Optimal Matrix Transposition and Bit Reversal on Hypercubes: All–to–All Personalized Communication

Alan Edelman
Department of Mathematics
University of California
Berkeley, CA 94720

Key words and phrases: bit reversal, communication, Connection Machine, hypercube, matrix transpose, parallel computing

Running head: All–to–all personalized communication.

Alan Edelman
Department of Mathematics
University of California
Berkeley, CA 94720
(510) 643-5695

Abstract

In a hypercube multiprocessor with distributed memory, messages have a street address and an apartment number, i.e., a hypercube node address and a local memory address. Here we describe an optimal algorithm for performing the communication described by exchanging the bits of the node address with that of the local address. These exchanges occur typically in both matrix transposition and bit reversal for the fast Fourier transform.

1 Introduction

Imagine the residents of $N = 2^d$ cities creating zoos each with N distinguishable specimens of some native animal. The Sydney zoo might have N very different looking koalas, the New Delhi zoo might have N identifiable elephants, while the Boston zoo might have N automobile drivers. Assume these N cities are serviced by an airplane network forming a d dimensional hypercube. What would be the most efficient way to use the air routes to reorganize the zoo specimens so that now each zoo has one of every animal?

Consider the example of an $N \times N$ matrix A stored on a d-dimensional hypercube with $N = 2^d$ nodes. A natural data structure stores a_{ij} in node i and at local memory location j. Denote this address by $i|j$. Matrix transpose requires that the N messages at node i $(\{i|j : j = 0, 1, \ldots, N-1\})$ be divided across the machine into the N memory locations with local address i $(\{j|i : j = 0, 1, \ldots, N-1\})$.

As another example, let us calculate the FFT of a vector of length $N^2 = 2^{2d}$. A natural data structure splits the $2d$ address bits into a d bit node address and a d-bit local address. As is well known (see [8], for example), the result of an FFT is in bit-reversed order, and a separate communication is needed to reorganize the data. This simply means that the real number located at (binary) address $i_{2d} \ldots i_d | i_{d-1} \ldots i_0$ is swapped with the number at address $i_0 \ldots i_{d-1} | i_d \ldots i_{2d}$. Thus bit reversal is similar to the matrix transpose: the N messages at node i need to be divided across the machine into the the N memory locations with local address i.

These two examples illustrate a communication pattern in which d node address bits are interchanged with d local address bits. In this situation, each node wants to send a different message to each other node, and thus this pattern is denoted all–to–all personalized communication by the authors of [5, 6, 7] who noticed that these examples and others merit implementing this type of communication as a programming primitive. The pattern is given a different name, "total exchange" in [1], because all pairs of nodes trade (unique) data.

This is different from "all–to–all (non–personalized) broadcasting" in which every message is replicated and sent to every node ([6]). We consider only the personalized case.

The total exchange algorithm presented in [1] optimally solves the problem presented. Optimal algorithms for the case when the number of messages for each pair of nodes is a multiple of the cube dimension can be found in [6] and [9]; these algorithms are not optimal for our situation. More recently, Johnsson and Ho have devised optimal algorithms for the single packet case as well [4].

We have independently derived an optimal algorithm that takes advantage of the symmetry of the hypercube. We do this by computing a schedule whose computation time is negligible compared to the communication time.

Our algorithm is based on performing one–to–all personalized communication ([6]) from node 0. In "one–to–all personalized communication," one node has a unique message for each of the other nodes (see Figure 1). We construct an optimal algorithm for routing the data from node 0 in such a way that if every other node performs the same sequence of actions in its own relativized coordinate system, there will be no contention for communications channels.

For uses of this algorithm in FFTs and for index digit permutations, see [2], [3], and [9]. Our implementation on the Connection Machine is used in the bit reversal for the FFT and other IO reorderings, but we do not discuss this in detail here. We take the view that the problem on the hypercube is interesting, even if no machine exists on which to implement it.

2 Hypercubes

As is well known, a d dimensional hypercube (sometimes known as a Boolean d-cube) consists of 2^d nodes. Nodes j and k are nearest neighbors if the d-bit binary representations of j and k differ in exactly one bit. Let $j \oplus k$ be the bitwise exclusive OR of the binary representations of j and k, and let $\mathrm{Sum}(j)$ denote the number of ones in the bitwise repre-

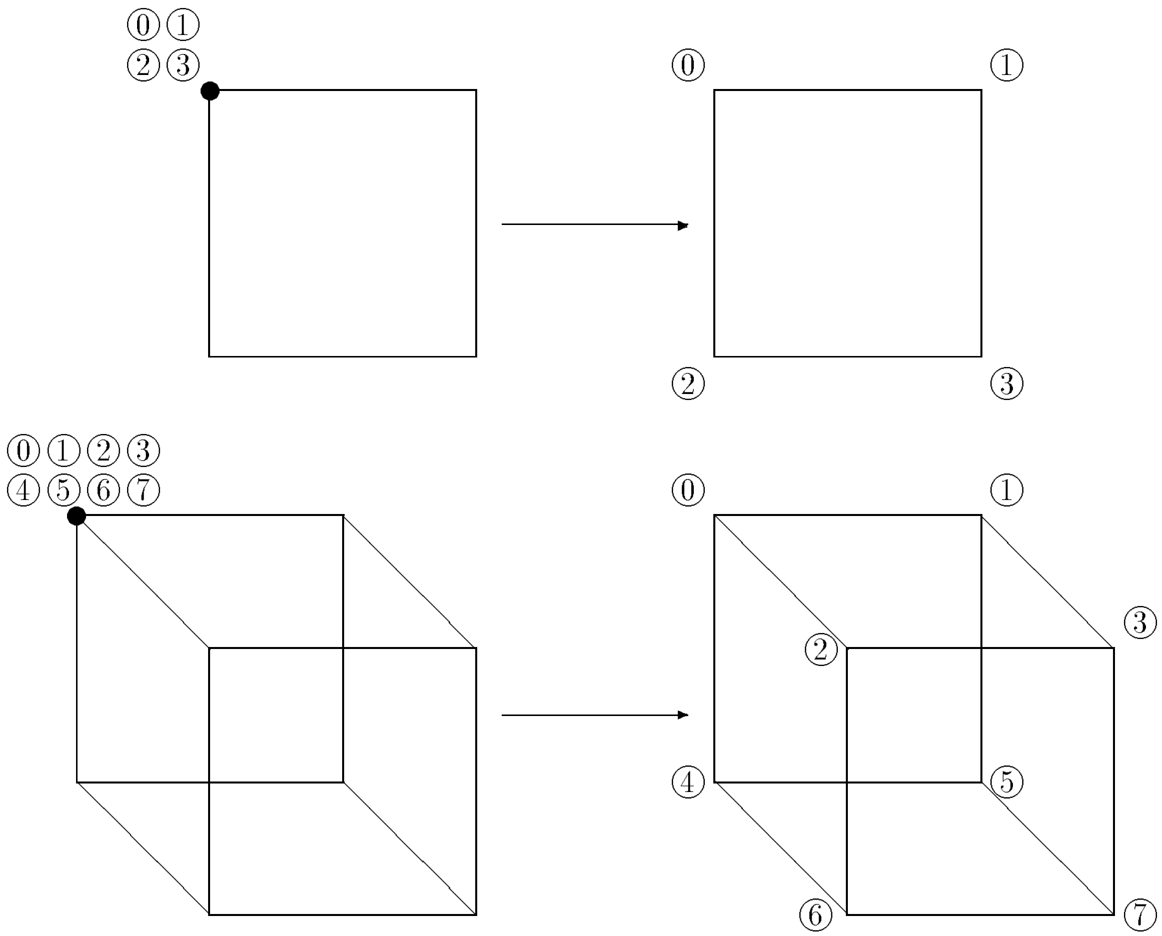

Figure 1: One–to–all personalized communication on a 2-cube and a 3-cube: Unique messages are sent from the leader to every other node.

sentation of j. The minimum path between nodes j and k has length $\text{Sum}(j \oplus k)$. It is then obvious that the average distance between two random nodes is $d/2$ links.

There are $d2^{d-1}$ non-directed edges or links in a d dimensional cube. In this paper, we typically assume that each non-directed edge is replaced by two directed edges, so that there are exactly $d2^d$ directed edges. At each node, there are d links for outgoing data and d links for incoming data. Let outgoing or incoming link k be the link that changes bit k.

3 Basic Algorithm

In this section, we describe our optimal algorithm for performing the matrix transpose described as our first example. Thus, we assume that we have a d-dimensional hypercube with 2^d local storage locations at each node, and that we have an $N \times N$ matrix, where $N = 2^d$, that we wish to transpose. We begin by assuming that the message (32-bit word) a_{ij} is located at the address $i|j$, i.e., node i memory location j.

We also make use of a relative address of $i|j$:

Definition 3.1 *The* **relative address** *of* $i|j = i \oplus j$.[1]

We immediately observe that the relative address of $i|j$ and that of $j|i$ are the same; i.e., the relative address of a message is preserved under transposition. Our method is to use the links effectively to move the message from $i|j$ to $j|i$ while preserving the relative address:

Definition 3.2 *A* **relative address preserving data movement** *is a communication where at each intermediate step, the data preserve their relative addresses.*

The relative address is a powerful tool as it captures the underlying symmetry of the communication. With it, we can focus on one particular node, say node 0, with the assurance that all the other nodes are performing the appropriate operations relative to their address.

[1] The difference between matrix transpose and bit reversal is in this definition. For bit reversal, the relative address is $\text{Reverse}(i) \oplus j$. For arbitrary all–to–all personalized communication, the data in node i destined for node j needs to be loaded into local memory location j.

Furthermore, a relative address preserving data movement allows the motion of data to occur without storing any information regarding where the data was coming from or going to. Thus no address bits need to be sent and the information is tied to the memory location.

We make some simple observations.

Lemma 3.1 *The minimum number of links that the word at* $i|j$ *needs to cross to reach* $j|i$ *is exactly* $\mathrm{Sum}(i \oplus j)$*. Let* L *be the set of bits of* $i \oplus j$ *that are 1. If the word at* $i|j$ *traverses all the dimensions in* L *in any order, it will reach its destination. Moreover, if in each node the word at relative address* $i \oplus j$ *marches through the same numbered dimensions simultaneously, after* $\mathrm{Sum}(i \oplus j)$ *steps, the words will all reach their destination.*

As an example, let d=3, and consider the words in each node with relative address 5. In all eight nodes, the word with relative address 5 needs to travel over links 0 and 2. If we send all these words simultaneously first over link 0 and then over link 2, they will all arrive at their destination in two communication steps.

Theorem 3.1 *The transpose requires at least* 2^{d-1} *communications steps.*

Proof A lower bound is given by

$$\frac{\text{total number of links that must be crossed}}{\text{total number of links available}}$$

or

$$\frac{\sum_{i,j} \mathrm{Sum}(i \oplus j)}{d2^d} = \frac{(d/2)(2^{2d})}{d2^d} = 2^{d-1},$$

since the average value for $\mathrm{Sum}(i \oplus j)$ is $d/2$. (Other proofs of this result can be found in [1, 6].) □

Theorem 3.2 *An optimal schedule is given by a* 2^{d-1} *by* d *array of numbers* w_{ij}*,* $i = 0, 1, \ldots, 2^{d-1} - 1$*,* $j = 0, 1, \ldots, d-1$ *whose kth binary bit is denoted by* w_{ij}^k *satisfying*

1. Wire appropriateness: $w_{ik}^k = 1$ for all i, k.

2. Row uniqueness: $w_{ij_1} \neq w_{ij_2}$, if $j_1 \neq j_2$ for all i.

3. Column uniqueness: $w_{i_1 j} \neq w_{i_2 j}$, if $i_1 \neq i_2$.

Given such an array, the optimal algorithm is

For $i = 0, 1, \ldots, 2^{d-1} - 1$

In each node, the expression $w_{ij} \oplus$ **(node-address)** specifies the local memory location for the word sent over link j, ($j = 0, 1, \ldots, d-1$).

The word arriving over link j takes its place.

Proof In Lemma 3.1, we observed that the relative address must have a 1 in the kth bit for it to be appropriate to send the data over link k. Row uniqueness guarantees that we do not move the same message over two different links at the same time step; column uniqueness guarantees that we do not move any message twice over the same link (which would be backtracking). These conditions guarantee that the relative address w appears exactly Sum(w) times in the array, once in each column j for which $w^j = 1$. Thus, the data at each relative address crosses all the appropriate links exactly once. □

We now propose an inexpensive algorithm for computing an array with the three conditions described in Theorem 3.2.

Direct algorithm: Let $n_i = 2i + 1$ and n_i^k be the kth bit in its binary representation. Compute w_{ij} by performing the following operations on n_i: 1) complement bit $j+1$ and 2) interchange bit 0 and bit j. When $j = d-1$, complementing bit $j+1$ should be considered a vacuous operation.

Proof The wire appropriateness condition is satisfied since $w_{ik}^k = n_0^i = 1$. Row uniqueness for w_{ik}, where $k \neq d-1$ follows from the complementing. For $k = d-1$ a separate trivial argument confirms row uniqueness once again. Column uniqueness follows from the clear fact that column j is some permutation of the 2^{d-1} numbers with bit j equal to 1. $\square$

Figure 2 explicitly shows the data movement when $d = 3$. The data starting at node i at location j is indicated by i^j. The data starting at node 0 is denoted with a ♠ to facilitate tracking the data from one particular node. Though it is satisfying to see all the data arriving at their destinations explicitly, we do not recommend this diagram as a means for understanding our algorithm. Rather Table i contains the same information as Figure 2, in a more compact manner.

Tables I, II, and III illustrate the schedule for $d = 3, 4$, and 5. Note that the schedules list the relative address of the data that is moved out of each node on a particular wire at a particular time. The three conditions: wire appropriateness, row uniqueness, and column uniqueness are readily observed in these tables. Also observe that each binary number w appears exactly Sum(w) times in the array, once in each column k for which w^k=1.

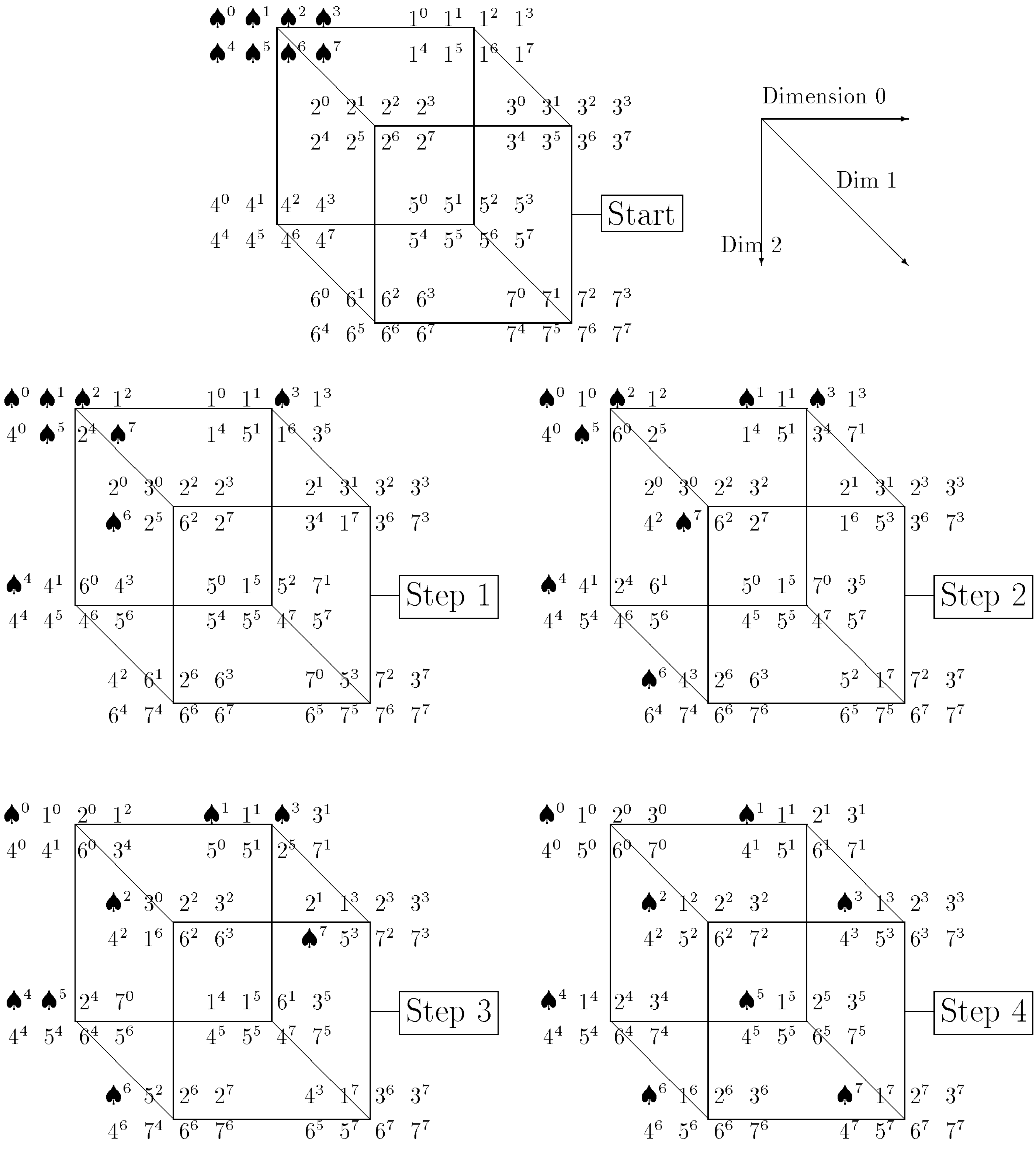

Figure 2: Data movement for all–to–all personalized communication (see text and compare Table i)

Table i: Schedule for $d = 3$

.

		Dimension 0	Dimension 1	Dimension 2
Communication Step	1	011	110	100
	2	001	111	110
	3	111	010	101
	4	101	011	111

Table ii: Schedule for $d = 4$

Communication Step		Dimension 0	1	2	3
	1	0011	0110	1100	1000
	2	0001	0111	1110	1010
	3	0111	0010	1101	1100
	4	0101	0011	1111	1110
	5	1011	1110	0100	1001
	6	1001	1111	0110	1011
	7	1111	1010	0101	1101
	8	1101	1011	0111	1111

Table iii: Schedule for $d = 5$

Communication Step	Dimension 0	1	2	3	4
1	00011	00110	01100	11000	10000
2	00001	00111	01110	11010	10010
3	00111	00010	01101	11100	10100
4	00101	00011	01111	11110	10110
5	01011	01110	00100	11001	11000
6	01001	01111	00110	11011	11010
7	01111	01010	00101	11101	11100
8	01101	01011	00111	11111	11110
9	10011	10110	11100	01000	10001
10	10001	10111	11110	01010	10011
11	10111	10010	11101	01100	10101
12	10101	10011	11111	01110	10111
13	11011	11110	10100	01001	11001
14	11001	11111	10110	01011	11011
15	11111	11010	10101	01101	11101
16	11101	11011	10111	01111	11111

Acknowledgements

I thank Lennart Johnsson for several interesting discussions, and for introducing me to this problem and related work. Without his motivating me to attack this problem, I never would have worked on it. I further thank Ching–Tien Ho for relating previous work in this area. I also acknowledge Mark Bromley, Steve Heller, Mike McKenna, and Walter Mascarenhas for all of their help implementing this and related algorithms on the Connection Machine.

Biography

Alan Edelman received his Ph.D. degree in mathematics from the Massachusetts Institute of Technology in 1989, and the BS and MS degrees in mathematics from Yale University in 1984.

He is currently a Morrey Assistant Professor at the mathematics department at the University of California at Berkeley after finishing an NSF-Nato postdoctoral fellowship at the CERFACS computing center in Toulouse, France.

Alan's research interests include numerical linear algebra, parallel computing, eigenvalues of random matrices, and approximation theory. He recently shared the Gordon Bell prize for parallel computing and the Householder prize for numerical linear algebra.

References

[1] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, Optimal communication algorithms for hypercubes, *J. Parallel Distributed Comput.*, to appear.

[2] 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.

[3] D. Fraser, Array permutation by index-digit permutation, *J. ACM 22* (1976), 298-308.

[4] C.T. Ho, personal communication, August 1, 1990.

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

[6] S.L. Johnsson and C.T. Ho, Optimum broadcasting and personalized communication in hypercubes, *IEEE Transactions on Computers. 38* (1989), 1249-1268.

[7] 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.

[8] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press, Cambridge, 1986.

[9] P.N. Swarztrauber, Multiprocessor FFTs, *Parallel Computing 5* (1987), 197-210.