[5] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors*, volume I. Prentice Hall, 1988.

[6] S.L. Johnnson. Communication efficient basic linear algebra computations on hypercube architectures. *JPDC*, 4:133–172, 1987.

[7] S.L. Johsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Comput.*, 38:1249–1268, 1989.

[8] Y. Saad and M.H. Schultz. Data communications in hypercubes. *JPDC*, 6:115–135, 1989.

[9] Q.F. Stout and B. Wagar. Intensive hypercube communications. *J. Parallel Distrib. Comput.*, 10:167–181, 1990.

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

[11] R.A. van de Geijn. On global combine operations. LAPACK Working Note 29, Technical Report CS-91-129, University of Tennessee, 1991. Modified version to appear in JPDC.

Acknowledgements

The author would like to thank Dr. Israel Nelken, Dr. Bernard Tourancheau, and the referees for helpful comments.

References

[1] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn. LAPACK for distributed memory architectures: Progress report. In *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, pages 625–630. SIAM, 1992.

[2] J.J. Dongarra and R.A. van de Geijn. Reduction to condensed form for the eigenvalue problem on distributed memory architectures. LAPACK Working Note 30, Technical Report CS-91-130, University of Tennessee, 1991. To appear in *Parallel Computing*.

[3] J.J. Dongarra and R.A. van de Geijn. Two dimensional basic linear algebra subprograms. LAPACK Working Note 37, Technical Report CS-91-138, University of Tennessee, 1991.

[4] T.H. Dunigan. Performance of the Intel iPSC/860 hypercube. Technical Report ORNL/TM-11491, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1990.

Otherwise, a strategy that halves the vector at each stage until the vector length is less than or equal to 25, and communicates and combines full vectors thereafter, is used. The results are given in Fig. 6. In this case the modified hybrid method can be regarded as providing an automatic transition between Versions 2 and 3.

7 Conclusions

The implementation of global combine operations is greatly influenced by the machine on which the operations are to be performed. In this paper, we have proposed a hybrid approach that outperforms standard implementations under the restrictions mentioned in §2.

The Intel iPSC/860 failed to meet Assumption 4, requiring a modification to the hybrid algorithm. Other parallel architectures may be more flexible than the assumptions in §2. For example, if a hypercube architecture has the feature that it can communicate to all neighbors simultaneously, the hybrid algorithm can be generalized in the following way: At each stage, the vectors to be combined and communicated are partitioned into d (the dimension of the cube) equal parts, where one part is exchanged in each direction. This kind of approach to increasing the utilization of the communication network for different communications is discussed in [6, 7, 8, 9], as well as [5], Chapter 21. The effect is to reduce the constant multiplying the β term in the time complexities of Versions 2 and 3 by a factor d, which can be carried through to reduce the execution time of the hybrid algorithm as well. Details go beyond the scope of this paper.

$\alpha = 75\mu$sec. To overcome this complication, we padded all communication for our first experiments with an additional 30 floating point numbers. Moreover, the timings in [4] were obtained by passing messages around a ring. This means a receiving node is busy-waiting for the message when it arrives, which is not generally the case for the global combine. This, together with the overhead associated with making recursive subroutine calls and other general bookkeeping yields an effective $\alpha = 525\mu$sec. We also observed a slightly higher per item overhead: $\beta = 2.0\mu$sec.

The first series of experiments measured the time required for executing the various approaches described in §3. The results are reported in Fig. 5. Version 1 has a lower communication startup time than predicted in Fig. 5. We believe that this can be attributed to the fact that this algorithm allows some communication and computation to overlap.

Fig. 6 shows the performance of the global combine algorithms when the vectors are not padded to compensate for the lower communication startup time when the length of vectors being communicated is less than 25. While there is a dramatic drop in execution time for Version 1 and 2 when $n \leq 25$, Version 3 is particularly aided by the reduction in startup time, since the vectors being communicated are halved at each step. For example, on a cube of dimension 6, if the original vector is of length 512, the vectors communicated during the last two stages are of length 16 and 8, respectively, thereby reducing the execution time. This drastically affects the optimal choice $\bar{S}$ in §4. The following modification appears to give statisfactary results: If $n > 25 \times 2^d$, the optimal choice as described in §4 is used.

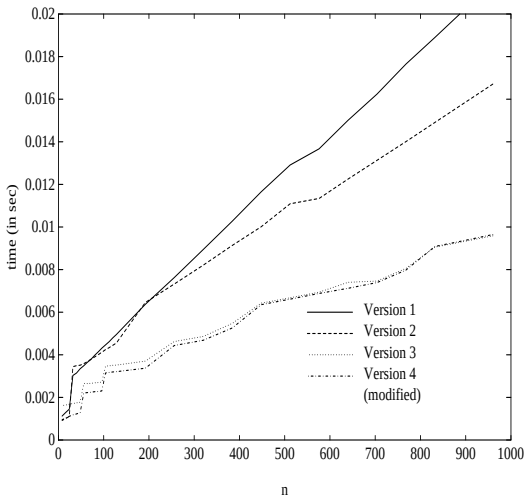
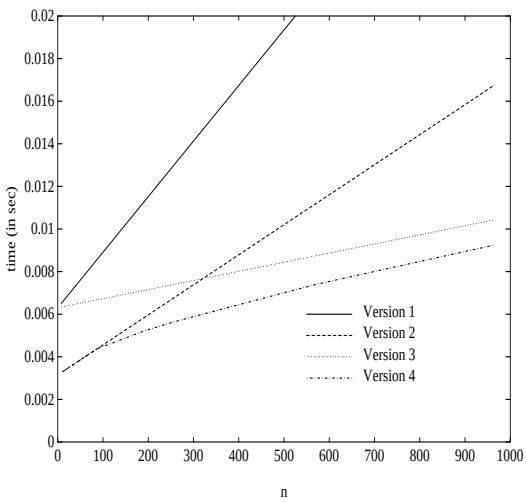

0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
time (in sec)
0
100
200
300
400
500
600
700
800
900
1000
n
Version 1
Version 2
Version 3
Version 4
(modified)

Figure 6: Observed time as a function of vector length n on 64 nodes. In this version, the length of vectors being communicated has not been adjusted, and the algorithm has been modified as described in §6, with $\alpha = 480\mu$sec, $\beta = 2\mu$sec, and $\gamma = .35\mu$sec.

assumed in §2, it can be programmed in such a way that all assumptions are more or less satisfied. It should be noted when we performed our experiments, the performance of the global summation operation provided by the system library for the iPSC/860 was essentially identical to Version 2. This leads us to believe that this is the version implemented by Intel at that time.

In [4] it is shown that the cost for communicating a floating point vector of length n on the iPSC/860 is roughly given by Assumption 5, with $\alpha = 136\mu$sec and $\beta = 1.6\mu$sec. The cost of adding two single precision floating point numbers is $\gamma = .35\mu$sec. Unfortunately, when vectors of length 25 or less are communicated, the communication startup time becomes


0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
time (in sec)
0
100
200
300
400
500
600
700
800
900
1000
n
Version 1
Version 2
Version 3
Version 4

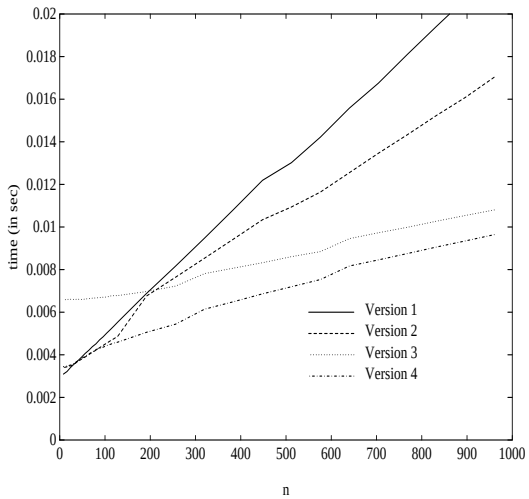

0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
time (in sec)
0
100
200
300
400
500
600
700
800
900
1000
n
Version 1
Version 2
Version 3
Version 4

Figure 5: Predicted (*left*) and observed (*right*) time as a function of vector length n on 64 nodes when $\alpha = 525\mu$sec, $\beta = 2\mu$sec, and $\gamma = .35\mu$sec.

ation, very much like Version 4 in §3. Since the time complexity of the two approaches being combined are identical to those being combined in Version 4, Thm. 1 holds in this case as well. Details for the optimal hybrid algorithm for combining to a root are given in [11].

6 Experiments on the Intel iPSC/860

To test our theoretical results, we implemented the various global combine operations on the Intel iPSC/860. Our experiments centered around a specific global combine operation, the global summation of single precision floating point vectors.

The Intel iPSC/860 is a commercial parallel processor that consists of Intel i860 processors connected in a hypercube topology. Although this machine is somewhat more general than

nating `S` from the calling sequence, and replacing the condition in the first `if` clause by `n` $<2\alpha/((\mathtt{d}-1)(\beta+\gamma)+\gamma)$.

The time complexity of this optimal hybrid strategy is given by

$$2(d-\min(k,d))\alpha+(1-2^{-(d-\min(k,d))})n(2\beta+\gamma)+\min(k,d)(\alpha+2^{-(d-\min(k,d))}n(\beta+\gamma))$$

where n, d, and k are as given in Thm. 1.

5 Combine-to-Root

Some algorithms require the result of the global combine to only be known to one node. Without loss of generality, we can take this node to be $\mathbf{P}_0$. Again, there are two commonly used algorithms.

The first approach progresses exactly like Version 1 in the previous section, except that no broadcast of the result to all nodes is necessary (see Steps 1 and 2 of Fig. 1). The time complexity for this approach is identical to the time complexity of Version 2 in the previous section.

A second approach is identical to Version 3 in the previous section, except that once each node has computed its part of the global combine, the results are gathered to the root using a minimum spanning tree. The time complexity for this second approach is identical to the time complexity for Version 3.

Naturally, these two approaches can be combined to form a hybrid global combine oper-

```
hybridCOMB(n, x, y, d, S)
begin
     if S_{d-1} = 1 then
          send (n, x, ngbr(i,d-1))
          recv (n, y, ngbr(i,d-1))
          combine (n, x, y)
          if d-1 > 0 call hybridCOMB(n, x, y, d-1, S_{1:d-1})
     else
          let x0 = x[0,...,n/2-1], x1 = x[n/2,...,n]
          if bit(i,d-1)=0 then
               send(n/2, x1, ngbr(i,d-1))
               recv(n/2, y, ngbr(i,d-1))
               combine(n/2, x0, y)
               if d-1>0 then
                    call hybridCOMB(n/2, x0, y, d-1, S_{1:d-1})
               send(n/2, x0, ngbr(i,d-1))
               recv(n/2, x1, ngbr(i,d-1))
          else
               send(n/2, x0, ngbr(i,d-1))
               recv(n/2, y, ngbr(i,d-1))
               combine(n/2, x1, y)
               if d-1>0 call hybridCOMB(n/2, x1, y, d-1, S_{1:d-1})
               send(n/2, x1, ngbr(i,d-1))
               recv(n/2, x0, ngbr(i,d-1))
end
```

Figure 4: Hybrid global combine routine

Notice that this difference is monotonically increasing in k, hence an optimal strategy is strategy S where k equals the smallest integer so that $T(R,n,d) \geq T(S,n,d)$. Eq. (1) follows immediately from this inequality.

Careful examination of boundary cases completes the proof. □

The optimal hybrid algorithm is now derived from the algorithm given in Fig. 4 by elimi-

Theorem 1 *Given n equal to an integer multiple of p, let k equal the smallest integer so that*

$$n \geq 2^{d-k} \frac{\alpha}{k(\beta+\gamma)+\gamma} \tag{1}$$

holds. Then the optimal choice of hybrid strategy equals $S = (1^{d-k}, 0^k)$. (If $k \leq 0$ then $S = 1^d$, if $k \geq d$ then $S = 0^d$.)

Proof: We start by showing that there exists an optimal strategy of the form $(1^{d-k}, 0^k)$ (i.e., Version 3 followed by Version 2): Assume $R \in \{0,1\}^d$ is an optimal strategy and assume $R = (R_{0:j-1}, 0, 1, R_{j+1:d-1})$. Then

$$T(R,n,d) = T(R_{0:j-1}, n, j) + T((0,1), m, 2) + T(R_{j+1:d-1}, m/2, d-j-2)$$

where $m = n/2^r$ and r equals the number times Version 3 is chosen during the first j steps.

Define strategy $S = (R_{0:j-1}, 1, 0, R_{j+1:d-1})$. Then

$$T(S,n,d) = T(R_{0:j-1}, n, j) + T((1,0), m, 2) + T(R_{j+1:d-1}, m/2, d-j-2) < T(R,n,d)$$

We conclude that there always exists an optimal strategy of the desired form.

Next, we need to check that of all solutions of this form, the one given by the theorem is optimal. Given k, let $S = (0^k, 1^{d-k})$ and $R = (0^{k+1}, 0^{d-(k+1)})$, i.e., R is the strategy that does one additional step of Version 2. Then

$$\begin{aligned} T(R,n,d) - T(S,n,d) &= T(1^{d-(k+1)}, n, d-(k+1)) + T(0^{k+1}, n/2^{d-(k+1)}, k+1) \\ &\quad - T(1^{d-k}, n, d-k) - T(0^k, n/2^{d-k}, k) \\ &= -\alpha + n\frac{k(\beta+\gamma)+\gamma}{2^{d-k}} \end{aligned}$$

For a given strategy S, the time complexity of the hybrid strategy is given by the recursion

$$T(S,n,d) = \begin{cases} 0 & \text{if } d < 0 \\ T(S_{1:d-1}, n, d-1) + \alpha + n\beta + n\gamma & \text{if } S_0 = 0 \\ T(S_{1:d-1}, \frac{n}{2}, d-1) + 2\alpha + n\beta + \frac{n}{2}\gamma & \text{if } S_0 = 1 \end{cases}$$

where $S_{i:j} = (S_i, \ldots, S_j)$.

4 Optimal Choice of Hybrid Strategy

Naturally, we would like to know how to choose S given vectors of length n and cube dimension d.

Clearly, Version 3 is better than Version 2 for long vectors. So, intuitively, we are looking for the best k such that, starting with vectors of length n, Version 3 is applied $(d-k)$ times, followed by k steps of Version 2, with vectors of length $n/2^{(d-k)}$. To determine k, it must be decided at what point there is no benefit in performing an additional step of Version 2. I.e., k equals the smallest integer such that

$$T((1^{d-(k+1)}, 0^{k+1}), n, d) \geq T((1^{d-k)}, 0^k), n, d)$$

where $0^j = (0, \ldots, 0) \in \{0,1\}^j$, $1^j = (1, \ldots, 1) \in \{0,1\}^j$, and, if $S \in \{0,1\}^r$ and $R \in \{0,1\}^s$, $(S,R) = (S_0, \ldots, S_{r-1}, R_0, \ldots R_{s-1})$.

The proof of the following theorem shows that this intuitive argument is indeed correct.

or Version 3 can yield a faster execution time. This is illustrated in Fig. 5 for a cube of dimension 6 with $\alpha = 525\mu$sec, $\beta = 2\mu$sec, and $\gamma = .35\mu$sec, which we shall see are typical parameters when performing combine operations on the Intel iPSC/860.

3.4 Version 4

Notice that in all of the above strategies the global combine can be factored into a combine of vectors on neighboring nodes along direction $(d-1)$, followed by recursively performing global combines of the partial result on subcubes of dimension $d-1$. This suggests a family of hybrid strategies that combine Versions 2 and 3.

We can generate 2^d separate strategies by considering $S \in \{0,1\}^d = (S_0, \ldots, S_{d-1})$. Here $S_j = 0$ indicates that for the step of the global combine that combines in direction j, node $\mathbf{P}_i$ and $\mathbf{P}_{\text{ngbr}(i,j)}$ exchange current partial results and both combine them as in Version 2, while $S_j = 1$ implies that only half of the current vector is to be exchanged and combined, like in Version 3. Notice that $S = (0, \ldots, 0)$ yields Version 2, while $S = (1, \ldots, 1)$ yields Version 3.

If we assume x_i is initially stored in vector `x = x[0,...,n-1]` on node i, pseudo code that drives $\mathbf{P}_i$ is given in Fig. 4.

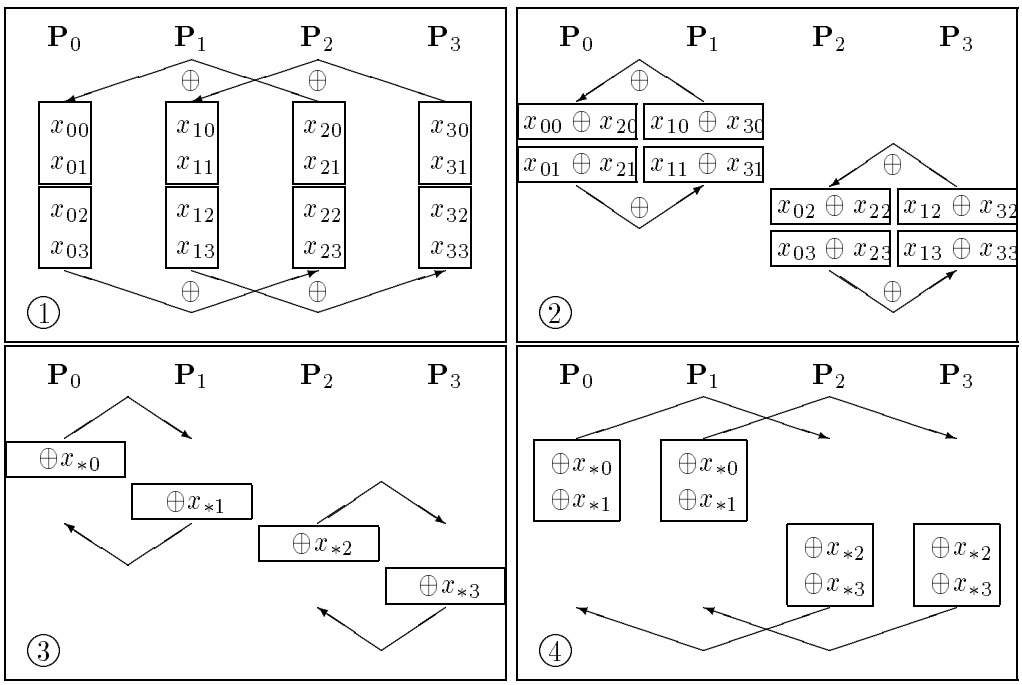


Figure 3: Version 3 on 4 nodes

$d-2,\ldots,0$, where the size of the partial result being communicated and combined is halved at each step. Finally, each node owns an equal part of the vector that results from combining all vectors x_i. Next, these results are distributed to all nodes by first sending the local results in direction 0, concatenating arriving results with the local results, and proceeding similarly with directions $1,\ldots,d-1$. The whole process is illustrated for $d=2$ in Fig. 3. In this figure, each vector x_i is divided into p equal parts, $x_{i0},\ldots,x_{i(p-1)}$.

The total time complexity for Version 3 equals

$$\sum_{i=1}^{d}\Big(2\alpha+2^{-i}n(2\beta+\gamma)\Big)=2d\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Depending on the relative values of α, β, and γ, as well as d and n, either Version 2

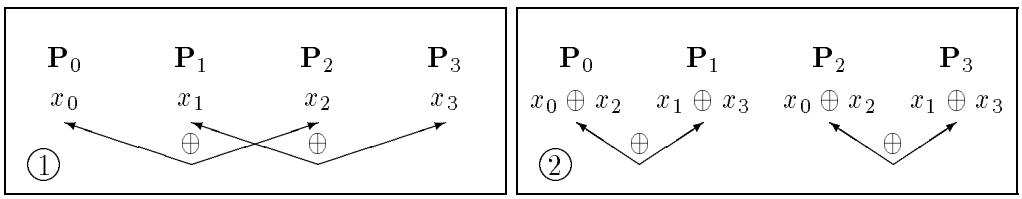


Figure 2: Version 2 on 4 nodes

a copy of the desired result, as illustrated for $d = 2$ in Fig. 2.

The total time complexity of this approach equals $d(\alpha + n\beta + n\gamma)$. Notice that under our assumptions, the time complexity of Version 1 always exceeds that of Version 2 by $d(\alpha + n\beta)$.

It should be noted that Version 2 is the most commonly used algorithm.

3.3 Version 3

Neither Version 1 nor Version 2 are optimal in the following sense: even if $\alpha = \beta = 0$, combining p vectors of length n requires time $dn\gamma$ on p nodes versus $(p-1)n\gamma$ on a single node, yielding a speedup of only $(p-1)/\log_2(p)$.

The following (much less known) approach, discussed for the global vector sum in [5], Chapter 19, does yield optimal use of the nodes if $\alpha = \beta = 0$: Assume n equals an integer multiple of $p = 2^d$. First, each node $\mathbf{P}_i$ divides its vector x_i into two equal subvectors and sends the first half to $\mathbf{P}_j$, where $j = \text{ngbr}(i, d-1)$, if the $(d-1)$st binary digit of i equals 0. Otherwise, the second half is sent to $\mathbf{P}_j$. Next, each node combines the half of vector x_j it receives with the half of vector x_i that was not sent. This process proceeds for direction

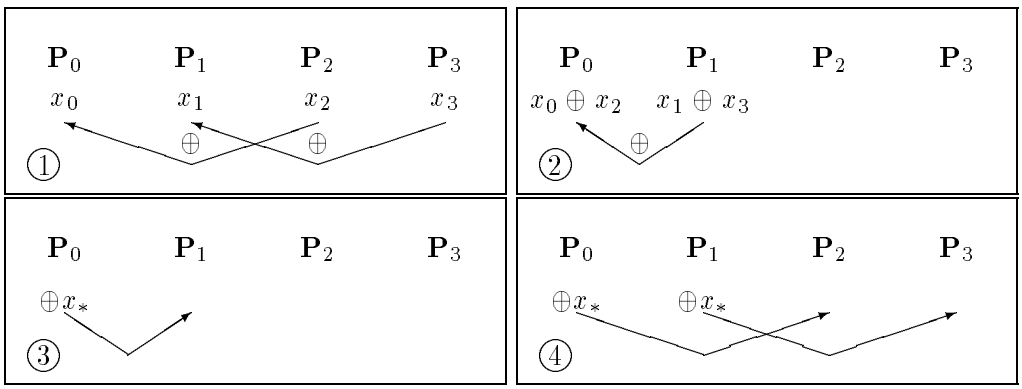


Figure 1: Version 1 on 4 nodes

3.1 Version 1

The first approach we will discuss embeds a minimum spanning tree in the hypercube network, rooted at node $\mathbf{P}_0$. The vectors are pairwise combined until the result is left at the root, after which the result is broadcasted, again utilizing the minimum spanning tree. This process is illustrated for $d = 2$ in Fig. 1.

The total time complexity for this strategy equals $2d(\alpha + n\beta) + dn\gamma$.

3.2 Version 2

A second approach, described in [5], §14-5.4, starts by each node $\mathbf{P}_i$ exchanging the contents of vector x_i with its neighbor in direction $d-1$, after which the contents of x_i are combined with the contents of the vector that arrived from direction $d-1$, leaving a partial result in x_i. The process proceeds for directions $d-2$ through 0, after which all nodes own

$k = 0, \ldots, d-1$ where i_k equals the integer that differs from i in the kth binary digit. We will use the notation $\texttt{ngbr}(i,k)$ to denote i_k and $\texttt{bit}(i,k)$ to denote the kth binary digit of i. For fixed k, $\mathbf{P}_{i_k}$ is called the neighbor of $\mathbf{P}_i$ in direction k. 3. A node can exchange data with only one neighbor at a time, i.e., it can simultaneously send to and receive from the same neighbor [1], but not to and from different neighbors. 4. Communication and computation do not overlap, i.e., the communication is blocking. 5. Exchanging messages of length n between two neighbors requires time $\alpha + n\beta$, where α and β represents the communication startup time and per item transfer time, respectively [1]. 6. Combining two vectors of length n items requires time $n\gamma$. 7. Splitting and concatenation of vectors does not consume any time.

3 Combine-to-All

In this section, we discuss strategies for implementing global combine operations that leave the result on all nodes. In all cases, it is assumed that before initiating the combine, each node $\mathbf{P}_i$ owns a vector x_i of length n. Upon completion of the global combine, this vector is overwritten on all nodes with the result of performing an element-wise combine on corresponding elements of the vectors x_i. For simplicity, we will assume n equals a multiple of p in the subsequent discussion.

[1] When a node cannot simultaneously send and receive from the same neighbor, both α and β can be adjusted appropriately.

1 Introduction

In this paper, we address the implementation of the combine operation when vectors of data to be combined are distributed among the processors (nodes) of a MIMD hypercube architecture. There are several solutions to this problem. These can be found in [5]. The contribution of this paper is to examine how such solutions can be "glued together" to yield a faster hybrid algorithm.

Our interest in the implementation of global combine operations stems from their use in parallel implementations of numerical algorithms for dense matrix problems. For example, the global summation of vectors is useful when implementing the reduction of a matrix to Hessenberg or tridiagonal form [2]. Several global combine operators are included in a proposed set of communication primitives, the Basic Linear Algebra Communication Subprograms (BLACS) [1, 3].

2 Assumptions

Target architectures for our algorithm are distributed memory hypercube multicomputers, including Multiple Instruction Multiple Data (MIMD) machines like Intel's iPSC/860, NCUBE Inc.'s NCUBE2 and Transputer based machines.

For our theoretical analysis, we will assume the following model: 1. The hypercube consists of $p = 2^d$ nodes, labeled $\mathbf{P}_0, \ldots, \mathbf{P}_{p-1}$. 2. Node $\mathbf{P}_i$ has physical neighbors $\mathbf{P}_{i_k}$,

Proposed running head:

On Global Combine Operations

Robert A. van de Geijn

Department of Computer Sciences

University of Texas at Austin

Austin, Texas 78712

(512) 471-9720

rvdg@cs.utexas.edu

Abstract

We discuss a hybrid strategy for implementing global combine operations on distributed memory MIMD multicomputers. A theoretical analysis is given and results from its implementation on the Intel iPSC/860 are reported.

On Global Combine Operations *

Robert A. van de Geijn

Department of Computer Sciences

University of Texas at Austin

Austin, Texas 78712

rvdg@cs.utexas.edu

July 30, 1993

*This work was supported by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615 while the author was on leave at the University of Tennessee.