January 1989 LIDS - P - 1847

Optimal Communication Algorithms for Hypercubes[1]

by

D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis[2]

Abstract

We consider several basic communication problems in a hypercube network of processors. These include the problem of simultaneous broadcast of the same packet from every processor to all other processors and the problem of simultaneous exchange of different packets between every pair of processors. The algorithms proposed for these problems are optimal in terms of execution time and communication resource requirements, that is, they require the minimum possible number of time steps and packet transmissions. This is a particularly strong form of optimality, which has not been considered in earlier investigations.

[1] Research supported by NSF under Grants ECS-8519058 and ECS-8552419, with matching funds from Bellcore Inc., the ARO under Grant DAAL03-86-K-0171, and the AFOSR under Grant AFOSR-88-0032.

[2] Laboratory for Information and Decision Systems, M.I.T, Cambridge, Mass. 02139.

1. INTRODUCTION AND PROBLEM FORMULATION

When algorithms are executed in a network of processors, it is necessary to exchange some intermediate information between the processors. The interprocessor communication time may be substantial relative to the time needed exclusively for computations, so it is important to carry out the information exchange as efficiently as possible. There are a number of generic communication problems that arise frequently in numerical and other algorithms. In this paper, we describe new algorithms for solving some of these problems on a hypercube. An important characteristic of these algorithms is that they are *optimal*, in the sense that they execute the required communication tasks in the minimum possible number of time steps and link transmissions.

To define a hypercube network (or d-cube), we consider the set of points in d-dimensional space with each coordinate equal to zero or one. We let these points correspond to processors, and we consider a communication link for every two points differing in a single coordinate. We thus obtain an undirected graph with the processors as nodes and the communication links as arcs. The binary string of length d that corresponds to the coordinates of a node of the d-cube is referred to as the *identity number* of the node. We recall that a hypercube of any dimension can be constructed by connecting lower-dimensional cubes, starting with a 1-cube. In particular, we can start with two $(d-1)$-dimensional cubes and introduce a link connecting each pair of nodes with the same identity number (see e.g. [BeT89], Section 1.3). This constructs a d-cube with the identity number of each node obtained by adding a leading 0 or a leading 1 to its previous identity, depending on whether the node belongs to the first $(d-1)$-dimensional cube or the second (see Fig. 1). When confusion cannot arise, we refer to a d-cube node interchangeably in terms of its identity number (a binary string of length d) and in terms of the decimal representation of its identity number. Thus, for example, the nodes $(00\cdots0)$, $(00\cdots1)$, and $(11\cdots1)$ will also be referred to as nodes 0, 1, and 2^d-1, respectively.

The *Hamming distance* between two nodes is the number of bits in which their identity numbers differ. Two nodes are directly connected with a communication link if and only if their Hamming distance is unity, that is, if and only if their identity numbers differ in exactly one bit. The number of links on any path connecting two nodes cannot be less than the Hamming distance of the nodes. Furthermore, there is a path with a number of links that is equal to the Hamming distance, obtained, for example, by switching in sequence the bits in which the identity numbers of the two nodes differ (equivalently, by traversing the corresponding links of

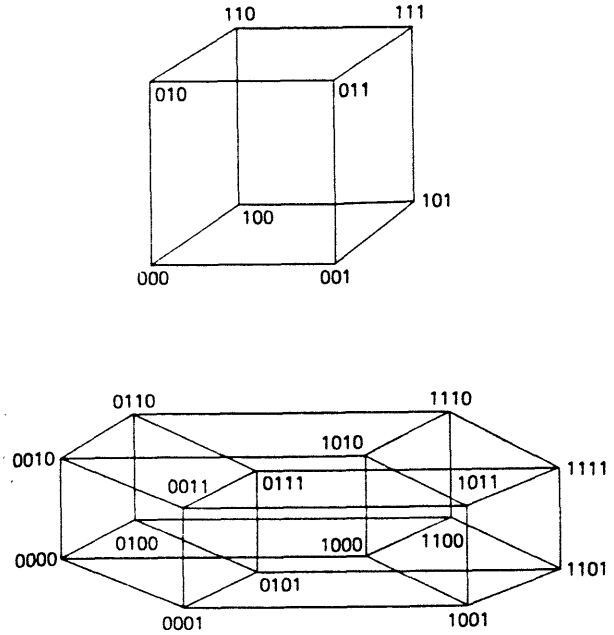

Figure 1: Construction of a 3-cube and a 4-cube by connecting the corresponding nodes of two identical lower-dimensional cubes. A node belongs to the first lower-dimensional cube or the second depending on whether its identity has a leading 0 or a leading 1.

the hypercube). Such a path is referred to as a *shortest path* in this paper and a tree consisting of shortest paths from some node to all other nodes is referred to as a *shortest path tree*.

Information is transmitted along the hypercube links in groups of bits called *packets*. In our algorithms we assume that the time required to cross any link is the same for all packets, and is taken to be one unit. Thus, our analysis applies to communication problems where all packets have roughly equal length. We assume that packets can be simultaneously transmitted along a link in both directions, and that their transmission is error free. Only one packet can travel along a link in each direction at any one time; thus, if more than one packets are available at a node and are scheduled to be transmitted on the same incident link of the node, then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in queue.

Each node is assumed to have infinite storage space. We will consider two different cases regarding the availability of links for transmission at each time period. In the first case, called the *Multiple Link Availability (or MLA) assumption*, we assume that all incident links of a node can be used simultaneously for packet transmission and reception. In the second case, called the *Single Link Availability (or SLA) assumption*, we assume that at any time, a node can transmit a packet along at most one incident link and can simultaneously receive a packet along at most one incident link. Finally, we assume that each of the algorithms proposed in this paper is simultaneously initiated at all processors. This is a somewhat restrictive assumption,

essentially implying that all processors can be synchronized with a global clock.

We will be concerned with several communication problems, which we now describe:

Single Node and Multinode Broadcast

In the first problem, we want to send the same packet from a given node, called the *root*, to every other node (we call this a *single node broadcast*). In a generalized version of this problem, we want to do a single node broadcast simultaneously from all nodes (we call this a *multinode broadcast*).

A typical example of a multinode broadcast arises in linear iterations of the form

$$x := Ax + b,$$

where A is an $n \times n$ matrix and b is an n-dimensional vector. Here we assume that each node i of an n-processor system holds the ith row of A and the ith coordinate of b, and updates x_i. Thus at the end of an iteration, it is necessary for every node i to send the updated value of x_i to every other node, which is a multinode broadcast.

Clearly, to solve the single node broadcast problem, it is sufficient to transmit the packet along a directed spanning tree emanating from the root node, that is, a spanning tree of the network together with a direction on each link of the tree such that there is a unique directed path on the tree from the root to every other node (all links of the path must be oriented away from the root). To solve the multinode broadcast problem, we need to specify one spanning tree per root node. The difficulty here is that some links may belong to several spanning trees; this complicates the timing analysis, because several packets can arrive simultaneously at a node, and require transmission on the same link with a queueing delay resulting.

Single Node and Multinode Accumulation

There are two important communication problems that are dual to the single and multinode broadcasts, in the sense that the spanning tree(s) used to solve one problem can also be used to solve the dual in the same amount of communication time. In the first problem, called *single node accumulation*, we want to send to a given node a packet from every other node; we assume, however, that packets can be "combined" for transmission on any communication link, with a "combined" transmission time equal to the transmission time of a single packet. This

problem arises, for example, when we want to form at a given node a sum consisting of one term from each node as in an inner product calculation; we can view addition of scalars at a node as "combining" the corresponding packets into a single packet. The second problem, which is dual to a multinode broadcast, is called *multinode accumulation*, and involves a separate single node accumulation at each node.

It can be shown that a single node (or multinode) accumulation problem can be solved in the same time as a single node (multinode, respectively) broadcast problem. To see this, note that any single node broadcast algorithm consists of the list of links on which the packet is transmitted, together with the time that transmission begins on each link of the list. From this information, one can construct a set of times for starting transmission of a packet along each link of the list in the reverse direction, which specifies a single node accumulation algorithm. The process is illustrated in Fig. 2; the detailed mathematical proof is left for the reader. A similar argument shows that any algorithm for single node (or multinode) accumulation can be used to produce a single node (multinode, respectively) broadcast algorithm that takes the same amount of time.

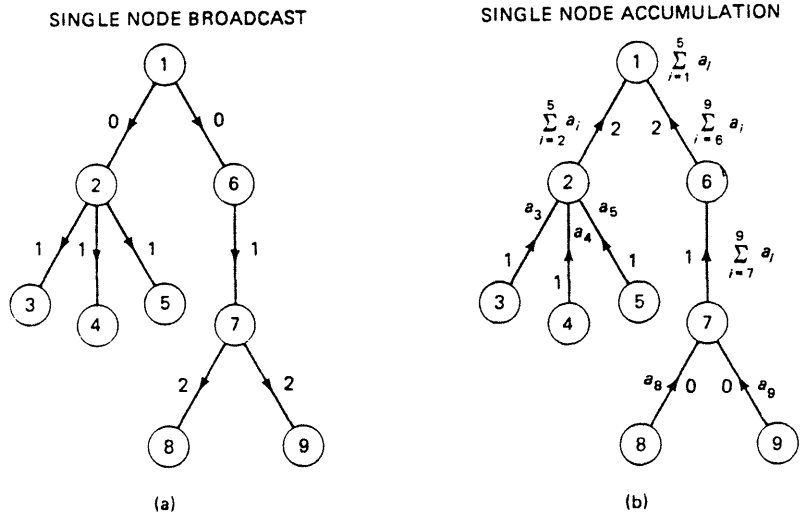

Figure 2: (a) A single node broadcast requires a tree that is rooted at a given node (which is node 1 in the figure). The time next to each link is the time at which transmission of the packet on the link begins. (b) A single node accumulation problem involving summation of n scalars $a_1, \ldots, a_n$ (one per node) at the given node (which is node 1 in the figure). The time next to each link is the time at which transmission of the "combined" packet on the link begins, assuming that the time for scalar addition is negligible relative to the time required for packet transmission. The time for single node accumulation (or broadcast) is the maximum length of a path from a node to the root (or from the root to a node, respectively), counting each link as one unit. Thus, the single node accumulation and the single node broadcast take the same amount of time if a single packet in the latter problem corresponds to a scalar in the former problem.

Single Node Scatter and Gather; Total Exchange

Another interesting communication problem is to send a packet from every node to every other node (here a node sends different packets to different nodes in contrast with the multinode broadcast problem, where a node sends the same packet to every other node). We call this the *total exchange problem*, and note that it arises frequently in connection with matrix computations. A related problem, called the *single node scatter* problem, involves sending a separate packet from a single node, called the *root*, to every other node. A dual problem, called *single node gather*, involves collecting a packet at a given node from every other node. An algorithm that solves the single node scatter (or gather) problem consists of a schedule of packet transmissions on each link that properly takes into account queueing. By reversing this schedule as discussed in connection with the single node accumulation problem, it can be seen that for every algorithm that solves the single node scatter (or gather) problem, there is a corresponding algorithm that solves the single node gather (scatter, respectively) problem, and takes the same amount of communication time.

Note that in a multinode broadcast, each node receives a different packet from every other node, thereby solving the single node gather problem. Note also that the total exchange problem may be viewed as a multinode version of both a single node scatter and a single node gather problem, and also as a generalization of a multinode broadcast, whereby the packets sent by each node to different nodes are different. We conclude that the communication problems of the preceding discussion form a hierarchy in terms of difficulty, as illustrated in Fig. 3. An algorithm solving one problem in the hierarchy can also solve the next problem in the hierarchy in no additional time. In particular, a total exchange algorithm can also solve the multinode broadcast (accumulation) problem; a multinode broadcast (accumulation) algorithm can also solve the single node gather (scatter) problem; and a single node scatter (gather) algorithm can also solve the single node broadcast (accumulation) problem. Therefore, the communication requirements for these problems decrease in the order just given, regardless of the network being used.

Our results are summarized in Tables 1 and 2. Table 1 gives the number of time units required to solve the communication problems described earlier on a d-cube. We show that each of these numbers is a lower bound on the number of time units taken by any algorithm that solves the corresponding problem, and we describe an algorithm that attains the lower bound. Similarly, Table 2 gives the number of packet transmissions required to solve the

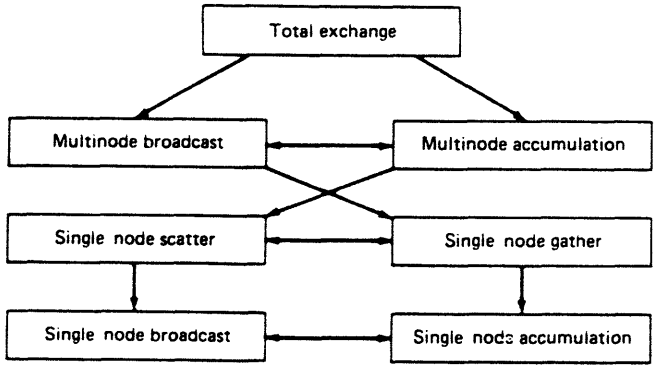

Figure 3: Hierarchy of communication problems. A directed arc from problem A to problem B indicates that an algorithm that solves A can also solve B, and that the optimal time for solving A is not more than the optimal time for solving B. A horizontal bidirectional arc indicates a duality relation.

corresponding communication problems. These numbers are lower bounds on the number of packet transmissions taken by any algorithms that solve the corresponding problems, and the lower bounds are attained by the same algorithms that attain the corresponding lower bounds of Table 1. We have thus obtained algorithms that are simultaneously optimal in terms of execution time and number of packet transmissions for our communication problems.

Problem	**Number of Transmissions (MLA and SLA)**
Single node broadcast (or single node accumulation)	$2^d - 1$
Single node scatter (or single node gather)	$d2^{d-1}$
Multinode broadcast (or multinode accumulation)	$2^d(2^d - 1)$
Total exchange	$d2^{2d-1}$

Table 2: Optimal number of packet transmissions for solving the basic communication problems on a d-cube.

Problem	Time (MLA)	Time (SLA)
Single node broadcast (or single node accumulation)	d	d
Single node scatter (or single node gather)	$\lceil \frac{2^d-1}{d} \rceil$	2^d-1
Multinode broadcast (or multinode accumulation)	$\lceil \frac{2^d-1}{d} \rceil$	2^d-1
Total exchange	2^{d-1}	$d2^{d-1}$

Table 1: Optimal times for solving the basic communication problems on a d-cube for the case where simultaneous transmission along all incident links of a node is allowed (the MLA assumption), and for the case where it is not (the SLA assumption). We assume that each packet requires unit time for transmission on any link.

Algorithms for the communication problems of this paper were first considered in [SaS85], which also discusses the effects of the transmission times of the packet overhead and the packet data (denoted by β and $m\tau$, respectively, in [SaS85]). The problems are named differently in [SaS85] than here. Also the hypercube links are assumed to be unidirectional in [SaS85], thus increasing the algorithm execution times by a factor of 2. Thus, to compare the results of [SaS85] with the ones of the present paper, the times of [SaS85] should be used with $\beta = 0$ and $m\tau = 1$, and should be divided by 2. A multinode broadcast algorithm under the MLA assumption is given in [SaS85], which is slightly suboptimal (by no more than d time units). This algorithm is constructed by specifying a packet transmission schedule at a single node and then properly replicating that schedule at each node, exploiting the symmetry of the d-cube. By contrast, we obtain optimal multinode broadcast algorithms starting from a suitable single node broadcast algorithm and replicating that algorithm at each node. This approach was first introduced for meshes in general in [Ozv87], where a slightly suboptimal (by no more than $d-3$ time units) multinode broadcast algorithm was given. Optimal algorithms for single node scatter and total exchange were also given in [SaS85] under the SLA assumption (even though the SLA assumption does not explicitly appear in [SaS85]). The single node scatter algorithm of [SaS85] has the additional property that no node transmits and receives a packet at the same time; more

precisely, each node, before beginning its transmissions, waits to receive all of the packets that are supposed to be transmitted through it. An algorithm similar to the total exchange algorithm of [SaS85] is also given in [McV87]. An optimal total exchange algorithm, given in reference [SaS85] under the MLA assumption, assumes also that each node has m packets to send to every other node, where m is a multiple of d; this is a different total exchange problem than the one we are considering here. Some of the results of the present paper have also appeared in the recent textbook [BeT89], but a complete set of optimal algorithms for the communication problem hierarchy of Fig. 3 (particularly for single node scatter and total exchange) is given here for the first time. Indeed, except for our own preliminary work reported in [Ozv87] and [BeT89], the strong form of optimality for communication algorithms investigated here has not been considered elsewhere. We also note that in addition to [SaS85] and [McV87], there are several other works dealing with communication problems and network architectures not discussed in the present paper; see [BhI85], [DNS81], [Joh87], [KVC88], [SaS86], [SaS87], [SaS88], [Saa86], and [Top85].

A device that is often useful in reducing the communication delay in various algorithms is to divide each packet into smaller packets that can travel independently through the network. The idea here is to parallelize communication through pipelining the smaller packets over paths with multiple links, and is inherent in proposals for virtual cut-through and wormhole routing [KeK79], [DaS87]. A little thought shows that (as long as the associated extra overhead is not excessive) the single node broadcast time can be reduced by dividing packets into smaller packets. On the other hand this is essentially impossible for the other basic communication problems under the MLA assumption; from Tables 1 and 2 it is seen that in an optimal algorithm, there is almost 100% utilization of some critical communication resource (the d links outgoing from the root in single node scatter, and all of the $d2^d$ directed network links in multinode broadcast and total exchange). Any communication algorithm for these problems that divides packets into smaller packets cannot reduce the total usage of the corresponding critical resource and therefore, cannot enjoy any pipelining advantage. By modifying the lower bound arguments given later in this paper, it can be shown that a similar conclusion holds under the SLA assumption.

2. SINGLE NODE AND MULTINODE BROADCAST

Optimal single node broadcast algorithms for the d-cube under the MLA and SLA assumptions can be generated using the spanning tree in Fig. 4. With this tree, a single node broadcast from the root node $(00\cdots0)$ to all other nodes takes d time units under the MLA assumption, because the number of links on the path from the root to any node is at most d. Under the SLA assumption, the single node broadcast using the same tree takes also d time units by giving priority to neighbors with larger identity numbers, that is, node $(00\cdots0)$ transmits the packet to $(10\cdots0)$ at time 1, to $(01\cdots0)$ at time 2, etc., and finally to $(00\cdots1)$ at time d; node $(10\cdots0)$ transmits the packet to $(110\cdots0)$ at time 2, to $(101\cdots0)$ at time 3, and so on. Note also that d is a lower bound on the required time for any single node broadcast algorithm, since any path joining nodes $(00\cdots0)$ and $(11\cdots1)$ has at least d links. Therefore, the single node broadcast algorithms we gave are optimal in terms of execution time.

Regarding the number of packet transmissions, we note that under both the MLA and the SLA assumptions, a single packet is transmitted on each link of the tree, so the number of link transmissions for our single node broadcast algorithms is $2^d - 1$. This number is also a lower bound on the number of link transmissions, since there must be at least one packet reception by each one of the $2^d - 1$ nodes. Therefore, our algorithms are optimal in terms of the number of packet transmissions.

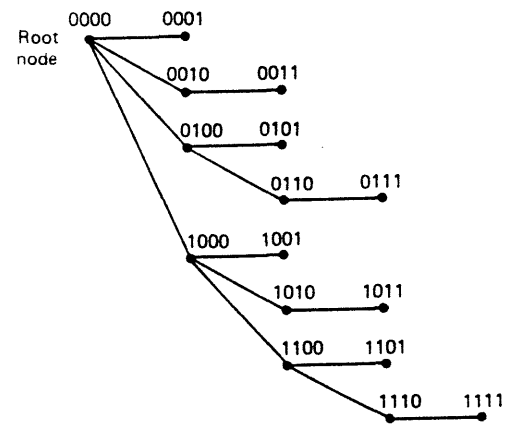

Figure 4: Spanning tree of a d-cube rooted at node $(00\cdots0)$, and used in optimal single node broadcast algorithms.

Multinode Broadcast Under the MLA Assumption

We first note that in a multinode broadcast each node must receive a total of $2^d - 1$ packets over its d incident links, so $\lceil (2^d - 1)/d \rceil$ is a lower bound for the time required by any multinode broadcast algorithm under the MLA assumption. We will obtain an algorithm that attains this lower bound.

As a first step towards constructing such an algorithm, we represent any single node broadcast algorithm from node $(00\cdots0)$ to all other nodes that takes q time units, by a sequence of sets of directed links $A_1, A_2, \ldots, A_q$. Each A_i is the set of links on which transmission of the packet begins at time $i-1$ and ends at time i. Naturally, the sets A_i must satisfy certain consistency requirements for accomplishing the single node broadcast. In particular, if S_i and E_i are the sets of identity numbers of the start nodes and end nodes, respectively, of the links in A_i, we must have $S_1 = \{(00\cdots0)\}$, and $S_i \subset \{(00\cdots0)\} \cup \left(\cup_{k=1}^{i-1} E_k\right)$. Furthermore, every nonzero node identity must belong to some E_i. The set of all nodes together with the set of links $(\cup_{i=1}^{q} A_i)$ must form a subgraph that contains a spanning tree [see Fig. 5(a)]; in fact, to minimize the number of packet transmissions, the sets of links $A_1, A_2, \ldots, A_q$ should be disjoint and should form a spanning tree.

Consider now a d-bit string t representing the identity number of some node on the d-cube. For any node identity z, we denote by $t \oplus z$ the d-bit string obtained by performing modulo 2 addition of the jth bit of t and z for each $j = 1, 2, \ldots, d$. It can be seen that an algorithm for broadcasting a packet from the node with identity t can be specified by the sets

$$A_i(t) = \{(t \oplus x, t \oplus y) \mid (x, y) \in A_i\}, \qquad i = 1, 2, \ldots, q,$$

where $A_i(t)$ denotes the set of links on which transmission of the packet begins at time $i-1$ and ends at time i. The proof of this is based on the fact that $t \oplus x$ and $t \oplus y$ differ in a particular bit if and only if x and y differ in the same bit, so $(t \oplus x, t \oplus y)$ is a link if and only if (x, y) is a link. Figure 5 illustrates the sets $A_i(t)$ corresponding to all possible t for the case where $d = 3$.

We now describe a procedure for generating a multinode broadcast algorithm specified by the sets $A_i(t)$ for all possible values of i and t, starting from a single node broadcast algorithm specified by the sets $A_1, A_2, \ldots, A_q$. Let us say that *a link (x, y) is of type j* if x and y differ on the jth bit. We make the following key observation: consider a single node broadcast algorithm specified by the link sets $A_1, \ldots, A_q$. If for each i, the links in A_i are of different types, then for each i, the sets $A_i(t)$, where t ranges over all possible identities, are disjoint. [If,

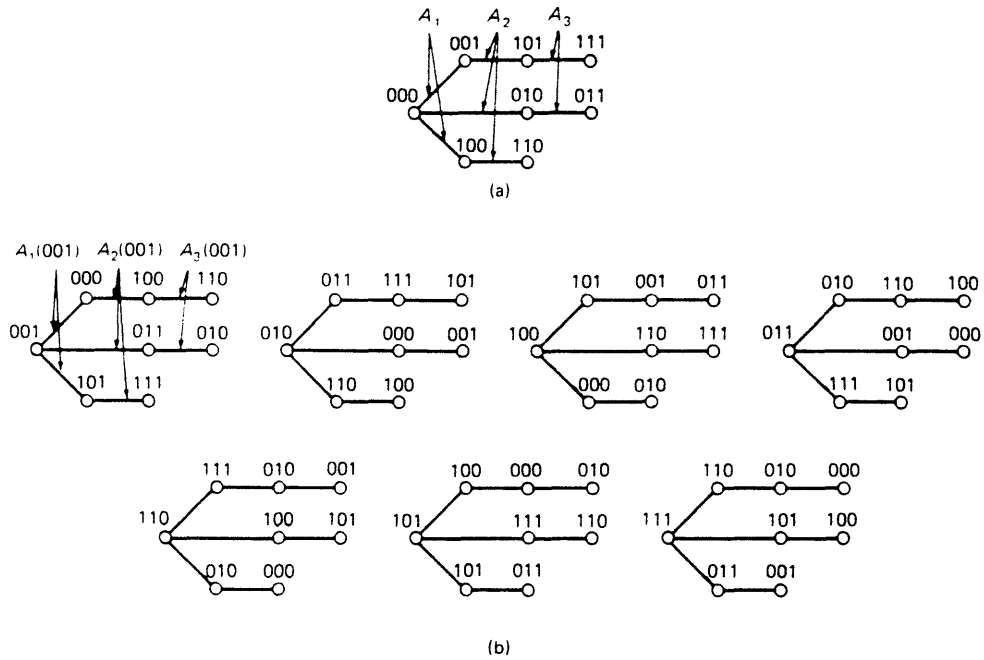

Figure 5: Generation of a multinode broadcast algorithm for the d-cube, starting from a single node broadcast algorithm. (a) The algorithm that broadcasts a packet from the node with identity $(00\cdots0)$ to all other nodes is specified by a sequence of sets of directed links $A_1, A_2, \ldots, A_q$. Each A_i is the set of links on which transmission begins at time $i-1$ and ends at time i. (b) A corresponding broadcast algorithm for each root node identity t is specified by the sets of links

$$A_i(t) = \{(t \oplus x, t \oplus y) \mid (x,y) \in A_i\},$$

where we denote by $t \oplus z$ the d-bit string obtained by performing modulo 2 addition of the jth bit of t and z for $j = 1, 2, \ldots, d$. The multinode broadcast algorithm is specified by the requirement that transmission of the packet of node t starts on each link in $A_i(t)$ at time $i-1$. The figure shows the construction for an example where $d = 3$. Here the set A_2 has two links of the same type and the multinode broadcast cannot be executed in 3 time units. However, if the link $(000, 010)$ belonged to A_1 instead of A_2, the required time would be the optimal 3 time units.

for $t \neq t'$, two links $(t \oplus x, t \oplus y) \in A_i(t)$ and $(t' \oplus x', t' \oplus y') \in A_i(t')$ were the same, then the links (x,y) and (x',y') would be different (since $t \neq t'$), and they would be of the same type because (x,y) and (x',y') are of the same type as $(t \oplus x, t \oplus y)$ and $(t' \oplus x', t' \oplus y')$, respectively, which contradicts the fact that (x,y) and (x',y') belong to A_i.] This implies that the single node broadcasts of all nodes t can be executed simultaneously, without any further delay. In particular, we have a multinode broadcast algorithm that takes q time units. We proceed to give a method for selecting the sets A_i with the links in each A_i being of different types. Furthermore, we will ensure that each one of the sets $A_1, \ldots, A_{q-1}$ has exactly d elements, which is the maximum possible (since there exist only d link types), thereby resulting in the minimum possible execution time of $q = \lceil (2^d - 1)/d \rceil$ units.

Let N_k, $k = 0, 1, \ldots, d$, be the set of node identities having k unity bits and $d-k$ zero bits. The number of elements in N_k is $\binom{d}{k} = d!/(k!\,(d-k)!)$. In particular, N_0 and N_d contain

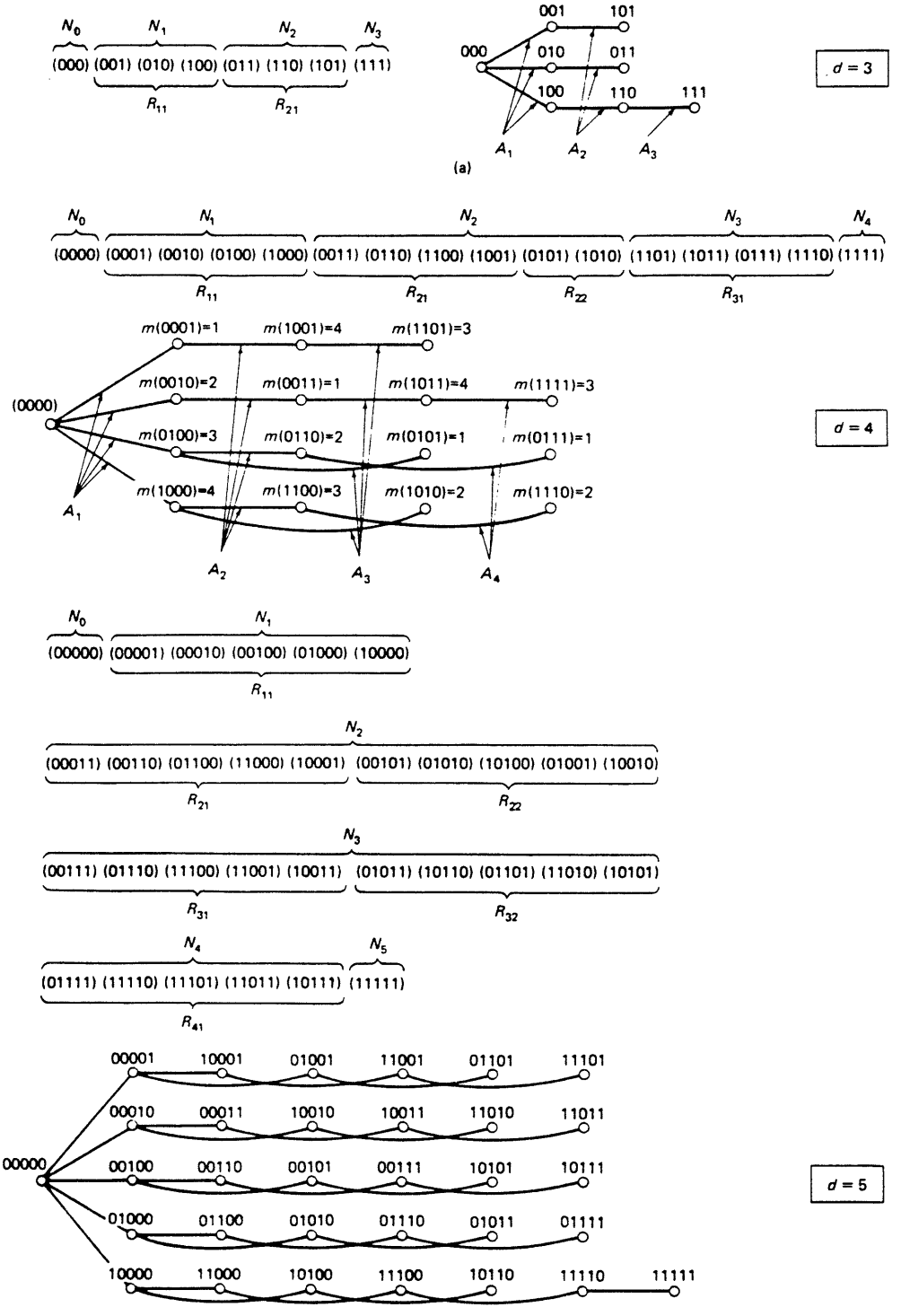

Figure 6: Construction of a multinode broadcast algorithm for a d-cube that takes $\lceil (2^d - 1)/d \rceil$ time.

one element, the strings $(00\cdots0)$ and $(11\cdots1)$, respectively; the sets N_1 and N_{d-1} contain d elements; and for $2 \leq k \leq d-2$ and $d \geq 5$, N_k contains at least $2d$ elements (when $d = 4$, the number of elements of N_2 is 6, as shown in Fig. 6). We partition each set N_k, $k = 1, \ldots, d-1$, into disjoint subsets $R_{k1}, \ldots, R_{kn_k}$ which are equivalence classes under a single bit rotation to the left, and we select R_{k1} to be the equivalence class of the element whose k rightmost bits are unity. Then, we associate each node identity t with a distinct number $n(t) \in \{0, 1, 2, \ldots, 2^d - 1\}$ in the order

$$(00\cdots0)R_{11}R_{21}\cdots R_{2n_2}\cdots R_{k1}\cdots R_{kn_k}\cdots R_{(d-2)1}\cdots R_{(d-2)n_{d-2}}R_{(d-1)1}(11\cdots1)$$

[i.e., $n(00\cdots0) = 0$, $n(11\cdots1) = 2^d - 1$, and the other node identities are numbered consecutively in the above order between 1 and $2^d - 2$]. Let

$$m(t) = 1 + [(n(t) - 1)(\text{mod d})].$$

Thus, the sequence of numbers $m(t)$ corresponding to the sequence of node identities

$$R_{11}R_{21}R_{22}\ldots R_{(d-1)1}$$

is $1, 2, \ldots, d, 1, 2, \ldots, d, 1, 2, \ldots$ (cf. Fig. 6 for the case $d = 4$). We specify the order of node identities within each set R_{kn} as follows: the first element t in each set R_{kn} is chosen so that the relation

$$\text{the bit in position } m(t) \text{ from the right is a one} \tag{1}$$

is satisfied, and the subsequent elements in R_{kn} are chosen so that each element is obtained by a single bit rotation to the left of the preceding element. Also, for the elements t of R_{k1}, we require that the bit in position $m(t) - 1$ [if $m(t) > 1$] or d [if $m(t) = 1$] from the right must be a zero. For $i = 1, 2, \ldots, \lceil (2^d - 1)/d \rceil - 1$, define

$$E_i = \{t \mid (i-1)d + 1 \leq n(t) \leq id\},$$

and for $i = 0$, and $i = q = \lceil (2^d - 1)/d \rceil$, define

$$E_0 = \{(00\cdots0)\}, \qquad E_q = \{t \mid (q-1)d + 1 \leq n(t) \leq 2^d - 1\}.$$

We define the set of links A_i as follows:

For $i = 1, 2, \ldots, q$, each set A_i consists of the links that connect the node identities $t \in E_i$ with the corresponding node identities of $\cup_{k=0}^{i-1} E_k$ obtained from t by reversing the bit in

position $m(t)$ [which is always a one by property (1)]. In particular, the node identities in each set in R_{k1} are connected with corresponding node identities in $R_{(k-1)1}$, because, by construction, the bit in position $m(t)$ lies next to a zero for each node identity t in the set R_{k1}.

To show that this definition of the sets A_i is legitimate, we need to verify that by reversing the specified bit of a node identity $t \in E_i$, we indeed obtain a node identity t' that belongs to $\cup_{k=1}^{i-1} E_k$, as opposed to E_i. [It cannot belong to E_k for $k > i$, because $n(t') < n(t)$.]

To see this in the exceptional case where $t = (11\cdots1)$, note that by the preceding rule, t' is the element of $R_{(d-1)1}$ with a zero bit in position $m(t)$ from the right. The elements of $R_{(d-1)1}$ are ordered so that the bit of t' in position $m(t') - 1$ is a zero (if $m(t') > 1$) or in position d (if $m(t') = 1$). Since $2^d - 1$ is not divisible by d (see the Appendix), we have $m(t) \neq d$. Thus, the zero bit of t' cannot be in position d, so it must be in position $m(t') - 1$, implying that $m(t) = m(t') - 1$. The set $R_{(d-1)1}$ has d elements and as a result, its first element t'' satisfies $m(t'') = m(t)$, so t' must be the second element of $R_{(d-1)1}$. Since $m(t) \neq d$, E_q has at most $d - 1$ elements, and thus, we obtain the desired conclusion $t' \notin E_q$.

In the case where $t \neq (11\cdots1)$, it is sufficient to show that $n(t) - n(t') \geq d$. We consider two cases: a) If $t \in R_{kn}$ for some $n > 1$, then all of the d elements of R_{k1} are between t' and t, and the inequality $n(t) - n(t') \geq d$ follows. b) If $t \in R_{k1}$, then $t' \in R_{(k-1)1}$, and all the elements of the sets $R_{(k-1)2}, \ldots, R_{(k-1)n_{k-1}}$ are between t' and t. There are $\binom{d}{k-1} - d$ such elements. If $2 < k < d$ and $d \geq 5$, it can be verified that $\binom{d}{k-1} - d \geq d$ and we are done. The cases $d = 3$ and $d = 4$ can be handled individually (see Fig. 6). The cases $k = 1, 2$ create no difficulties because $R_{11} = E_1$, $R_{21} = E_2$.

We have thus shown that the sets A_i are properly defined, and we also note that any two links in each set A_i are of different types, implying that the corresponding multinode broadcast algorithm takes $q = \lceil (2^d - 1)/d \rceil$ time units. Thus, the algorithm attains the lower bound of execution time over all multinode broadcast algorithms under the MLA assumption and is optimal.

Multinode Broadcast Under the SLA Assumption

We can construct a multinode broadcast algorithm under the SLA assumption by using the optimal algorithm just given for the case of the MLA assumption. Instead of transmitting the packet of node t over all the links of the set $A_i(t)$ simultaneously, we transmit, for each

t, this packet on one link of $A_i(t)$ at a time, taking at most d time units. In particular, if (x, y) is a link in a set A_i and $n(y)$ is the number associated with the node identity y as in the preceding algorithmic construction, then at time $n(y)$, we transmit the packet of node t on link $(t \oplus x, t \oplus y)$. Note that at each time unit, each node will transmit one packet and will receive one packet. The time required for the algorithm is $2^d - 1$ time units. This is also a lower bound for the required time, since in a multinode broadcast, each node receives a total of $2^d - 1$ packets and can receive at most one in each time unit under the SLA assumption.

A simpler alternative possibility for an optimal multinode broadcast algorithm under the SLA assumption is to embed a ring on the d-cube (see [BeT89]) and to require each node to transmit at each time unit, to its clockwise neighbor on the ring the packet it received at the previous time unit from its counterclockwise neighbor. This algorithm starts with each node transmitting its packet to its clockwise neighbor and terminates after $2^d - 1$ time units.

Number of Packet Transmissions

Each of the multinode broadcast algorithms given requires $2^d(2^d - 1)$ packet transmissions. This is also a lower bound on the required number, since each of the 2^d nodes must receive a total of $2^d - 1$ different packets (one from each of the other nodes). Therefore the algorithms are optimal in terms of total required communication resource.

3. SINGLE NODE SCATTER AND GATHER IN THE HYPERCUBE

Consider the d-cube and the problem of single node scatter and gather with root node s. Since $2^d - 1$ different packets must be transmitted (in the case of scatter) or received (in the case of gather) by the root node over its d incident links, any algorithm solving these problems requires at least $\lceil (2^d - 1)/d \rceil$ time units under the MLA assumption and $2^d - 1$ time units under the SLA assumption. These times can be achieved by using the corresponding optimal multinode broadcast algorithms of the previous section, thereby justifying the entries of Table 1 for single node scatter and gather.

The multinode broadcast algorithms are not, however, optimal for the scatter and gather problems with respect to number of packet transmissions. To see this, note that a packet destined for some node must travel a number of links at least equal to the Hamming distance

between that node and the root. Therefore, a lower bound for the optimal number of packet transmissions is the sum of the Hamming distances of all nodes to the root. There are $\binom{d}{k} = d!/(k!\,(d-k)!)$ nodes that are at distance k from the root, so this bound is

$$\sum_{k=1}^{d} k\binom{d}{k} = d2^{d-1}. \tag{2}$$

The lower bound of Eq. (2) is much smaller than the $2^d(2^d-1)$ packet transmissions required by a multinode broadcast.

It is possible to extract from an optimal multinode broadcast algorithm a single node gather algorithm which attains the lower bound of Eq. (2) as follows: for each node t, consider the single node broadcast tree of the previous section, consisting of the links $A_i(t)$, and let $p(t,s)$ be the path on this tree leading from t to the root s. A single node gather algorithm is obtained by executing the multinode broadcast algorithm of the previous section with the modification that the packet of each node t is transmitted only on the path $p(t,s)$. Since the number of links on $p(t,s)$ is equal to the Hamming distance of t and s, it follows that the total number of packet transmissions in this algorithm is equal to the lower bound of Eq. (2), thereby justifying the corresponding entry in Table 2. This algorithm still requires $\lceil (2^d-1)/d \rceil$ time units, so it is also optimal in terms of execution time. On the other hand, this algorithm is somewhat complex to visualize and implement. For this reason, we present an alternative algorithm that requires the construction of only one tree.

Single Node Scatter Under the MLA Assumption

For any spanning tree T of the d-cube, let r be the number of neighbor nodes of the root node s in T, and let T_i be the subtree of nodes that are connected with s via a path that lies in T and passes through the ith neighbor of s. Consider the following rule for s to send packets to each subtree T_i:

> Continuously send packets to distinct nodes in the subtree (using only links in T), giving priority to nodes furthest away from s (break ties arbitrarily).

With this rule, s starts transmitting its last packet to the subtree T_i no later than time $N_i - 1$, where N_i denotes the number of nodes in T_i, and all nodes in T_i receive their packet no later than time N_i. [To see the latter, note that all packets destined for the nodes in T_i that are k links away from s are sent no later than time $N_i - k$, and each of these packets completes its journey

in exactly k time units.] Therefore, all packets are received at their respective destinations in $\max\{N_1, N_2, \ldots, N_r\}$ time units. Hence, the above algorithm attains the optimal time if and only if T has the property that s has d neighbors in T and that each subtree T_i, $i = 1, \ldots, d$, contains at most $\lceil (2^d - 1)/d \rceil$ nodes. If T is in addition a shortest path tree from s, then each packet travels along the shortest path to its destination and this algorithm also attains the optimal number of packet transmissions.

We will assume without loss of generality that $s = (00\cdots0)$ in what follows. To construct a spanning tree T with the above two properties, let us consider the equivalence classes R_{kn} introduced in Section 2 in connection with the multinode broadcast problem. As in Section 2, we order the classes as

$$(00\cdots0)R_{11}R_{21}\cdots R_{2n_2}\cdots R_{k1}\cdots R_{kn_k}\cdots R_{(d-2)1}\cdots R_{(d-2)n_{d-2}}R_{(d-1)1}(11\cdots1)$$

and we consider the numbers $n(t)$ and $m(t)$ for each identity t, but for the moment, we leave the choice of the first element in each class R_{kn} unspecified. We denote by m_{kn} the number $m(t)$ of the first element t of R_{kn} and we note that this number depends only on R_{kn} and not on the choice of the first element within R_{kn}.

We say that class $R_{(k-1)n'}$ is *compatible* with class R_{kn} if $R_{(k-1)n'}$ has d elements (node identities) and there exist identities $t' \in R_{(k-1)n'}$ and $t \in R_{kn}$ such that t' is obtained from t by changing some unity bit of t to a zero. Since the elements of $R_{(k-1)n'}$ and R_{kn} are obtained by left shifting the bits of t' and t, respectively, it is seen that for every element x' of $R_{(k-1)n'}$ there is an element x of R_{kn} such that x' is obtained from x by changing one of its unity bits to a zero. The reverse is also true, namely that for every element x of R_{kn} there is an element x' of $R_{(k-1)n'}$ such that x is obtained from x' by changing one of its zero bits to unity.

An important fact for the subsequent spanning tree construction is that for every class R_{kn} with $2 \leq k \leq d-1$, there exists a compatible class $R_{(k-1)n'}$. Such a class can be obtained as follows: take any identity $t \in R_{kn}$ whose rightmost bit is a one and its leftmost bit is a zero. Let σ be a string of consecutive zeros with maximal number of bits and let t' be the identity obtained from t by changing to zero the unity bit immediately to the right of σ. [For example, if $t = (0010011)$, then $t' = (0010001)$ or $t' = (0000011)$, and if $t = (0010001)$, then $t' = (0010000)$.] Then the equivalence class of t' is compatible with R_{kn}, because it has d elements [$t' \neq (00\cdots0)$ and t' contains a unique substring of consecutive zeros with maximal number of bits, so it cannot be replicated by left rotation of less than d bits].

The spanning tree T with the desired properties is constructed sequentially by adding links

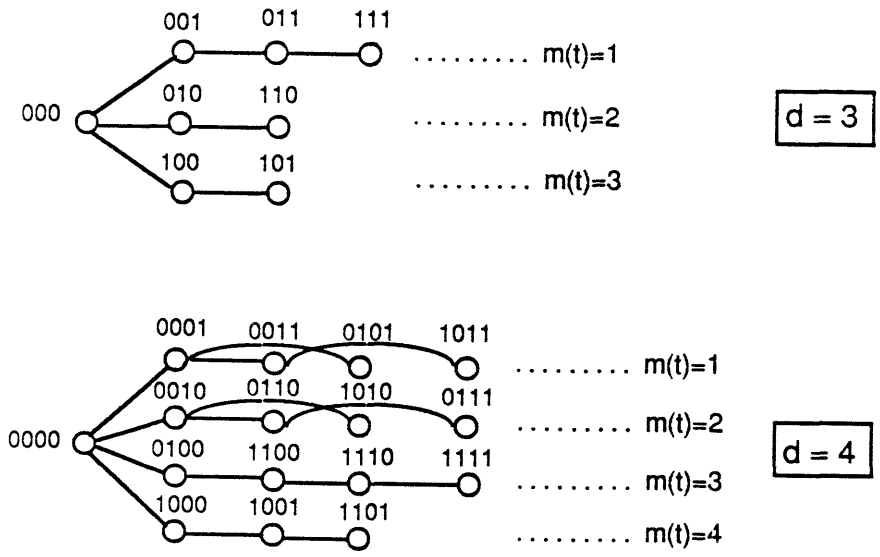

Figure 7: Spanning tree construction for optimal single node scatter under the MLA assumption for $d = 3$ and $d = 4$.

incident to elements of the classes R_{kn} as follows (see Fig. 7):

Initially T contains no links. We choose arbitrarily the first element of class R_{11} and we add to T the links connecting $(00\cdots0)$ with all the elements of R_{11}. We then consider the classes R_{kn} $(2 \leq k \leq d-1)$ one-by-one in the order indicated above, and for each R_{kn}, we find a compatible class $R_{(k-1)n'}$ and the element t' in $R_{(k-1)n'}$ such that $m(t') = m_{kn}$ (this is possible because $R_{(k-1)n'}$ has d elements). We then choose as the first element of R_{kn} an element t such that t' is obtained from t by changing one of its unity bits to a zero. Since $R_{(k-1)n'}$ has d elements and R_{kn} has at most d elements, it can be seen that, for any x in R_{kn}, we have $m(x') = m(x)$, where x' is the element of $R_{(k-1)n'}$ obtained by shifting t' to the left by the same amount as needed to obtain x by shifting t to the left. Moreover, x' can be obtained from x by changing some unity bit of x to a zero. We add to T the links (x', x), for all $x \in R_{kn}$ (with x' defined as above for each x). After exhausting the classes R_{kn}, $2 \leq k \leq d-1$, we finally add to T the link $\big(x, (11\cdots1)\big)$, where x is the element of $R_{(d-1)1}$ with $m(x) = m(11\cdots1)$.

The construction of T is such that each node $x \neq (00\cdots0)$ is in the subtree $T_{m(x)}$. Since there are at most $\lceil (2^d - 1)/d \rceil$ nodes x having the same value of $m(x)$, each subtree contains

at most $\lceil (2^d - 1)/d \rceil$ nodes. Furthermore, the number of links on the path of T connecting any node and $(00\cdots0)$ is the corresponding Hamming distance. Hence, T is also a shortest path tree from $(00\cdots0)$, as desired.

Single Node Scatter Under the SLA Assumption

To construct an algorithm attaining the lower bound of $2^d - 1$ on the single node scatter time under the SLA assumption, consider sending continuously packets from s along any spanning tree T, giving priority to the packets destined for nodes that are furthest away from s (break ties arbitrarily). It can be seen that all packets arrive at their respective destination at time $2^d - 1$ or earlier. [The argument for this is analogous to the one given earlier in this section. The only difference here is that s can send a packet to only one (instead of all) of the subtrees per time unit. Hence the time of the algorithm is the sum of the N_i's (instead of the maximum of the N_i's).] Furthermore, if T is chosen to be a tree of shortest paths from s, then each packet travels along the shortest path to its destination and this algorithm also attains the optimal number of packet transmissions.

4. TOTAL EXCHANGE IN THE HYPERCUBE

Consider the total exchange problem under the MLA assumption. We showed in the preceding section that for any single node scatter algorithm in the d-cube, the number of packet transmissions is bounded below by $d2^{d-1}$, and it is equal to $d2^{d-1}$ if and only if packets follow shortest paths from the root to all other nodes. Since a total exchange can be viewed as 2^d separate versions of single node scatter, a lower bound for the total number of transmissions is

$$d2^d 2^{d-1}. \tag{3}$$

Since each node has d incident links, at most $d2^d$ transmissions may take place at each time unit. Therefore, if T_d is the execution time of a total exchange algorithm in the d-cube, we have

$$T_d \geq 2^{d-1}. \tag{4}$$

For an algorithm to achieve this lower bound, it is necessary that packets follow shortest paths and that all links are busy (in both directions) during all of the 2^{d-1} time units. In what

follows, we present an algorithm for which $T_d = 2^{d-1}$. In light of the above, this algorithm is optimal with respect to both the time and the number of packet transmissions criteria, and achieves 100% link utilization.

We will construct the algorithm recursively. We will assume that we have an optimal algorithm for total exchange in the d-cube with certain properties to be stated shortly, and we will use this algorithm in order to perform an optimal total exchange in the $(d+1)$-cube. The construction is as follows: we decompose the $(d+1)$-cube into two d-cubes, denoted as C_1 and C_2 (cf. the construction of Fig. 1). Without loss of generality we assume that C_1 contains nodes $0, \ldots, 2^d - 1$, and that their counterparts in C_2 are nodes $2^d, \ldots, 2^{d+1} - 1$, respectively. The total exchange algorithm for the $(d+1)$-cube consists of three phases. In the first phase, there is a total exchange (using the optimal algorithm for the d-cube) within each of the cubes C_1 and C_2 (each node in C_1 and C_2 exchanges its packets with the other nodes in C_1 and C_2, respectively). In the second phase, each node transmits to its counterpart node in the opposite d-cube all of the 2^d packets that are destined for the nodes of the opposite d-cube. In the third phase, there is an optimal total exchange in each of the two d-cubes of the packets received in phase two (see Fig. 8). Phase three must be carried out after phase one, because during phase one all the links of the cubes C_1 and C_2 are continuously busy (since the d-cube total exchange algorithm is assumed optimal). On the other hand, phase two may take place simultaneously with both phases one and three. In an algorithm presented in [BeT89], phase three starts after the end of phase two, resulting in an execution time of $2^d - 1$ units. Here, we improve on this time by allowing phase three to start before phase two ends. To illustrate how this is possible, consider the packet originating at some node $i \in C_1$ and destined for its counterpart node in C_2, namely $i + 2^d$, and the packet originating at $i + 2^d$ and destined for i. These packets are not transmitted at all during phase three. Therefore, if they are transmitted last in phase two, then phase three can start one time unit before the end of phase two. This idea can be generalized as follows: clearly, if it were guaranteed that packets going from C_1 to C_2 and from C_2 to C_1 arrive sufficiently early in C_2 and in C_1, respectively, then phase three may be carried out just after phase one, without completing phase two. In such a case, the first half of phase two would be carried out simultaneously with phase one, while the second half would be carried out simultaneously with phase three, and we would have $T_{d+1} = 2T_d$. Since, by assumption, T_d is equal to the lower bound 2^{d-1} of Eq. (4) for a total exchange in the d-cube, we would have $T_{d+1} = 2^d$, implying that such an algorithm would achieve the lower bound of Eq. (4) for the $(d+1)$-cube. We prove that this is indeed feasible.

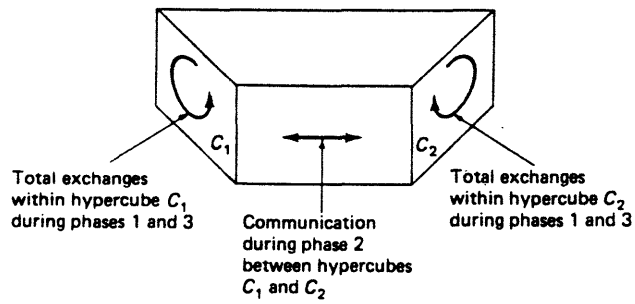

Figure 8: Recursive construction of a total exchange algorithm for the d-cube. Let the $(d+1)$-cube be decomposed into two d-cubes denoted C_1 and C_2. The algorithm consists of three phases. In the first phase, there is a total exchange within each of the cubes C_1 and C_2. In the second phase, each node transmits to its counterpart node in the opposite d-cube all of the 2^d packets that are destined for the nodes of the opposite d-cube. In the third phase, there is a total exchange in each of the two d-cubes of the packets received in phase two.

Suppose that an optimal total exchange algorithm has already been devised for the d-cube. Let $N_d(i,n)$ denote the number of its own packets that node i has transmitted up to and including time n, for $n = 1, \ldots, 2^{d-1}$, [$N_d(i,n)$ ranges from 1 to $2^d - 1$]. We can use $N_d(i,n)$ to express the requirement that phase three packets originating at nodes of C_1 are available in time at the appropriate nodes of C_2, so that phase three begins right after phase one and continues without delay. In particular, it is necessary that

$$N_d(i,n) \leq 2^{d-1} + n - 1, \qquad \forall\ n = 1, \ldots, 2^{d-1},\ i = 0, \ldots, 2^d - 1. \tag{5}$$

To see this, note that the left-hand quantity in Eq. (5) is the number of packets of node $i \in C_1$ that must be transmitted by node $i + 2^d$ during the first n time units of phase three, while the right-hand quantity in Eq. (5) is the number of available time units within phase two for transferring these packets from node i to node $i + 2^d$. There is also a requirement analogous to Eq. (5) for the nodes i of C_2.

We will proceed by induction, using the requirement of Eq. (5) as part of the inductive hypothesis. In particular, we will prove that for every d, there exists a total exchange algorithm for the d-cube, satisfying

$$T_d = 2^{d-1} \qquad \text{and} \qquad N_d(i,n) \leq 2^{d-1} + n - 1,\ \ \forall\ n = 1, \ldots, 2^{d-1},\ i = 0, \ldots, 2^d - 1. \tag{6}$$

We have $T_1 = 1$ and $N_1(i,1) = 1$, for $i = 0,1$, which proves the inductive hypothesis for $d = 1$. Assume that for some d, we have a total exchange algorithm for the d-cube that satisfies the inductive hypothesis (6), and let $s(i,j,d)$ denote the time unit in this algorithm during which node i transmits its own packet that is destined for node j. We will construct a three-phase total exchange algorithm for the $(d+1)$-cube of the type described above that satisfies the inductive hypothesis. Suppose that packets are transmitted in phase two according to the following rules: (In view of the symmetry of the transmissions of nodes of the d-cubes C_1 and C_2, we describe the rules for phase two packet transmissions only for the nodes of C_1.)

(a) Each node $i \in C_1$ transmits its packets to node $i + 2^d$ in the order in which the latter node forwards them in phase three (ties are broken arbitrarily), i.e., the packet destined for $j \in C_2$, $j \neq i + 2^d$, is transmitted before the packet destined for $j' \in C_2$, $j' \neq i + 2^d$, if

$$s(i, j - 2^d, d) < s(i, j' - 2^d, d).$$

(b) Each node $i \in C_1$ transmits its packet destined for node $i + 2^d$ last.

We claim that, under the above rules, phase three can proceed uninterrupted after phase one. To show this, consider any $i \in C_1$ (the case of $i \in C_2$ can be treated analogously). At the end of phase one, node i has received exactly 2^{d-1} packets from node $i + 2^d$ (since phase one lasts 2^{d-1} time units by induction). Hence, n time units after the end of phase one, node i has received exactly $2^{d-1} + n$ packets from node $i + 2^d$. On the other hand, the total number of packets of node $i + 2^d$ that node i forwards after $n + 1$ time units of phase three is exactly $N_d(i, n+1)$. Since [cf. Eq. (6)] $N_d(i, n+1) \leq 2^{d-1} + n$ for all $n = 0, 1, \ldots, 2^{d-1} - 1$, and node $i + 2^d$ transmits its packets to node i according to the above rules, node i always has enough packets from node $i + 2^d$ for transmission if phase three begins immediately after phase one. Since $i \in C_1$ was chosen arbitrarily, this holds for all $i \in C_1$.

Consider the total exchange algorithm for the $(d+1)$-cube whereby phase three proceeds uninterrupted immediately following phase one as described above. Since according to the inductive hypothesis, each of phases one and three takes time $T_d = 2^{d-1}$, this algorithm takes time $2T_d = 2^d$. There remains to show that the second part of the inductive hypothesis is satisfied for $d + 1$. For any node i, let $N_{d+1}(i, n)$ denote the number of node i's own packets that i has transmitted up to and including time n in this algorithm. Since in the first 2^{d-1}

time units of this algorithm, phases one and two execute simultaneously, we obtain

$$N_{d+1}(i,n) = N_d(i,n) + n, \qquad \forall\, n = 1,\ldots,2^{d-1}.$$

By combining this equation with the inequality $N_d(i,n) \leq 2^d - 1$, which holds for all n, we obtain

$$N_{d+1}(i,n) \leq 2^d + n - 1, \qquad \forall\, n = 1,\ldots,2^{d-1}.$$

Since in the next 2^{d-1} time units of this algorithm, phases three and two execute simultaneously (and i does not transmit any packet of its own in phase three), we have

$$N_{d+1}(i,n) = 2^d - 1 + n, \qquad \forall\, n = 2^{d-1}+1,\ldots,2^d.$$

By combining the last two relations, it follows that $N_{d+1}(i,n)$ satisfies

$$N_{d+1}(i,n) \leq 2^d + n - 1, \qquad \forall\, n = 1,\ldots,2^d.$$

Since the choice of i was arbitrary, this implies that the inductive hypothesis (6) holds for the $(d+1)$-cube.

Implementation of the Optimal Algorithm

In what follows, we present the rules used by the nodes of the d-cube for transmitting their own packets and forwarding the packets they receive from other nodes, whenever they require further transmission. We write the identity of node i as $(i_d,\ldots,i_1)$, where each i_k, $k = 1,\ldots,d$, is a 0 or a 1. Moreover, we denote by e_k the identity of node 2^{k-1}, for $k = 1,\ldots,d$. The link between i and $i \oplus e_k$ is called the *kth link* incident to i. Finally, $\bar{i}_k$ denotes the reverse of bit i_k, namely $\bar{i}_k = (i_k + 1) \bmod 2$.

We first describe the order in which an arbitrary node i transmits its own packets. It can be seen that during time units $1,\ldots,2^{k-1}$, node i transmits all its packets destined for nodes

$$(i_d,\ldots,i_{k+1},\bar{i}_k,x_{k-1},\ldots,x_1), \qquad \text{where } x_m = 0 \text{ or } 1, \text{ for } m = 1,\ldots,k-1,$$

through its kth incident link, for $k = 1,\ldots,d$. The last packet to be transmitted in this group is the one destined for node $i \oplus e_k$. For $i = 0$, the exact order in which i transmits its packets on each of its incident links may be derived by using a sequence of d tables, which may be constructed iteratively. The kth table consists of k columns, the mth of which contains the destinations of the packets transmitted through link m. The first table contains only e_1. For

$k = 2, \ldots, d$, the first $k-1$ columns of the kth table are identical to those of the $(k-1)$st table, whereas its last column consists of e_k and the entries of the $(k-1)$st table with their kth bit being set to 1. In the last column, entries corresponding to the same row of the $(k-1)$st table, appear one after the other, ordered (arbitrarily) from left to right; entries corresponding to different rows of the $(k-1)$st table are ordered from top to bottom. The last element of the last column is e_k. This scheme follows from the recursive construction of the algorithm. As an example, we present below the scheme for $d = 4$.

Time	**Link 1**
1	0001

Time	**Link 1**	**Link 2**
1	0001	0011
2		0010

Time	**Link 1**	**Link 2**	**Link 3**
1	0001	0011	0101
2		0010	0111
3			0110
4			0100

Time	**Link 1**	**Link 2**	**Link 3**	**Link 4**
1	0001	0011	0101	1001
2		0010	0111	1011
3			0110	1101
4			0100	1010
5				1111
6				1110
7				1100
8				1000

For any other node i, the corresponding order of destinations may be obtained by forming the $\oplus$ operation of each entry of the dth table of 0 with i.

We now consider the packets arriving at some node i and present the rules under which these packets are forwarded by i (whenever necessary). Packets arrive in i, through the kth link, in groups of 2^{k-1}, for $k = 1, \ldots, d$. Each group contains all the packets originating from the same node $(y_d, \ldots, y_{k+1}, \bar{i}_k, i_{k-1}, \ldots, i_1)$ (where $y_m = 0$ or 1, for $m = k+1, \ldots, d$) and destined for all nodes of the form $(i_d, \ldots, i_k, x_{k-1}, \ldots, x_1)$ (where $x_m = 0$ or 1, for $m = 1, \ldots, k-1$). The order of group arrivals is lexicographic on $(y_d \oplus i_d, \ldots, y_{k+1} \oplus i_{k+1})$. Routing is accomplished as follows:

A packet destined for node $(i_d, \ldots, i_k, x_{k-1}, \ldots, x_1)$ is placed in the queue which contains packets to be transmitted by i through the k_0th link, where

$$k_0 = \max_{1 \le m \le k-1} \{m \mid x_m = \bar{i}_m\}.$$

Packets originating from different nodes j, j' and placed in the same queue, are ordered according to the lexicographic order between $j \oplus i$ and $j' \oplus i$. Packets originating from the same node and placed in the same queue preserve their order of arrival. Forwarding packets in the kth link starts at time $2^{k-1} + 1$, for $k = 1, \ldots, d-1$; no forwarding takes place in the dth link.

The rules presented above follow from the recursive construction of the algorithm. Our earlier analysis guarantees that packets are always in time at the intermediate nodes (if any) of the paths they have to traverse.

Total Exchange Under the SLA Assumption

Let T_d denote the time required by some total exchange algorithm in the d-cube under the SLA assumption. Since the total number of packet transmissions is at least $d2^d2^{d-1}$ [cf. Eq. (3)], and at most 2^d transmissions may take place during each time unit, we obtain

$$T_d \ge d2^{d-1}.$$

A total exchange algorithm that requires $d2^{d-1}$ time units and satisfies the SLA assumption is presented in [SaS85]. This algorithm, named MTADEA by the authors of [SaS85], uses an optimal number of packet transmissions. It may be constructed recursively, by modifying the construction presented earlier in this section (cf. Fig. 8), in such a way that phases one, two, and three are carried out one after the other. In this case, we have $T_{d+1} = 2T_d + 2^d$. Since $T_1 = 1$, we obtain by induction $T_d = d2^{d-1}$. Moreover, assuming that the algorithm in the d-cube involves one packet transmission and one packet reception by each node per time unit, it is straightforward to verify that the algorithm for the $(d+1)$-cube also has this property.

5. CONCLUSIONS

Excessive communication time is widely recognized as the principal obstacle for achieving large speedup in many problems using massively parallel computing systems. This emphasizes the importance of optimal communication algorithms. In this paper, we have shown that a very strong form of optimality can be achieved for some basic communication problems in the hypercube architecture. However, some of our assumptions (all packets have equal transmission time and all nodes initiate the algorithm simultaneously), are fairly restrictive and it is interesting to investigate the performance of our algorithms or variations thereoff, under weakened forms of these assumptions.

Our methodological ideas may find application in other related contexts. In particular, some of our algorithmic constructions can be applied in other architectures to obtain optimal or nearly optimal algorithms for the communication problems of this paper. Furthermore, it is worth considering the potential existence of optimal algorithms for specialized communication tasks, arising in the context of specific numerical and other methods.

REFERENCES

[BeT88] Bertsekas, D. P., and Tsitsiklis, J. N., "Parallel and Distributed Computation: Numerical Methods", Prentice-Hall, Englewood Cliffs, N.J., 1989.

[BhI85] Bhatt, S. N., and Ipsen, I. C. F., "How to Embed Trees in Hypercubes", Yale University, Dept. of Computer Science, Research Report YALEU/DCS/RR-443, 1985.

[DaS87] Dally, W. J., and Seitz, C. L., "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", IEEE Trans. on Computers, Vol. C-36, pp. 547-553, 1987.

[DNS81] Dekel, E., Nassimi, D., and Sahni, S., "Parallel Matrix and Graph Algorithms", SIAM J. Comput., Vol. 10, pp. 657-673, 1981.

[Joh85a] Johnsson, S. L., "Cyclic Reduction on a Binary Tree", Computer Physics Communications, Vol. 37, 1985, pp. 195-203.

[KVC88] Krumme, D. W, Venkataraman, K. N., and Cybenko, G., "The Token Exchange Problem", Tufts University, Technical Report 88-2, 1988.

[KeK79] Kermani, P., and Kleinrock, L., "Virtual Cut-Through: A New Computer Communicating Switching Technique", Comput. Networks, Vol. 3, pp. 267-286, 1979.

[Ozv87] Ozveren, C., "Communication Aspects of Parallel Processing", Laboratory for Information and Decision Systems Report LIDS-P-1721, M.I.T., Cambridge, MA, 1987.

[SaS85] Saad, Y., and Schultz, M. H., "Data Communication in Hypercubes", Yale University Research Report YALEU/DCS/RR-428, October 1985 (revision of August 1987).

[SaS86] Saad Y., and Schultz, M. H., "Data Communication in Parallel Architectures", Yale University Report, March 1986.

[SaS87] Saad Y., and Schultz, M. H., "Parallel Direct Methods for Solving Banded Linear Systems", Linear Algebra and its Applications, 88/89, pp. 623-650, 1987.

[SaS88] Saad Y., and Schultz, M. H., "Topological Properties of Hypercubes", IEEE Trans. on Computers, Vol. 37, 1988, pp. 867-872.

[Saa86] Saad, Y., "Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors", Linear Algebra and its Applications, Vol. 77, pp. 315-340,1986.

[Top85] Topkis, D. M., "Concurrent Broadcast for Information Dissemination", IEEE Trans. Software Engineering, Vol. 13, pp. 207-231, 1983.

APPENDIX: INDIVISIBILITY OF $2^d - 1$ BY d

The proof of the following result is due to David Gillman and Arthur Mattuck of the M.I.T. Department of Mathematics, and is given here with their kind permission.

Proposition : For any integer $d \geq 2$, $2^d - 1$ is not divisible by d.

Proof: For any positive integers a, b, and d we use the notation $a \equiv b \pmod{d}$ to indicate that a and b give the same remainder when divided by d; this remainder is denoted $a \bmod d$ or $b \bmod d$. We note that for all positive integers a, b, d, and t we have

$$a \equiv b \pmod{d} \qquad \Rightarrow \qquad ta \equiv tb \pmod{d}. \tag{A.1}$$

(Write $a = p_a d + w$, $b = p_b d + w$, $tw = pd + r$, where $w = a \bmod d = b \bmod d$ and $r = (tw) \bmod d$, and note that $r = (ta) \bmod d = (tb) \bmod d$.)

It suffices to consider odd $d \geq 2$. We will argue by contradiction. If the claim does not hold, let d be the smallest odd integer, which is larger than one and is such that

$$2^d \equiv 1 \pmod{d}. \tag{A.2}$$

Let m be the smallest positive integer for which

$$2^m \equiv 1 \pmod{d}. \tag{A.3}$$

We claim that $m < d$. To see this, note that the numbers

$$2 \bmod d, 2^2 \bmod d, \ldots, 2^d \bmod d$$

belong to $\{1, \ldots, d-1\}$. Since there are d such numbers, some of them must repeat, i.e., for some integers r and s with $1 \leq r < s \leq d$,

$$2^r \equiv 2^s \pmod{d}.$$

Using Eq. (A.1) with $a = 2^r$, $b = 2^s$, and $t = 2^{d-s}$, and using also Eq. (A.2), we obtain

$$2^{d-s+r} \equiv 2^d \equiv 1 \pmod{d}.$$

Since m is the smallest positive integer for which Eq. (A.3) holds, we obtain $m \leq d - s + r$, so finally $m < d$.

Let us now express d as $d = pm + r$, where $r = d \bmod m$. By multiplying with 2^m in Eq. (A.3) and by using Eq. (A.1), we obtain

$$2^{2m} \equiv 2^m \equiv 1 \pmod{d}.$$

By multiplying again with 2^m and by using Eq. (A.3) and (A.1), we have

$$2^{3m} \equiv 2^m \equiv 1 \pmod{d},$$

and by continuing similarly,

$$2^{pm} \equiv 1 \pmod{d}.$$

By multiplying with 2^r in this equation and by using again Eq. (A.1) together with Eq. (A.2), we obtain

$$2^r \equiv 2^{pm+r} \equiv 2^d \equiv 1 \pmod{d}.$$

Since $r < m$, our hypothesis on m implies that $r = 0$. Thus, d is divisible by m. This implies that m is odd (since d is odd) and that $2^m \equiv 1 \pmod{m}$ (since Eq. (A.3) holds). In view of the definition of d and the fact $d > m$, we must have $m = 1$, which contradicts Eq. (A.3). **Q.E.D.**