Index Transformation Algorithms in a Linear Algebra Framework*

Alan Edelman†
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139
`edelman@math.mit.edu`

Steve Heller
& S. Lennart Johnsson‡
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142
`heller@think.com`
`johnsson@think.com`

April 1992

Abstract

We present a linear algebraic formulation for a class of index transformations such as Gray code encoding and decoding, matrix transpose, bit reversal, vector reversal, shuffles, and other index or dimension permutations. This formulation unifies, simplifies, and can be used to derive algorithms for hypercube multiprocessors. We show how all the widely known properties of Gray codes and some not so well-known properties as well, can be derived using this framework. Using this framework, we relate hypercube communications algorithms to Gauss-Jordan elimination on a matrix of 0's and 1's.

Keywords and phrases: binary-complement/permute, binary hypercube, Connection Machine, Gray code, index transformation, multiprocessor communication, routing, shuffle

---

*Simultaneously appears as Thinking Machines technical report TMC–223.

†Supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

‡Also affiliated with the Division of Applied Sciences, Harvard University.

1 Introduction

We present a theory for a class of index transformation algorithms that should be properly thought of as a matrix-vector product, though they rarely are. This class is strictly a superset of the class known as BCP (bit-permute/complement) [20, 21]. In spirit this theory is linked with the ideas in Van Loan's new book [26], particularly the notion that matrix factorizations can define algorithms. The principal idea is not the discussion of matrix factorization algorithms, per se. The idea is a different way of viewing and generating algorithms.

Van Loan [26] covers computational frameworks for the Fast Fourier Transform. Despite differences in our approach, on this quote we firmly agree:

> The proper way to discuss a matrix-vector product such as the discrete Fourier transform is with matrix-vector notation, not with vectors of subscripts and multiple summations. We should be as repelled by scalar notation as we are by assembly language coding for both retard algorithmic development.

Although it has always been clear that BCP and larger classes of communications problems *can* be formulated as matrix-vector products, they rarely have been. Keohane and Stearns address a similar class of permutations [19], but do not formulate the problem as a matrix-vector product. A notable exception is the contemporaneous work of Cormen [2] for permuting data on disk arrays.

Our motivation stems from communications algorithms for real applications on hypercube multiprocessors such as the Connection Machine model CM–2 multiprocessor, though we believe these ideas to have wider applicability. Our matrices only contain 0's and 1's: they describe transformations on a vector of length 2^n indirectly through binary encodings. The most familiar example is bit reversal, an operation used in conjunction with FFT's. Bit reversal is a permutation of a vector of length 2^n induced by a permutation on n objects: the n bits of the vector's indices. One can represent this transformation as a $2^n \times 2^n$ permutation matrix on the components of the vector [11, 26]. For our purposes it is more convenient to consider the more compact representation of the $n \times n$ matrix describing the index transformation, which in the bit reversal case has 1's on the northeast-southwest diagonal and is otherwise 0. Also familiar are so-called dimension transformations or index permutations. These are arbitrary permutations of the n bit indices, which induce permutations on 2^n elements. Why use matrices of order 2^n when matrices of order n suffice?

We define a linear index transformation by

$$i \to Ai,$$

where i is a bit vector with n components, A is an $n \times n$ 0,1 matrix, and the matrix-vector multiply is performed modulo 2. So long as A is nonsingular, this $n \times n$ matrix induces a permutation on the 2^n indices. Dimension permutations are trivial examples of such transformations; other examples include Gray code encoding and decoding of arbitrary axes. Many real applications on hypercube multiprocessors require complicated compositions of these transformations.

We show that this is not a matter of notation, but rather that the existence of a certain kind of convenient algorithm on a hypercube to perform the data movement given by a linear index transformation is equivalent to the ability to perform Gauss-Jordan elimination on A without pivoting. This ability, in turn, is related to a familiar condition on the principal submatrices of A. Thus the complicated combinatorial problem of devising an algorithm is reduced to the algebraic problem of decomposing a matrix. We believe that this is the first time that the existence of a hypercube communications algorithm has been related to the ability to perform Gauss-Jordan elimination.

In Section 2, we fix notation that will be useful throughout the paper, while Section 3 contains our main results. In Section 4, we apply these results towards the special case of Gray code encoding and decoding while Section 5 considers dimension permutations. We conclude in Section 6.

2 Notation

Let F_2 be the field of elements $\{0,1\}$ with addition and multiplication defined modulo 2. In this paper, addition and multiplication are always performed modulo 2.

We denote the vector space of n-vectors with elements in F_2 as F_2^n. Similarly, the set of $m \times n$ matrices with elements in F_2 is denoted by $\mathrm{F}_2^{m,n}$. For clarity, we sometimes display such matrices with empty spaces where the entries are 0. We sometimes consider i or its binary encoding as the node address of a hypercube in the usual manner.

Any integer i such that $0 \leq i < 2^n$ can be identified with an element of F_2^n by the use of the binary encoding of the number. Thus, if $i = \sum_{k=0}^{n-1} i_k 2^k$, then we identify i with the vector $(i_0, \ldots, i_{n-1})^T$. Notice that the vector is written with the least significant bit first. Of course F_2^n can be naturally included as a subset of F_2^{n+1} by appending an extra zero.

We admit that this vector notation for the binary representation of a number seems to clash with the usual representation, $i_{n-1} \ldots i_1 i_0$, in that the order appears backwards, but the definition as presented is appropriate and consistent for matrix-vector notation. We have resisted the temptation to refer to the first row of a matrix in F_2^{mn} or the first component of a vector in F_2^n as the 0th, but rather chose the more familiar index origin of one.

Some useful vectors are $e_n = 2^{n-1}$ in which only the nth component is 1 and $j_n = 2^n - 1$ in which only the first n components are 1. These vectors can be thought of as elements of F_2^k for any $k \geq n$ using the natural embedding. Also we can avoid difficulties by letting $e_0 = j_0 = 0$.

If $(x_1, \ldots, x_k)$ is any ordered sequence of numbers, then its **reversal** is the sequence $(x_k, \ldots, x_1)$.

3 Linear and Affine Index Transformations

We now define the transformations of interest to us which we refer to as **affine** or **linear**:

Definition 3.1 *An index transformation is defined to be* **affine** *if the data in node i is sent to node $f(i)$, where*

$$f(i) = Ai + b.$$

Cormen [2] calls this class of transformations BMMC for bit-matrix-multiply/complement.

Definition 3.2 *An index transformation is defined to be* **linear** *if the data in node i is sent to node $f(i)$, where*

$$f(i) = Ai.$$

Thus a linear index transformation is an affine transformation that fixes the data in node 0.

The simplest hypercube communication is the unconditional exchange of data across a fixed dimension. Algebraically this can be described by $f(i) = i + e_k$. Another simple hypercube communication sends data to the opposite corner of the hypercube. This is $f(i) = i + j_n$, which describes vector reversal.

Another example of a linear index transformation is a **dimension permutation** considered by such authors as Stone [22], Fraser [6], Nassimi and Sahni [20, 21], Flanders [5], Johnsson and Ho [14], Stout and Wagar [23, 24], and Swarztrauber [25]. A dimension permutation is defined as the map $f(i) = Pi$, where P is a permutation matrix. Since permutation matrices are orthogonal ($PP^T = I$), if it is also symmetric, then it is a square root of the identity ($P^2 = I$). Thus a **symmetric permutation matrix** corresponds to a disjoint set of dimension pairs being exchanged. On the other hand **circulant permutation matrices** correspond to relabeling dimensions in a way that preserves the circular order of the indices. The shuffle and unshuffle operations give two such matrices. Circulant permutation matrices form a subset of the **irreducible permutation matrices**. A matrix A is said to be irreducible if it has no nontrivial invariant subspaces. The irreducible permutation matrices correspond to the dimension exchange represented by a cycle.

In Section 4, we will consider the example of Gray code encoding and decoding.

The basic theorems of algebra tell us that if $f(i) = Ai + b$, where A is nonsingular, then the map is one-to-one. Otherwise, if the rank of A is r, then A maps the hypercube to an r-dimensional subcube. This map sends the data in 2^{n-r} nodes to one.

Definition 3.3 *A* **conditional exchange** *across dimension k, denoted E_k, is a communication pattern defined by $f(i) = Ai$, where A is any matrix whose diagonal consists of 1's, and whose off-diagonal may possibly be 1 only in the kth row.*

An example of a conditional exchange across dimension 3 is represented by the matrix:

$$E_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The mapping $f(i) = E_3 i$ describes a conditional change of the third bit, depending on the first and fourth bits. We will extend our use of the term "conditional exchange" to also refer to the associated matrix without loss of clarity.

Lemma 3.1 *If E_k is a conditional exchange, then E_k is nonsingular, $E_k e_k = e_k$ and $E_k^2 = I$ (i.e., $E_k = E_k^{-1}$).*

Proof From the form of the matrix, it is clear that the determinant of E_k is 1, and that $E_k e_k = e_k$. Either $E_k i = i$ so that $E_k^2 i = i$ or $E_k i = i + e_k$ and $E_k(i + e_k) = i$. Either way, $E_k^2 i = i$ for all nodes i, and thus $E_k^2 = I$. $\square$

Notice that if the kth diagonal entry were 0, then the kth column is 0 and the matrix would be singular. In fact the rank of the matrix would be exactly $n - 1$. Such a communication might be termed a conditional projection.

A conditional exchange can be implemented directly on a hypercube. Each node either sends all its data across the dimension specified in the exchange, or does nothing. Only one dimension of the hypercube is traversed in this operation, and this algorithm achieves fifty percent overall utilization of that dimension.

A hypercube communication operation that uses all the dimensions simultaneously is called **cube swap**. In this operation, each node sends one message along each hypercube dimension.

If an $n \times n$ matrix A can be decomposed as a sequence of conditional exchange matrices, $A = E_n \ldots E_1$, then this factorization describes an algorithm for implementing the linear index transformation given by A as a sequence of conditional exchange operations across dimensions 1 through n respectively. More generally, if A admits a factorization of the form $A = E_{d_n} \ldots E_{d_2} E_{d_1}$, where $d_1, d_2, \ldots, d_n$ is a reordering of the dimensions 1 through n, then the factorization defines an algorithm for implementing the linear index transformation as a sequence of conditional exchanges in a different order. Any sequence of exchanges on disjoint dimensions can be implemented in a pipelined fashion on a hypercube as a sequence of identical cube swap operations, as long as there is a nontrivial amount of data at the node. The pipeline will have one start-up and one wind-down step for each dimension traversed. Once the pipe is started the algorithm achieves fifty percent utilization of the total bandwidth available. Of course, this leaves us short by a factor of two in total use of cube swap bandwidth, but allows us to consider very general situations.

We now present our main theorem relating hypercube communications algorithms algebraically to Gauss-Jordan elimination performed columnwise and modulo 2 instead of over the reals:

Theorem 3.1 *The following statements are equivalent:*

1. *A may be decomposed as a product of conditional exchanges:*
$$A = E_n \ldots E_1.$$
2. *The index transformation defined by A can be accomplished on a hypercube as a pipelined sequence of cube swaps, accomplishing a sequence of conditional exchanges traversing dimensions 1 through n consecutively.*
3. *The columnwise Gauss-Jordan elimination algorithm (modulo 2) on A runs to completion without the need for pivoting.*
4. *All n principal submatrices of A are nonsingular.*

Proof The equivalence of *1.* and *2.* is discussed before the theorem. By columnwise Gauss-Jordan elimination we mean an algorithm whose ith step consists of adding multiples of column i to the other columns so that the resulting matrix matches the identity in the first i rows. In modulo 2 arithmetic one can verify that the algorithm takes the following simple form:

```
A^0 = A
for i=1,2,...,n
   E_i := E(A^{i-1}, i)
   A^i := A^{i-1} E^i
end
```

Here $E(A, j)$ denotes a matrix that is the identity except in the jth row, which is defined to match that of A. It is well-known that the Gauss-Jordan algorithm requires no pivoting at the ith step if $A_{ii}^{i-1} \neq 0$ which is exactly the condition that $E(A^{i-1}, i)$ is nonsingular. If the Gauss-Jordan algorithm above can run to completion without generating any singular matrices E_i then

$$A^n = I = AE_1E_2 \ldots E_n$$

or

$$A = E_n \ldots E_1.$$

Conversely, suppose A can be decomposed as in *1*. Then

$$AE_1 \ldots E_i = E_{i+1} \ldots E_n. \tag{3.1}$$

For $i = 1, \ldots, n$, the product on the right side of (3.1) does not change bits 1 through i and thus, as a matrix it agrees with the identity matrix in its first i rows. This determines E_i as the unique matrix that describes the ith step of column-wise Gauss-Jordan elimination without pivoting. This establishes the equivalence of *1* and *3*.

Finally, since at step i the Gauss-Jordan procedure adds multiples of column i to the other columns, the determinants of the principal submatrices do not change. Thus, if the Gauss-Jordan algorithm runs to completion, then the principal submatrices are all nonsingular. Conversely, if the principal submatrices are all nonsingular, the ith pivot cannot be 0, for the product of the first i pivots is the determinant of the ith principal submatrix. Having now established the equivalence of *3* and *4*, the proof is complete. $\square$

Corollary 3.1 *If $A = LU$ where L and U are nonsingular lower and upper triangular matrices, then A can be decomposed as $A = E_n \ldots E_1$. Thus Gaussian elimination, rather than Gauss-Jordan elimination, can be used to test whether A has this decomposition, though Gauss-Jordan is needed to construct the decomposition.*

Corollary 3.2 *Let $d_1, \ldots, d_n$ be a reordering of the numbers 1 through n. Then A can be decomposed as $A = E_{d_n} \ldots E_{d_1}$ if and only if all the diagonal submatrices of A given by rows and columns $d_1, \ldots, d_i$ are nonsingular for $i = 1, \ldots, n$. Equivalently, if $A = PLUP^T$, where P is a permutation matrix, then the index transformation corresponding to A can be performed as a sequence of conditional exchanges in an order specified by P.*

Proof The Gauss-Jordan algorithm, when run consecutively on rows d_1 through d_n, gives the desired decomposition if it exists, or breaks down through the need for pivoting if it does not. $\square$

Corollary 3.3 *If A is a nonsingular upper (or lower) triangular matrix, then an algorithm exists that traverses the dimensions in any order.*

Proof All diagonal minors of A are determinants of upper (or lower) nonsingular triangular matrices. $\square$

Corollary 3.4 *A cycle or any matrix at all that has all diagonal entries equal to* 0 *cannot be written as a product of conditional exchanges in any order.*

Proof No 1×1 principal submatrix is equal to 1. $\square$

Corollary 3.5 *No permutation matrix can be written as a product of conditional exchanges in any order.*

Proof All principal submatrices that include exactly one row and column from one of the component cycles are singular. $\square$

Corollary 3.6 *Any nonsingular A defines an index transformation that can be performed as a pipelined sequence of conditional exchanges followed by a dimension permutation algorithm.*

Proof Any nonsingular A can be written as PLU by performing Gaussian elimination with partial pivoting. $\square$

Since we have shown how to construct an algorithm corresponding to any LU, and since algorithms for accomplishing address permutations exist, we can now accomplish any linear transformation.

Corollary 3.7 *If A has the form $U_1 P U_2$ where U_1 and U_2 are upper triangular, then $A = PA'$ where A' has all nonsingular principal submatrices. Therefore A' can be implemented as a sequence of conditional exchanges in standard order.*

Proof Let $A' = P^T U_1 P U_2$. Since U_1 is upper triangular, every diagonal minor of U_1 and hence $P^T U_1 P$ is nonzero. The kth principal submatrix of A' is given by the product of the kth principal submatrix of $P^T U_1 P U_2$ and that of U_2 and hence is nonsingular. $\square$

The triple product $U_1 P U_2$ arises on the CM–2 multiprocessor when transposing a matrix, collapsing or separating axes, or changing the layout of an array on the machine. In this case, U_1 and U_2 denote Gray coding and decoding operations respectively. The Gray code is decoded, the address bits are permuted, and then the bits are encoded in possibly a new way. This type of operation is explored in the next section.

4 Gray Codes and Hypercube Multiprocessors

Gray coding and decoding of arbitrary axes is an important communication pattern on hypercube multiprocessors. The outline of this section is as follows:

1. A brief digression into the history of Gray coding, which is not as well-known as perhaps it ought to be.

2. Derivation of widely known properties of the Gray code using the linear algebra framework.

3. Applications of the theory from the previous section toward new results about Gray coding.

The binary-reflected Gray code has had a most curious history in that it has appeared in so many different applications. It was invented by the French engineer Emile Baudot (1845-1903) for the purpose of sending and receiving telegraphs [10]. In 1872, it appeared in the solution of the so-called Chinese ring puzzle (see Gardner [7]), and it is also the solution of the famous Tower of Hanoi puzzle. Frank Gray developed the code that now bears his name during the 1940's, though it was first published in 1953 in a patent for a so-called pulse code modulation tube. Later, the Gray code has been used in many ways in analog-to-digital converters.

Though probably obvious to many, we believe that Gilbert [8] in 1958 was the first to point out explicitly that the consecutive numbers in the Gray code sequence form a Hamiltonian path on a hypercube. During that time it was fashionable to enumerate other Hamiltonian paths on the hypercube as well.

With the invention of multiprocessor computers with hypercube networks, it became possible for the first time to make use of these paths on real physical hypercubes. Many authors independently observed the utility of this property for embedding rings and higher dimensional meshes. CM–2 system software uses these embeddings to store grids in such a manner that it is invisible to the programmer. Indeed it would be easy to believe erroneously that the CM-2 has a separate network for grid communication.

On the CM-2, data is considered to be in "grid" order (also known as "NEWS" order) if the data labeled i is located in the processor with the label Gi, where G is the gray coding operator. The data is in "cube" order (also known as "send" order) if the data labeled i is in fact located in node i. Since certain algorithms run more efficiently if the data is in "grid" order while other algorithms run faster in "cube" order, there has been need for routines to convert between the two ordering schemes. The communication pattern that converts a single one-dimensional axis from "cube" to "grid" order is $f(i) = Gi$ and from "grid" to "cube" order is given by $f(i) = G^{-1}i$, where

G and G^{-1} are given below. The key point is that they are linear index transformations.

In numerical linear algebra [9], it is common to embed Householder reflections or Givens rotations inside a larger identity matrix so as to operate on selected components of a vector. Analogously, one can "Gray code" certain components of a vector. On hypercubes it is usual to associate blocks of components with various axes, and then one refers to Gray coding an axis.

The Gray code encoding operator G is deceptively simple, defined by the condition that G be a linear operator on vectors modulo 2 and that

$$G(j_n) = e_n, \quad n = 1, 2, \ldots\ . \tag{4.2}$$

Since $e_n = j_n + j_{n-1}$, it follows that

$$G(e_n) = G(j_n + j_{n-1}) = e_n + e_{n-1}. \tag{4.3}$$

Let G_n denote the restriction of the Gray code encoding operator G to the finite dimensional space F_2^n. We then have that G_n is a linear transformation on F_2^n whose $n \times n$ matrix representation is

$$G_n = \begin{pmatrix} 1 & 1 & & & & \\ & 1 & 1 & & 0 & \\ & & 1 & & & \\ & & & \ddots & 1 & \\ & 0 & & & 1 & 1 \\ & & & & & 1 \end{pmatrix}.$$

The Gray code decoding operator G^{-1} is uniquely defined by

$$G^{-1}(e_n) = j_n, \quad n = 1, 2, \ldots\ . \tag{4.4}$$

The restriction of G^{-1} to the finite dimensional space F_2^n has the $n \times n$ matrix representation

$$G_n^{-1} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ & 1 & 1 & \cdots & 1 & 1 \\ & & 1 & \cdots & 1 & 1 \\ & 0 & & \ddots & 1 & 1 \\ & & & & 1 & 1 \\ & & & & & 1 \end{pmatrix}.$$

We now let S_n be the sequence of 2^n elements of F_2^n in numerically increasing order. To obtain the same sequence in reverse order, add j_n to each element; hence the name **vector reversal**. Let $G(S_n)$ denote the sequence of Gray codes of elements of S_n. Since

$$G(i + j_n) = G(i) + G(j_n) = G(i) + e_n, \tag{4.5}$$

we have proved a very important property of the binary-reflected Gray code that is often taken as part of the standard definition:

Theorem 4.1 (Reversal Property) *The reversal of the sequence $G(S_n)$ is equal to the sequence $G(S_n)$ with the bit in the nth position complemented.*

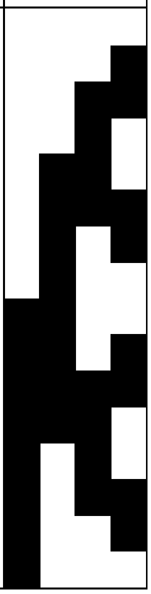
A related observation is

Theorem 4.2 *Consecutive members of the sequence $G(S_n)$ differ in exactly one bit.*

Proof Two consecutive numbers can always be written as $i + j_{k-1}$ and $i + e_k$, where neither i nor Gi has a 1 in the k least significant bits. Since $Gj_{k-1} = e_{k-1}$ and $Ge_k = e_{k-1} + e_k$, the bit in which the Gray codes differ is the kth. $\square$

Following Gilbert [8], the reversal property is readily grasped by the eye from the diagram below in which 0 is represented by a blank space, and 1 with a black square.

S_4	$G(S_4)$
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000



Since G and G^{-1} are both upper triangular, by Corollary 3.3 Gray coding and decoding can be accomplished in any order. For example, when $n = 4$, we express the algorithm from Johnsson [12] in our notation:

$$G = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & 1 \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & 1 & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

and

$$G^{-1} = \begin{pmatrix} 1 & 1 & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & 1 & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & 1 \\ & & & 1 \end{pmatrix}.$$

Notice that the algorithms perform encoding from low-order bits to high-order bits, while decoding is performed from high-order bits to low-order bits. Algorithms for the reverse order were first developed by Johnsson [16], and the existence and use of algorithms for any order are discussed by Johnsson and Ho [15, 17].

One particularly interesting example is decoding starting from the least significant bit. In this case F_k^p has a 1 in row $p(k)$ and column n. It readily follows that if an edge is used in the subcube defined by $v_n = 0$, then it is not used in the subcube $v_n = 1$. This is the basis for a new algorithm given by Johnsson and Ho [15] that takes better advantage of the available bandwidth.

More generally, if A can be decomposed as the product of conditional exchanges E_i over distinct dimensions, then if the element in the ith row and jth column of E_i is 1 for every i and if the jth row of A matches the identity matrix, then the wires along dimension j can be used to take better advantage of the available bandwidth.

We define a code change operation as any $G_1 G_2^{-1}$ combination. As an example, treating a two-dimensional matrix as a one-dimensional vector on a hypercube involves a code change.

Corollary 4.1 *All code change operations have pipelined algorithms.*

Proof Since decode and encode operations are both upper triangular, so is their composition. $\square$

Corollary 4.2 *All code change operations have pipelined algorithms for each permutation of the dimensions.*

5 Dimension Permutations and Hypercube Multiprocessors

We have seen previously that dimension permutations correspond to permutation matrices. Why use n^2 elements to describe an object only requiring n? There are two answers. One is that on a hypercube multiprocessor it is frequently desirable to combine coding, decoding, and dimension permutation operations [13]. Matrix notation allows us to put all of these operations into the same setting. The other answer is that we can derive results about these matrices without actually explicitly writing down the entries of the matrix. In this latter context, we are really only deriving algebraic results for the symmetric group on n objects.

On hypercube multiprocessors, dimension permutations induce a fairly complicated motion on the machine. Remember that a dimension permutation is an index transformation on n objects that induces a more complicated permutation of 2^n objects. Factorizing the permutation matrix into simpler matrices allows a compact way of thinking about algorithms.

A dimension permutation on all dimensions forming a shuffle is represented by a circulant matrix as shown below for five dimensions.

$$S_{1,5} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

An unshuffle is also represented as a circulant matrix,

$$S_{1,5}^{-1} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In our next definition we precisely define shuffle permutations.

Definition 5.1 *A* **shuffle permutation** *on indices $i, i+1, \ldots, j$ is the transformation whose matrix $S_{i,j}$ is given as the identity except in columns $i, i+1, \ldots, j$, which are $e_{i+1}, \ldots, e_{j-1}, e_j, e_i$ respectively; in other words, the appropriate columns are shifted left circularly.*

On hypercube multiprocessors, it is convenient to implement dimension permutations as sequences of elementary bit-exchanges:

Definition 5.2 *An index transformation is defined to be an* **elementary bit-exchange** *if its matrix representation is a permutation matrix that is the identity except in two rows and columns. We denote such a matrix $E_{i\leftrightarrow j}$, where i and j are the distinguished rows and columns.*

Definition 5.3 *An index transformation is defined to be a* **bit-exchange** *if its matrix representation is a symmetric permutation matrix.*

Lemma 5.1 *A bit-exchange matrix can be expressed as the product of independent elementary bit-exchange matrices, and, conversely, the product of independent elementary bit-exchange matrices can be reduced to a bit-exchange matrix.*

Lemma 5.2 *Any shuffle permutation can be expressed as the product of two bit-exchange matrices.*

Proof Renumber the shuffle, if necessary, to be $S_{1,n}$. $S_{1,n}$ is the product of the following two bit-exchange matrices: $E_1 = E_{1\leftrightarrow n}E_{2\leftrightarrow n-1}\ldots$ and $E_2 = E_{1\leftrightarrow n-1}E_{2\leftrightarrow n-2}\ldots$. $\square$

Lemma 5.3 *Any permutation matrix can be expressed as the product of two bit-exchange matrices.*

Proof The proof is similar to the proof of Lemma 5.2 once the permutation matrix is separated into disjoint cycles. $\square$

These facts can be quite useful in practice. Code written for the CM–2 to accomplish the bit-reverse operation [4] was easily generalized to the bit-exchange operation. Using Lemma 5.3, any dimension permutation had an implementation. This was the motivation for a large software project, known as the "twuffler," to accomplish operations of the form $G_1 P G_2^{-1}$.

Notice that if $j = i + 1$, then $E_{i\leftrightarrow j} = S_{i,j}$.

As is well-known [14], a shuffle or unshuffle can be carried out as a sequence of dimension exchanges in two convenient ways, as illustrated by the following examples when $n = 5$:

$$\text{Example 1: } S_{1,5} = E_{1\leftrightarrow 2}E_{2\leftrightarrow 3}E_{3\leftrightarrow 4}E_{4\leftrightarrow 5}$$

and

$$\text{Example 2: } S_{1,5} = E_{1\leftrightarrow 5}E_{1\leftrightarrow 4}E_{1\leftrightarrow 3}E_{1\leftrightarrow 2}.$$

In fact, there are exactly n factorizations of the shuffle matrix into elementary bit-exchanges with $n-1$ factors. Since elementary bit-exchanges are their own inverses, factorizations of $S_{1,5}^{-1}$ are obtained by reversing the order of the factors of $S_{1,5}$.

Generalizing the two examples, we see that

$$S_{i,j} = E_{i\leftrightarrow i+1}E_{i+1\leftrightarrow i+2}\ldots E_{j-1\leftrightarrow j} \tag{5.6}$$

and

$$S_{i,j} = E_{i\leftrightarrow j}E_{i\leftrightarrow j-1}\ldots E_{i\leftrightarrow i+1} \tag{5.7}$$

where the product is in increasing order in Equation (5.6) and in decreasing order in Equation(5.7). Note how in Example 1, all dimensions but the first and last are used twice, while in Example 2 only dimension 1 is used more than once. With $n-1$ factors, the total use of dimensions must be $2n-2$, so that Example 1 best load balances all of the dimensions, while Example 2 represents the worst case of load balancing the dimensions. However, the data motion in Example 1 accounting for the factor-of-two difference between the two approaches is unnecessary and can be eliminated [14].

Furthermore, even though Example 2 appears unfavorable, if the fixed dimension is a dimension local to a node, then all bit-exchanges are between adjacent nodes in a binary cube, while the factorization given in Example 1 requires communication between nodes at distance two. The factorization given by Example 2 is the basis for the algorithms by Swartztrauber [25], and several of the algorithms by Johnsson and Ho [14, 18].

These algorithms are based on the following observation. From (5.6) we see that $E_{i\leftrightarrow i+1}S_{i,j} = S_{i+1,j}$. Combining this with (5.7), we obtain that

$$E_{i\leftrightarrow i+1} \prod_{k=j,\ldots,i+1} E_{i\leftrightarrow k} = S_{i+1,j}.$$

Thus, a shuffle on $n-1$ dimensions can be expressed as the product of $n+1$ elementary bit-exchanges, with the same dimension used in every bit-exchange. If dimension i in fact represents local memory, the advantages of this approach are clear. Each elementary bit-exchange represents one-hop communication on the hypercube.

Another approach that has proved convenient is to express a shuffle permutation as a composition of several shuffle permutations on fewer dimensions. This method can be used to devise algorithms with optimal concurrency in communication [20, 21, 14, 18].

Again using Equations (5.6) and (5.7),

$$\begin{aligned} S_{i,k} &= S_{i,j}S_{j,k} \\ &= E_{i\leftrightarrow j}S_{i,j-1}S_{j,k} \\ &= E_{i\leftrightarrow j}S_{j,k}S_{i,j-1}, \end{aligned}$$

taking advantage of the fact that $S_{j,k}$ and $S_{i,j-1}$ commute. Thus, if there are several elements per node, some elements can be permuted according to $S_{j,k}$ first, others according to $S_{i,j-1}$ first.

6 Conclusion

We have cast index transformation algorithms in a linear algebraic framework with applications towards hypercube algorithms. Such a framework has multiple purposes. One is to express ideas that are already commonly known, but in a more concise language. Another more important purpose is to shed light on the existence of algorithms and to construct them automatically. We have demonstrated both.

Acknowledgements

We would like to thank Thinking Machines Corporation, and particularly Marshall Isman and Ted Tabloski, for supporting this work and related projects on the Connection Machine. They provided financial support and a sense of spirit that makes such a project a pleasure. In addition, the first author would like to thank Thinking Machines Corporation and the third author for inviting him to work on this project as it related to the "twuffler" project in the summer of 1990 during which the earliest version of this paper was drafted [3]. Finally, we would like to thank Emily Stone for her excellent assistance in editing.

References

[1] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, Optimal communication algorithms for hypercubes, submitted for publication.

[2] T.H. Cormen, Fast Permuting on Disk Arrays, *Brown/MIT VLSI Conference*, (1992).

[3] A. Edelman, *The algebra of shuffling and Gray-coding on a hypercube*, Thinking Machines Corporation, Semi-internal note, August 9, 1990.

[4] A. Edelman, Optimal matrix transposition and bit reversal on hypercubes: all-to-all personalized communication, *J. Parallel Dist. Comp., 11*, (1991), 328–331.

[5] P.M. Flanders, A Unified Approach to a Class of Data Movements on an Array Processor, *IEEE Transactions on Computers. C-31*, (1982), 809–819.

[6] D. Fraser, Array permutations by index-digit permutation, *Journal of the Association for Computing Machinery. 22*, (1976), 298–308.

[7] M. Gardner, Mathematical Games. The curious properties of the Gray code and how it can be used to solve puzzles, *Scientific American. 227* (August 1972), 106–109.

[8] E.N. Gilbert, Gray codes and paths on the n-cube, *Bell System Technical Journal. 37* (1958), 815–826.

[9] G. Golub and C.F. Van Loan, *Matrix Computations, second edition*, Johns Hopkins University Press, Baltimore, 1989.

[10] F.G. Heath, Origins of the binary code, *Scientific American. 227* (August 1972), 76–83.

[11] M. van Heel, A fast algorithm for transposing large multidimensional image data sets, *Ultramicroscopy 38*, (1991), 75–83.

[12] S.L. Johnsson, Communication efficient basic linear algebra computations on hypercube architectures, *J. Parallel Distributed Comput. 4* (1987), 133–172.

[13] S.L. Johnsson and C.T. Ho, Algorithms for matrix transposition on Boolean N-Cube Configured Ensemble Architectures, *SIAM J. Matrix Anal. Appl. 9*, (1988), 419–454.

[14] S.L. Johnsson and C.T. Ho, Optimal Algorithms for Stable Dimension Permutations on Boolean Cubes, *The Third Conference on Hypercube Concurrent Computers and Applications*, ACM Press, 725–736, (1988).

[15] S.L. Johnsson and C.T. Ho, On the conversion between binary code and binary-reflected Gray code on Boolean cubes, *Harvard University Technical Report 20-91*, (1991).

[16] S.L. Johnsson, Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures, *Models of Massively Parallel Computation*, Morgan Kaufman, (1990), 223–389.

[17] S.L. Johnsson and C.T. Ho, The Complexity of Reshaping Arrays on Boolean Cubes, *The Fifth Distributed Memory Computing Conference*, IEEE Computer Society, (1990), 370–377.

[18] S.L. Johnsson and C.T. Ho, Generalized Shuffle Permutations on Boolean Cubes, *J. Parallel and Distributed Computing. 1992*, to appear.

[19] J. Keohane and R.E. Stearns, Routing linear permutations through the omega network in two passes, *Proceedings of The 2nd Symposium on the Frontiers of Massively Parallel Computing*, IEEE, No. 88CH2649–2, 476–82.

[20] D. Nassimi and S. Sahni, An optimal routing algorithm for mesh connected parallel computers, *Journal of the Association for Computing Machinery, 27*, (1980), 6–29.

[21] D. Nassimi and S. Sahni, Optimal BPC Permutations on a Cube Connected SIMD Computer, *IEEE Transactions on Computers. C-31*, (1982), 338-341.

[22] H. Stone, Parallel processing with the perfect shuffle, *IEEE Transactions on Computers. 20*, (1971), 153–161.

[23] Q.F. Stout and B. Wagar, Passing Messages in Link-Bound Hypercubes, *Proceedings of Hypercube Multiprocessors 1987*, SIAM Publications, Philadelphia, 1987.

[24] Q.F. Stout and B. Wagar, Intensive hypercube communication I: prearranged communication in link-bound machines, *Computing Research Laboratory, University of Michigan, Technical Report CRL-TR-9-87*, (1987).

[25] P.N. Swarztrauber, Multiprocessor FFT's, *Journal of Parallel Computing. 5*, (1987), 197–210.

[26] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM Publications, Philadelphia, 1992.