An Efficient Parallel Algorithm for Matrix–Vector Multiplication.

Article *in* International Journal of High Speed Computing · March 1995

DOI: 10.1142/S0129053395000051 · Source: DBLP

3 authors, including:

Bruce Hendrickson
Sandia National Laboratories
154 PUBLICATIONS **10,548** CITATIONS

SEE PROFILE

Robert Leland
Sandia National Laboratories
58 PUBLICATIONS **3,195** CITATIONS

SEE PROFILE

SANDIA REPORT

SAND92—2765 • UC—405
Unlimited Release
Printed March 1993

An Efficient Parallel Algorithm for Matrix—Vector Multiplication

Bruce Hendrickson, Robert Leland, Steve Plimpton

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND92-2765
Unlimited Release
Printed March 1993

Distribution
Category UC-405

An Efficient Parallel Algorithm for Matrix–Vector Multiplication

Bruce Hendrickson, Robert Leland and Steve Plimpton
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

Abstract.
The multiplication of a vector by a matrix is the kernel computation of many algorithms in scientific computation. A fast parallel algorithm for this calculation is therefore necessary if we are to make full use of the new generation of parallel supercomputers. This paper presents a high performance, parallel matrix–vector multiplication algorithm that is particularly well suited to hypercube multiprocessors. For an $n \times n$ matrix on p processors, the communication cost of this algorithm is $O(n/\sqrt{p} + \log(p))$, independent of the matrix sparsity pattern. The performance of the algorithm is demonstrated by employing it as the kernel in the well–known NAS conjugate gradient benchmark, where a run time of 6.09 seconds was observed. This is the best published performance on this benchmark achieved to date using a massively parallel supercomputer.

Key words. matrix–vector multiplication, parallel computing, hypercube, conjugate gradient method

AMS(MOS) subject classification. 65Y05, 65F10

Abbreviated title. Parallel Matrix–Vector Multiplication.

This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-76DP00789.

1. Introduction. The multiplication of a vector by a matrix is the kernel computation in many linear algebra algorithms, including, for example, the popular Krylov methods for solving linear and eigen systems. Recent improvements in such methods, coupled with the increasing use of massively parallel computers, require the development of efficient parallel algorithms for matrix–vector multiplication. This paper describes such an algorithm. Although the method works on all parallel architectures, it is particularly well suited to machines with hypercube interconnection topology, for example the Intel iPSC/860 and the nCUBE 2.

The algorithm described here was developed independently in connection with research on efficient methods of organizing parallel many-body calculations (see [5]). We subsequently learned that our algorithm is very similar in structure to a parallel matrix-vector multiplication algorithm described in [4]. We have, nevertheless, chosen to present our algorithm because it improves upon that in [4] in several ways: First, we specify how to overlap communication and computation and thereby reduce the overall run time. Second, we show how to map the blocks of the matrix to processors in a novel way which improves the performance of a critical communication operation on current hypercube architectures. And third, we consider the actual use of the algorithm within the iterative conjugate gradient solution method and show how in this context a small amount of redundant computation can be used to further reduce the communication requirements. By integrating these improvements we have been able to achieve significantly better performance on a well known benchmark than has been previously possible with a massively parallel machine.

A very attractive property of the new algorithm is that its communication operations are independent of the sparsity pattern of the matrix, making it applicable to all matrices. For an $n \times n$ matrix on p processors, the cost of the communication is $O(n/\sqrt{p} + \log(p))$. However, many sparse matrices exhibit structure which allows for other algorithms with even lower communication requirements. Typically this structure arises from the physical problem being modeled by the matrix equation and manifests itself as the ability to reorder the rows and columns to obtain a nearly block–diagonal matrix, where the p diagonal blocks are about equally sized, and the number of matrix elements not in the blocks is small. This structure can also be expressed in terms of the size of the separator of the graph describing the nonzero structure of the matrix. Our algorithm is clearly not optimal for such matrices, but there are many contexts where the matrix structure is not helpful (e.g. dense matrices, random matrices), or the effort required to identify the structure is too large to justify. It is these settings in which our algorithm is most appropriate and provides high performance.

This paper is structured as follows. In the next section we describe the algorithm and its communication primitives. In §3 we present refinements and improvements to the basic algorithm, and develop a performance model. In §4 we apply the algorithm to the NAS conjugate gradient benchmark problem to demonstrate its utility. Conclusions are drawn in §5.

2. A parallel matrix–vector multiplication algorithm. Iterative solution methods for linear and eigen systems are one of the mainstays of scientific computation. These methods involve repeated matrix–vector products or *matvecs* of the form $y_i = Ax_i$ where the the new iterate, x_{i+1}, is generally

some simple function of the product vector y_i. To sustain the iteration on a parallel computer, it is necessary that x_{i+1} be distributed among processors in the same fashion as the previous iterate x_i. Hence, a good matvec routine will return a y_i with the same distribution as x_i so that x_{i+1} can be constructed with a minimum of data movement. Our algorithm respects this distribution requirement.

We will simplify notation and consider the parallel matrix–vector product $y = Ax$ where A is an $n \times n$ matrix and x and y are n-vectors. The number of processors in the parallel machine is denoted by p, and we assume for ease of exposition that n is evenly divisible by p and that p is an even power of 2. It is fairly straightforward to relax these restrictions.

Let A be decomposed into square blocks of size $(n/\sqrt{p}) \times (n/\sqrt{p})$, each of which is assigned to one of the p processors, as illustrated by Fig. 1. We introduce the Greek subscripts α and β running from 0 to $\sqrt{p}-1$ to index the row and column ordering of the blocks. The (α, β) block of A is denoted by $A_{\alpha\beta}$ and owned by processor $P_{\alpha\beta}$. The input vector x and product vector y are also conceptually divided into $\sqrt{p}$ pieces indexed by β and α respectively. Given this block decomposition, processor $P_{\alpha\beta}$ must know x_β in order to compute its contribution to y_α. This contribution is a vector of length $n/\sqrt{p}$ which we denote by $z_{\alpha\beta}$. Thus $z_{\alpha\beta} = A_{\alpha\beta}x_\beta$, and $y_\alpha = \sum_\beta z_{\alpha\beta}$ where the sum is over all the processors sharing row block α of the matrix.

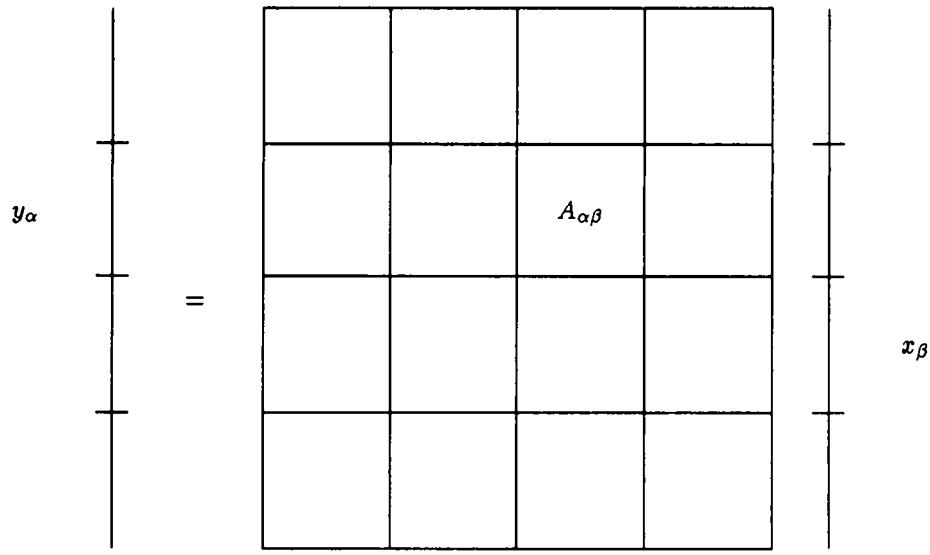

Fig. 1. Structure of matrix product $y = Ax$.

2.1. Communication primitives. Our algorithm requires three distinct patterns of communication. The first of these is an efficient method for summing elements of vectors owned by different processors, and is called a *fold* operation in [4]. We will use this operation to combine contributions to y owned by the processors that hold a block row of A. The fold operation is sketched in Fig. 2 for communication among processors with the same block row index α. Each processor begins the fold operation with a vector $z_{\alpha\beta}$ of length $n/\sqrt{p}$. The operation requires $\log_2(\sqrt{p})$ stages, halving the length of the vectors involved at each stage. Within each stage, a processor first divides its vector z into two equal sized subvectors, z_1 and z_2, as denoted by $(z_1|z_2)$. One of these subvectors is sent to another

processor, while the other processor sends back its contribution to the subvector which remained. The received subvector is summed element–by-element with the retained subvector to finish the stage. At the conclusion of the fold, each processor has a unique, length n/p portion of the fully summed vector. We denote this subvector with Greek superscripts, hence $P_{\alpha\beta}$ owns portion $y^{\alpha\beta}$. The fold operation requires no redundant floating point operations, and the total number of values sent and received by each processor is $n/\sqrt{p} - n/p$.

```
Processor P_αβ knows z_αβ ∈ R^{n/√p}
z := z_αβ
For i = 0, ..., log_2(√p) − 1
    (z_1|z_2) = z
    P_αβ' := P_αβ with i^th bit of β flipped
    If bit i of β is 1 Then
        Send z_1 to processor P_αβ'
        Receive w_2 from processor P_αβ'
        z_β := z_2 + w_2
    Else
        Send z_2 to processor P_αβ'
        Receive w_1 from processor P_αβ'
        z := z_1 + w_1
y^αβ := z
Processor P_αβ now owns y^αβ ∈ R^{n/p}
```

Fig. 2. The fold operation for processor $P_{\alpha\beta}$ as part of block row α.

In the second communication operation each processor knows some information that must be shared among all the processors in a column. We use a simple algorithm called *expand* [4], that essentially uses the inverse communication pattern of the fold operation. The expand operation is outlined in Fig. 3 for communication between processors with the same column index β. Each processor in the column begins with a subvector of length n/p, and when the operation finishes all processors in the column know all $n/\sqrt{p}$ values in the union of their subvectors. At each step in the operation a processor sends all the values it knows to another processor and receives that processor's values. These two subvectors are concatenated, as indicated by the "|" notation. As with the fold operation, only a logarithmic number of stages are required, and the total number of values sent and received by each processor is $n/\sqrt{p} - n/p$.

The optimal implementation of the fold and expand operations depends on the machine topology and various hardware considerations, *e.g.* the availability of multiport communication. There are, however, efficient implementations on most architectures. On hypercubes, for example, these operations can be implemented using only nearest neighbor communication if the blocks in each row and column of the matrix are owned by a subcube with $\sqrt{p}$ processors. On meshes, if the blocks of the matrix are

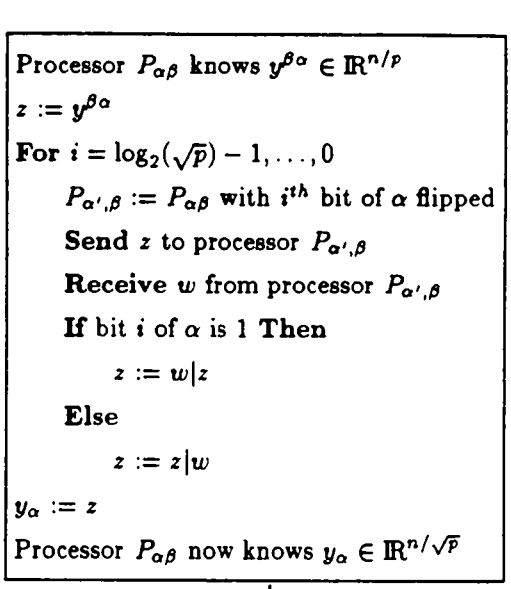

Fig. 3. The expand operation for processor $P_{\alpha\beta}$ as part of block column β.

mapped in the natural way to a square grid of processors, then the fold and expand operations can be implemented efficiently [9].

The third communication operation in our matvec algorithm requires each processor to send a message to the processor owning the transpose portion of the matrix, *i.e.* $P_{\alpha\beta}$ sends to $P_{\beta\alpha}$. Since we want row and column communication to be efficient for the fold and expand operations, this transpose communication can be difficult to implement efficiently. This is because a large number of messages must travel to architecturally distant processors, so the potential for message congestion is great. We have devised an optimal, congestion-free algorithm for this operation on hypercubes which is discussed in §3.1. Implementations of our matvec algorithm on other architectures may benefit from a similarly tailored transpose algorithm. However, even if congestion is unavoidable, the length of the message in the transpose communication step of our matvec algorithm is about $\sqrt{p}$ less than the volume of data exchanged in the fold and expand steps. Consequently, the transpose messages can be delayed by $O(\sqrt{p})$ without changing the overall scaling of the algorithm.

2.2. The matrix–vector multiplication algorithm. We can now present our algorithm for computing $y = Ax$ in Fig. 4. Further details and enhancements are presented in the following section. All the numerical operations in the algorithm are performed in steps (1) and (2). First, in step (1), each processor performs the local matrix–vector multiplication involving the portion of the matrix it owns. These values are summed within processor rows in step (2) using the fold operation from Fig. 2, after which each processor owns n/p of the values of y. Unfortunately, the values owned by processor $P_{\alpha\beta}$ are just a subvector of y_α, whereas to perform the next matvec $P_{\alpha\beta}$ must know all of y_β. This is accomplished in steps (3) and (4). In step (3), each processor exchanges its subsegment of y with the processor owning the transpose block of the matrix. After the transposition, the values of y_β are distributed among the processors in column block β of A. The expand operation among these processors gives each of them all of y_β, so the result is distributed as required for a subsequent matvec.

We note that at this level of detail, the algorithm is identical to the one described in [4] for dense matrices, but as we discuss in the next section, the details of steps (1), (2) and (3) are different and result in a more efficient overall algorithm.

Processor $P_{\alpha\beta}$ owns $A_{\alpha\beta}$ and x_β
(1) **Compute** $z_{\alpha\beta} = A_{\alpha\beta} x_\beta$
(2) **Fold** $z_{\alpha\beta}$ within rows to form $y^{\alpha\beta}$
(3) **Transpose** the $y^{\alpha\beta}$, *i.e.*
    a) **Send** $y^{\alpha\beta}$ to $P_{\beta\alpha}$
    b) **Receive** $y^{\beta\alpha}$ from $P_{\beta\alpha}$
(4) **Expand** $y^{\beta\alpha}$ within columns to form y_β

Fig. 4. Parallel matrix–vector multiplication algorithm for processor $P_{\alpha\beta}$.

3. Algorithmic details and refinements.

3.1. Transposition on parallel computers. The expand and fold primitives used in the matvec algorithm are most efficient on a parallel computer if rows and columns of the matrix are mapped to subsets of processors that allow for fast communication. On a hypercube a natural subset is a subcube, while on a 2–D mesh rows, columns or submeshes are possible. Unfortunately, such a mapping can make the transpose operation inefficient since it requires communication between processors that are architecturally distant. Modern parallel computers use *cut–through routing* so that a single message can be transmitted between non–adjacent processors at nearly the same speed as if it were sent between adjacent processors. Nevertheless, if multiple messages are simultaneously trying to use the same wire, all but one of them must be delayed. Hence machines with cut through routing can still suffer from serious message congestion.

On a hypercube the scheme for routing a message is usually to compare the bit addresses of the sending and receiving processors and flip the bits in a fixed order (and transmit along the corresponding channel) until the two addresses agree. On the nCUBE 2 and Intel iPSC/860 hypercubes the order of comparisons is from lowest bit to highest, a procedure known as *dimension order routing*. Thus a message from processor 1001 to processor 0100 will route from 1001 to 1000 to 1100 to 0100. The usual scheme of assigning matrix blocks to processors uses low order bits to encode the column number and the high order bits to encode the row number. Unfortunately, dimension order routing on this mapping induces congestion since messages from all the $\sqrt{p}$ processors in a row route through the diagonal processor. A similar bottleneck occurs with mesh architectures where the usual routing scheme is to move within a row before moving within a column. Fortunately, the messages being transposed in our algorithm are shorter than those in the fold and expand operations by a factor of $\sqrt{p}$. So even if congestion delays the transpose messages by $\sqrt{p}$, the overall communication scaling of the algorithm will not be affected.

On a hypercube, a different mapping of matrix blocks to processors can avoid transpose congestion

altogether. With this mapping we still have nearest neighbor communication in the fold and expand operations, but now the transpose operation is as fast as sending and receiving a single message of length n/p. Consider a d–dimensional hypercube where the address of each processor is a d–bit string. For simplicity we assume that d is even. The row block number α is a $d/2$–bit string, as is the column block number β. For fast fold and expand operations, we require that the processors in each row and column form a subcube. This is assured if any set of $d/2$ bits in the d–bit processor address encode the block row number and the other $d/2$ bits encode the block column number. Now consider a mapping where the bits of the block row and block column indices of the matrix are interleaved in the processor address. For a 64–processor hypercube (with 3–bit row and column addresses for the 8x8 blocks of the matrix) this means the 6–bit processor address would be $r_2c_2r_1c_1r_0c_0$ where the three bits $r_2r_1r_0$ encode the block row index and $c_2c_1c_0$ encodes the block column index.

Note that in this mapping each row of blocks and column of blocks of the matrix still resides on a subcube of the hypercube, so the expand and fold operations can be performed optimally. However, the transpose operation is now contention–free as demonstrated by the following theorem. Although the proof assumes a routing scheme where bits are flipped in order from lowest to highest, a similar contention free mapping is possible for any fixed routing scheme as long as row and column bits are forced to change alternately.

THEOREM 3.1. *Consider a hypercube using dimension order routing, and map processors to elements of an array in such a way that the bit–representations of a processor's row number and column number are interleaved in the processor's bit–address id. Then the wires used when each processor sends a message to the processor in the transpose location in the array are disjoint.*

Proof. Consider a processor P with bit–address $r_bc_br_{b-1}c_{b-1}\cdots r_0c_0$, where the row number is encoded with $r_b\cdots r_0$, and the column number with $c_b\cdots c_0$. The processor P^T in the transpose array location will have with bit–address $c_br_bc_{b-1}r_{b-1}\cdots c_0r_0$. Under dimension order routing, a message is transmitted in as many stages as there are bits, flipping bits in order from right to left to generate a sequence of intermediate patterns. After each stage, the message will have been routed to the intermediate processor denoted by the current intermediate bit pattern. The wires used in routing the message from P to P^T are those that connect two processors whose patterns occur consecutively in the sequence of intermediate patterns. After $2k$ stages, the intermediate processor will have the pattern $r_bc_b\cdots r_kc_kc_{k-1}r_{k-1}\cdots c_0r_0$. The bits of this intermediate processor are a simple permutation of the original bits of P in which the lowest k pairs of bits have been swapped. Also, after $2k-1$ stages, the values in the bit positions $2k$ and $2k-1$ are equal.

Now consider another processor $P' \neq P$, and assume that the message being routed from P' to P'^T uses the same wire employed in step i of the transmission from P to P^T. Denote the two processors connected by this wire by P_1 and P_2. Since they differ in bit position i, P_1 and P_2 can only be encountered consecutively in the transition between stages $i-1$ and i of the routing algorithm. Either $i-1$ or i is even, so a simple permutation of pairs of bits of P must generate either P_1 or P_2; say P_*. Similarly, the same permutation applied to P' must also yield either P_1 or P_2; say P'_*. If $P_* = P'_*$ then $P = P'$ which is a contradiction. Otherwise, both P_1 and P_2 must appear after an odd number of

stages in one of the routing sequences. If i is odd then bits i and $i+1$ of P must be equal, and if i is even then bits i and $i-1$ of P are equal. In either case, $P_1 = P_2$ which again implies the contradiction that $P = P'$. □

3.2. Overlapping computation and communication. If a processor is able to both compute and communicate simultaneously, then the algorithm in Fig. 4 has the shortcoming that once a processor has sent a message in the fold or expand operations, it is idle until the message from its neighbor arrives. This can be alleviated in the fold operation in step (2) of the algorithm by interleaving communication with computation from step (1). Rather than computing all the elements of $z_{\alpha\beta}$ before beginning the fold operation, we should compute just those that are about to be sent. Then whichever values will be sent in the next pass through the fold loop get computed between the send and receive operations in the current pass. In the final pass, the values that the processor will keep are computed. In this way, the total run time is reduced on each pass through the fold loop by the minimum of the message transmission time and the time to compute the next set of elements of $z_{\alpha\beta}$.

3.3. Balancing the computational load. The discussion above has concentrated on the communication requirements of our algorithm, but an efficient algorithm must also ensure that the computational load is well balanced across the processors. For our algorithm, this requires balancing the computations within each local matvec. If the region of the matrix owned by a processor has m' nonzeros, the number of floating point operations (flops) required for the local matvec is $2m' - n/\sqrt{p}$. These will be balanced if $m' \approx m/p$ for each processor, where m is the total number of nonzero elements in the matrix. For dense matrices or random matrices in which $m \gg n$, the load is likely to be balanced. However for matrices with some structure it may not be. For these problems, Ogielski and Aiello have shown that randomly permuting the rows and columns gives good balance with high probability [8]. A random permutation has the additional advantage that zero values encountered when summing vectors in the fold operation are likely to be distributed randomly among the processors.

Most matrices used in real applications have nonzero diagonal elements. We have found that when this is the case, it may be advantageous to force an even distribution of these among processors and to randomly map the remaining elements. This can be accomplished by first applying a random symmetric permutation to the matrix. This preserves the diagonal while moving the off-diagonal elements. The diagonal can now be mapped to processors to match the distribution of the $y^{\alpha\beta}$ subsegment that each processor owns. The contribution of the diagonal elements can then be computed in between the send and receive operations in the transpose communication, saving either the transpose transmission time or the diagonal computation time, whichever is smaller.

3.4. Complexity model. The algorithm described above can be implemented to require the minimal $2m - n$ flops to perform a matrix–vector multiplication, where m is the number of nonzeros in the matrix. Some of these flops will occur during the calculation of the local matvecs, and the rest during the fold summations. We make no assumptions about the data structure used on each processor to compute its local matrix–vector product. This allows for the implementation of whatever algorithm works best on the particular hardware. If we assume the computational load is balanced by using the

techniques described in §3.3, the time to execute these floating point operations should be very nearly $(2m-n)T_{\text{flop}}/p$, where T_{flop} is the time required for a single floating point operation.

The algorithm requires $\log_2(p)+1$ read/write pairs for each processor, and a total communication volume of $n(2\sqrt{p}-1)$ floating point numbers. Accounting for the natural parallelism in the communication operations, the effective communication volume is $n(2\sqrt{p}-1)/p$. Unless the matrix is very sparse, the computational time required to form the local matvec will be sufficient to hide the transmission time in the fold operation, as discussed in §3.2. We will assume that this is the case. Furthermore, we will assume that the transpose transmission time can be hidden with computations involving the matrix diagonal, as described in §3.3. The effective communication volume therefore reduces to $n(\sqrt{p}-1)/p$. The total run time, T_{total} can now be expressed as

$$T_{\text{total}} = \frac{2m-n}{p}T_{\text{flop}} + (\log_2(p)+1)(T_{\text{send}} + T_{\text{receive}}) + \frac{n(\sqrt{p}-1)}{p}T_{\text{transmit}}, \tag{1}$$

where T_{flop} is the time to execute a floating point operation, T_{send} and T_{receive} are the times to initiate a send and receive operation respectively, and T_{transmit} is the transmission time per floating point value. This model will be most accurate if message contention is insignificant, as it is with the mapping for hypercubes described in §3.1.

4. Application to the Conjugate Gradient algorithm. To examine the efficiency of our parallel matrix–vector multiplication algorithm, we used it as the kernel of a conjugate gradient (CG) solver. A version of the CG algorithm for solving the linear system $Ax = b$ is depicted in Fig. 5. There are a number of variants of the basic CG method; the one we present is a slightly modified version of the algorithm given in the NAS benchmark [1, 3] discussed later. In addition to the matrix–vector multiplication, the inner loop of the CG algorithm requires three vector updates (of x, r and p), as well as two inner products (forming γ and ρ').

An efficient parallel implementation of the CG algorithm should divide the workload evenly among processors while keeping the cost of communication small. Unfortunately, these goals are in conflict because when the vector updates are distributed, the inner product calculations require communication among all the processors. In addition, if the algorithm in Fig. 5 is implemented in parallel, each processor must know the value of α before it can update r to compute ρ' and hence β. The calculation of $\gamma = p^T y$, the distribution of γ, and the calculation of $\rho' = r^T r$ can actually be condensed into two global operations because the first two operations can be accomplished simultaneously with a binary exchange algorithm. However these global operations are still very costly. One way to reduce the communication load of the algorithm is to modify it as shown in Fig. 6.

This modified algorithm is algebraically equivalent to the original, but instead of updating r and then calculating $r^T r$, the new algorithm exploits the identity $r_{i+1}^T r_{i+1} = (r_i - \alpha y)^T (r_i - \alpha y) = r_i^T r_i - \alpha y^T r_i + \alpha^2 y^T y$, as suggested by Van Rosendale [10]. The values of γ, ϕ and ψ can be summed with a single global communication, essentially halving the communication time required outside the matvec routine. In exchange for this communication reduction, there is a net increase of one inner product calculation since $\phi = y^T r$ and $\psi = y^T y$ must now be computed, but $\rho' = r^T r$ need not

$x := 0$
$r := b$
$p := b$
$\rho := r^T r$
For i=1,...
    $y := Ap$
    $\gamma := p^T y$
    $\alpha := \rho/\gamma$
    $x := x + \alpha p$
    $r := r - \alpha y$
    $\rho' := r^T r$
    $\beta := \rho'/\rho$
    $\rho := \rho'$
    $p := r + \beta p$

Fig. 5. A conjugate gradient algorithm.

be calculated explicitly. Since the vectors are distributed across all the processors, this requires an additional $2n/p$ floating point operations by each processor in order to avoid a global communication. Whether this is a net gain depends upon the relative sizes of n and p, as well as the cost of flops and communication on a particular machine, but since communication is typically much more expensive per unit than computation, the modified algorithm should generally be faster. For the nCUBE 2, the machine used in this study, we estimate that this recasting of the algorithm is worthwhile when $n \leq 5 \times 10^5$.

This restructuring of the CG algorithm can in principle be carried further to hide more of the communication cost of the linear solve. That is, by repeatedly substituting for the residual and search vectors r and p we can express the current values of these vectors in terms of their values k steps previously. (General formulas for this process are given in [7].) By proper choice of k it is possible to completely hide the global communication in the CG algorithm. Unfortunately this leads to a serious loss of stability in the CG process which is expensive to correct [6]. We therefore recommend only limited application of this restructuring idea.

The vector and scalar operations associated with CG fit conveniently between steps (3) and (4) of the matrix–vector multiplication algorithm outlined in Fig. 4. At the end of step (3) the product vector y is distributed across all p processors, and it is trivial to achieve the identical distribution for x, r and p. Now all the vector updates can proceed perfectly in parallel. At the end of the CG loop, the vector p can be shared through an expand operation within columns and hence the processors will be ready for the next matvec. The resulting algorithm is sketched in Fig. 7.

We implemented a double precision version of this algorithm in C on the 1024 processor nCUBE 2 hypercube at Sandia's Massively Parallel Computing Research Laboratory. The resulting code was

$x := 0$
$r := b$
$p := b$
$\rho := r^T r$
For i=1,...,n
    $y := Ap$
    $\gamma := p^T y$
    $\phi := y^T r$
    $\psi := y^T y$
    $\alpha := \rho/\gamma$
    $\rho' := \rho - \alpha\phi + \alpha^2\psi$
    $\beta := \rho'/\rho$
    $\rho := \rho'$
    $x := x + \alpha p$
    $r := r - \alpha y$
    $p := r + \beta p$

Fig. 6. A modified conjugate gradient algorithm.

tested on the well-known NAS parallel benchmark problem proposed by researchers at NASA Ames [1, 3]. The benchmark uses a conjugate gradient iteration to approximate the smallest eigenvalue of a random, symmetric matrix of size 14,000, with an average of just over 132 nonzeros in each row. The benchmark requires 15 calls to the conjugate gradient routine, each of which involves 25 passes through the innermost loop containing the matvec.

This benchmark problem has been addressed by a number of different researchers on several different machines [2]. A common theme in this previous work has been the search for some exploitable structure within the benchmark matrix. Since arbitrary restructuring of the matrix is permitted by the benchmark rules as a pre-processing step, the computational effort expended in this search for structure is not counted in the benchmark timings.

In contrast, our algorithm is completely generic and does not require any special structure in the matrix. The communication operations are entirely independent of the zero/nonzero pattern of the matrix, and the only advantage of reordering would be to lessen the load on the most heavily burdened processor. Because the benchmark matrix diagonal is dense, we did partition the diagonal across all processors, as described in §3.3. Otherwise, we accepted the matrix as given, and made no effort to exploit structure.

Our implementation solved the benchmark problem in 6.09 seconds, which compares quite favorably with all other published results on massively parallel machines [3]. For comparison, the recently published times for the 128 processor iPSC/860 and 32K CM-2 are 8.61 and 8.8 seconds respectively, which is more than 40% longer than our result. Although this problem is highly unstructured, our

Processor $P_{\mu\nu}$ owns $A_{\mu\nu}$

$x, r, p, b, y \in \mathbb{R}^{n/p}$, $z_\mu, p_\nu \in \mathbb{R}^{n/\sqrt{p}}$

$x := 0$

$r := b$

$p := b$

$\bar{\rho} := r^T r$

Sum $\bar{\rho}$ over all processors to form ρ

Expand p within columns to form p_ν

For $i = 1, \ldots$

  Compute $z_\mu = A_{\mu\nu} p_\nu$

  Fold z_μ within rows to form $y^{\mu\nu}$

  Transpose $y^{\mu\nu}$, *i.e.*

    Send $y^{\mu\nu}$ to $P_{\nu\mu}$

    Receive $y := y^{\nu\mu}$ from $P_{\nu\mu}$

  $\bar{\gamma} := p^T y$

  $\bar{\phi} := y^T r$

  $\bar{\psi} := y^T y$

  Sum $\bar{\gamma}$, $\bar{\phi}$ and $\bar{\psi}$ over all processors to form γ, ϕ and ψ

  $\alpha := \rho / \gamma$

  $\rho' := \rho - \alpha\phi + \alpha^2 \psi$

  $\beta := \rho' / \rho$

  $\rho := \rho'$

  $x := x + \alpha p$

  $r := r + \alpha y$

  $p := r + \beta p$

  Expand p within columns to form p_ν

Fig. 7. A parallel CG algorithm for processor $P_{\mu\nu}$.

C code achieves about 250 Mflops, which is about 12% of the raw speed achievable by running pure assembly language BLAS on each processor without any communication.

5. Conclusions. We have presented a parallel algorithm for matrix–vector multiplication, and shown how this algorithm can be used very effectively within the conjugate gradient algorithm. The communication cost of this algorithm is independent of the zero/nonzero structure of the matrix and scales as $n/\sqrt{p}$. Consequently, the algorithm is most appropriate for matrices in which structure is either difficult or impossible to exploit. This is clearly the case for dense and random matrices, and it is also true more generally for sparse matrices in many contexts. For example, our algorithm could serve as an efficient black-box routine for prototyping sparse matrix linear algebra algorithms or could be embedded in a sparse matrix library where few assumptions about matrix structure can be made.

On the NAS conjugate gradient benchmark, an nCUBE 2 implementation of this algorithm runs more than 40% faster than any other reported algorithm running on any massively parallel machine.

The particular mapping we employ for hypercubes is likely to be of independent interest. This mapping ensures that rows and columns of the matrix are owned entirely by subcubes, and that with cut–through routing the transpose operation can be performed without message contention. This mapping has already proved useful for parallel many–body calculations [5], and is probably applicable to other linear algebra algorithms.

Acknowledgements. We are indebted to David Greenberg for assistance in developing the hypercube transposition algorithm in §3.1.

REFERENCES

[1] D. H. BAILEY, E. BARSZCZ, J. T. BARTON, D. S. BROWNING, R. L. CARTER, L. DAGUM, R. A. FATOOHI, P. O. FREDERICKSON, T. A. LASINSKI, R. S. SCHREIBER, , H. D. SIMON, V. VENKATAKRISHNAN, AND S. K. WEERATUNGA, *The NAS parallel benchmarks*, Intl. J. Supercomputing Applications, 5 (1991), pp. 63–73.

[2] D. H. BAILEY, E. BARSZCZ, L. DAGUM, AND H. D. SIMON, *NAS parallel benchmark results*, in Proc. Supercomputing '92, IEEE Computer Society Press, 1992, pp. 386–393.

[3] D. H. BAILEY, J. T. BARTON, T. A. LASINSKI, AND H. D. SIMON, EDITORS, *The NAS parallel benchmarks*, Tech. Rep. RNR-91-02, NASA Ames Research Center, Moffett Field, CA, January 1991.

[4] G. C. FOX, M. A. JOHNSON, G. A. LYZENGA, S. W. OTTO, J. K. SALMON, AND D. W. WALKER, *Solving problems on concurrent processors: Volume 1*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[5] B. HENDRICKSON AND S. PLIMPTON, *Parallel many–body calculations without all–to–all communication*, Tech. Rep. SAND 92-2766, Sandia National Laboratories, Albuquerque, NM, December 1992.

[6] R. W. LELAND, *The Effectiveness of Parallel Iterative Algorithms for Solution of Large Sparse Linear Systems*, PhD thesis, University of Oxford, Oxford, England, October 1989.

[7] R. W. LELAND AND J. S. ROLLETT, *Evaluation of a parallel conjugate gradient algorithm*, in Numerical methods in fluid dynamics III, K. W. Morton and M. J. Baines, eds., Oxford University Press, 1988, pp. 478–483.

[8] A. T. OGIELSKI AND W. AIELLO, *Sparse matrix computations on parallel processor arrays*, SIAM J. Sci. Stat. Comput., 14 (1993). To appear.

[9] R. VAN DE GEIJN, *Efficient global combine operations*, in Proc. 6th Distributed Memory Computing Conf., IEEE Computer Society Press, 1991, pp. 291–294.

[10] J. VAN ROSENDALE, *Minimizing inner product data dependencies in conjugate gradient iteration*, in 1983 International conference on parallel processing, H. J. Siegel et al., eds., IEEE, 1983, pp. 44–46.

EXTERNAL DISTRIBUTION:

R. W. Alewine
DARPA/RMO
1400 Wilson Blvd.
Arlington, VA 22209

Firooz Allahadi
US Air Force Weapons Lab
Nuclear Technology Branch
Kirtland, AFB, NM 87117-6008

Michael W. Allen
The Wall Street Journal
1233 Regal Row
Dallas, TX 75247

M. Alme
Alme and Associates
6219 Bright Plume
Columbia, MD 21044

Charles E. Anderson
Southwest Research Institute
PO Drawer 28510
San Antonio, TX 78284

Dan Anderson
Ford Motor Co.,Suite 1100
Village Place
22400 Michigan Ave.
Dearborn, MI 48124

Andy Arenth
National Security Agency
Savage Road
Ft. Meade, MD 20755
Attn: C6

Greg Astfalk
Convex Computer Corp.
3000 Waterview Parkway
PO Box 833851
Richardson, TX 75083-3851

Susan R. Atlas
TMC, MS B258
Center for Nonlinear Studies
Los Alamos National Labs
Los Alamos, NM 87545

L. Auslander
DARPA/DSO
1400 Wilson Blvd.
Arlington, VA 22209

D. M. Austin
Army High Per Comp. Res. Cntr.
University of Minnesota
1100 S. Second St.
Minneapolis, MN 55415

Scott Baden
University of California, San Diego
Dept. of Computer Science
9500 Gilman Drive
Engineering 0114
La Jolla, CA 92091-0014

F. R. Bailey
MS200-4
NASA Ames Research Center
Moffett Field, CA 94035

D. H. Bailey
NAS Applied Research Branch
NASA Ames Research Center
Moffett Field, CA 94035

Raymond A. Bair
Molecular Science Resch. Cntr.
Pacific NW Laboratory
Richland, WA 99352

R. E. Bank
Dept. of Mathematics
Univ. of CA at San Diego
La Jolla, CA 92093

Ken Bannister
US Army Ballistic Res. Lab
Attn: SLCBR-IB-M
Aberdeen Prov. Grnd., MD 21005-5066

Edward Barragy
Dept. ASE/EM
University of Texas
Austin, TX 78712

E. Barszcz
NAS Applied Research Branch
NASA Ames Research Center
Moffett Field, CA 94035

W. Beck
AT&T Pixel Machines, 4J-214
Crawfords Corner Rd.
Homdel, NJ 07733-1988

David J. Benson
Dept. of AMES R-011
Univ. of California at San Diego
La Jolla, CA 92093

Myles R. Berg
Lockheed, O/62-30, B/150
1111 Lockheed Way
Sunnyvale, CA 94089-3504

Stephan Bilyk
US Army Ballist. Resch. Lab
SLCBR-TB-AMB
Aberdeen Proving Ground, MD
21005-5066

Rob Bisseling
Shell Research B.V.
Postbus 3003
1003 AA Amsterdam
The Netherlands

Matt Blumrich
Dept. of Comp. Science
Princeton University
Princeton, NJ 08544

B. W. Boehm
DARPA/ISTO
1400 Wilson Blvd.
Arlington, VA 22209

R. R. Borcher
L-669
Lawrence Livermore Nat'l Labs
PO Box 808
Livermore, CA 94550

Dan Bowlus
Mail Code 8123
Attn: D. Bowlus & G. Letiecq
Naval Underwater Sys. Cntr.
Newport, RI 02841-5047

Dr. Donald Brand
MS - N8930
Geodynamics
Falcon AFB, CO 80912-5000

Bud Brewster
410 South Pierce
Wheaton, IL 60187

Carl N. Brooks
Brooks Associates
414 Falls Road
Chagrin Falls, OH 44022

John Brunet
245 First St.
Cambridge, MA 02142

John Bruno
Cntr for Comp. Sci and Engr
College of Engineering
University of California
Santa Barbara, CA 93106-5110

H. L. Buchanan
DARPA/DSO
1400 Wilson Blvd.
Arlington, VA 22209

D. A. Buell
Supercomputing Resch. Cntr.
17100 Science Dr.
Bowie, MD 20715

B. L. Buzbee
Scientific Computing Dept.
NCAR
PO Box 3000
Boulder, CO 80307

G. F. Carey
Dept. of Engineering Mechanics
TICOM ASE-EM WRW 305
University of Texas at Austin
Austin, TX 78712

Art Carlson
NOSC Code 41
New London, CT 06320

Bonnie C. Carroll
Sec. Dir.
CENDI, Information Int'l
PO Box 4141
Oak Ridge, TN 37831

Charles T. Casale
Aberdeen Group, Inc.
92 State Street
Boston, MA 02109

J. M. Cavallini
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

John Champine
Clay Res. Inc., Software Div.
655F Lone Oak Dr.
Eagan, MN 55121

Tony Chan
Department of Computer Science
The Chinese University of Hong Kong
Shatin, NT
Hong Kong

J. Chandra
Army Research Office
PO Box 12211
Resch Triangle Park, NC 27709

Siddartha Chatterjee
RIACS
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035-1000

Warren Chernock
Scientific Advisor DP-1
US Department of Energy
Forestal Bldg. 4A-045
Washington, DC 20585

R.C.Y. Chin
L-321
Lawrence Livermore Nat'l Lab
PO Box 808
Livermore, CA 94550

Mark Christon
L-122
Lawrence Livermore Nat'l Lab
PO Box 808
Livermore, CA 94550

M. Ciment
Adv. Sci. Comp/ Div. RM 417
National Science Foundation
Washington, DC 20550

Richard Claytor
US Department of Energy
Def. Prog. DE-1
Forestal Bldg. 4A-014
Washington, DC 20585

Andy Cleary
Centre for Information Research
Australia National University
GPA Box 4
Canberra, ACT 2601
Australia

T. Cole
MS180-500
Jet Prop. Lab
4800 Oak Grove Dr.,
Pasadena, CA 91109

Monte Coleman
US Army Bal. Resch Lab
SLCBR-SE-A (Bldg. 394/216)
Aberdeen Prov. Grnd., MD 21005-5006

Tom Coleman
Dept. of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853

S. Colley
NCUBE
19A Davis Drive
Belmont, CA 94002

J. Corones
Ames Laboratory
236 Wilhelm Hall
Iowa State University
Ames, IA 50011-3020

Steve Cosgrove
E6-220
Knolls Atomic Power Lab
PO Box 1072
Schenectady, NY 12301-1072

C. L. Crothers
IBM Corporation
472 Wheelers Farms Road
Milford, CT 06460

J. K. Cullum
Thomas J. Watson Resch. Center
PO Box 218
Yorktown Heights, NY 10598

Leo Dagum
Computer Sciences Corp.
NASA Ames Research Center
Moffett Field, CA 94035

Kenneth I. Daugherty
Chief Scientist
HQ DMA (SA), MS -A-16
8613 Lee Highway
Fairfax, VA 22031-2138

L. Davis
Cray Research Inc.
1168 Industrial Blvd.
Chippawa Falls, WI 54729

Mr. Frank R. Deis
Martin Marietta
Falcon AFB, CO 80912-5000

R. A. DeMillo
Comp. & Comput. Resch.,
Rm. 304
National Science Foundation
Washington, DC 20550

L. Deng
Applied Mathematics Dept.
SUNY at Stony Brook
Stony Brook, NY 11794-3600

A. Trent DePersia
Prog. Mgr.
DARPA/ASTO
1400 Wilson Blvd.
Arlington, VA 22209-2308

Sean Dolan
nCUBE
919 E. Hillsdale Blvd.
Foster City, CA 94404

Jack Dongarra
Department of Computer Science
University of Tennessee
Knoxville, TN 37996

L. Dowdy
Computer Science Department
Vanderbilt University
Nashville, TN 37235

Joanne Downey-Burke
8030 Sangor Dr.
Colorado Springs, CO 80920

I. S. Duff
CSS Division
Harwell Laboratory
Oxfordshire, OX11 ORA
United Kingdom

Alan Edelman
University of California, Berkeley
Dept. of Mathematics
Berkeley, CA 94720

S. C. Eisenstat
Computer Science Dept.
Yale University
PO Box 2158
New Haven, CT 06520

H. Elman
Computer Science Dept.
University of Maryland
College Park, MD 20842

M. A. Elmer
DARPA/RMO
1400 Wilson Blvd.
Arlington, VA 22209

J. N. Entzminger
DARPA/TTO
1400 Wilson Blvd.
Arlington, VA 22209

A. M. Erisman
MS 7L-21
Boeing Computer Services
PO Box 24346
Seattle, WA 98124-0346

R. E. Ewing
Mathematics Dept.
University of Wyoming
PO Box 3036 University Station
Laramie, WY 82071

El Dabaghi Fadi
Charge of Research
Institute Nat'l De Recherche en
Informatique et en Automatique
Domaine de Voluceau Rocouencourt
BP 105
78153 Le Chesnay Cedex (France)

H. D. Fair
Institute for Adv. Tech.
4032-2 W. Braker Lane
Austin, TX 78759

Kurt D. Fickie
US Army Ballistic Resch. Lab
ATTN: SLCBR-SE
Aberdeen Proving Ground, MD
21005-5066

Tom Finnegan
NORAD/USSPACECOM J2XS
STOP 35
PETERSON AFB, CO 80914

J. E. Flaherty
Computer Science Dept.
Rensselaer Polytech Inst.
Troy, NY 12181

L. D. Fosdick
University of Colorado
Computer Science Department
Campus Box 430
Boulder, CO 80309

G. C. Fox
Northeast Parallel Archit. Cntr.
111 College Place
Syracuse, NY 13244

R. F. Freund
NRaD- Code 423
San Diego, CA 99152-5000

Sverre Froyen
Solar Energy Research Inst.
1617 Cole Blvd.

Golden, CO 80401

David Gale
Intel Corp
600 S. Cherry St.
Denver, CO 80222-1801

D. B. Gannon
Computer Science Dept.
Indiana University
Bloomington, IN 47401

C. W. Gear
NEC Research Institute
4 Independence Way
Princeton, NJ 08548

J. A. George
Needles Hall
University of Waterloo
Waterloo, Ont., Can.
N2L 3G1

Shomit Ghose
nCUBE
919 E. Hillsdale Blvd.
Foster City, CA 94404

Clarence Giese
8135 Table Mesa Way
Colorado Springs, CO 80919

Dr. Horst Gietl
nCUBE GmbH
Hanauer Strasse 85
8000 Munich 50
Germany

John Gilbert
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Michael E. Giltrud
DNA
HQ DNA/SPSD
6801 Telegraph Rd.
Alexandria, VA 22310-3398

Alastair M. Glass
AT&T Bell Labs Rm 1A-164
600 Mountain Avenue
Murray Hill, NJ 07974

J. G. Glimm
Dept. of App Math. & Stat.
State U. of NY at Stony Brook
Stony Brook, NY 11794

Dr. Raphael Gluck
TRW-DSSG, R4/1408
One Space Park
Redondo Beach, CA 90278

G. H. Golub
Computer Science Dept.
Stanford University
Stanford, CA 94305

Marcia Grabow
AT&T Bell Labs 1D-153
600 Mountain Ave.
PO Box 636
Murray Hill, NJ 07974-0636

Nancy Grady
MS 6240
Oak Ridge Nat'l Lab
Bos 2008
Oak Ridge, TN 37831

Anne Greenbaum
New York University
Courant Institute
251 Mercer Street
New York, NY 10012-1185

Satya Gupta
Intel SSD
Bldg. C)6-09, Zone 8
14924 NW Greenbriar,Pwky
Beaverton, OR 97006

J. Gustafson
Computer Science Department
236 Wilhelm Hall
Iowa State University
Ames, IA 50011

R. Grayson Hall
USDOE/HQ
1000 Independence Ave, SW
Washington, DC 20585

Cush Hamlen
Minnesota Supercomputer Cntr.
1200 Washington Ave. So.
Minneapolis, MN 55415

Steve Hammond ,
NCAR
PO Box 3000
Boulder, CO 80307

CDR. D. R. Hamon
Chief, Space Integration Div.
USSPACECOM/J5SI
Peterson AFB, CO 80914-5003

Dr. James P. Hardy
NTBIC/GEODYNAMICS
MS N 8930
Falcon AFB, CO 80912-5000

Doug Harless
NCUBE
2221 East Lamar Blvd., Suite 360
Arlington, TX 76006

Mike Heath
University of Illinois
4157 Beckman Institute
405 N. Mathews Ave.
Urbana, IL 61801

Greg Heileman
EECE Department
University of New Mexico
Albuquerque, NM 87131

Brent Henrich
Mobile R &D Laboratory
13777 Midway Rd.
PO Box 819047
Dallas, TX 75244-4312

Michael Heroux
Cray Research Park
655F Lone Oak Drive
Eagan, MN 55121

A.J. Hey
University of Southampton
Dept. of Electronics and Computer Science
Mountbatten Bldg., Highfield
Southampton, SO9 5NH
United Kingdom

W. D. Hillis
Thinking Machines, Inc.
245 First St.
Cambridge, MA 02139

Dan Hitchcock
US Department of Energy
SCS, ER-30 GTN
Washington, DC 20585

LTC Richard Hochbewrg
SDIO/SDA
The Pentagon
Washington, DC 20301-7100

C. J. Holland
Director
Math and Information Sciences
AFOSR/NM, Bolling AFB
Washington, DC 20332-6448

Dr. Albert C. Holt
Ofc. of Munitions
Ofc of Sec. of St.-ODDRE/TWP
Pentagon, Room 3B1060
Washington, DC 203301-3100

Mr. Daniel Holtzman
Vanguard Research Inc.
10306 Eaton Pl., Suite 450
Fairfax, VA 22030-2201

David A. Hopkins
US Army Ballistic Resch. Lab.
Attention: SLCBR-IB-M
Aberdeen Prov. Grnd., MD 21005-5066

Graham Horton
Universitat Erlangen-Nurnberg
IMMD III
Martensstrase 3
8520 Erlangen
Germany

Fred Howes
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Chua-Huang Huang
Assist. Prof. Dept. Comp. & Info Sci
Ohio State Univ.
228 Bolz Hall-2036 Neil Ave.
Columbus, OH 43210-1277

R. E. Huddleston
L-61
Lawrence Livermore Nat'l Lab
PO Box 808
Livermore, CA 94550

Zdenek Johan
Thinking Machines Corp.
245 First Street
Cambridge, MA 02142-1264

Gary Johnson
US Department of Energy
SCS, ER-30 GTN
Washington, DC 20585

S. Lennart Johnsson
Thinking Machines Corp.
245 First Street
Cambridge, MA 02142-1264

G. S. Jones
Tech Program Support Cntr.
DARPA/AVSTO
1515 Wilson Blvd.
Arlington, VA 22209

T. H. Jordan
Dept of Earth, Atmos & Pla. Sci.
MIT
Cambridge, MA 02139

M. H. Kalos
Cornell Theory Center
514A Eng. and Theory Center
Hoy Road, Cornell University
Ithaca, NY 14853

H. G. Kaper
Math. and Comp. Sci. Division
Argonne National Laboratory
Argonne, IL 60439

S. Karin
Supercomputing Department
9500 Gilman Drive
University of CA at San Diego
La Jolla, CA 92093

Herb Keller
Applied Math 217-50
Cal Tech
Pasadena, CA 91125

M. J. Kelley
DARPA/DMO
1400 Wilson Blvd.
Arlington, VA 22209

K. W. Kennedy
Computer Science Department
Rice University
PO Box 1892
Houston, TX 77251

Aram K. Kevorkian
Codje 7304
Naval Ocean Systems Center
271 Catalina Blvd.
San Diego, CA 92152-5000

John E. Killough
University of Houston
Dept. of Chem. Engineering
Houston, TX 77204-4792

D. R. Kincaid
Cntr. for Num. Analy.,
RLM 13-150
University of Texas at Austin
Austin, TX 78712

T. A. Kitchens
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Thomas Klemas
394 Briar Lane
Newark, DE 19711

Dr. Peter L. Knepell
NTBIC/GEODYNAMICS
MS N 8930
Falcon AFB, CO 80912-5000

Max Koontz
DOE/OAC/DP 5.1
Forestal Bldg
1000 Independence Ave.
Washington, DC 20585

Ann Krause
HQ AFOTEC/OAN
Kirtland AFB, NM 87117-7001

V. Kumar
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

J. Lannutti
MS B-186
Director, SC. Resch. Institute
Florida State University
Tallahassee, FL 32306

P. D. Lax
New York University-Courant
251 Mercer St.
New York, NY 10012

Lawrence A. Lee
NC Supercomputing Center
PO Box 12889
3021 Cornwallis Rd.
Research Triangle Park, NC 27709

Dr. H.R. Leland
Calspan Corporation
PO Box 400,
Buffalo, NY 14225

David Levine
Math & Comp. Science
Argonne National Laboratory
9700 Cass Avenue South
Argonne, IL 60439

Peter Littlewood
Theoret. Phy. Dept.
AT&T Bell Labs
Rm 1D-335
Murray Hill, NJ 07974

Peter Lomdahl
T-11, MS B262
Los Alamos Nat'l Lab
Los Alamos, NM 87545

Louis S. Lome
SDIO/TNI
The Pentagon
Washington, DC 20301-7100

Col. Gordon A. Long
Deptuty Director for Adv. Comp.
HQ USSPACECOM/JOSDEPS
Peterson AFB, CO 80914-5003

John Lou
3258 Caminito Ameca
La Jolla, CA 92037

Daniel Loyens
Koninklijke/Shell-Laboratorium
Postbus 3003
1002 AA Amsterdam
The Netherlands

Robert E. Lynch
Dept. of CS
Purdue University
West Lafayete, IN 47907

Kathy MacLeod
AFEWC/SAT
Kelly AFB
San Antonio, TX 78243-5000

H. Mair
Naval Surface Warfare Center
10901 New Hampshire Ave.
Silver Springs, MD 20903-5000

Henry Makowitz
MS - 2211-INEL
EG&E Idaho Incorporated
Idaho Falls, ID 83415

David Mandell
MS F663
Hydrodynamic App. Grp. X-3
Los Alamos Nat'l Labs
Los Alamos, NM 87545

T. A. Manteuffel
Department of Mathematics
University of Co. at Denver
Denver, CO 80202

William S. Mark
Lockheed - Org, 96-01
Bldg. 254E
3251 Hanover Street
Palo Alto, CA 94303-1191

Kapil Mathur
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

John May
Kaman Sciences Corporation
1500 Garden of the Gods Road
Colorado Springs, CO 80933

William McColl
Oxford Univ. Computing Lab
8-11 Keble Road
Oxford, OX1 3QD
United Kingdom

S. F. McCormick
Computer Mathematics Group
University of CO at Denver
1200 Larimer St.
Denver, CO 80204

J. R. McGraw
L-316
Lawrence Livermore Nat'l Lab
PO Box 808
Livermore, CA 94550

Jill Mesirov
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

P. C. Messina
158-79
Mathematics & Comp Sci. Dept.
Caltech
Pasadena, CA 91125

Prof. Ralph Metcalfe
Dept. of Mech. Engr.
University of Houston
4800 Calhoun Road
Houston, TX 77204-4792

G. A. Michael
L-306
Lawrence Livermore Nat Lab

PO Box 808
Livermore, CA 94550

Loren K. Miller
Goodyear Tire & Rubber
PO Box 3531
Akron, OH 44309-3531

Robert E. Millstein
TMC
245 First Street
Cambridge, MA 02142

G. Mohnkern
NOSC - Code 73
San Diego, CA 92152-5000

C. Moler
The Mathworks
24 Prime Park Way
Natick, MA 01760

Gary Montry
Southwest Software
11812 Persimmon, NE
Albuquerque, NM 87111

N. R. Morse
C-DO, MS B260
Comp. & Comm .Division
Los Alamos National Lab
Los Alamos, NM 87545

J. R. Moulic
IBM
Thomas J. Watson Resch Cntr.
PO Box 704
Yorktown Heights, NY 10598

D. B. Nelson
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Jeff Newmeyer
Org. 81-04, Bldg. 157
1111 Lockheed Way
Sunnyvale, CA 94089-3504

D. M. Nosenchuck
Mech. and Aero. Engr. Dept.
D302 E Quad
Princeton University
Princeton, NJ 08544

C. E. Oliver
Offc of Lab Comp Bldg. 4500N,
Oak Ridge Nat'l Laboratory
PO Box 2008
Oak Ridge, TN 37831-6259

Dennis L. Orphal
Calif Resch & Technology Inc.
5117 Johnson Dr.
Pleasanton, CA 94588

J. M. Ortega
Applied Math Department
University of Virginia
Charlottesville, VA 22903

John Palmer
TMC
245 First St.
Cambridge, MA 02142

Robert J . Paluck
Convex Computer Corp.
3000 Waterview Parkway
PO Box 733851
Richardson, TX 75083-3851

Anthony C. Parmee
Counsellor and Attache (Def.)
British Embassy
3100 Mass. Ave, NW
Washington, DC 20008

S. V. Parter
Department of Mathematics
Van Vleck Hall
University of Wisconsin
Madison, WI 53706

Dr. Nisheeth Patel
US Army Ballistic Resch. Lab.
AMXBR-LFD
Aberdeen Prov. Grnd., MD 21005-5066

A. T. Patera
Mechanical Engineering Dept.
77 Massachusetts Ave.
MIT
Cambridge, MA 02139

A. Patrinos
Atmos. and Clim. Resch. Div
Office of Engy Resch, ER-74
US Department of Energy
Washington, DC 20545

R. F. Peierls
Math. Sciences Group, Bldg. 515
Brookhaven National Lab
Upton, NY 11973

Donna Perez
NOSC - MCAC Resource Cntr.
Code 912
San Diego, CA 92152-5000

K. Perko
Supercomputing Solutions, Inc.
6175 Mancy Ridge Dr.
San Diego, CA 92121

John Petresky
Ballistic Research Lab
SLCBR-LF-C
Aberdeen Prov. Grnd., MD 21005-5006

Linda Petzold
L-316
Lawrence Livermore Natl . Lab.
Livermore, CA 94550

Wayne Pfeiffer
San Diego SC Center
PO Box 85608
San Diego, CA 92186

Frank X. Pfenneberger
Martin Marietta
MS-N83104
National Test Bed
Falcon AFB, CO 80912-5000

Dr. Leslie Pierre
SDIO/ENA
The Pentagon
Washington, DC 20301-7100

Paul Plassman
Math and Computer Science Division
Argonne National Lab
Argonne, IL 60439

R. J. Plemmons
Dept. of Math. & Comp Sci.
Wake Forest University
PO Box 7311
Winston Salem, NC 27109

Alex Pothen
Computer Science Department
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

John K. Prentice
Amparo Corporation
3700 Rio Grande NW, Suite 5
Albuq., NM 87107-3042

Peter P. F. Radkowski
PO Box 1121
Los Alamos, NM 87544

J. Rattner
Intel Scientific Computers
15201 NW Greenbriar Pkwy.
Beaverton, OR 97006

J. P. Retelle
Org. 94-90
Lockheed - Bldg. 254G
3251 Hanover Street
Palo Alto, CA 94304

C. E. Rhoades
L-298
Computational Physics Div.
PO Box 808
Lawrence Livermore Nat'l Lab
Livermore, CA 94550

J. R. Rice
Computer Science Dept.
Purdue University
West Lafayette, IN 47907

John Richardson
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

Lt. Col. George E. Richie
Chief, ADv. Tech Plans
JOSDEPS
Peterson AFB, CO 80914-5003

John Rollett
Oxford University Computing Laboratory
8-11 Keble Road
Oxford, OX1 3QD
United Kingdom

R. Z. Roskies
Physics and Astronomy Dept.
100 Allen Hall
University of Pittsburg
Pittsburg, PA 15206

Diane Rover
Michigan State University
Dept. of Electrical Engineering
260 Engineering Bldg.
East Lansing, MI 48824

Y. Saad
University of Minnesota
4-192 EE/CSci Bldg.
200 Union St.
Minneapolis, MN 55455-0159

P. Sadayappan
Dept. of Comp. & Info Science
Ohio State Univ.-228 Bolz Hall
2036 Neil Ave.
Columbus, OH 43210-1277

Joel Saltz
Computer Science Department
A.V. Williams Building
University of Maryland
College Park, MD 20742

A. H. Sameh
CSRD
305 Talbot Laboratory
University of Illinois
104 S. Wright St.
Urbana, IL 61801

P. E. Saylor
Dept. of Comp. Science
222 Digital Computation Lab
University of Illinois
Urbana, IL 61801

LCDR Robert J. Schoppe
Chief, Operations Rqmts
USSPACECOM/JOSDEPS (Stop 35)
Peterson AFB, CO 80914

Rob Schreiber
RIACS
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035-1000

M. H. Schultz
Department of Computer Science
Yale University
PO Box 2158
New Haven, CT 06520

Dave Schwartz
NOSC, Code 733
San Diego, CA 92152-5000

Mark Seager
LLNL, L-80
PO box 803
Livermore, CA 94550

A. H. Sherman
Sci. Computing Assoc. Inc.
Suite 307, 246 Church Street
New Haven, CT 06510

Horst Simon
NAS Systems Division
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035

Richard Sincovec
Oak Ridge National Laboratory
P.O. Box 2008, Bldg 6012
Oak Ridge, TN 37831-6367

Vineet Singh
HP Labs, Bldg. 1U, MS 14
1501 Page Mill Road
Palo Alto, CA 94304

Ashok Singhal
CFD Resch. Center
3325 Triana Blvd.
Huntsville, AL 35805

Anthony Skjellum
Lawrence Livermore National Laboratory
7000 East Ave., Mail Code L-316
Livermore, CA 94550

L. Smarr
Director, Supercomputer Apps.
152 Supercomputer Applications
Bldg. 605 E. Springfield
Champaign, IL 61801

William R. Somsky
Ballistic Research Laboratory
SLCBR-SE-A, Bldg. 394
Aberdeen Proving Ground, MD 21005

D. C. Sorenson
Department of Math Sciences
Rice University
PO Box 1892
Houston, TX 77251

S. Squires
DARPA/ISTO
1400 Wilson Blvd.
Arlington, VA 11109

N. Srinivasan
AMOCO Corp
PO 87703
Chicago, IL 60680-0703

Thomas Stegmann
Digital Equipment Corporation
8085 S. Chester Street
Englewood, CO 80112

D. E. Stein
AT&T
100 South Jefferson Rd.
Whippany, NJ 07981

M. Steuerwalt
Division of Math Sciences
National Science Foundation
Washington, DC 20550

Mike Stevens
nCUBE
919 E. Hillsdale Blvd.
Foster City, CA 94404

G. W. Stewart
Computer Science Department
University of Maryland
College Park, MD 20742

O. Storassli
MS-244
NASA Langley Research Cntr.
Hampton, VA 23665

C. Stuart
DARPA/TTO
1400 Wilson Blvd.
Arlington, VA 22209

LTC James Sweeder
SDIO/SDA
The Pentagon
Washington, DC 20301-7100

R. A. Tapia
Mathematical Sci. Dept
Rice University
PO Box 1892
Houston, TX 77251

Gligor A. Tashkovich
PO Box 296
Pound Ridge, NY 10576-0296

H. Teuteberg
Cray Research, Suite 830
6565 Americas Pkway, NE
Albuquerque, NM 87110

A. Thaler
Division of Math Sciences
National Science Foundation
Washington, DC 20550

Allan Torres
125 Lincoln Ave., Suite 400
Santa Fe, NM 87501

Harold Trease
Los Alamos National Lab
PO Box 1666, MS F663
Los Alamos, NM 87545

Randy Truman
Mechanical Engineering Dept.
University of NM
Albuquerque, NM 87131

Ray Tuminaro
CERFACS
42 Ave Gustave Coriolis
31057 Toulouse Cedex
France

Mark Urrutia
Intel Corp., CO1-03
5200 NE Elam Young Pkwy.
Hillsboro, ORE 97124-6497

Mike Uttormark
UW-Madison
1500 Johnson Dr.
Madison, WI 53706

R. VanDeGeijn
Computer Science Department
University of Texas
Austin, TX 78712

George Vandergrift
Dist. Mgr.
Convex Computer Corp.
3916 Juan Tabo, NE, Suite 38
Albuquerque, NM 87111

H. VanDerVorst
Delft University of Technology
Faculty of Mathematics
POB 356
2600 AJ Delft
The Netherlands

C. VanLoan
Department of Computer Science
Cornell University, Rm. 5146
Ithaca, NY 14853

John VanRosendale
ICASE, NASA Langley Research Center
MS 132C
Hampton, VA 23665

Steve Vavasis
Dept. of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853

R. G. Voigt
MS 132-C
NASA Langley Resch Cntr, ICASE
Hampton, VA 36665

Phuong Vu
Cray Research, Inc.
19607 Franz Road
Houston, TX 77084

David Walker
Bldg 6012
Oak Ridge National Lab
PO Box 2008
Oak Ridge, TN 37831

Steven J. Wallach
Convex Computer Corp.
3000 Waterview Parkway
PO Box 833851
Richardson, TX 75083-3851

R. C. Ward
Bld. 9207-A
Mathematical Sciences
Oak Ridge National Lab
PO Box 4141
Oak Ridge, TN 37831-8083

Thomas A. Weber
National Science Found.
1800 G. Street, NW
Washington, DC 20550

G. W. Weigand
DARPA/CSTO
3701 N. Fairfax Ave.
Arlington, VA 22203-1714

M. F. Wheeler
Math Sciences Dept
Rice University
PO Box 1892
Houston, TX 77251

A. B. White
MS-265
Los Alamos National Lab
PO Box 1663
Los Alamos, NM 87544

B. Wilcox
DARPA/DSO
1400 Wilson Blvd.
Arlington, VA 22209

Roy Williams
California Institute of Technology
206-49
Pasadena, CA 91104

C. W. Wilson
MS MI02-3/B11
Digital Equipment Corp.
146 Main Street
Maynard, MA 00175

K. G. Wilson
Physics Dept.
Ohio State University
Columbus, OH 43210

Leonard T. Wilson
NSWC
Code G22
Dahlgren, VA 22448

Peter Wolochow
Intel Corp., CO1-03
5200 NE Elam Young Pkwy.
Hillsboro, OR 97124-6497

P. R. Woodward
University of Minnesota
Department of Astronomy
116 Church Street, SE
Minneapolis, MN 55455

M. Wunderlich
Math. Sciences Program
National Security Agency
Ft. George G. Mead, MD 20755

Hishashi Yasumori
KTEC-Kawasaki Steel
Techno-research Corporation
Hibiya Kokusai Bldg. 2-3
Uchisaiwaicho 2-chrome
Chiyoda-ku, Tokyo 100

David Young
Center for Numerical Analysis
RLM 13.150
The University of Texax
Austin, TX 78713-8510

Robert Young
Alcoa Laboratories
Alcoa Center, PA 15069
Attn: R. Young & J. McMichael

William Zierke
Applied Research Lab-Penn State.
PO Box 30
State College, PA 16804

INTERNAL DISTRIBUTION:

Paul Fleury	1000
Ed Barsis	1400
Sudip Dosanjh	1402
Bill Camp	1421
Doug Cline	1421
David Gardner	1421
Grant Heffelfinger	1421
Scott Hutchinson	1421
Martin Lewitt	1421
Steve Plimpton	1421
Mark Sears	1421
John Shadid	1421
Julie Swisshelm	1421
Dick Allen	1422
Bruce Hendrickson (25)	1422
David Womble	1422
Ernie Brickell	1423
Kevin McCurley	1423
Robert Benner	1424
Carl Diegert	1424
Art Hale	1424
Rob Leland (25)	1424
Courtenay Vaughan	1424
Steve Attaway	1425
Johnny Biffle	1425
Mark Blanford	1425
Jim Schutt	1425
Michael McGlaun	1431
Allen Robinson	1431
Paul Yarrington	1432
David Martinez	1434
Dona Crawford	1900
William Mason	1952
Technical Library (5)	7141
Technical Publications	7151
Document Processing for DOE/OSTI (10)	7613-2
Central Technical File	8523-2
Charles Tong	8117