figure 2: Observed performance of six global combine algorithms on a 16x32 mesh of the Intel DELTA computer system. All algorithms used "forced" messages and were timed with the "fast" kernel.

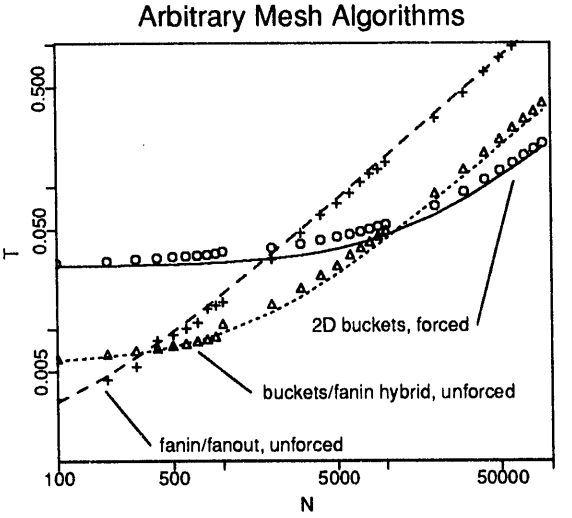

Figure 3: Observed and expected performance of three algorithms suitable for use on non-power-of-two meshes (measured on a 16x32 mesh). Where noted, unforced messages were used.

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

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.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] R.A. van de Geijn. Efficient global combine operations. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 291–294. IEEE Computer Society Press, 1991. To appear, Journal of Parallel and Distributed Computing.

DATE
FILMED
10/20/93
END