This paper appears in *J. of Parallel and Distributed Computing* **10** (1990), pp. 167–181.

Intensive Hypercube Communication:
Prearranged Communication in Link-Bound Machines [1] [2]

Quentin F. Stout and Bruce Wagar

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2121 USA

September 7, 1988

[1] A preliminary condensed version of this paper appeared as "Passing messages in link-bound hypercubes", *Hypercube Mutiprocessors 1987*, M. Heath, ed., pp. 251-257.

[2] This research was partially supported by National Science Foundation grant DCR-8507851 and an Incentives for Excellence award from Digital Equipment Corp.

Abstract

Hypercube algorithms are developed for a variety of communication-intensive tasks such as transposing a matrix, histogramming, one node sending a (long) message to another, broadcasting a message from one node to all others, each node broadcasting a message to all others, and nodes exchanging messages via a fixed permutation. The algorithm for exchanging via a fixed permutation can be viewed as a deterministic analogue of Valiant's randomized routing. The algorithms are for link-bound hypercubes in which local processing time is ignored, communication time predominates, message headers are not needed because all nodes know the task being performed, and all nodes can use all communication links simultaneously. Through systematic use of techniques such as pipelining, batching, variable packet sizes, symmetrizing, and completing, for all problems algorithms are obtained which achieve a time with an optimal highest-order term.

1 Introduction

This paper gives efficient hypercube algorithms for a variety of communication-intensive tasks. The emphasis is on problems where the communication pattern (i.e., which nodes are sending information and where it is going) is known in advance by all processors, and all messages are of the same size. This situation is common in SIMD (Single Instruction Multiple Data) or SCMD (Single Code Multiple Data) applications such as matrix multiplication or inversion, some database operations, solving PDEs on a regular grid, image manipulation, and histogramming. By systematic use of a few basic techniques, algorithms are developed which are significantly faster than the simplest or most common ones, and faster than those published previously. Our algorithms show that these techniques are quite powerful, and also show that it is advantageous to be able to use all communication links simultaneously when communication becomes a bottleneck.

1.1 Definition of a Hypercube

A *d-dimensional hypercube* computer is a distributed-memory multiprocessor consisting of 2^d separate processing elements (or *nodes*), linked together in a d-dimensional binary cube network (see Figure 1). Each node is given a unique d-bit identification number (henceforth referred to as the *node i.d.*) and two nodes linked if and only if their node i.d.'s differ in exactly one bit position. Two nodes in a hypercube are said to be *adjacent* or *neighboring* if they share a link.

Hypercube networks have some useful properties such as a logarithmic communication diameter and a totally symmetric layout. Because the number of links per node grows only linearly with the dimension, reasonably-sized hypercubes (of say, dimension 10 or 12) can be practically built with current technology. NCUBE, Intel, Floating Point Systems, Ametek, and Thinking Machines have already introduced hypercube computers on the commercial market.

1.2 The Link-Bound Model

This paper uses a model of hypercubes in which communication time is assumed to predominate, and local processing time by the nodes can be ignored. We are interested in problems where extensive communication is required because very long messages are being sent, and where all nodes are available to participate in the task and know the communication task being performed. This latter point helps reduce the communication time by insuring that messages do not need to include header information such as message destination, message route, packet sequence number, etc. Further, we assume that each node can utilize all of its communication links simultaneously, where the links between neighboring nodes can be used in both directions simultaneously. Thus, a node in a d-dimensional hypercube may be handling $2d$ messages simultaneously, receiving d and sending d. This property we call *link-bound*, as opposed to other possibilities such as processor-bound (in which each node can do only one operation at a time) or DMA-bound (in which there is an upper bound on the number of messages which can go in and out of a node at one time). While no hypercube can currently use all of its communication links simultaneously, several manufacturers are trying to attain such ability. The NCUBE machines apparently come the closest [4], and the FPS T-Series machines have nodes capable of 4 bi-directional communications at the same time [3]. The link-bound model has been studied in [1, 5, 6, 7, 8, 9, 10, 11, 12, 13].

1.3 Message-Passing Problems

Throughout the paper, various hypercube communication patterns will be looked at. Each will involve sending messages between some known combination of the nodes. These messages will be the same length m, but their contents may vary between different pairs of communicating nodes. It

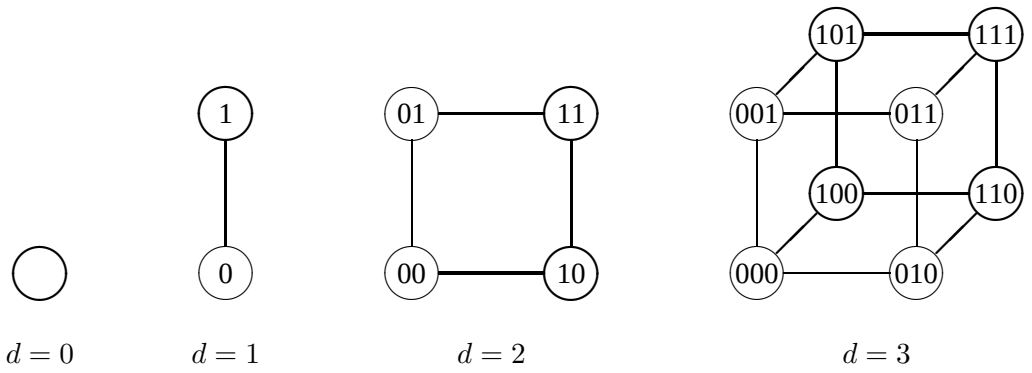

Figure 1: Some Hypercubes for Small d

will be assumed that messages may be broken down into packets at any time, while existing packets may be either recombined or broken down still further, in order to facilitate sending, so long as the ultimate destination eventually receives the entire original message. Another key assumption is that a node receiving a packet must finish receiving it before any of its contents can be utilized. This is sometimes called the *store-and-forward* or *packet-switched* model, as opposed to a circuit-switched model. All existing hypercubes use store-and-forward.

We assume that it takes $\tau m + \beta$ time for a node to send a packet of length m to a neighbor, where

- τ represents the transfer rate and
- β the time for start-up and termination.

In real systems it is generally the case that $\beta \gg \tau$, and hence it is important to include the effect of start-up costs. Also note that the total internal I/O bandwidth of the computer is $2^d d\tau$. In general, τ and β are treated as constants and the algorithms are analyzed as functions of d and m. As in [5, 6, 7, 8, 11], we ignore the effects of rounding or truncating. Furthermore, because special cases arise for various relative values of the parameters, the exact formulas will be given only for m sufficiently large. Because we are most interested in processing long messages, the term containing the highest power of m will be called the *highest-order* term.

Some of the problems studied in this paper, such as broadcasting a message from one node to all others, have been previously considered by [5, 9, 11]. These papers use a similar model to the link-bound model, but give slightly slower algorithms. In particular, [11] considered a model where communication between nodes can only take place one direction at a time, so where this made a difference, the times of their algorithms have been divided by two so that they can be fairly compared with ours. For most of the problems, several algorithms are developed, the last being the fastest (and most sophisticated). In all problems, simple arguments show that the best versions of our algorithms have optimal high-order terms. In some cases we believe that our algorithms are completely optimal, but are unable to prove this because our arguments can bound terms involving τ, m, and d, or terms involving β and d, but not sums of terms of different types.

1.4 Notation

Several of the patterns considered here are oriented around one node. For such patterns, it will be assumed without loss of generality that node 0 is the special node. The set of all nodes (or their corresponding i.d.'s, depending on context) which are distance k from node 0 will be denoted by C_k. Note that because of the symmetry of the hypercube, any algorithm written for node 0 can be

converted to an identical algorithm for node $n \in \{1, 2, \ldots, 2^d - 1\}$ by exclusive-ORing all node i.d.'s referenced in the node 0 algorithm with n.

Some of the algorithms utilize somewhat subtle message routing schemes, for which the following notation will be useful. Let

$$\oplus_k(n)$$

denote the node i.d. formed by taking the bit-wise exclusive-OR of n and 2^k. I.e., it is just n with the kth bit flipped. For example, if $d = 5$, then $\oplus_1(01001) = 01011$ and $\oplus_3(01001) = 00001$. Observe then that the neighbors of node n are nodes $\oplus_0(n), \oplus_1(n), \ldots, \oplus_{d-1}(n)$. Let

$$\overset{i}{\hookleftarrow}(n)$$

denote the left circular-shift (rotation) of the d-bit representation of n by i bits, where $i \in \{0, 1, \ldots, d-1\}$ and $n \in \{0, 1, \ldots, 2^d - 1\}$. For example, if $d = 5$, then $\overset{2}{\hookleftarrow}(01001) = 00101$. For each $n \in \{1, 2, \ldots, 2^d - 1\}$, let W_n denote the set of all bit positions in the binary representation of n which are 1. That is, W_n is the unique set of integers such that

$$n = \sum_{w \in W_n} 2^w.$$

Likewise, let $Z_n = \{0, 1, \ldots, d-1\} \setminus W_n$ denote the corresponding set of 0-bit positions of n. Define $\lceil_n \colon Z_n \to W_n$ as follows:

$$\lceil_n(z) \quad = \quad \begin{cases} \min W_n & \text{if } z > \max W_n \\ \min\{w \in W_n : w > z\} & \text{otherwise} \end{cases}$$

for each $z \in Z_n$. I.e., $\lceil_n(z)$ is the bit position of the first 1 (in left circular order) after z in n. For example, when $d = 5$, then $\lceil_{01001}(1) = 3$ and $\lceil_{01001}(4) = 0$.

2 Sending the Same Message From One Node to All Others

One of the fundamental hypercube communication patterns is *broadcasting*, in which one node has to send the same message to all the other nodes in the hypercube. This problem has been previously examined in [5, 11], but our algorithms are somewhat faster. It is interesting to note that the NCUBE machine has special hardware instructions to allow a node to simultaneously broadcast to all of its neighbors, though the current operating system does not make use of these instructions.

Assume node 0 is to do the broadcasting. The most common and straightforward way to broadcast in a hypercube is recursive doubling.

Algorithm: BROADCAST 1 (Recursive Doubling)

There are d stages, numbered $0, 1, \ldots, d-1$. During stage k, nodes $0, 1, \ldots, 2^k - 1$ send the message to nodes $\oplus_k(0), \oplus_k(1), \ldots, \oplus_k(2^k - 1)$, respectively (and concurrently).

Analysis of BROADCAST 1

Each of the stages takes $\tau m + \beta$ time, so the whole algorithm requires time

$$d(\tau m + \beta) \quad = \quad \boxed{d\tau m + d\beta}\,.$$

BROADCAST 1 works on only one dimension at a time and thus fails to take advantage of the concurrent read/write channels of the link-bound hypercube. To alleviate this, it is necessary to *symmetrize* the algorithm. That is, break up the problem into d subproblems and run them simultaneously along separate dimensions.

Algorithm: BROADCAST 2

Symmetrize BROADCAST 1. I.e., break up the message into d packets $P_0, P_1, \ldots, P_{d-1}$, each of size m/d. During stage k, nodes $\overset{i}{\hookleftarrow}(0), \overset{i}{\hookleftarrow}(1), \ldots, \overset{i}{\hookleftarrow}(2^k-1)$ send P_i to nodes $\oplus_{(i+k) \bmod d}(\overset{i}{\hookleftarrow}(0))$, $\oplus_{(i+k) \bmod d}(\overset{i}{\hookleftarrow}(1)), \ldots, \oplus_{(i+k) \bmod d}(\overset{i}{\hookleftarrow}(2^k-1))$, respectively, for each i in $\{0, 1, \ldots, d-1\}$.

Notice that packet P_0 reaches each processor at the same stage as the single packet in BROADCAST 1, and in general at the end of stage k each packet has been broadcast to a k-dimensional subcube. The modular packet routing guarantees that no processor is trying to send two packets along the same link at the same time.

Analysis of BROADCAST 2

The same as for BROADCAST 1, only now each stage takes time $\tau(m/d) + \beta$ for a total time of

$$d\left(\tau\frac{m}{d} + \beta\right) \quad = \quad \boxed{\tau m + d\beta}.$$

Although much improved from the previous version, BROADCAST 2 still suffers from the fact that a large percentage of the links are idle most of the time. In fact, half of the links are *never* used (those which are directed towards 0). A better algorithm would be one in which each link is always busy sending data to a node which has not yet seen it. This is impossible to fully achieve, but it provides a goal to aim for. The next version attempts this in a *systolic* fashion, for the message travels across the hypercube in a "wave" going from one C_k to the next, with every node actively sending along all of its links when the "wave" hits it.

Algorithm: BROADCAST 3A

There are $d+1$ stages, numbered $0, 1, \ldots, d$. Break up the message into packets as in BROADCAST 2. During stage k, only the nodes in C_k actively send along their their d outgoing links, each node sending exactly one packet along each such link. Every node in C_k receives k distinct packets from C_{k-1} during stage $k-1$ and the remaining $d-k$ packets from C_{k+1} during stage $k+1$.

More specifically, during stage k, each node $n \in \mathsf{C}_k$ starts out with P_w for every $w \in W_n$. For each such w, it sends P_w to node $\oplus_w(n) \in \mathsf{C}_{k-1}$. In addition, for each $z \in Z_n$, it sends $P_{\lceil_n(z)}$ to node $\oplus_z(n) \in \mathsf{C}_{k+1}$. Figure 2 shows this process for $d = 3$. By induction, it is easy to verify that during stage $k-1$, a node $n \in \mathsf{C}_k$ receives P_w for every $w \in W_n$, receiving it from node $\oplus_v(n) \in \mathsf{C}_{k-1}$, where $v \in W_n$ is such that $\lceil_n(v) = w$. During stage $k+1$ node n receives P_i from $\oplus_i(n) \in \mathsf{C}_{k+1}$ for each $i \notin W_n$. Therefore, at the end of stage $k+1$, every node in C_k has received all of the packets.

Analysis of BROADCAST 3A

Similar to that of BROADCAST 2, only now there are $d+1$ stages and thus, a total time of

$$(d+1)\left(\tau\frac{m}{d} + \beta\right) \quad = \quad \boxed{\frac{d+1}{d}\tau m + (d+1)\beta}.$$

Although slower than the previous broadcast, 3A has the special property that only nodes in C_k are sending in the kth stage. This will be exploited later, but first notice that the only purpose of the last stage is to send $P_0, P_1, \ldots, P_{d-1}$ from node 2^d-1 to nodes $\oplus_0(2^d-1), \oplus_1(2^d-1), \ldots,$ $\oplus_{d-1}(2^d-1)$, respectively.

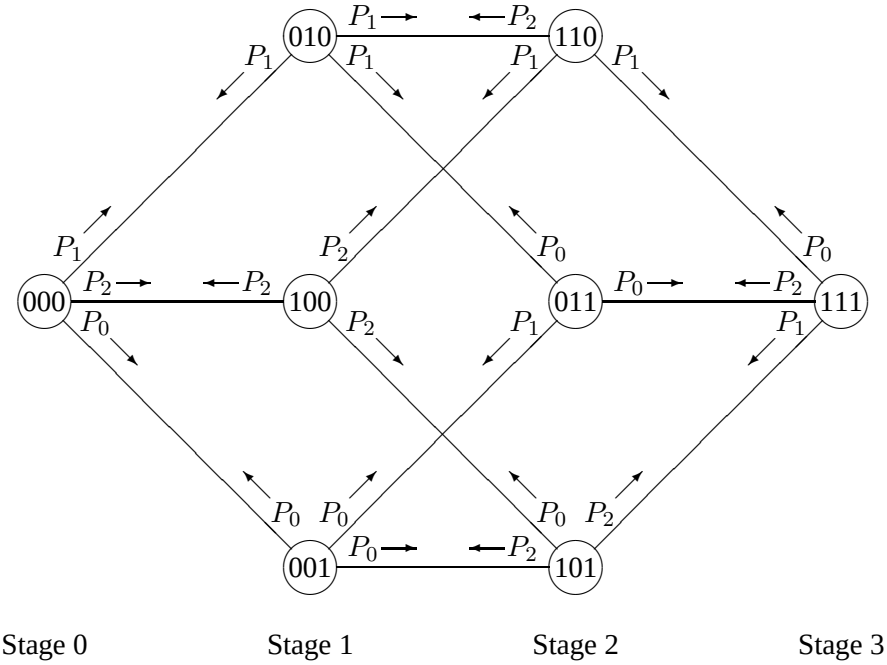

Figure 2: BROADCAST 3A for $d = 3$

Algorithm: BROADCAST 3B

Identical to BROADCAST 3A, but the last stage is eliminated by sending a second "wave" right after the first. During stage 1, node 0 relabels $P_0, P_1, P_2, \ldots, P_{d-1}$ as $Q_{d-1}, Q_0, Q_1, \ldots, Q_{d-2}$, respectively, and starts a second BROADCAST 3A, only this time using $Q_0, Q_1, \ldots, Q_{d-1}$ in place of $P_0, P_1, \ldots, P_{d-1}$, respectively. This second broadcast is run for only $d-1$ stages, after which each node in C_{d-1} will have received every packet (and then some).

Analysis of BROADCAST 3B

The one stage saving results in a final time of

$$d\left(\tau\frac{m}{d}+\beta\right) \quad = \quad \boxed{\tau m + d\beta},$$

the same as for BROADCAST 2.

BROADCAST 3 can be significantly sped up through the use of *pipelining*. Basically, pipelining consists of breaking up a problem into smaller pieces and sending these out as separate "waves", one after another, in order to keep more of the nodes busy at the same time. It is a useful tool for speeding up asymmetrical communication algorithms.

Algorithm: BROADCAST 4

Pipeline BROADCAST 3. That is, divide up the message into g groups, numbered $0, 1, \ldots, g-1$, each of length m/g. Execute $g-1$ separate BROADCAST 3A's, one for each of the first $g-1$ groups, followed by a BROADCAST 3B for the last group. These should be done concurrently, but staggered one stage apart so that no node is working on more than one at a time. To be more precise, there are $d+g-1$ stages, numbered $0, 1, \ldots, d+g-2$. During stage k, each node in C_j, $k-g+1 \leq j \leq k$, executes stage j of BROADCAST 3 for group $k-j$.

Analysis of BROADCAST 4

Each of the $d+g-1$ stages now takes $\tau(m/dg)+\beta$ time, so the whole algorithm requires time

$$(d+g-1)\left(\tau\frac{m}{dg}+\beta\right) \;=\; \frac{(d+g-1)\tau}{dg}m+(d+g-1)\beta\,.$$

By simple calculus, it can be shown that this time is minimized when

$$g \;=\; \begin{cases} 1 & d=1 \\ \sqrt{\dfrac{(d-1)\tau}{d\beta}}m^{1/2} & d\geq 2 \end{cases},$$

which produces a final time of

$$\boxed{\begin{cases} \tau m+\beta & d=1 \\ \dfrac{\tau}{d}m+2\sqrt{\dfrac{(d-1)\beta\tau}{d}}m^{1/2}+(d-1)\beta & d\geq 2 \end{cases}}.$$

For purposes of comparison, the best broadcast in [5] has a running time of

$$\frac{\tau}{d}m+2\sqrt{\beta\tau}m^{1/2}+d\beta.$$

One last observation. BROADCAST 4 is absolutely optimal if all sends have to use fixed-size packets. To see this, suppose the packet size is s. Then each node will have to receive at least m/s of these packets and, since only d packets can be sent out at a time from node 0, at least one such packet will have to wait $(m/ds-1)(\tau s+\beta)$ time for a free link before it can leave node 0. To reach node 2^d-1, this packet will then have to travel d links, which takes time $d(\tau s+\beta)$, for a total minimum time of

$$\left(d+\frac{m}{ds}-1\right)(\tau s+\beta)\,.$$

But this is just what BROADCAST 4 takes when you let $g=m/ds$.

3 Sending From One Node to the Opposite Corner Node

Another important communication pattern is when some node n sends to its *opposite corner (o.c.)* node 2^d-1-n. Obviously, this is similar to, and could be accomplished by, broadcasting. Although this might appear to waste time, the analysis of BROADCAST 4 shows that that algorithm already performs an optimal o.c. send, if one is limited to fixed-size packets. Which is not to say that variable-sized packets will help much, for at least $(\tau/d)m+\beta$ time is needed just to get all of the data out of node n. More important, though, is to cut down all of the unnecessary communication. Assume that node 0 wants to send to node 2^d-1.

Algorithm: O.C. SEND 1

There are d stages, numbered $0,1,\ldots,d-1$. During stage k, node 2^k-1 sends the message to node $\oplus_k(2^k-1)$.

Analysis of O.C. SEND 1

Identical to BROADCAST 1:

$$d(\tau m + \beta) \quad = \quad \boxed{d\tau m + d\beta}\,.$$

Algorithm: O.C. SEND 2

Symmetrize O.C. SEND 1. I.e., break up the data into d packets $P_0, P_1, \ldots, P_{d-1}$, each of size m/d. During stage k, node $\stackrel{i}{\hookleftarrow}(2^k - 1)$ sends P_i to node $\oplus_{(i+k) \bmod d}(\stackrel{i}{\hookleftarrow}(2^k - 1))$ for each $i \in \{0, 1, \ldots, d-1\}$.

Analysis of O.C. SEND 2

Identical to BROADCAST 2:

$$d\left(\tau\frac{m}{d} + \beta\right) \quad = \quad \boxed{\tau m + d\beta}\,.$$

These algorithms reduce the total communication needed in their broadcast equivalents considerably. Unfortunately, they don't save any time. In order to do that, it is necessary to speed up the intermediate stages.

Algorithm: O.C. SEND 3

There are d stages, numbered $0, 1, \ldots, d-1$. The packets will vary in size depending on the stage. During stage k, only the nodes of C_k will be sending, while only nodes in C_{k+1} will be receiving. At the start of stage k, the data starts out evenly distributed among each of the $\binom{d}{k}$ nodes in C_k. Each such node then breaks up its data into $d-k$ equal-sized packets and sends a different one out along each of its $d-k$ links to C_{k+1}, at which point the message will be evenly distributed among C_{k+1}'s nodes.

Analysis of O.C. SEND 3

Stage k takes time

$$\tau\frac{m}{\binom{d}{k}(d-k)} + \beta \quad = \quad \frac{\tau}{\binom{d-1}{k}d}m + \beta,$$

so the whole algorithm requires

$$\begin{aligned}\sum_{k=0}^{d-1}\left[\frac{\tau}{\binom{d-1}{k}d}m + \beta\right] \quad &= \quad \left[\sum_{k=0}^{d-1}\frac{1}{\binom{d-1}{k}}\right]\frac{\tau}{d}m + d\beta \\ &= \quad \boxed{\frac{K_{d-1}\tau}{d}m + d\beta}\,,\end{aligned}$$

where

$$K_d \quad = \quad \sum_{k=0}^{d}\frac{1}{\binom{d}{k}}\,.$$

To help understand K_d, note that it's first six values are $1, 2, 2\frac{1}{2}, 2\frac{2}{3}, 2\frac{2}{3}$, and $2\frac{3}{5}$. For $d \geq 5$,

$$2 + \frac{2}{d} \;\leq\; K_d \;\leq\; 2 + \frac{2}{d} + \frac{4}{d(d-1)} + \frac{6(d-5)}{d(d-1)(d-2)} \;<\; 2 + \frac{2}{d} + \frac{10}{d(d-1)}\,,$$

and hence,

$$K_d \;\approx\; 2 + \frac{2}{d}\,.$$

The upshot of all this is that O.C. SEND 3 is faster than BROADCAST 3 by a factor of about $d/2$. Like BROADCAST 3, O.C. SEND 3 also lends itself well to pipelining.

Algorithm: O.C. SEND 4

Pipeline O.C. SEND 3. I.e., break up the message into g groups of length m/g and execute g separate O.C. SEND 3's, one for each group. These should be done concurrently, but staggered one stage apart. Because O.C. SEND 3's stages have varying lengths, there will be no synchronization between the various stages of O.C. SEND 3 being worked on for each of the groups. Since O.C. SEND 3's first stage, which takes time $\tau(m/dg) + \beta$, is at least as long as any other of its stages, it sets the rate at which the groups can be started. Each node will be able to complete sending the previous group by the time it has finished receiving the next group.

Analysis of O.C. SEND 4

Due to the fact that there is no chance of conflict, the time needed for this algorithm is just the time needed to start the first $g-1$ O.C. SEND 3's, as well as all of the time needed for the last one. This works out to be

$$(g-1)\left(\tau\frac{m}{dg} + \beta\right) + \frac{K_{d-1}\tau}{d}\frac{m}{g} + d\beta \;=\; \frac{g+K_{d-1}-1}{dg}\tau m + (g+d-1)\beta\,,$$

which is minimized when

$$g \;=\; \begin{cases} 1 & d = 1 \\ \sqrt{\dfrac{(K_{d-1}-1)\tau}{d\beta}}\,m^{1/2} & d \geq 2 \end{cases}.$$

This produces a final time of

$$\boxed{\begin{cases} \tau m + \beta & d = 1 \\ \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(K_{d-1}-1)\beta\tau}{d}}\,m^{1/2} + (d-1)\beta & d \geq 2 \end{cases}}.$$

Notice that this is identical to the time for BROADCAST 4, except that the coefficient of the $m^{1/2}$ term has been reduced by a factor of approximately $\sqrt{d-1}$.

A final observation about the O.C. SEND algorithms: they only use links in one direction (i.e., towards node $2^d - 1$). Consequently, another o.c. send could be done from node $2^d - 1$ to node 0 concurrently (that is, an opposite corner *exchange*) since they each would use different links.

4 Sending Messages Between Two Arbitrary Nodes

One of the more interesting questions that can be asked is, given the full use of the link-bound hypercube, what is the fastest possible time for one node to send to an arbitrary node distance n away ($1 \leq n \leq d$)? For simplicity, assume node 0 is sending to node $2^n - 1$.

Algorithm: ARBITRARY SEND 1

Use O.C. SEND 4 on the n-dimensional subcube containing nodes $0, 1, \ldots, 2^n - 1$.

Analysis of ARBITRARY SEND 1

$$\boxed{\begin{cases} \tau m + \beta & n = 1 \\ \dfrac{\tau}{n} m + 2\sqrt{\dfrac{(K_{n-1}-1)\beta\tau}{n}} m^{1/2} + (n-1)\beta & n \geq 2 \end{cases}}.$$

Like previous algorithms, this approach fails to make use of the tremendous bandwidth available and is slower than broadcasting for all $n < d$.

Algorithm: ARBITRARY SEND 2

Use BROADCAST 4, deleting the last $d-2-n$ stages if $n < d-2$. These stages were only needed to get the message to nodes farther than distance n from node 0.

Analysis of ARBITRARY SEND 2

There are

$$\begin{cases} n+g-1 & 0 \leq n \leq d-2 \\ d+g-1 & d-2 \leq n \leq d \end{cases}$$

stages, which produce minimum times when

$$g = \begin{cases} 1 & d = 1 \\ \sqrt{\dfrac{(n+1)\tau}{d\beta}} m^{1/2} & 1 \leq n \leq d-2 \\ \sqrt{\dfrac{(d-1)\tau}{d\beta}} m^{1/2} & 0 \leq d-2 \leq n \leq d \end{cases}.$$

The corresponding times are then

$$\boxed{\begin{cases} \tau m + \beta & d = 1 \\ \dfrac{\tau}{d} m + 2\sqrt{\dfrac{(n+1)\beta\tau}{d}} m^{1/2} + (n+1)\beta & 1 \leq n \leq d-2 \\ \dfrac{\tau}{d} m + 2\sqrt{\dfrac{(d-1)\beta\tau}{d}} m^{1/2} + (d-1)\beta & 0 \leq d-2 \leq n \leq d \end{cases}}.$$

Closer inspection of BROADCAST 3 reveals that yet another stage can be saved when $d/2 \leq n \leq d-2$ and a fraction of a stage saved when $1 \leq n < d/2$, but these only reduce the $m^{1/2}$ and β coefficients by negligible amounts. What's needed is a way to combine the link utilization of BROADCAST 4 with the efficiency of O.C. SEND 4. With this in mind, it is necessary to look at a new communication pattern, the *extended* o.c. (x.o.c.) send. It is similar to a regular o.c. send, except that the sending and receiving nodes are connected to opposite corners of an n-dimensional subcube S, and all communication must go through S.

Algorithm: X.O.C. SEND 1

Identical to O.C. SEND 3, except two stages, numbered -1 and n, are added to get the items into and out of S.

Analysis of X.O.C. SEND 1

The first and last stages each take time $\tau m + \beta$, so the whole algorithm requires

$$2(\tau m + \beta) + \frac{K_{n-1}\tau}{n} m + n\beta \quad = \quad \boxed{\frac{(2n + K_{n-1})\tau}{n} m + (n+2)\beta}\,.$$

Algorithm: X.O.C. SEND 2

Pipeline X.O.C. SEND 1 as in O.C. SEND 4. There are still g groups of m/g items apiece, only in this case the $\tau(m/g) + \beta$ time needed to get a group out of the sending node sets the pace for the rest of the stages.

Analysis of X.O.C. SEND 2

Similar to O.C. SEND 2. The total time needed is

$$(g-1)\left(\tau\frac{m}{g} + \beta\right) + \frac{(2n + K_{n-1})\tau}{n}\frac{m}{g} + (n+2)\beta$$

$$= \quad \frac{(gn + n + K_{n-1})\tau}{gn} m + (g+n+1)\beta\,,$$

which is minimized when

$$g \quad = \quad \sqrt{\frac{(n+K_{n-1})\tau}{n\beta}}\, m^{1/2}$$

with a resulting time of

$$\boxed{\tau m + 2\sqrt{\frac{(n+K_{n-1})\beta\tau}{n}}\, m^{1/2} + (n+1)\beta}\,.$$

Algorithm: ARBITRARY SEND 3

Break up the data into $d-n+1$ packets, $d-n$ of which contain m/d apiece while the other has the remaining nm/d items. Send the larger packet via an O.C. SEND 4 through the subcube T containing nodes 0 and 2^n-1 as opposite corners. Send each of the other $d-n$ packets via an X.O.C. SEND 2 through a different one of the $d-n$ n-dimensional subcubes which both run parallel to T and are distance 1 from T (i.e., the subcubes containing nodes 2^{n+1} through $2^{n+1}+2^n-1$, 2^{n+2} through $2^{n+2}+2^n-1$,..., 2^{d-1} through $2^{d-1}+2^n-1$).

Analysis of ARBITRARY SEND 3

The $d-n+1$ separate sends use different links, so they can all be done concurrently with no conflicts. Each of the X.O.C. SEND 2's of length m/d takes time

$$\frac{\tau}{d}m + 2\sqrt{\frac{(n+K_{n-1})\beta\tau}{dn}}m^{1/2} + (n+1)\beta$$

whereas the O.C. SEND 4 of length nm/d requires

$$\begin{cases} \dfrac{\tau}{d}m + \beta & n = 1 \\ \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(K_{n-1}-1)\beta\tau}{d}}m^{1/2} + (n-1)\beta & 2 \le n \le d \end{cases}.$$

Hence, the X.O.C. SEND 2's will be slower than the O.C. SEND 4 whenever

$$\frac{n+K_{n-1}}{n} \quad \ge \quad K_{n-1}-1\,,$$

which is true by inspection for $1 \le n \le 4$ and is false for bigger n since

$$2 + \frac{2}{n} \quad < \quad K_n$$

holds for all $n \ge 4$. What this all means is that the actual time for ARBITRARY SEND works out to

$$\boxed{\begin{cases} \tau m + \beta & d = n = 1 \\ \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(n+K_{n-1})\beta\tau}{dn}}m^{1/2} + (n+1)\beta & 1 \le n \le 4 \text{ and } n \le d-1 \\ \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(K_{n-1}-1)\beta\tau}{d}}m^{1/2} + (n-1)\beta & 5 \le n \le d \text{ or } n = d \ge 2 \end{cases}}.$$

This compares to the

$$\begin{cases} \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(n+1)\beta\tau}{d}}m^{1/2} + (n+1)\beta & n \le d-1 \\ \dfrac{\tau}{d}m + 2\sqrt{\dfrac{(d-1)\beta\tau}{d}}m^{1/2} + (d-1)\beta & n = d \end{cases}$$

needed by [11].

As was the case with O.C. SEND 4, ARBITRARY SEND 3 is only a slight improvement over broadcasting. More importantly, it uses only one link between nodes, so an arbitrary exchange is possible between two nodes in the same amount of time by using the links in the other direction.

5 Sending Different Messages From One Node to Every Other Node

The next communication pattern to be looked at is where one node needs to send a different message to each of the other nodes. This operation has no standard name (it was referred to as "scatter" in [11] and "personalized communications" in [5]), so we will call it *distributing*. Its dual operation, *collecting*, where one node has to receive a message from each of the other nodes, is exactly the same operation, only run in reverse. Hence, it suffices to design and analyze distributing algorithms.

Both of these operations are useful in asymmetrical situations where one node of the hypercube acts as a master processor and the others as its slaves. The master distributes different data sets to each of the slaves, which in turn perform computations on them. Then the master collects all of the results.

Assume without loss of generality that node 0 is the distributing node. The following algorithm makes use of O.C. SEND 2, which gets executed on every subcube containing node 0.

Algorithm: DISTRIBUTE 1

There are d stages, numbered $0, 1, \ldots, d-1$. Each node is sent its corresponding message via a separate O.C. SEND 2 applied to the subcube containing it and node 0 as opposite corner nodes. These 2^d-1 o.c. sends are run concurrently, with batching, and staggered so that the one to C_d goes first, followed by the ones to C_{d-1}, then C_{d-2}, etc. Specifically, during stage k, the $\binom{d}{j}$ nodes in C_j, $0 \le j \le k$, do their share of the work for the $\binom{d}{d+j-k} = \binom{d}{k-j}$ O.C. SEND 2's whose destinations are in C_{d+j-k}.

Analysis of DISTRIBUTE 1

The time for each stage depends on the maximal amount of data being sent out over a single link. For stage k, C_j contains $\binom{d}{k-j}$ messages of length m to be sent out evenly along its $\binom{d}{j}(d-j) = \binom{d-1}{j}d$ links to C_{j+1}. This means that each such link sends a packet of length

$$\frac{\binom{d}{k-j}m}{\binom{d-1}{j}d},$$

which is maximized when $j = 0$, for

$$\frac{\binom{d}{k-i}}{\binom{d-1}{i}} \;>\; \frac{\binom{d}{k-(i+1)}}{\binom{d-1}{i+1}}$$

holds for all $i \in \{0, 1, \ldots, k-1\}$. Consequently, the time spent by node 0 during each stage is longer than nodes in any other C_j, so the total time for the algorithm is

$$\begin{aligned}\sum_{k=0}^{d-1}\left[\tau\frac{\binom{d}{k}m}{d}+\beta\right] &= \left[\sum_{k=0}^{d-1}\binom{d}{k}\right]\frac{\tau}{d}m + d\beta \\ &= \boxed{\frac{(2^d-1)\tau}{d}m + d\beta}.\end{aligned}$$

The distribute algorithm in [11] had a time of

$$(2^d-1)\tau m + d\beta$$

and [5]'s had a time strictly greater than ours. The time in [5] is dificult to represent, but has the property that for fixed $d > 1$, the coefficient of m is strictly greater than $(2^d-1)/d$, but it tends to $(2^d-1)/d$ as d approaches ∞.

Note that, as was the case with the o.c. send algorithms, DISTRIBUTE 1 uses links in only one direction, namely towards node 2^d-1. Hence, a DISTRIBUTE 1 from node 2^d-1 can be done concurrently using the opposite set of links. Also, for m sufficiently large (depending on the values of τ, β, and d), it is possible to reduce the β coefficient, which represents the number of stages or "waves" of data leaving node 0, by grouping together some of the waves as they leave node 0 and then breaking them apart in C_1. For example, using $d = 3$, first a wave containing messages for C_2 and C_3 is sent out, taking $\frac{4}{3}\tau m + \beta$ time to leave node 0. When this wave arrives at the nodes of C_1, the portion destined for C_3 is sent, taking $\frac{1}{6}\tau m + \beta$ time, and then the portion destined for C_2 is sent, taking $\frac{1}{2}\tau m + \beta$ time. When the portion destined for C_3 reaches the nodes of C_2, it is sent on to C_3, taking $\frac{1}{3}\tau m + \beta$ time. Meanwhile, the second wave of messages sent by node 0 are those destined for C_1, taking $\tau m + \beta$ time. Messages for C_1 finish arriving at time $\frac{7}{3}\tau m + 2\beta$, messages for C_2 finish at time $2\tau m + 3\beta$, and messages for C_3 finish at time $\frac{11}{6}\tau m + 3\beta$. If $m/3 > \beta$, then all messages arrive by $\frac{7}{3}\tau m + 2\beta$, which has improved upon the coefficient of β. This can be shown to be absolutely optimal. This approach is extended in DISTRIBUTE 2.

Algorithm: DISTRIBUTE 2

Fix d, let k be the smallest integer such that

$$d\frac{(d-1)^k - 1}{d-2} \geq 2^d - 1,$$

let r be such that

$$d\frac{r^k - 1}{r-1} = 2^d - 1.$$

(Since r may be irrational, in practice one may prefer to use some rational r' such that $r \leq r' < d-1$.) There will be exactly k waves. Wave k will be those messages destined for C_1, where each link from node 0 carries a message of size m. Wave $k-1$ starts from node 0 with packets of size rm along each link, and will contain all of the messages for C_2 and (for large d) portions of each of the messages for C_3, where the portion is chosen to fill the packet size. Wave $k-2$ will start with packets of size r^2m, containing the rest of each of the messages for C_3, plus messages for C_4, plus (for sufficiently large d) portions of messages for C_5. Each wave starts with packets r times larger than the following wave, and contains messages destined for a set of further C_i's. When a wave reaches the nodes of C_1 it is broken into wavelets, one for each of the destination C_i in the wave. These wavelets continue on to their destination, adjusting the packet sizes at each step as in DISTRIBUTE 1, but not subdividing into smaller wavelets.

Analysis of DISTRIBUTE 2

Since the bandwidth from C_1 to C_2 is $d-1$ times the bandwidth from 0 to C_1, and $r < d-1$, for m sufficiently large each wave can be sent on from C_1 before the next wave arrives. The reason m must be sufficiently large is that the breaking into wavelets introduces additional β terms, but since r is less than $d-1$ there is a slight bit of extra bandwidth, which can mask the extra start-up for sufficiently large m. As in DISTRIBUTE 1, it can be shown that, for suffiently large m, all wavelets reach their destination by the time the last wave reaches C_1. Therefore the total time is determined

by the time it takes node 0 to send all k waves, which is

$$\frac{(2^d-1)\tau}{d}m + k\beta \quad \approx \quad \boxed{\frac{(2^d-1)\tau}{d}m + \frac{d}{\log_2 d}\beta}.$$

This algorithm shows that one cannot obtain a lower bound by simply adding the bandwidth lower bound, which determines the optimal coefficient of m, to the start-up lower bound which shows that at least $d\beta$ time is needed to move any message across the hypercube. In general one can only take the maximum of these two components as a lower bound, since operations can be overlapped.

6 Completing Hypercube Algorithms

Completing a hypercube operation refers to taking an operation centered around one node and then simultaneously performing it on all of the nodes. This produces highly symmetrical communication patterns which utilize all of the available bandwidth.

The simplest way to complete an operation is just to run 2^d single-node operations concurrently. In terms of algorithms, this amounts to using the same number of stages as the single-node version. During each stage of the complete algorithm, however, each node does all the work necessary for the corresponding stage in all of the single node algorithms. Link conflicts are resolved by *batching*. That is, grouping together all of the separate packets that have to be sent along a particular link during the same stage and sending them as one big packet. This also reduces communication overhead (i.e., β terms) considerably.

The best single-node algorithms to complete are usually the simplest versions which still take advantage of the concurrent link capability. Sophisticated techniques such as pipelining and link balancing aren't necessary because the complete operations are so symmetric. The first operation to be completed will be the broadcast. This pattern is useful for various matrix operations as well as vector multiplication.

Algorithm: COMPLETE BROADCAST

Complete BROADCAST 2. There are d stages, numbered $0, 1, \ldots, d-1$, and during stage k, each node does its share of the work for the corresponding stage of BROADCAST 2 for all $\binom{d}{k}$ nodes which are distance k from it.

Analysis of COMPLETE BROADCAST

During stage k of BROADCAST 2, the total amount of data being sent out is $2^k dm/d = 2^k m$, so the corresponding amount being sent out in COMPLETE BROADCAST is $2^d 2^k m$. Due to the overall symmetry of COMPLETE BROADCAST, this outgoing data will be evenly divided among all $2^d d$ links of the hypercube, so each link ends up sending a packet of size $2^k m/d$. Therefore, the time for the algorithm is

$$\sum_{k=0}^{d-1}\left(\tau\frac{2^k m}{d} + \beta\right) \quad = \quad \boxed{\frac{(2^d-1)\tau}{d}m + d\beta}.$$

For purposes of comparison, [11] produced an "optimal" complete broadcast (which they referred to as a "total exchange") with a running time of

$$\frac{(2^d+d^2)\tau}{d}m + d\beta.$$

The next operation to be completed will be the o.c. send. A complete o.c. send, henceforth referred to as an *inversion* is another fundamental communication pattern useful for reversing the order of data which is stored by node i.d. and for transposing matrices (to be discussed later).

Algorithm: INVERSION (Complete Opposite Corner Send)

Complete O.C. SEND 2 in exactly the same manner as BROADCAST 2 was in COMPLETE BROADCAST.

Analysis of INVERSION

During each stage of O.C. SEND 2, a total of d packets of size m/d were being sent over separate links. Now there are $2^d d$ such packets, but there are also that many links and the symmetry of the algorithm guarantees that no more than one packet will be sent along the same link during a stage. Thus, the time for INVERSION is identical to O.C. SEND 2:

$$d\left(\tau\frac{m}{d}+\beta\right) \quad = \quad \boxed{\tau m + d\beta}.$$

Now consider the ultimate communication pattern, the *complete exchange*. This is when every node wants to send (as well as receive) a different message to (from) each of the other nodes. In other words, it's the same thing as completing the distributing or collecting operations. The complete exchange turns out to be useful for matrix transpositions as well as random communication patterns (both to be discussed later). In [11], complete exchange was called multigather/scatter.

Algorithm: COMPLETE EXCHANGE

Just complete DISTRIBUTE in the same manner that O.C. SEND 1 was completed to produce INVERSION. There are still d stages, numbered $0, 1, \ldots, d-1$.

Analysis of COMPLETE EXCHANGE

As in INVERSION's analysis, all that has to be determined is the amount of data each node has to pass along each stage. Basically, every node starts out with $(2^d-1)m$ items which have to be sent out to the other nodes. During stage k, it starts sending messages to the $\binom{d}{k}$ nodes which are distance k away. No messages reach their proper destinations until the last stage, which means that a total of

$$2^d \sum_{j=0}^{k} \binom{d}{j}$$

messages are being worked on during stage k. Due to the symmetric pattern of the sends, each of the $2^d d$ links thus sends a packet of size

$$\sum_{j=0}^{k} \frac{\binom{d}{j}}{d} m \quad = \quad \sum_{j=0}^{k} \frac{\binom{d-1}{j}}{d-j} m,$$

so the algorithm needs time

$$\sum_{k=0}^{d-1}\left[\tau\sum_{j=0}^{k}\frac{\binom{d-1}{j}}{d-j}m+\beta\right] \quad = \quad \sum_{j=0}^{d-1}\sum_{k=j}^{d-1}\frac{\binom{d-1}{j}}{d-j}\tau m + d\beta$$

$$= \sum_{j=0}^{d-1} \binom{d-1}{j} \tau m + d\beta$$
$$= \boxed{2^{d-1}\tau m + d\beta},$$

compared to the

$$2^{d-1} d\tau m + d\beta$$

needed by [11]'s corresponding algorithm.

Finally, sometimes a situation arises where each node wants to send to one other node as well as receive from just one node. This will be termed a *permuted send*, although in some ways it is analogous to a complete arbitrary send. An obvious example is an inversion. Another one is when each node wants to send to the next higher-numbered node (mod 2^d), which can be thought of as a *rotation*.

There are $2^d!$ such permutations, so determining the most efficient algorithm for each one seems neither possible nor practical, though recently some papers have appeared analyzing specific permutations [8, 10]. For arbitrary permutatations, however, a deterministic analogue of Valiant's randomized routing [12, 13] can be employed. It consists of two complete exchanges: one to disperse all the data evenly throughout the cube and another to collect it all up at the appropriate destinations. We explicitly use the fact that all nodes know the permutation being performed so that destination information need not be sent with the data.

Algorithm: PERMUTED SEND

Each node breaks up its m items into 2^d packets of $m/2^d$ items apiece. These packets are distributed throughout the cube via a complete exchange so that each node has one packet from every node in the cube. Since the communication pattern is a permutation, this also means that each node has a different packet to send to every other node in the cube. As a result, another complete exchange can be used to route all of the packets to their correct destinations.

Analysis of PERMUTED SEND

There are two complete exchanges, each involving message lengths of $m/2^d$ items, so the time needed for PERMUTED SEND is

$$2\left(2^{d-1}\tau\frac{m}{2^d} + d\beta\right) = \boxed{\tau m + 2d\beta}.$$

7 Matrix Transposition

Transposing a matrix in a hypercube is an interesting communication problem which can make good use of some of the algorithms developed so far. It has been previously considered in [9, 11], but faster algorithms will be developed here. Suppose you want to transpose an $N \times N$ matrix M stored in a d-dimensional hypercube, with each node containing $N^2/2^d$ entries. It is necessary to specify exactly how M is stored, where the usual ways are either by rows(columns) or as square submatrices. Storage by rows is the easier of the two, so it will be considered first.

7.1 Storage by Rows(Columns)

There are many ways of storing by rows(columns), where we assume that N is evenly divisible by 2^d. For example, the rows may be stored by partitioning the rows into blocks of $N/2^d$ consecutive rows, where the assignment of blocks to nodes may or may not use a Gray code. Or it may be that a striped pattern is used, partitioning the rows into sets of rows 2^d apart, again with variations possible on how the sets are mapped onto the nodes. However, no matter what method is used to assign rows to nodes (as long as the assignment evenly distributes the data), to perform transposition each node must send exactly $N^2/2^{2d}$ entries to each other node. In other words, a complete exchange has to be performed with a message length of $N^2/2^{2d}$. This takes time

$$2^{d-1}\tau\frac{N^2}{2^{2d}} + d\beta \quad = \quad \boxed{\frac{\tau}{2^{d+1}}N^2 + d\beta},$$

as compared to

$$\frac{d\tau}{2^{d+1}}N^2 + d\beta$$

needed by [11].

7.2 Storage by Submatrices

When stored as submatrices, it is convenient to assume that d is even, say $d = 2c$, and that N is evenly divisible by 2^c. Assume that M has been partitioned into submatrices $\mathsf{M}_{x,y}$, $x, y \in \{0, 1, \ldots, 2^c-1\}$, where $\mathsf{M}_{x,y}$ is formed by the intersection of rows

$$xN/2^c + 1 \quad \text{through} \quad (x+1)N/2^c$$

with columns

$$yN/2^c + 1 \quad \text{through} \quad (y+1)N/2^c.$$

Let G denote any permutation of $\{0, \ldots, 2^c-1\}$, and assume that $\mathsf{M}_{x,y}$ is stored in node $G(x)2^c + G(y)$. Typical choices for G include the identity, in which case this is known as *row-major ordering*, or a Gray code, in which case adjacent submatrices are stored in adjacent nodes. No matter what G is used, transposition reduces to the problem of node $a2^c + b$ exchanging its entries with node $b2^c + a$, for all $a, b \in \{0, 1, \ldots, 2^c-1\}$. We provide an algorithm for this operation.

First observe that this is a permuted send with message length $N^2/2^d$. Hence, it can be accomplished by using PERMUTED SEND in time

$$\tau\frac{N^2}{2^d} + 2d\beta \quad = \quad 2\left(\frac{\tau}{2^{d+1}}N^2 + d\beta\right).$$

This is twice as long as when M is stored by rows, yet on average, each item moves only half as far. Consequently, it would not be unreasonable to expect there to be an algorithm which works in half the time.

In fact, such an algorithm does exist, but describing and analyzing it requires examining the bit patterns of the node i.d.'s. Consider node $a2^c + b$, where $a, b \in \{0, 1, \ldots, 2^c-1\}$. In their base two representations, a and b differ by say, k bits, and agree on the other $c-k$. Now let $\mathsf{S}_{a,b}$ denote the set of all nodes whose first c bits of their i.d.'s differ from their last c bits in exactly the same k positions that a and b do.

Observe that $\mathsf{S}_{a,b}$ contains 2^k nodes. By themselves, they do not form a proper subcube, but something close to one. The distance between any two nodes of $\mathsf{S}_{a,b}$ is always even, and if there

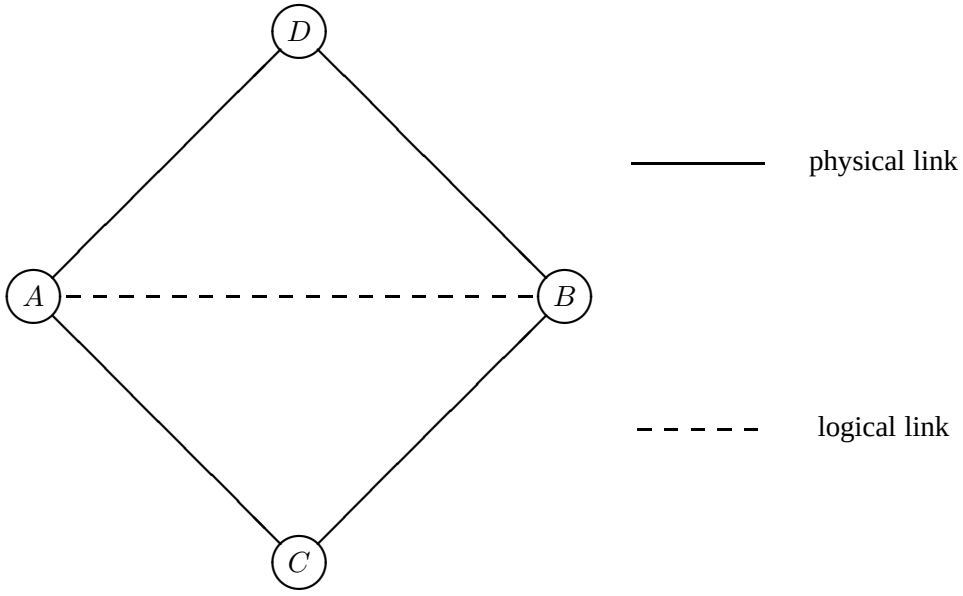

Figure 3: Logical and Physical Links Between two Nodes

were links between the nodes that are distance two apart, then $\mathsf{S}_{a,b}$ plus these new connections would form a k-dimensional hypercube.

With these thoughts in mind, define a *logical link* between two hypercube nodes A and B, which are distance two apart, to be the four physical links which connect A and B along the two possible paths of length 2 (see Figure 3). Let C and D denote the *intermediate nodes* connecting A and B. A *logically connected (l.c.)* subcube can then be defined to be a subset of nodes whose logical links connect them together in a hypercube network.

That said, $\mathsf{S}_{a,b}$ is a l.c. subcube. Furthermore, its intermediate node i.d.'s have the property that their first c bits differ from their last c bits in precisely $k-1$ positions. Finally, from the definition of $\mathsf{S}_{a,b}$, every node in the hypercube belongs to exactly one such l.c. subcube. Combining these last two statements, it becomes apparent that no two such subcubes can share the same physical link. As a consequence, algorithms can be run concurrently on all of the these l.c. subcubes without the possibility of link conflict.

Returning to the original problem, node $a2^c + b$ can exchange data with node $b2^c + a$ by simply performing an o.c. exchange in $\mathsf{S}_{a,b}$. In fact, every node in $\mathsf{S}_{a,b}$ can exchange data with their corresponding node for the transposition by performing an inversion in $\mathsf{S}_{a,b}$. This brings up the need then for an inversion algorithm for l.c. subcubes.

Inversion in a logically connected subcube

Sending data along a logical link is equivalent to doing an o.c. send in a 2-dimensional hypercube. Hence, a standard send would take time

$$\frac{\tau}{2}m + 2\sqrt{\frac{\beta\tau}{2}}m^{1/2} + \beta$$

to perform, so a l.c. inversion could be accomplished by performing a regular INVERSION using these logical sends. For a k-dimensional l.c. subcube, this would require time

$$k\left(\frac{\tau}{2}\frac{m}{k} + 2\sqrt{\frac{\beta\tau}{2}}\left(\frac{m}{k}\right)^{1/2} + \beta\right) \quad = \quad \frac{\tau}{2}m + 2\sqrt{\frac{k\beta\tau}{2}}m^{1/2} + k\beta.$$

As long as the number of packets sent out along each physical link in a logical send is at least two, however, then there is no point in waiting for all of the incoming packets to arrive before starting to pass them along. That is, after all, the whole idea behind pipelining.

Algorithm: L.C. INVERSION

Perform a regular INVERSION along the logical links of the l.c. subcube, making sure to pipeline the stages together and breaking the data up into at least four packets so that incoming packets start arriving no later than when the last of the outgoing packets are being sent out.

Analysis of L.C. INVERSION

Let p denote the number of packets to be sent out along each outgoing physical link (note that p has to be at least 2). Then the packet size for each send is $m/2kp$. With pipelining then, it takes a total of kp sends for each node to pass along its data, with the intermediate nodes being one send behind the regular nodes. Therefore, $kp+1$ send stages are needed, so the algorithm runs in time

$$(kp+1)\left(\frac{\tau}{2kp}m+\beta\right),$$

which is minimized when

$$p \;=\; \frac{1}{k}\sqrt{\frac{\tau}{2\beta}}m^{1/2}.$$

This produces a final time of

$$\boxed{\frac{\tau}{2}m+2\sqrt{\frac{\beta\tau}{2}}m^{1/2}+\beta}.$$

Observe that this time is the same time as needed by INVERSION for a 2-dimensional cube. Also, it is independent of k so long as m is large enough to insure $p \geq 2$.

Algorithm: MATRIX TRANSPOSITION (stored by submatrices)

With L.C. INVERSION, transposing M becomes trivial. Just perform it on every l.c. subcube of M.

Analysis of MATRIX TRANSPOSITION

Since the l.c. inversions can all be done concurrently without overlap, and all take the same amount of time, the transposition is completed in time

$$\frac{\tau}{2}\frac{N^2}{2^d}+2\sqrt{\frac{\beta\tau}{2}}\left(\frac{N^2}{2^d}\right)^{1/2}+\beta \;=\; \boxed{\frac{\tau}{2^{d+1}}N^2+2\sqrt{\frac{\beta\tau}{2^{d+1}}}N+\beta},$$

which is nearly the same time needed when M was stored by rows. This is approximately half of the

$$\frac{\tau}{2^d}N^2+(d-1)\tau$$

time required by [9], where it is assumed that β is zero.

8 Histogramming

The same techniques used previously can be applied to the problem of histogramming. We consider a simple variant in in which there are m different "bins", each node starts with a value for each of the bins, and the goal is to find the sum of the values for each bin. The sum for bins $im/2^d$ through $(i+1)m/2^d - 1$ will be in node i. (If it is desired that all nodes contain all sums, then a complete broadcast can be used at the end.) We assume that each value and sum is of unit length.

Algorithm: HISTOGRAM

First consider the algorithm where the data is exchanged one dimension at a time, using recursive halving to decrease the number of subtotals in each node. During the first stage, nodes in the bottom half of the subcube send up their values for the second half of the bins to their neighbors in the upper half, which are concurrently sending down their values for the first half of the bins. Each node adds the values received to its own, and recursively continues on to the next stage. The final HISTOGRAM algorithm is just the symmetrized version of this simple algorithm.

Analysis of HISTOGRAM

For the unsymmetric algorithm, stage k, $1 \leq k \leq d$, takes $(\tau/2^k)m + \beta$ time, so for the symmetric algorithm it takes $(\tau/d2^k)m + \beta$ time. The total time is

$$\boxed{\frac{(1-2^{-d})\tau}{d}m + d\beta}.$$

9 Optimality of the Algorithms

As was mentioned earlier, the coefficients of the high-order terms are the least possible. In all cases, a proof can be given based on a simple counting argument. The easiest such approach is

- pick a subset S of links,
- show that the total message load that must be sent over S is at least some amount a, and
- conclude that at least one link sends at least $a/|\mathsf{S}|$ and thus, takes at least

$$\frac{a\tau}{|\mathsf{S}|}m + \beta$$

  time doing so.

For example, in BROADCAST, O.C. SEND, ARBITRARY SEND, DISTRIBUTE, and HISTOGRAM, let S be the d outgoing links of node 0. Then a is m, m, m, $(2^d-1)m$, and $(1-2^{-d})m$, respectively. In the case of COMPLETE BROADCAST, pick S to be the d incoming links to node 0 and set a to $(2^d-1)m$, since node 0 receives a different message from each other broadcast. For INVERSION and COMPLETE EXCHANGE, consider S to be the 2^d links connecting nodes $0, 1, \ldots, 2^{d-1}-1$ in the "lower" subcube L with nodes $2^{d-1}, 2^{d-1}+1, \ldots, 2^d-1$ in the "upper" subcube U. Then a is $2^d m$ and $2^d 2^{d-1} m$, respectively, since every node in L sends one and 2^{d-1} messages, respectively, to the nodes in U (and vice versa). PERMUTED SEND is optimal since it is slower than the special case INVERSION by only an additive $d\beta$. Finally, for MATRIX TRANSPOSITION, when the

matrix is stored by rows or columns the problem is just complete exchange, which was shown to be optimal. When the data is stored as submatrices, let S be all $d\,2^d$ links. Then a is $dN^2/2$ since the total distance traveled by all messages, each of size $N^2/2^d$, is $d\,2^{d-1}$.

The lower bound for the permutation *reflection*, where every node in L exchanges with its corresponding neighbor in U, has the same high-order term as does inversion (by the same argument). This occurs despite the fact that reflection is a fixed-point free permuation with the smallest total message distance, while inversion has the greatest total message distance. Given this, and the fact that PERMUTED SEND shows that all permutations can be routed with this highest-order term, one might guess that all fixed-point free permutations require the same highest-order term. (If permutations with fixed points are considered, then the identity can be completed in zero time.) However, it has been shown that some fixed-point free permutations can be routed with a highest-order term smaller than that of reflection [10], and therefore PERMUTED SEND is only worst-case optimal among fixed-point free permutations.

Beyond the highest-order term, we believe that some of the algorithms herein are absolutely optimal. Unfortunately, we have generally been unable prove this because of the difficulty in finding good lower bounds which go beyond the highest-order term. Such bounds must incorporate both bandwidth considerations and an accounting of start-up times, but, as was noted in Section 5, one cannot simply add these components together to obtain a correct lower bound.

We can, however, prove absolute optimality for DISTRIBUTE 2 for any d, if m is sufficiently large. Notice that at least one of node 0's neighbors must receive at least $(2^d-1)m/d$ items, and all except perhaps m of these items need to be forwarded. Therefore it suffices to show that if a node 0 is connected to a node 1, which in turn is connected to $d-1$ additional nodes, and if node 0 starts with m items destined for node 0, and $(2^d-1)m/d-m$ items destined for the additional nodes, then the time needed is at least the time taken by DISTRIBUTE 2. We assume that we have complete freedom in deciding which additional node to deliver a specific item to.

Without increasing the time, we can alter any algorithm so that the items destined for node 1 are the last items sent from node 0. Suppose the first packet to arrive at node 1 has size p, and the second packet has size q, and both are destined for the additional nodes. If $p < (d-1)q$, then the items cannot finish arriving at the additional nodes until time $(p\tau+\beta)+(q\tau+\beta)+(q\tau/(d-1)+\beta)$. By moving some of the items from the second message to the first, creating new messages with lengths p' and q', where $p'=(d-1)q'$ and $p'+q'=p+q$, the messages can finish arriving at the additional nodes at time $(p'\tau+\beta)+(q'\tau+\beta)+(q'\tau/(d-1)+\beta)$. Since $q'<q$, this is faster. A similar argument applies if $p>(d-1)q$, and therefore without increasing the time, we can assume that the first packet is $d-1$ times as long as the second.

This argument can be applied inductively, showing that we may assume that each packet sent from node 0 is $(d-1)$ as long as the following one. (Temporarily ignore the fact that this argument does not apply to the last packets, since some of the items in them are not forwarded). Suppose node 0 sends k packets, with sizes $x(d-1)^{k-1}$, $x(d-1)^{k-2}$, $\ldots$, x, where x is such that the sum of the message sizes is $(2^d-1)m/d$. The time for node 1 to receive these messages and send on the items destined for the additional nodes is at least

$$\frac{(2^d-1)\tau}{d}m+k\beta\;+\;\frac{(x-m)\tau}{d-1}+\beta,$$

where the last two terms are included only if $x>m$. For fixed d, τ, and β, and sufficiently large m, this is minimized when $k=\lceil\log_{d-1}[1+(d-2)(2^d-1)/d]\rceil$, which gives the time taken by DISTRIBUTE 2. To be correct, this argument must be modified to deal with the sizes of the last packets, since the argument showing each packet must be $d-1$ times as long as the following one assumed that all items were destined for the additional nodes. An analysis by cases shows that the same time bound holds.

10 Conclusion

We have shown that link-bound hypercubes can make effective use of all of their communication links to perform some common communication-intensive tasks. Since a lower bound for some of these tasks is the time needed to send out the data from an originating node, such tasks would take longer on more restricted machines in which nodes cannot use all of their communication links at one time. Thus our algorithms provide support for the belief that it is useful to build machines where all communication links can be used simultaneously.

By systematically applying a few techniques such as pipelining, symmetrizing, and completing, we were able to develop a collection of algorithms giving efficient solutions to a wide range of problems. We concentrated on communication problems that are rather fundamental, and have not tried to develop all of their uses. However, we note that several additional matrix manipulation problems can be solved by our algorithms. For example, if a matrix is stored by rows or columns, then switching between blocked and striped storage, or rotating by a quarter-turn, are all examples of complete exchange. If a matrix is stored via submatrices, and the G function used in the assignment is either the identity or a reflexive Gray code, then rotation via quarter-turns or half-turns can be accomplished by algorithms closely related to MATRIX TRANSPOSITION. Since the initial announcement of our results in [14] and the submission of this paper, additional papers have appeared which pursue the use of such techniques for matrix problems [6, 8, 9]. These papers include experimental results on Intel and Thinking Machines hypercubes, showing that our techniques do indeed result in faster message transmission.

Though our algorithms are deterministic, this paper has ties to Valiant's work on randomized routing [12, 13]. He showed that indivisible unit-length messages in a link-bound hypercube could be routed in $\Theta(d)$ expected time, no matter what the permutation, by routing each message to a random intermediate destination and then on to its original destination. For long divisible messages and a known permutation (so that header information need not be attached), PERMUTED SEND eliminates the random destination by sending a portion of the message to every processor. Further, in [12] he used four "bad" examples to empirically show the usefulness of randomization. One of these is equivalent to matrix transposition for a matrix stored as submatrices, and the worst one was inversion. MATRIX TRANSPOSITION and INVERSION show that there are efficient deterministic routing schemes for these permutations.

Finally, despite the intense interest in hypercube communication [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13], still little is known about optimal hypercube performance on communication-intensive tasks such as sorting, routing, data balancing, database operations, and image warping. For example, it is not known if a d-dimensional hypercube, starting with one item per node, can sort the items in $\Theta(d)$ worst-case time. Additional open questions include extending analyses to processor-bound and DMA-bound hypercubes, and to problems where the communication pattern is not known in advance and/or the message lengths are not uniform.

References

[1] Baru, C. K., and Frieder, O. Implementing relational database operations in a cube-connected multicomputer system. *Proc. 3rd Int'l. Conf. on Data Engineering*, 1987.

[2] Cybenko, G. Dynamic load balancing for distributed memory multiprocessors. Tufts Univ. Dept. of Computer Science Tech. Report 87-1, Jan. 1987.

[3] Gustafson, J. L., Hawkinson, S., and Scott, K. The architecture of a homogeneous vector supercomputer. *Proc. 1986 Int'l. Conf. on Parallel Proc.*, IEEE, 1986, pp. 649-652.

[4] Hayes, J., Mudge,T., Stout, Q. F., Coley, S., and Palmer, J. A microprocessor-based hypercube supercomputer. *IEEE Micro* 6 (1986), pp. 6-17.

[5] Ho, C.-T., and Johnsson, S. L. Distributed routing algorithms for broadcasting and personalized communications in hypercubes. *Proc. 1986 Int'l. Conf. on Parallel Proc.*, IEEE, 1986, pp. 640-648.

[6] Ho, C.-T., and Johnsson, S. L. Algorithms for matrix transposition on boolean n-cube configured ensemble architectures, *Proc. 1987 Int'l. Conf. on Parallel Proc.*, IEEE, 1987, pp. 621-629.

[7] Ho, C.-T., and Johnsson, S. L. Optimal algorithms for stable dimension permutations on boolean cubes, *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applic.*, ACM, 1988, pp. 725-736.

[8] Ho, C.-T., and Johnsson, S. L. Expressing boolean cube matrix algorithms in shared memory primitives. *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applic.*, ACM, 1988, pp. 1599-1609.

[9] Johnsson, S. L. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel and Distributed Computing* 4 (1987), pp. 133-172.

[10] Livingston, M. and Stout, Q. F. Good permutations for hypercube communication, in preparation.

[11] Saad, Y. and Schultz, M. H. Data communications in hypercubes, Yale Univ. Dept. of Computer Science Research Report YALEU/DCS/RR-428, 1985.

[12] Valiant, L. G. Experiments with a parallel communication scheme, *Proc. 18th Allerton Conf. on Communication, Control, and Computing*, 1980, pp. 802-811.

[13] Valiant, L. G. A scheme for parallel communication. *SIAM J. Computing* 11 (1982), pp. 350-361.

[14] Wagar, B., and Stout, Q. F. Passing messages in link-bound hypercubes. *Hypercube Multiprocessors 1987*, SIAM, pp. 251-257.