On Optimal and Practical Routing Methods for a Massive Data Movement Operation on Hypercubes[1]

by

Rajendra V. Boppana and C. S. Raghavendra

Technical Report No. CENG 90-20
July 1990

Dept. of Electrical Engineering–Systems
University of Southern California
Los Angeles, CA 90089–0781

Phone: (213) 743-5532
Email: raghu@surya.usc.edu

[1]This research is supported by the NSF grant no. DCI-8452003.

On Optimal and Practical Routing Methods for a Massive Data Movement Operation on Hypercubes

Rajendra V. Boppana and C. S. Raghavendra

11 July 1990

Abstract. The all-to-all personalized communication is a massive data movement operation that arises frequently in problems like large matrix transposition on hypercube computers. In literature, various researchers proposed routing methods to solve this problem using multiport packet switching communication, which are not practical. In this paper, we present a simple routing algorithm to solve this routing problem. This algorithm is suitable for implementation in hypercube computers with circuit switching communication, for example, Intel's iPSC/2. During the routing process, a processor need to send and receive data from at most one communication link. For large data transmissions, the presented algorithm is very close to optimal.

Key words: circuit switching, data movement, distributed computing, hypercube computers, vector reversal permutation.

1 Introduction

Hypercube based parallel architectures are popular with the parallel processing community because of their versatile and high bandwidth interconnection network. This has prompted several researchers to use them to solve various communication intensive problems such as matrix-matrix multiplication, LU-factorization, matrix transposition, etc. Some of the well known and frequently used data movement operations in solving these problems are [5]: (a) one-to-all broadcasting—a processor has to send a common message to all the other processors in the system; (b) one-to-all personalized communication—a processor has to send every other processor in the system a private message; (c) all-to-all broadcasting—each processor in the system has to send a message to all other processors in the system; and (d) all-to-all personalized communication—each processor has to send a private message to every other processor in the system. Of these four, the all-to-all personalized communication has the most bandwidth requirements. For fast and efficient solution of a variety of linear algebra problems on hypercubes, the above four data movement operations need to be solved most efficiently.

Majority of the studies on providing fast data movement operations [5, 7, 8] assume *packet switching* (*store-and-forward*) method for communication among the processors in a hypercube. However, for a massive data movement operation, packet switching is not attractive because of the overhead (due to header and control information) associated with each packet and the large delays involved with the store-and-forward operation at the intermediary nodes in the path. In literature [5, 7, 8], various optimal routing methods are given for the above four data communication problems with the assumption: a processor can receive, process (i.e., disassemble and repack, if necessary), and send message packets on all of its communication links in the same time it would take for a *single* message packet. This capability, termed as *multiport* communication, is offered by none of the existing machines; and there are no satisfactory solutions offered in literature to achieve this. Hence, results obtained

with packet switching communication are relevant only when *one-port* communication (that is, a processor can receive and/or send a message at a time) is used. For each of the four data movement operations, the routing times with one port packet are proportionately larger than those obtained with multiport communication [5].

An alternative to the packet switching communication is the *circuit switching* communication. To send a message in the circuit switching communication, first a physical path between the source and the destination processors is established and, then, the complete message is transmitted at, essentially, the hardware clock speed. (Alternatively, one could use the *wormhole routing* approach of Dally and Seitz [2].) If the path traversed by a message goes through one or more intermediary processors, then a physical path is established within each of the intermediary processors. Hence, in this scheme, messages *pass* through the intermediary processors thereby eliminating the large delays involved with the store-and-forward approach. It is noteworthy that several messages could be passing through a processor, at the same time, without requiring that the processor be capable of processing messages on all of its links. (See figure 1.) Some of the recent hypercube machines such as Intel's iPSC/2 and iPSC/860 provide this type of communication mechanism [1, 3].

In this paper, we discuss optimal data movement methods in the context of how they can be realized in hypercubes with circuit switching communication capabilities. The main contribution of this paper is in developing a near optimal algorithm to realize the truly massive data movement operation, namely, the all-to-all personalized communication among processors, for hypercube machines with circuit switching capability. The basic argument put forth in this study is that, for regular and massive data movement operations, circuit switching mechanism can be used advantageously to provide very fast routing methods.

2 Preliminaries

In this section, we give some basic definitions, explain the communication model used, and discuss lower bounds on the routing times for each of the four data movement operations described in the previous section.

An n-dimensional, $n \geq 0$, hypercube or a boolean n-cube computer, $\mathcal{Q}_n$, consists of $N = 2^n$ processors, indexed, 0 through $N-1$, interconnected such that a communication link exists between a pair of processors if and only if the binary representation of their indices differ in exactly one bit. Hence, each processor is connected to exactly n other processors, one in each dimension of the cube. Due to its recursive construction, $\mathcal{Q}_n$ can be decomposed into 2^{n-i} i-cubes, $0 \leq i \leq n$, by removing links between processors adjacent in some $(n-i)$ dimensions [6].

The communication model. We assume that a processor can send data on a communication link and receive data on a communication link concurrently. Each communication link allows full duplex operation. If paths for any two or more messages do not use the same communication link in the same direction (i.e., *edge-disjoint*), then all such message transmissions can occur simultaneously, without any loss of time. A message can pass through a processor, if path (hardware support) is established to transfer data from from the "in" link (through which the message enters the processor) to the "out" link (through which the message leaves the processor),

without disturbing the other activities of the processor. Thus, multiple messages can pass through a processor, if paths for each of these messages are already established through the processor. A processor can establish a path for, at most, one message at a time. Even though the hypercube computer might operate in asynchronous mode, communication between any two processors, once a physical path is established, is synchronous, and the data movement is pipelined.

Messages consist of k elements. Typically an element of data is the number of bits transmitted in one direction of a communication link at a time. The time to transfer an element of data over a communication link between adjacent processors is taken as one time unit. To establish a path of length i, $1 \leq i \leq n$, it takes $(\xi + i\tau)$ time units, where ξ and τ indicate, respectively, the constant start up overhead and the time to set the path (of length 1) between two adjacent processors. To send a message of k elements between processors at a distance i, $1 \leq i \leq n$, it takes $(\xi + i\tau + i + k - 1)$ time units.

We note that these assumptions hold for Intel's iPSC/2 and iPSC/860 hypercube computers [1, 3]. The above formula for calculating delays to send messages is valid for some of the commercially available hypercube machines such as iPSC/860 [1, 4].

The lower bound arguments. Let us define a *unit transfer* as the transfer of an element of data between adjacent processors. Thus, if a message of k elements is to be moved between processor at a distance i, then the number of unit transfers specified by this operation is ik. The bandwidth specified by an operation is the the number of unit transfers required to complete the operation. For example, the number of unit transfers in the one-to-all personalized communication is $\sum_{i=1}^{n} ki \begin{pmatrix} n \\ i \end{pmatrix} = nkN/2$. In the all-to-all personalized communication, which is the N-body version of the one-to-all personalized communication, the number of unit transfers is $nkN^2/2$. The available bandwidth is the unit transfers that can be done in one time unit, which, for Q_n, is nN. One lower bound on the time to complete an operation is given by the ratio of bandwidth specified by an operation to the maximum available bandwidth (the bandwidth argument).

The minimum time to complete a data movement operation is also bounded by the time to move data into or out of a processor (the send/receive argument). These two lower bounds for each of the four communication operations are given in table 1.

For the data movement operations other than the all-to-all personalized communication, the lower bound specified by the send/receive argument is, at least, n times that specified by the bandwidth argument. This indicates that the routing delays for these operations with circuit switching communication are similar to those with one port packet switching communication.

For the all-to-all personalized communication, the lower bound time suggested by the send/receive argument is within a factor of 2 of that suggested by the bandwidth argument. Hence, for this operation, a routing algorithm optimal with respect to lower bound due to send and receive limitations is also close to optimal with respect to the lower bound due to bandwidth ratio. Furthermore, any routing algorithm with single port communication will have routing delay at least $\Omega(nkN)$, since only N links are used at any given time [5].

In the next section, we present an algorithm for the all-to-all personalized communication with routing delay, $O(kN)$, comparable to the delays of the best known algorithms based on multiport packet switching communication [5, 7, 8]. Furthermore, this routing algorithm is suitable for implementation on the existing hypercube machines such as Intel's iPSC/860.

3 All-to-all personalized communication

There are a total of $N(N-1)$ distinct messages to be moved among the processors to complete this operation. The simplistic approach of sending data along a Hamiltonian cycle of $\mathcal{Q}_n$, which works for all-to-all broadcasting and one-to-all personalized communication, is slow because, now, the amount of data *passing* through a processor is $O(nN^2)$ and the bandwidth used is $2N$. When messages are sent by the shortest paths, and the amount of data passing through a processor is $O(nkN)$. Hence, to realize this operation efficiently, messages should be sent by the shortest paths and all the available bandwidth should be used.

The basic idea of the algorithm is to show that the all-to-all personalized data movement operation is composed of several simple permutation based data movements. By providing optimal schemes to realize these permutations, it is possible complete the required data movement operation in near optimal time.

3.1 A basic data movement operation

The permutation useful here is the vector reversal permutation, in which, for $\mathcal{Q}_n$, processor x sends data to the processor $N-1-x$. In the vector reversal operation for an i-(sub)cube of $\mathcal{Q}_n$, each processor in the subcube sends data to the processor at distance i in the subcube.

A simple algorithm to realize the vector reversal permutation in $\mathcal{Q}_n$ is as follows. The path for message transmission between any pair of source and destination processors is set up through $(n-1)$ intermediary processors such that index of the ith intermediary processor, $1 \leq i \leq (n-1)$, matches that of the destination processor in the i least significant bits. For example, the path set up between processors 3 (in binary form, 0011) and 12 (1100) in $\mathcal{Q}_4$ goes through the intermediary processors 2 (0010), 0 (0000), and 4 (0100). This can be generalized to realize the vector reversal operation in the subcubes of a given hypercube (obtained by discarding links in some dimensions), by omitting the dimensions that do not constitute the subcube in establishing the paths. This is the *e-cube* routing algorithm [2].

Thus, if a message is to be routed from processor x to processor y, with the indices differing in bits $b_1, \ldots, b_i$, $0 \leq b_1 < \cdots < b_i < n$, $1 \leq i \leq n$, then it is routed through the set of intermediary processors $p_1 = x \oplus 2^{b_1}, p_2 = p_1 \oplus 2^{b_2}, \ldots, p_{i-1} = p_{i-2} \oplus 2^{b_{i-1}}$ such that $y = p_{i-1} \oplus 2^{b_i}$.

It can easily be seen that this algorithm provides edge-disjoint paths for all the messages to be moved in a vector reversal operation. Hence, for $\mathcal{Q}_n$, the vector reversal permutation can be completed in $(\xi + n\tau + n + k - 1)$ time, the same time it takes to move one message. An example showing the realization of the vector reversal permutation in $\mathcal{Q}_3$ is given in figure 2.

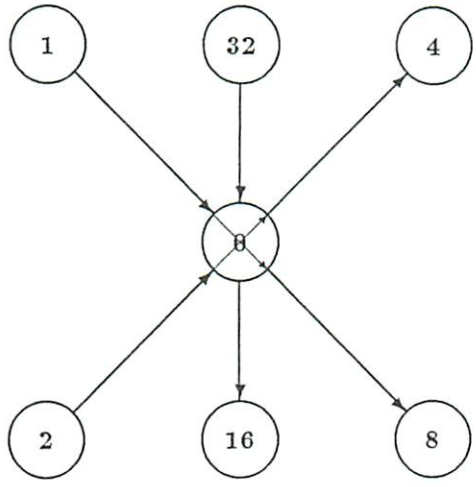


Figure 1: An illustration of multiple message transmissions through processor 0, while it is sending data to processor 16 and receiving data from processor 32, in a $\mathcal{Q}_6$ with circuit switched communication.

Operation	Lower bounds	
	Send/receive argument	Bandwidth argument
One-to-all broadcast	k	$\simeq k/n$
One-to-all personalized comm.	$k(N-1)$	$\simeq k/2$
All-to-all broadcast	$k(N-1)$	$k(N-1)/n$
All-to-all personalized comm.	$k(N-1)$	$kN/2$

Table 1: The lower bounds for the four data movement operations.

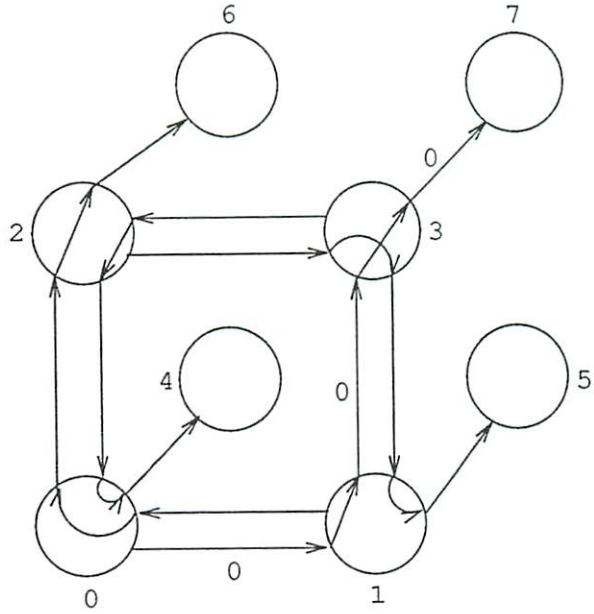


Figure 2: The set up of paths to realize the vector reversal operation in $\mathcal{Q}_3$. Only four paths are shown for clarity. Edges in the path from 0 to 7 are labelled 0.

3.2 The routing algorithm

The basic idea for the routing algorithm follows from the combinatorial identity (1).

$$N - 1 = \sum_{i=0}^{n-1} \binom{n}{i} \tag{1}$$

Algorithm AAP. The algorithm consists of n phases. Phase i, $0 \leq i < n$, consists of $\binom{n}{i}$ steps corresponding to the number of ways to decompose $\mathcal{Q}_n$ into 2^i $\mathcal{Q}_{n-i}$'s by removing some i dimension links. In each step, the vector reversal operation in the $(n-i)$-cubes as given by the corresponding decomposition of $\mathcal{Q}_n$ is performed. ∎

The steps within a phase of the algorithm are lexicographically ordered on the dimensions constituting the sub-cubes. Let $n_i = \binom{n}{i}$ and step (i, j) indicate step j of phase i, where $0 \leq i < n$ and $0 \leq j < n_i$. The all-to-all personalized communication on $\mathcal{Q}_3$ is realized using the sequence of the steps: (0,0), (1,0), (1,1), (1,2), (2,0), (2,1), and (2,2). See figure 3.

With the completion of each vector reversal permutation, the number of processors to which any given processor sent data increases by one. Hence, it is sufficient to do the vector reversal operations as specified by the combinatorial identity (1), to realize the all-to-all personalized communication.

Noting that a vector reversal operation in an i-cube takes $(\xi + i\tau + i + k - 1)$ time units, we derive the total time (T_{aap}) required to complete the all-to-all personalized communication as follows.

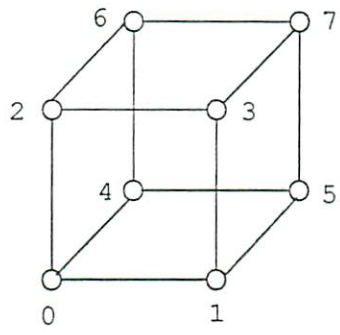
$$\begin{aligned} T_{aap} &= \sum_{i=0}^{n-1} (\xi + (n-i)\tau + (n-i) + k - 1) \binom{n}{i} \\ &= k(N-1) + (\xi - 1)(N-1) + \frac{(1+\tau)}{2} nN \end{aligned}$$

The lower bound for the all-to-all personalized communication operation is $T_{min} = k(N-1)$. (Section 2.) The ratio of T_{aap} and T_{min}, η, is as given below.

$$\begin{aligned} \eta &= T_{aap}/T_{min} \\ &= 1 + \frac{\xi - 1}{k} + \frac{(1+\tau)n}{2k} \frac{N}{N-1} \\ &= 1 + \frac{\frac{1+\tau}{2} n + \xi - 1}{k} + \frac{\frac{1+\tau}{2} n}{k(N-1)} \end{aligned}$$

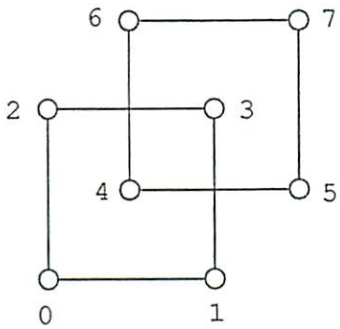
It is seen that, for large k, η approaches 1. Compared to the best routing delays given by the previous algorithms (using multiport packet communication and ideal packet sizes) [5, 7, 8], Algorithm AAP is optimal within a factor of 2.

For hypercube machines with bit-serial communication links between processors (such as the iPSC/2 [3]), k is the number of bits in a message. In such cases, even for messages of a few (32-bit) words of data, this routing method can be very close to optimal, provided ξ and τ are small. Even when these overheads are high, the routing delays are close to the minimum achievable with only moderate message sizes. For example, for iPSC/860 [1], the time to transmit $k \leq 100$ bytes between two processors over a distance i is measured to be $(65 + 0.425k + 10i)$ in μsec. This is unaffected by the other message transmissions in the machine. It should be

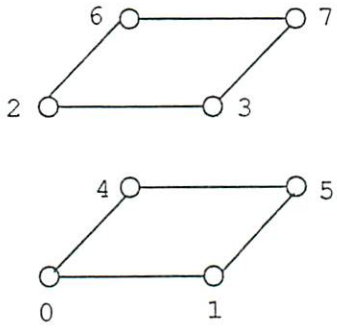


Step 0

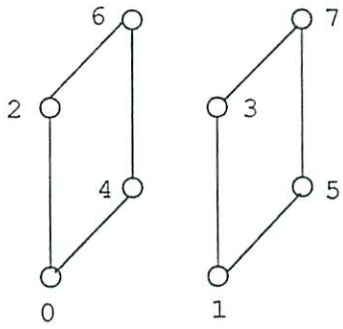
Phase 0: Vector reversal operation in the 3-cube.



Step 0

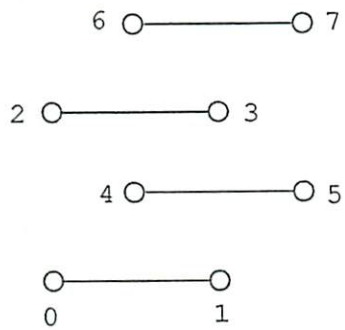


Step 1

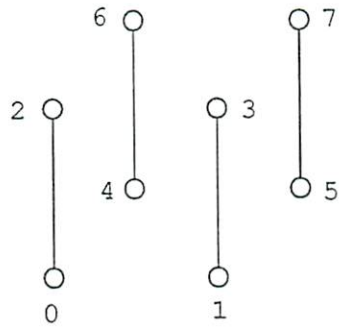


Step 2

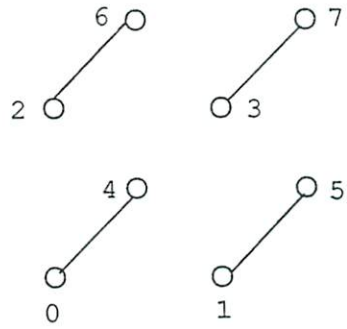
Phase 1: Vector reversal operation in 2-cubes in each step.



Step 0



Step 1



Step 2

Phase 2: Vector reversal operation in 1-cubes in each step.

Figure 3: The sequence of vector reversal operations to perform the all-to-all personalized communication in Q_3. For each step, only the links used are shown for clarity.

noted that the above formula includes the overheads due to the operating system of the machine also. Comparing this with our formula for the delay, we see that the time unit is 0.425 μsec., $\xi - 1 = 65/0.425 = 152.94$, and $1 + \tau = 10/0.425 = 23.53$. To realize the all-to-all personalized communication, on a 128-processor iPSC/860 for 100 byte messages, i.e., $k = 100$ and $n = 7$, the routing delay incurred is within a factor of 4 ($\eta = 3.37$) of the lower bound on moving data out of a processor.

3.3 A variation of Algorithm AAP.

Actually, one can fine tune Algorithm AAP to reduce the effect of the overheads, as suggested by Ho [4]. For this, we assume that a path can be set up through a processor without affecting any of its activities as long as the "out link" is free. This feature is supported by, for example, the iPSC/2. We assume that messages are long enough to completely overlap the message transmission time of a step with the set up time of paths for another step. (For this, $k > \xi + (n-1)\tau$. If this is not the case, then still overhead is reduced, proportionately, but not eliminated.) The modification to Algorithm AAP is as follows.

For $1 < i < n$, phase i has a *dual*, namely, the phase $(n - i)$. The number of steps in phase i are the same as those in phase $(n - i)$.

First, phase 0, the vector reversal operation in $\mathcal{Q}_n$, is done as before. For $1 < i \leq \lfloor (n-1)/2 \rfloor$, the steps in phases i and $(n - i)$ are interleaved: for $0 \leq j < n_i$, step (i, j) is followed up with step $(n - i, n_i - 1 - j)$, which, in turn, is followed up with step $(i, j + 1)$, and so on. Step $(n - i, n_i - 1)$ is followed up with step $(i + 1, 0)$, etc. For n even, phase $n/2$ is its own dual phase, in which case the steps are interleaved as before, for j up to $n_i/2 - 1$ only. Thus all the steps are of Algorithm AAP are done exactly once.

Due to the interleaving of the steps in dual phases, it is possible to set up paths for the next vector reversal operation in advance. The links not used in step (i, j) are exactly the ones used in step $(n - i, n_i - 1 - j)$. Note that the transition from step $(n - i, n_i - 1)$ to step $(i + 1, 0)$ does not pose any problem in setting up paths in advance.

For the all-to-all personalized communication on $\mathcal{Q}_3$, the sequence of steps are (0,0), (1,0), (2,2), (1,1), (2,1), (1,2), and (2,0).

Noting that set up overheads are not overlapped with message transmissions for the steps (0,0) and (1,0) only, the routing time ($T_{aap'}$) is calculated as follows.

$$\begin{aligned} T_{aap'} &= \xi + n\tau + \xi + (n-1)\tau + \sum_{i=0}^{n-1} (n - i + k - 1) \begin{pmatrix} n \\ i \end{pmatrix} \\ &= (k-1)(N-1) + nN/2 + 2(\xi + n\tau) - \tau. \end{aligned}$$

$$\text{And} \quad \eta' = T_{aap'}/T_{min} = 1 + \frac{n-2}{2k} + \frac{n/2 + 2(\xi + n\tau) - \tau}{k(N-1)}$$

4 Conclusions

In this paper, we provided a simple routing algorithm for the all-to-all personalized communication. The salient features of the algorithm are the use of permutation based communication operations as substeps and circuit

switched mechanism for possible practical application. We also showed that this algorithm is near optimal even compared to the minimum time achievable with multiport communication.

For regular data movement operations with extensive bandwidth requirements, circuit switching communication provides the basis for viable routing algorithms.

Acknowledgment

We would like to thank Dr. C.-T. Ho for the discussions on an earlier version of this paper. He suggested the modifications to the algorithm.

References

[1] S. Bokhari. Communication overhead on the Intel iPSC-860 hypercube. Technical Report 10, ICASE, NASA Langley Research Center, May 1990.

[2] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Computers*, C-36(5):547–553, 1987.

[3] A. L. DeCegama. *The Technology of Parallel Processing, Volume 1 (Parallel Processing Architectures and VLSI Hardware)*. Prentice-Hall Inc., 1989.

[4] C.-T. Ho, 1990. Personal communication.

[5] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. on Computers*, c-38(9):1249–1268, September 1989.

[6] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Trans. on Computers*, c-37(7):867–872, July 1988.

[7] Y. Saad and M. H. Schultz. Data communication in hypercubes. *J. of Parallel and Distributed Computing*, 6:115–135, 1989.

[8] Q. F. Stout and B. Wager. Passing messages in link-bound hypercubes. In M. T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 251–257. SIAM, 1987. Proceedings of the second conference on hypercube multiprocessors, 1986.