Matrix Multiplication on Hypercubes Using Full Bandwidth and Constant Storage

Ching-Tien Ho
S. Lennart Johnsson
Alan Edelman

TR-19-91

April 1991

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in *Proceedings of the 6th Distributed Memory Computing Conference.*

Matrix Multiplication on Hypercubes Using Full Bandwidth and Constant Storage

Ching-Tien Ho
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
ho@ibm.com

S. Lennart Johnsson
Harvard University and
Thinking Machines Corp.
Cambridge, MA
johnsson@think.com

Alan Edelman
Dept. of Mathematics
Univ. of California at Berkeley
Berkeley, CA 94270
edelman@math.berkeley.edu

Abstract

For matrix multiplication on hypercube multiprocessors with the product matrix accumulated in place a processor must receive about $P^2/\sqrt{N}$ elements of each input operand, with operands of size $P \times P$ distributed evenly over N processors. With concurrent communication on all ports, the number of element transfers in sequence can be reduced to $P^2/\sqrt{N}\log N$ for each input operand. We present a two-level partitioning of the matrices and an algorithm for the matrix multiplication with optimal data motion and constant storage. The algorithm has sequential arithmetic complexity $2P^3$, and parallel arithmetic complexity $2P^3/N$. The algorithm has been implemented on the Connection Machine model CM-2. For the performance on the 8K CM-2, we measured about 1.6 Gflops, which would scale up to about 13 Gflops for a 64K full machine.

1 Introduction

The multiplication of matrices is an important operation in many computationally intensive scientific applications. Effective use of the communication bandwidth is critical for maximum performance. The communication needs are minimized by a good choice of address map, i.e., data placement, and routing algorithms that minimize path lengths and congestion once an address map is given. We consider matrix multiplication on a Boolean n-cube. With sufficiently high data motion capability at each node, communication may be performed on all ports concurrently, and the full communications bandwidth of the network used.

Cannon [2] has given an algorithm for the multiplication of square matrices on two-dimensional meshes. Since a two-dimensional mesh is a subgraph of a Boolean cube [12], [5], it is possible to use Cannon's algorithm on a Boolean cube by emulating a mesh [6]. Dekel, Nassimi and Sahni [3] have described an algorithm (termed the DNS algorithm thereafter) for the multiplication of square matrices on a Boolean cube. Both the DNS and Cannon's algorithms assume that the number of matrix elements is equal to the number of processors. A generalization of Cannon's algorithm to matrices of arbitrary shapes and sizes is given in [7] and [8]. Cannon's algorithm may use up to four communication (unidirectional) channels per processor concurrently. The DNS algorithm only use two (bidirectional) channels at a time. The algorithm presented below concurrently use all $\log_2 N$ (bidirectional) channels in an N-processor Boolean cube, while preserving a constant storage, i.e., $O(\frac{P^2}{N})$ for $P \times P$ matrices.

The paper is organized as follows. In the next section, we introduce a few basic results regarding the specific communication operations used in the multiplication algorithms. In Section 3, we generalize the DNS algorithm to the multiplication of two $P \times P$ matrices on a Boolean n-cube, configured as a product cube of $\sqrt{N} \times \sqrt{N}$ processors, and show how the Boolean cube bandwidth can be fully utilized. We conclude in Section 4.

2 Preliminaries

In the following log denotes $\log_2$. The bit-wise exclusive-or operation is denoted "$\oplus$" and $\mathcal{Z}_n = \{0, 1, \cdots, n-1\}$. Also, $*^n$ denotes a string of n instances of $*$, where $*$ is either 0 or 1. Let $n' = n/2$ throughout the paper. We consider matrix multiplication $C \leftarrow A * B$ on a Boolean n-cube where A, B and C are $P \times P$ matrices, $N = 2^n$, n is even, and $P \geq n'\sqrt{N}$. For clarity, we assume P is a multiple of $n'\sqrt{N}$. Note that the assumption is made only to simplify the complexity analysis. The element in row i and column j of matrix A is $a(i,j)$, $i, j \in \mathcal{Z}_P$. $b(i,j)$ and $c(i,j)$ are similarly defined.

Let $\mathcal{S}(1,0) = (0)$ and

$$\mathcal{S}(n,0) = \mathcal{S}(n-1,0)|n-1|\mathcal{S}(n-1,0)$$

for $n > 0$, where "|" is the concatenation operator of two sequences. For instance, $\mathcal{S}(3,0) = (0,1,0,2,0,1,0)$. $\mathcal{S}(n,0)$ is the transition sequence in a *binary-reflected* n-bit Gray code [14]. If $\mathcal{S}(n,0) = (x_1, x_2, \cdots, x_{(2^n-1)})$, then we can shift modulo n to define

$$\mathcal{S}(n,s) = ((x_1+s) \bmod n, (x_2+s) \bmod n, \cdots,$$

$$(x_{(2^n-1)} + s) \bmod n), \quad 0 \leq s < n.$$

For instance, $\mathcal{S}(3,1) = (1,2,1,0,1,2,1)$. Let $\alpha(t,n,s)$ be the tth element of the sequence $\mathcal{S}(n,s)$, $1 \leq t \leq 2^n - 1$.

The communication times are measured by the number of elements transferred in sequence. Concurrent communication on all ports of all processors is assumed possible. All communications links are bidirectional.

A particular communication pattern that is used for one phase of the matrix multiplication algorithm is *bit-inversion* [15]. A bit-inversion in an n-cube implies that processor i sends its data to processor $\bar{i}$ for all i's, where $\bar{i}$ is the bit-complement of i.

Lemma 1 [15] *A tight bound for bit-inversion with K elements per processor on an n-cube is K.*

Proof: The required bandwidth is nNK and the available bandwidth is nN, which gives the lower bound K. An upper bound equal to the lower bound is given by the following algorithm. Divide the local data set into n parts, and exchange part i, $0 \leq i \leq n-1$, according to the sequence of dimensions $i, (i+1) \bmod n, \cdots, (i+n-1) \bmod n$. All n data sets can be exchanged concurrently without edge conflict. ∎

In general, with *bit-inversion* on only a subset of the processor address bits of every processor, the communications requirements are reduced, but not the lower bound.

Lemma 2 [9] *Any tight bound for communication in a Boolean n-cube is also a tight bound for the same communication in all disjoint n dimensional subcubes of an n'' dimensional cube, when the subcubes are identified by the same n dimensions, $n'' > n$.*

The significance of this lemma is that even though only a fraction $\frac{n}{n''}$ of the total bandwidth of the n''-cube is used, the communication time cannot be reduced when the communication in each subcube is optimal. Hence, in the case of the same *bit-inversion* on a subset of the bits of the address space the tight lower bound is still K for K elements per processor. The bits subject to inversion define a subcube, and the bits not inverted define the disjoint instances of the subcubes in which inversion is performed.

3 A block algorithm

In this section we first describe the DNS algorithm, which assumes $P \times P$ matrices distributed over P^2 processors. We then generalize it to the multiplication of $P \times P$ matrices distributed uniformly over an n-cube, factored as $\sqrt{N} \times \sqrt{N}$, where $P \geq \sqrt{N}$. Finally, we present an algorithm that use all communication channels of an n-cube when $P \geq n'\sqrt{N}$. For notational convenience, we assume P is a power of two and put $P = 2^p$.

3.1 The DNS algorithm

The DNS algorithm [3] assumes that A and B are $P \times P$ matrices and that the number of Boolean cube processors is P^2. The algorithm consists of two phases: alignment and multiplication.

Alignment:

$a(i,j) \leftarrow a(i, i \oplus j)$, $\forall i,j \in \mathcal{Z}_P$,
$b(i,j) \leftarrow b(i \oplus j, j)$, $\forall i,j \in \mathcal{Z}_P$.

Multiplication, step t, $0 \leq t \leq P-1$:

$a(i,j) \leftarrow a(i, j \oplus 2^{\alpha(t,p,0)})$, if $t \neq 0$, $\forall i,j \in \mathcal{Z}_P$,
$b(i,j) \leftarrow b(i \oplus 2^{\alpha(t,p,0)}, j)$, if $t \neq 0$, $\forall i,j \in \mathcal{Z}_P$,
$c(i,j) \leftarrow a(i,j) * b(i,j) + c(i,j)$, $\forall i,j \in \mathcal{Z}_P$.

With one element per processor and (i,j) being a processor address, the column index of an element of A is the same as the row index of an element of B for every processor (i,j) after the alignment phase, and for each step of the multiplication phase. Moreover, for any integer, complementing the bits of its binary encoding according to the transition sequence in a *binary-reflected* Gray code, such as the sequence $\mathcal{S}(p,0)$ for a p-bit number, produces every integer that can be encoded in p bits precisely once. Hence, during the course of the algorithm, processor (i,j) receives all the elements of row i of matrix A and column j of matrix B appropriately synchronized.

Replacing $\alpha(t,p,0)$ by $\alpha(t,p,s)$, $1 \leq s \leq p-1$, yields a matrix multiplication algorithm that for each t performs an exchange in dimension $(\alpha(t,p,0)+s) \bmod p$ instead of dimension $\alpha(t,p,0)$. This observation is the basis for defining an algorithm that fully uses the communications bandwidth of the Boolean cube.

3.2 Naive extension

Each processor holds a $2^{p-n'} \times 2^{p-n'}$ submatrix (consecutive assignment [6]). The data assignment is defined by the address map

$$(\underbrace{w^r_{p-1} w^r_{p-2} \cdots w^r_{p-n'}}_{rp^r} \underbrace{w^r_{p-n'-1} \cdots w^r_0}_{vp^r} \mid$$

$$\underbrace{w^c_{p-1} w^c_{p-2} \cdots w^c_{p-n'}}_{rp^c} \underbrace{w^c_{p-n'-1} \cdots w^c_0}_{vp^c}).$$

All operands have corresponding address maps. *Virtual processor* address bits (labeled vp) define local storage addresses, whereas the *real processor* address bits (labeled rp) define different physical processors. The superscripts "r" and "c" denote "row" and "column", respectively. The exchange operation defined by the exclusive-or operation on the virtual processor address bits reorders data in the local storage of *all* processors, but there is no exchange between real processors. An exclusive-or operation on bits in the real processor field implies an exchange of all data between pairs of processors. The local address map is preserved.

Lemma 3 *The alignment on the bits in the virtual processor address field, and the steps of the multiplication phase corresponding to bits in this address field defines a complete matrix multiplication on blocks of size* $2^{p-n'} \times 2^{p-n'}$.

Lemma 3 follows from the recursive nature of the binary-reflected Gray code. This block matrix multiplication can be replaced by any suitable matrix multiplication algorithm in each node, without affecting the part of the algorithm requiring interprocessor communication. For instance, a block, matrix-vector, or SAXPY [13] based algorithm may be used depending on the architecture of each node.

Theorem 1 *The multiplication of two square matrices of size* $P \times P$ *on an* n*-cube,* $2p \geq n$*, can be performed by applying the DNS algorithm [3] to the real processor address field, and by employing any suitable matrix multiplication algorithm for the local blocks of size* $2^{p-n'} \times 2^{p-n'}$*, assuming consecutive assignment of matrix elements to real processors.*

The time complexity of the algorithm is,

1. Communication:
   - Alignment: $n'\frac{P^2}{N}$.
   - Multiplication: $(\sqrt{N}-1)\frac{P^2}{N}$.
2. Arithmetic: $\frac{2P^3}{N}$.

The alignments of the matrices A and B are assumed to take place concurrently in the above estimates. The arithmetic time is reduced in proportion to the number of processors, but the largest communication term only in proportion to the square root of the number of processors. The data motion for the matrix A only uses one cube dimension per processor, and so does the data motion for B. A total of two cube dimensions are used for each processor, in each step of the multiplication algorithm. The communications capability of Boolean cubes of many dimensions is poorly utilized.

3.3 A block algorithm using all cube dimensions

By partitioning the matrix A into $1 \times n'$ blocks and the matrix B into $n' \times 1$ blocks the matrix multiplication is transformed into n' rank $\frac{P}{n'}$ updates. The idea in the algorithm below is to perform the communication for the different high rank updates concurrently. That is n' communication channels per processor are used for both A and B. The full communications bandwidth is used. We refer to the $P \times \frac{P}{n'}$ blocks of A and the $\frac{P}{n'} \times P$ blocks of B as *big blocks* in order to distinguish this blocking from the big blocks assigned to individual processors, the *small blocks*. The naive block algorithm modified as described below is used for the multiplication of each pair of big blocks.

3.3.1 Data allocation

Each big block is allocated to the processors with consecutive assignment [6] (as in the preceding section). The address map for A is

$$(\underbrace{w^r_{p-1}w^r_{p-2}\cdots w^r_{p-n'}}_{rp^r}\underbrace{w^r_{p-n'-1}\cdots w^r_0}_{vp0^r} \mid$$

$$\underbrace{w^c_{p-1}w^c_{p-2}\cdots w^c_{p-\mu}}_{vp1^c}\underbrace{w^c_{p-\mu-1}\cdots w^c_{p-\mu-n'}}_{rp^c}\underbrace{w^c_{p-\mu-n'-1}\cdots w^c_0}_{vp0^c})$$

and for B it is

$$(\underbrace{w^r_{p-1}w^r_{p-2}\cdots w^r_{p-\mu}}_{vp1^r}\underbrace{w^r_{p-\mu-1}\cdots w^r_{p-\mu-n'}}_{rp^r}\underbrace{w^r_{p-\mu-n'-1}\cdots w^r_0}_{vp0^r} \mid$$

$$\underbrace{w^c_{p-1}w^c_{p-2}\cdots w^c_{p-n'}}_{rp^c}\underbrace{w^c_{p-n'-1}\cdots w^c_0}_{vp0^c})$$

assuming that n' is a power of two and $\mu = \log n'$. This assumption is only made for notational convenience in the address map. The small blocks are defined by the fields labeled $vp0$. The small block size for A is $\frac{P}{\sqrt{N}} \times \frac{P}{n'\sqrt{N}}$, and for B it is $\frac{P}{n'\sqrt{N}} \times \frac{P}{\sqrt{N}}$. Big blocks are identified by the field labeled $vp1$. The concatenated rp and $vp0$ fields define the big blocks. Each such block is distributed uniformly over all $\sqrt{N} \times \sqrt{N}$ processors.

By Lemma 3 the alignment and subsequent exchange and multiplication operations related to the $vp0^c$ field of A and $vp0^r$ field of B define a block matrix multiplication local to every processor. Note that the lengths of the two fields $vp0^c$ and $vp0^r$ are the same. The exchange on the $vp1$ field is a local memory move. This exchange implies that in the next several steps a new pair of big blocks will be multiplied.

3.3.2 Alignment

The dimension of the least significant bit is zero. The number of 1-bits in the binary representation of i is $||i||$. $|S|$ denotes the cardinality of a set S. Let $\mathcal{D}(i)$ be the ordered set of dimensions (in an increasing order) for which the corresponding bits of the binary representation of i are one. For example, $\mathcal{D}((10110)) = \{1, 2, 4\}$. Clearly, $|\mathcal{D}(i)| = ||i||$.

The alignment is performed on the processor address fields alone, i.e., after the alignment processor (k, ℓ) has column indices

$$(\underbrace{**\cdots*}_{vp1^c}\underbrace{k \oplus \ell}_{rp^c}\underbrace{**\cdots*}_{vp0^c})$$

of the matrix A, and row indices

$$(\underbrace{**\cdots*}_{vp1^r}\underbrace{k \oplus \ell}_{rp^r}\underbrace{**\cdots*}_{vp0^r})$$

of the matrix B, where $** \cdots *$ denotes all numbers that can be represented by that bit-field. The matrices are properly aligned. For a processor in row k and column ℓ, the alignment of A involves the set of cube dimensions $\mathcal{D}(k|0^{n'})$ (the higher-order n' cube dimensions are used for the encoding of rows) and the alignment of B involves the set of cube dimensions $\mathcal{D}(\ell)$. Clearly, $\mathcal{D}(k|0^{n'}) \cap \mathcal{D}(\ell) = \phi$. Note that the set of dimensions involved in the alignment operation does not depend on the id of big block.

The number of processor dimensions involved in the alignment of A is $|\mathcal{D}(k|0^{n'})| = ||k||$ for processor row k. The number of dimensions involved in the alignment of B is $||\ell||$ for processor column ℓ. The data volume that needs to be communicated per processor is $\frac{P^2}{N}$ for A and B. The naive block algorithm does not fully use the communication bandwidth of the Boolean cube.

We constrain the alignment of a row to be confined to its row subcube, and the alignment of a column to be confined to its column subcube. By Lemma 1 and Lemma 2 the minimum number of element transfers in sequence under this constraint is $\frac{P^2}{N}$ for A and B. The alignment of each operand is sped up by a factor of n' by concurrent communication within subcubes, compared to the algorithm in the preceding section.

Lemma 4 *A lower bound for the alignment of A and B on a Boolean n-cube is $\frac{P^2}{N}$.*

Proof: Consider the $\frac{N}{4}$ processors in rows $\{1*^{n'-1}\}$ and columns $\{1*^{n'-1}\}$, i.e., the processors to which the lower right quarter submatrix of each operand is allocated. These $\frac{N}{4}$ processors form a $(n-2)$-dimensional subcube. Each processor in the subcube needs to exchange $\frac{P^2}{N}$ elements with the subcube storing the lower left quarter submatrix of A, and $\frac{P^2}{N}$ elements with the subcube storing the upper right quarter submatrix of B. The total number of elements that must be sent out of the subcube is $\frac{P^2}{2}$. The total number of links that connect to processors outside the subcube is $2\frac{N}{4}$. ∎

3.3.3 Multiplication

Lemma 5 [11] *A lower bound for the data transfer time of the matrices A and B during multiplication is $\frac{P^2}{n'N}(\sqrt{N}-1)$.*

Proof: Every processor needs to receive $\frac{P^2}{N}$ elements of A from each of $(\sqrt{N}-1)$ processors. The lower bound for this *all-to-all broadcasting* within row subcubes is $\frac{P^2}{n'N}(\sqrt{N}-1)$ [11]. But, since all row subcubes perform the same communication and are fully utilized for this lower bound the subcube lower bound is also the total lower bound by Lemma 2. The bound for B is derived similarly, and since the set of dimensions used for the broadcasting of A and B are disjoint the lemma follows. ∎

For the multiplication phase the binary-reflected Gray code exchange sequence accomplishes an *all-to-all broadcasting* [11] within columns for B, and within rows for A. Any sequence with this property applied to both A and B in the same order is acceptable. The exchange sequence $\mathcal{S}(n', s)$, $1 \leq s \leq n'-1$, is as appropriate as $\mathcal{S}(n', 0)$. It follows that n' pairs of blocks can be exchanged concurrently. The total data transfer time for the multiplication phase is $\frac{P^2}{n'N}(\sqrt{N}-1)$ for the matrix A and B.

For the rank $\frac{P}{n'}$ algorithm big block column m of A multiplies block row m of B. All small blocks of big block m of A and B are subject to the exchange sequence $\mathcal{S}(n', m)$. Let $A(k, \ell, m)$ be the small block assigned to processor (k, ℓ) of big block m of matrix A. The multiplication phase for big block column m of A and block row m of B involves the data motion defined by

$$\begin{aligned} A(k,\ell,m) &\leftarrow A(k, \ell \oplus 2^{\alpha(t,n',m)}, m), \\ &\quad \forall m \in \mathcal{Z}_{n'}, \forall k, \ell \in \mathcal{Z}_{\sqrt{N}} \text{ concurrently}, \\ B(k,\ell,m) &\leftarrow B(k \oplus 2^{\alpha(t,n',m)}, \ell, m), \\ &\quad \forall m \in \mathcal{Z}_{n'}, \forall k, \ell \in \mathcal{Z}_{\sqrt{N}} \text{ concurrently}. \end{aligned}$$

The index for time, t, ranges from 1 to $\sqrt{N}-1$. Note that the communication for exchanges of A and B can be performed concurrently. Moreover, since $\alpha(t, n', m_1) \neq \alpha(t, n', m_2)$, $m_1 \neq m_2$ for all t, the communication can be performed concurrently also for all $m \in \mathcal{Z}_{n'}$. In any communication step all cube dimensions are used.

Theorem 2 *The data transfer time for the described algorithm with concurrent communication on all ports of a Boolean n-cube is $\frac{P^2}{N} + \frac{P^2}{n'N}(\sqrt{N}-1)$, which is optimal within a small constant factor with the operands distributed uniformly over the processors configured as a product of two n'-cubes.*

4 Concluding Remarks

We have presented an algorithm for multiplying two $P \times P$ matrices on a Boolean n-cube where n is even and $P \geq n\sqrt{N}/2$. The algorithm has a parallel arithmetic complexity $2P^3/N$, a communication complexity $< \frac{P^2}{N} + \frac{2P^2}{n\sqrt{N}}$ and the minimal storage requirement $O(\frac{P^2}{N})$. The previous DNS algorithm, while having the same arithmetic complexity and minimal storage requirement, has communication complexity a factor of $n/2$ higher than our algorithm. Our algorithm has been implemented on the Connection Machine model CM-2. For the performance on the 8K CM-2, we measured about 1.6 Gflops, which would scale up to about 13 Gflops for a 64K full machine.

Note that if the storage is sufficiently large to allow all-to-all broadcasting within rows and columns to be performed by spanning tree algorithms then n steps suffice, and the communications bandwidth can be fully utilized [11]. But, the storage requirement per processor is proportional to $\frac{P^2}{\sqrt{N}}$, i.e., a factor of $\sqrt{N}$ higher than for the algorithm presented here, and is unlikely to be useful in practice.

It should be noted that the algorithm here can be generalized to non-square matrices distributed over an N processor cube configured into $N_1 \times N_2$. The choice of N_1, N_2 depends on the aspect ratios of the two input matrices. See [10] for a detailed discussion.

It is also possible to generalize Cannon's algorithm such that the full communication bandwidth of the cube is used. The generalization can be made since there exists n edge-disjoint Hamiltonian cycles in a $2n$-cube, [4], [1], [16]. The matrices are assigned to the processors by a two-level partitioning, as in the algorithm described here. The storage per processor is the same as for our algorithm. However, the local control at each processor is more complicated, because the known method for constructing the n edge-disjoint Hamiltonian cycles is quite complex (double recursion), and the path encoding is complicated.

References

[1] Jacques Aubert and Bernadette Schneider. Decomposition de la somme cartesienne d'un cycle et de l'union de deux cycles hamiltoniens en cycles hamiltoniens. *Descrete Mathematics*, 38:7–16, 1982.

[2] L.E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm.* PhD thesis, Montana State Univ., 1969.

[3] Eliezer Dekel, David Nassimi, and Sartaj Sahni. Parallel matrix and graph algorithms. *SIAM J. Computing*, 10:657–673, 1981.

[4] Marsha Foregger. Hamiltonian decompositions of products of cycles. *Descrete Mathematics*, 24:251–260, 1978.

[5] Geoffrey C. Fox, S.W. Otto, and A.J.G. Hey. Matrix algorithms on a hypercube i: Matrix multiplication. Technical Report Caltech Concurrent Computation Project Memo 206, California Institute of Technology, dept. of Theoretical Physics, October 1985.

[6] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Computing*, 4(2):133–172, April 1987.

[7] S. Lennart Johnsson. Data parallel programming and basic linear algebra subroutines. In John R. Rice, editor, *Mathematical Aspects of Scientific Software*, volume IMA series, 14, pages 183–196. Springer Verlag, 1987. YALE/DCS/RR-584, September 1987, Technical Report DP87-1, TMC-64, Thinking Machines Corp, 1987.

[8] S. Lennart Johnsson and Ching-Tien Ho. Algorithms for multiplying matrices of arbitrary shapes using shared memory primitives on a Boolean cube. Technical Report YALEU/DCS/RR-569, Dept. of Computer Science, Yale University, October 1987.

[9] S. Lennart Johnsson and Ching-Tien Ho. Shuffle permutations on Boolean cubes. Technical Report YALEU/DCS/RR-653, Department of Computer Science, Yale University, October 1988.

[10] S. Lennart Johnsson and Ching-Tien Ho. Multiplication of arbitrarily shaped matrices using the full communications bandwidth on Boolean cubes. Technical Report YALEU/DCS/RR-721, Department of Computer Science, Yale University, July 1989.

[11] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[12] S. Lennart Johnsson and Peggy Li. Solutionset for AMA/CS 146. Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[13] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM TOMS*, 5(3):308–323, September 1979.

[14] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.

[15] Quentin F. Stout and Bruce Wagar. Passing messages in link-bound hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[16] Alan Wagner, 1988. Personal communication.