BROADCASTING AND SCATTERING IN CUBE-CONNECTED CYCLES AND BUTTERFLY NETWORKS

by

Radoslav Nickolov

B.Sc., Technical University of Sofia, 1991

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

August, 1993

APPROVAL

Name: Radoslav Nickolov

Degree: Master of Science

Title of Thesis: Broadcasting and Scattering in
Cube-Connected Cycles and Butterfly Networks

Examining Committee:

Chairman: Dr.Z.N.Li

Dr.J.G.Peters, Senior Supervisor

Dr.A.L.Liestman

Dr.A.Proskurowski, External Examiner

Date Approved: 10/8/93

Title of Thesis/Project/Extended Essay

Broadcasting and Scattering in Cube-Connected Cycles and Butterfly Networks

Author: _____
(signature)

Radolsav Nickolov
(name)

11. Aug. 1993
(date)

ABSTRACT

The process of sending a message from one node of a communication network to all other nodes is called *broadcasting* when the message is the same for all nodes and *scattering* when each of the nodes receives a different message. In this thesis we prove several upper bounds on the time to broadcast and scatter in the Cube-Connected Cycles (CCCd) and Butterfly (BFd) interconnection networks. We use the *linear cost model* of communication, in which the time to send a single message from one node to its neighbour is the sum of the time to establish the connection and the time to send the data proportional to the length of the message. Our algorithms use *pipelining* in parallel along several *disjoint (spanning) trees.* We show how to construct 2 arc-disjoint spanning trees of depths $2d+\lfloor d/2 \rfloor+2$ and 3 arc-disjoint spanning trees of depths $3d+3$ in CCCd, and 2 and 4 arc-disjoint spanning trees in BFd, of depths $d+\lfloor d/2 \rfloor+1$ and $2d+1$ respectively, and compare the broadcasting times for different lengths of the broadcasted message. Our scattering algorithms consist of two phases. During the first phase we scatter along perfectly balanced *binary subtrees* of CCCd and BFd. In the second phase we scatter in parallel in all cycles of CCCd and BFd using several originators. The times to scatter are close to the existing lower bounds for both graphs. Algorithms are presented for *full-duplex* and *half-duplex* links and *processor-bound* and *link-bound* communication.

To my family.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my senior supervisor, Dr.Joseph Peters, for his help, for his encouragement, and for his valuable advice on the drafts of my thesis.

I would also like to thank Dr.Pavol Hell for his assistance during the first stages of my work on the thesis.

My special thanks to my fiancee, Karen, who was always there for me.

TABLE OF CONTENTS

APPROVAL ii

ABSTRACT iii

ACKNOWLEDGMENTS v

TABLE OF CONTENTS vi

INTRODUCTION 1

Communication Networks 1

Communication patterns 5

Routing strategies 6

Communication models 6

CHAPTER 1

LINEAR MODEL OF COMMUNICATION 10

1.1 Definition of the model and general lower bounds 10

1.2 Pipelining and disjoint spanning trees 12

1.2.1 Pipelining 12

1.2.2 Disjoint spanning trees 18

1.3 General upper bounds 19

CHAPTER 2

BINARY SUBTREES OF CCCd AND BFd 22

CHAPTER 3

BROADCASTING IN CCCd AND BFd 36

3.1 Three Arc-Disjoint Spanning Trees in the Cube-Connected Cycles graph 36

3.2 Four Arc-Disjoint Spanning Trees in the Butterfly graph 51

3.2.1 Phase 1. 52
3.2.2 Phase 2 57
3.3 Two Arc-Disjoint Spanning Trees in the Butterfly graph 61
3.4 Two Arc-Disjoint Spanning Trees in the Cube-Connected Cycles graph 64
3.5 Upper bounds on broadcasting in CCCd and BFd under the various models ... 66
3.5.1 Full-duplex links 66
3.5.2 Half-duplex links 67
3.5.3 Comparison between the bounds obtained using different number of ADST 69

CHAPTER 4
SCATTERING IN CCCd AND BFd 72

4.1 Cube-Connected Cycles 72
4.1.1 Phase 1 72
4.1.2 Phase 2 83
4.2 Butterflies 91
4.2.1 Phase 1 91
4.2.2 Phase 2 94
4.3 Upper bounds on scattering in CCCd and BFd under the various models 97

CONCLUSION 98

BIBLIOGRAPHY 100

INTRODUCTION

Communication Networks

Parallel computer architectures have gained increasing popularity over the past decade. The idea of distributing the algorithms to work simultaneously on several processor units is extremely attractive; however, there are still many problems to be solved at both hardware and algorithmic levels. Among the fundamental issues is the problem of communication between the processors within the parallel architecture.

It is easy to see that, from a communication point of view, the best solution would be if all processors within the *communication network* (we will stick to this more general term instead of parallel computer architecture) are directly connected. Thus messages would be passed directly between any two processors (nodes), and no special routing schemes would be necessary. Unfortunately, this approach requires a quadratic number of links -- $(N^2 - N)/2$, where N is the number of processors in the network. Another problem is that each processor would need N-1 links (to all other nodes of the communication network), and so far it is difficult to implement processor units with large numbers of I/O channels.

One approach to overcoming this problem is to use communication networks which provide relatively short paths between every two nodes (usually of length O(logN)) and also require a feasible number of processor links. One of the most popular networks used is the *hypercube* network [Pe77] (fig.1).

Definition:

> A *Hypercube* of dimension d, denoted by Hd, is a communication network consisting of $N = 2^d$ processors. Each processor is labelled by a binary string of length d. Two processors are connected iff their labels differ in exactly one bit.

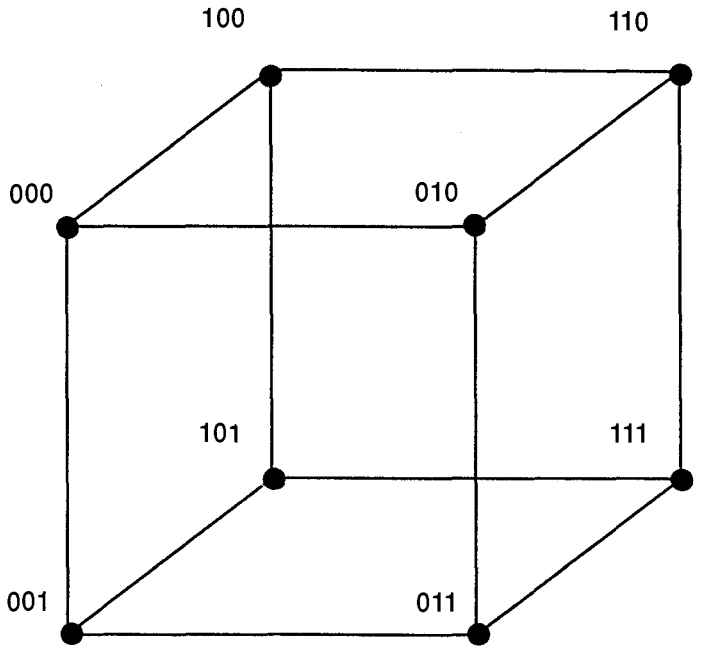


fig. 1. Hypercube H3.

It is clear from the definition above that every processor has exactly $d = \log_2 N$ links and therefore the total number of links is $d2^{d-1}$. It is also easy to see that the shortest path between any two processors is at most d. The longest shortest path within a communication network is usually called *diameter* and is denoted by D.

The simplicity and regularity of the hypercube make it very suitable for various communication patterns, some of which are discussed in the next section of this chapter. Its main disadvantage is that even though the number of processor links is only $\log_2 N$, the hypercube is still very difficult to implement for large values of N.

For example, in a hypercube with 1024 processors, each processor requires 10 links, while the units now available on the market usually have up to 4 I/O channels. For that reason, many variations of the hypercube have been proposed. They all try to reduce the number of processor links while attempting to preserve the properties of the hypercube. Usually these variations are aimed at achieving a constant number of processor links.

In this thesis we concentrate on two particular communication networks derived from the hypercube. They both have a constant number of processor links, and, in both of them, each hypercube processor is replaced by a cycle of processors.

Definition:

The *Cube-Connected Cycles* network of dimension d, denoted by CCCd, is a communication network consisting of $N = d2^d$ processors. Each processor is labeled by a pair (i, X), where $0 \le i \le d-1$ and $X = x_0x_1...x_{d-1}$ is a binary string of length d. Every processor (i, X) is connected to three other processors: (i-1 mod d, X), (i+1 mod d, X) and (i, X_i), where X_i denotes a binary label derived from X by replacing the i-th bit with its inverse.

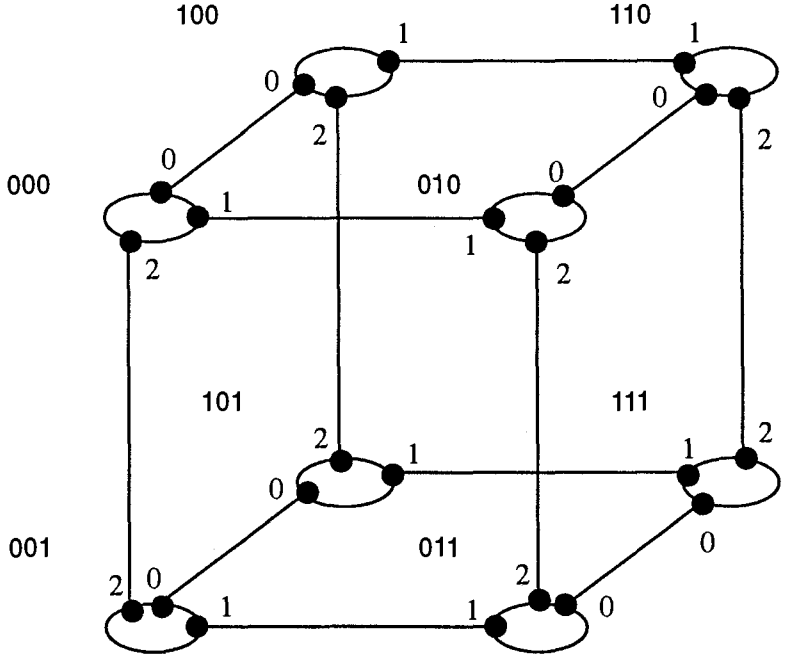


fig. 2. Cube-connected cycles CCC3.

CCCd was first proposed by Preparata and Vuillemin [PrVu81]. It is derived from Hd by replacing each processor by a cycle of d processors, each of which inherits a link to a hypercube

neighbour. In this way, the number of processor links is always 3. The diameter of CCCd is $2d + \lfloor d/2 \rfloor - 2$ for $d > 3$ and 6 for $d = 3$ [MeCh90]. Therefore, the maximum distance between two nodes is still O(logN).

Definition:

The *Butterfly* network of dimension d, denoted by BFd, is a communication network consisting of $N = d2^d$ processors. Each processor is labeled by a pair (i, X), where $0 \le i \le d-1$ and $X = x_0x_1...x_{d-1}$ is a binary string of length d. Every processor (i, X) is connected to four other processors: (i-1 mod d, X), (i+1 mod d, X), (i+1 mod d, X_i) and (i-1 mod d, $X_{i-1 \bmod d}$), where X_i denotes a binary label derived from X by replacing the i-th bit with its inverse.

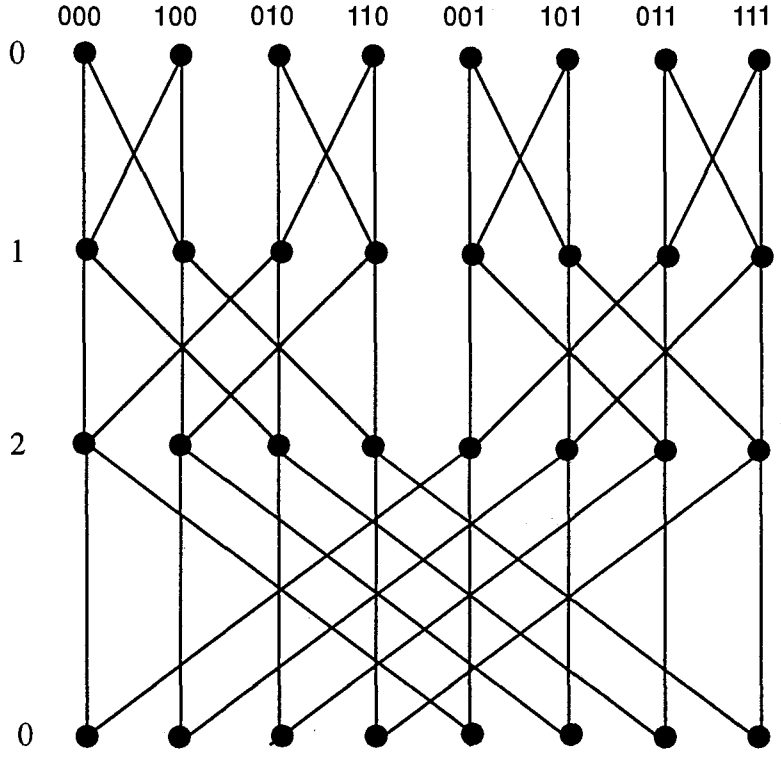


fig. 3. Butterfly BF3.

In this thesis we consider BFd for $d \geq 3$. Usually, BFd is displayed with the cycles represented by vertical lines and the 0-th node of each cycle duplicated for clarity. The resemblance between BFd (fig.3) and Hd (fig.1) is not as immediate as it is for the CCCd; however, we can see that each cycle of BFd corresponds to a processor in Hd. Every cycle is connected to its hypercube neighbours via two links, which improves the communication abilities of the network compared to the CCCd and yet preserves the processor link number constant -- 4. The diameter of BFd is $\lfloor 3d/2 \rfloor$, i.e., it is closer to the diameter of Hd than the diameter of CCCd is.

Communication patterns

In a communication network, processors need to exchange information. During the execution of a distributed program, communication schemes may vary, and they are very dependent on the particular problem. There are, however, several major communication patterns which appear very often. Here we briefly list some of them.

Routing:

Sending a message from one processor to another processor.

Broadcasting:

Sending a message from one processor to all other processors.

Scattering:

Sending different messages from one processor to all other processors.

Gossiping:

Sending a message from each processor to all other processors.

All these patterns are specific and require different approaches. Of course, given a routing algorithm all other patterns can be performed by sequentially sending messages from an originator to a destination. This brute force approach does not fully utilize the available parallelism in the network, and therefore other algorithms are needed.

In this work we study the problems of broadcasting and scattering in the CCCd and BFd

networks.

Routing strategies

Let us consider the task of routing a message between two processors within a communication network. In the general case those two processors are not adjacent and therefore intermediate nodes have to be used in order to send the message from the originator to the destination. It is possible to use several strategies. We briefly describe some of them.

Store-and-forward. The message proceeds from the source to the destination one "hop" at a time. This means that, first, the entire message is sent from the originator to the first node in the route. After the message is completely received (and stored in a buffer at the node), it is forwarded to the next node in the route. This is repeated until the message reaches the destination. *Packet-switching* is a type of store-and-forward routing in which the message is split into several packets, and then each of the packets is sent using the above described approach. We can even use different routes for the different packets to speed up the process.

Circuit Switching. Once the route is determined we occupy all links composing the route and send the message along the established channel. In a way, we switch the circuit to provide connection between the two nodes. Thus, it is not necessary to store the message at the intermediate nodes. The disadvantage, however, is that the route is busy during the entire message passing, and also that some time is needed to establish the connection.

We study only the store-and-forward strategy. Any circuit-switched communication network can also perform store-and-forward routing. Thus the algorithms described can be used in those networks as well.

Communication models

We model a communication network as a graph G defined by its set of vertices V(G) and set of edges E(G). Each vertex (node) of V(G) corresponds to a processor from the network and each edge $(u, v) \in E(G)$, $u, v \in V(G)$, corresponds to a link between the processors represented

by u and v.

In our work we concentrate on networks in which all processors and links have identical properties. Therefore, when we talk about the properties of a processor (link), we assume that they apply to all processors (links) in the network.

There are two possible cases for every link within a network:

1. The link could be used in only one direction at any given time. This is called a *half-duplex* link.

2. The link could be used simultaneously in both directions. This is called a *full-duplex* link.

Half-duplex links are modelled with arcs (i.e. using directed graphs) and full-duplex links are modelled with edges in the network graph. Another important parameter of the network is the ability of each processor to communicate over its links in parallel. There are two extreme cases, which we study further in our thesis:

1. A processor can use only one of its links at any given time. This case is called *processor-bound* communication.

2. A processor can use all of its links at the same time. In this case the communication is called *link-bound* communication.

Notice that the above described two properties are orthogonal. We further use some abbreviation to simplify the notation. F will stand for full-duplex links and H will stand for half-duplex links. F or H followed by 1 will mean processor-bound communication and F or H followed by * will mean link-bound communication. To summarize, there are four communication models which we study in this thesis: F1, F*, H1, H*.

Definition:

Given a connected graph G and a message originator $u \in V(G)$, the *broadcast time of vertex* u *(under model* M*)*, $b_M(u)$, is the

minimum number of time units required to complete broadcasting from vertex u under the M communication model and the *scatter time of vertex* u *(under model* M*)*, $s_M(u)$, is the minimum number of time units required to complete scattering from vertex u under the M communication model.

Definition:

The *broadcast time of a graph* G *(under model* M*)* is
$b_M(G) = \max\{b_M(u) \mid u \in V(G)\}$ and
the *scatter time of graph* G *(under model* M*)* is
$s_M(G) = \max\{s_M(u) \mid u \in V(G)\}$.

Once a graph corresponding to the network is built and the types of links and communication are determined, we have to decide how to measure the time that a communication between two processors takes. In earlier years, the time to send a message between two processors (nodes) was considered to be a constant. Thus, the algorithms were aimed to minimize the number of steps necessary to perform a given communication pattern. The estimated time to perform those algorithms did not reflect the fact that the time to finish a given algorithm depends on the length of the message being transferred across the communication network.

This approach is known as the *constant cost model* or the *telegraph model.* For surveys on the constant cost model refer to [HeHeLi88], [LiSo90], [BeFraPe92], [FrLa92]. This model appears to provide satisfactory solutions for several communication patterns and short messages. Unfortunately, there are communication patterns for which this is not the case. For example, scattering and broadcasting can use algorithms with the same number of steps; however, the amount of information sent during each step is different and thus, in real systems, scattering is always slower than broadcasting.

Another approach is to model the time needed for single communication between two processors (T) as a function of the message length. For the store-and-forward routing strategy, T is often modelled as:

$$T = \beta + L\tau$$

where β is the time to initiate the communication, τ is the time to send a single unit of information across the link and L is the length of the message. This model is known as the *linear cost model* of communication. For simplicity, we may refer to the linear cost model as *linear model.*

Many papers are devoted to the constant cost model, while the linear one is relatively new. It is also harder to use and it is less studied. A survey on the existing results under the linear model can be found in [FrLa92].

In this thesis, we use the linear model and store-and-forward routing in order to design algorithms for broadcasting and scattering in the CCCd and BFd networks under the F1, F*, H1 and H* models of communication. To the best of our knowledge, only the process of broadcasting in CCCd has been studied before (see [FrHo91], [FrLa91]). We improve the upper bounds presented by Fraigniaud and Ho in [FrHo91] and [FrHo91r]. All of our algorithms run in times very close to the existing lower bounds.

CHAPTER 1

LINEAR MODEL OF COMMUNICATION

In this chapter we describe the linear cost model of communication in details and give some general lower and upper bounds. We also show some common techniques for broadcasting and scattering which are used to design our algorithms presented in Chapters 3 and 4.

1.1 Definition of the model and general lower bounds

The time for a node to send a message to a neighbouring node greatly depends on the length of the message. This time can be modeled as a linear combination of the time to establish the connection called *start-up* time (β) and the time to send a unit of information of the message called *propagation* time (τ):

$$T = \beta + \tau L$$

where L is the length of the message measured in some units of information and τ is the time required to send one unit of information. We call $1/\tau$ the *bandwidth* of the link. The fact that the communication time depends on the length of the message implies that the number of steps is not the only parameter to be minimized when optimal algorithms are sought. It is also necessary to minimize the total amount of information sent at each step. Indeed, for cases in which $\beta \gg L\tau$ it is efficient enough to minimize the number of steps only. However, for long messages this approach is far from satisfactory and thus other techniques have to be used.

Before we proceed any further some general lower bounds on the communication time for broadcasting and scattering will be stated. Those were presented in [FrLa91]. The method to obtain the lower bounds is the one given by Ho [Ho90]: "The minimum data transfer time can be derived considering either of the following three cases: (1) *root dominance*, that is the minimum time required for the source node to send the data; (2) *latency dominance*, that is the propagation delay for the last element, and (3) *bandwidth dominance*, that is the total bandwidth required divided by the total bandwidth available."

Lemma 1.1: [FrLa91]

The minimum time to broadcast a message of length L in a Δ-regular graph G with diameter D is:

$$b_{F1}(G) \geq D\beta + (D-1+L)\tau, \quad b_{F^*}(G) \geq D\beta + (D-1+\frac{L}{\Delta})\tau,$$

$$b_{H1}(G) \geq \frac{(N-1)}{\lfloor N/2 \rfloor}L\tau, \quad b_{H^*}(G) \geq \frac{(N-1)}{N/2}\frac{L}{\Delta}\tau.$$

Proof: For the full-duplex model we use the latency dominance. The maximum distance between the originator and any other node is the diameter D (in the worst case) and therefore the first packet of the whole message sent to the furthermost node reaches it after a time of at least $D(\beta + \tau)$. Under the F1 model at most one unit of information has reached the node at that time and thus L - 1 other units have to be received as well. This takes time of at least $(L - 1)\tau$. Under the F* model parallel routes can be used to send the message and therefore Δ units of information could have been received after $D(\beta + \tau)$ time. Thus, there are $L - \Delta$ other units yet to be received and this requires at least $\frac{(L-\Delta)}{\Delta}\tau$ time. Adding up the two times gives the lower bounds for b_{F1} and b_{F^*}.

Under the half-duplex model for H* the nodes can communicate over all their links at the same time, therefore the total bandwidth is at most $\frac{N\Delta}{2}\frac{1}{\tau}$, since the bandwidth of every link is $1/\tau$. If only one port can be used at a given time by each processor then the total bandwidth is at most $\lfloor N/2 \rfloor \frac{1}{\tau}$. To broadcast a message the total number of exchanged information units is at least $(N-1)L$ (every node but the originator has to receive the full message). Using the bandwidth dominance the above bounds are obtained. □

Lemma 1.2: [FrLa91]

The minimum time to scatter messages of length L in a Δ-regular graph G with diameter D is:

$$s_{F1}(G) \geq (N-1)L\tau, \quad s_{F^*}(G) \geq (N-1)\frac{L}{\Delta}\tau,$$

$$s_{H1}(G) \geq (N-1)L\tau, \quad s_{H^*}(G) \geq (N-1)\frac{L}{\Delta}\tau.$$

Proof: The bounds for both the full-duplex model and the half-duplex models are obtained using the root dominance. The originator has to send at least (N - 1)L units of information -- a mes-

sage of length L to every other node. If it can only use one link at any given time then the time needed is at least $(N - 1)L\tau$. If all links can be used simultaneously then the time needed is at least $(N-1)\frac{L}{\Delta}\tau$. □

Having these general lower bounds we try to describe algorithms running in times as close to the bounds as possible.

1.2 Pipelining and disjoint spanning trees

There are two major techniques used to build information dissemination algorithms under the linear model of communication. They can be both applied for scattering and broadcasting and for that reason we describe them here. We want to emphasize that all our algorithms presented in chapters 3 and 4 are based on those two techniques.

1.2.1 Pipelining

Let us consider an array of N processors numbered from 0 to N-1. Let assume that node 0 has to broadcast a message M of length L to all other nodes. Under the constant model of communication an optimal algorithm would be to send the message from node 0 to node 1, then send it from node 1 to node 2, etc., until node N-1 receives M. If we applied the same approach under the linear model the time needed to broadcast M would be $(N - 1)(\beta + L\tau)$. Notice that at any given moment only one link of the array would be used. This implies that we could somehow split M into m packets of size $B < L$ and send the packets sequentially from node 0 and then forward a packet from any node immediately after the packet has been fully received. Thus, more than one link at a time will be used and the algorithm will run significantly faster (see [SaSch89] and [FrLa91]).

More precisely, the algorithm works as follows: First we send a packet of size B from node 0 to node 1. The moment node 1 receives the packet it sends it to node 2 while node 0 sends the next packet to node 1. We repeat this procedure for all nodes and until all packets reach the last node -- N-1. The L/B packets are *pipelined* from node 0 to node N-1. (see fig. 1.1).

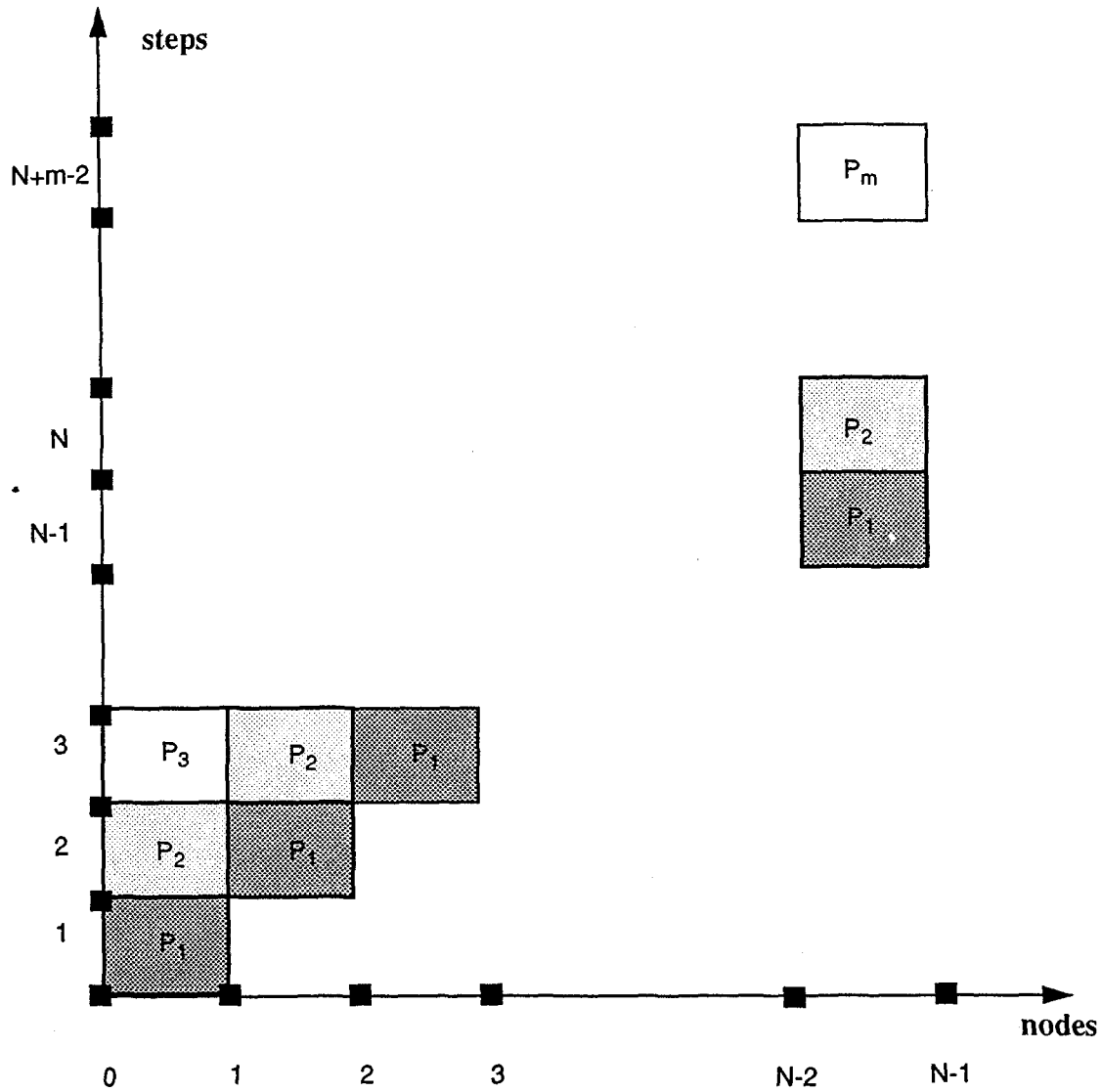


fig. 1.1. Pipelining in array of N processors.

Now let analyze the time needed to complete the algorithm. The first packet reaches node N-1 after N-1 steps, each of which takes time $\beta + B\tau$. At that time the second packet is already at node N-2 and thus one more step is needed for N-1 to receive the second packet. Thus, we can see that the remaining $\frac{L}{B}$ - 1 packets of size B successively reach node N-1 in $(\frac{L}{B} - 1)$ steps after the first one, that is in time $(\beta + B\tau)(\frac{L}{B} - 1)$. Therefore, the global time for all packets to reach node N-1 is $(N + \frac{L}{B} - 2)(\beta + B\tau)$. At that time all other nodes have already received all packets and thus this

is the broadcasting time. Minimizing the expression as a function of the packet size B gives $B_{opt} = \sqrt{\frac{L\beta}{(N-2)\tau}}$ and a total time of $(\sqrt{L\tau} + \sqrt{(N-2)\beta})^2$. Thus on an array of N processors using pipelining for large messages is asymptotically N-1 times faster than using the constant model algorithm.

Under the link-bound model this technique can be generalized for any graph G. Instead of pipelining M over an array we can do so over a spanning tree of G because each node can send the message in parallel to all its children in the tree. Thus, the time to broadcast will depend on the height of the spanning tree.

Definition:

The *height* of a tree is the largest distance between the root and a leaf.

Lemma 1.3: [BeFr91]

Let h be the height of a spanning tree of a graph G rooted at node r (the originator). Then there exists a protocol for link-bound broadcasting from r and the time to complete the protocol is $(\sqrt{L\tau} + \sqrt{(h-1)\beta})^2$.

We present an original approach for scattering. First, let us have the above described array of processors and let try to scatter messages of size L from node 0. If we attempt to split the messages into m packets of size B (assume that $m = L/B$), as we did for the broadcasting algorithm, and pipeline them to their destination nodes in reverse order, then after (N - 1) steps the first packet for node N-1 will reach its destination. Again after (L/B - 1) additional steps all packets for node N-1 will reach it. At that time node N-2 will have its first packet received and therefore another (L/B - 1) steps are necessary for all packets to reach node N-2. Continuing the same analysis for the rest of the nodes we can easily see that the total number of steps will be $(N-1) + (N-1)(\frac{L}{B} - 1)$ or $(N-1)\frac{L}{B}$. Each step takes time $\beta + B\tau$, therefore the total time needed will be $(\beta + B\tau)(N-1)\frac{L}{B}$.

Obviously the minimum of the above expression is for B = L. Therefore, the best approach is to send the messages in reverse other without splitting them (see fig. 1.2). This will

yield an algorithm for scattering in an array of time $(N - 1)\beta + (N - 1)L\tau$.

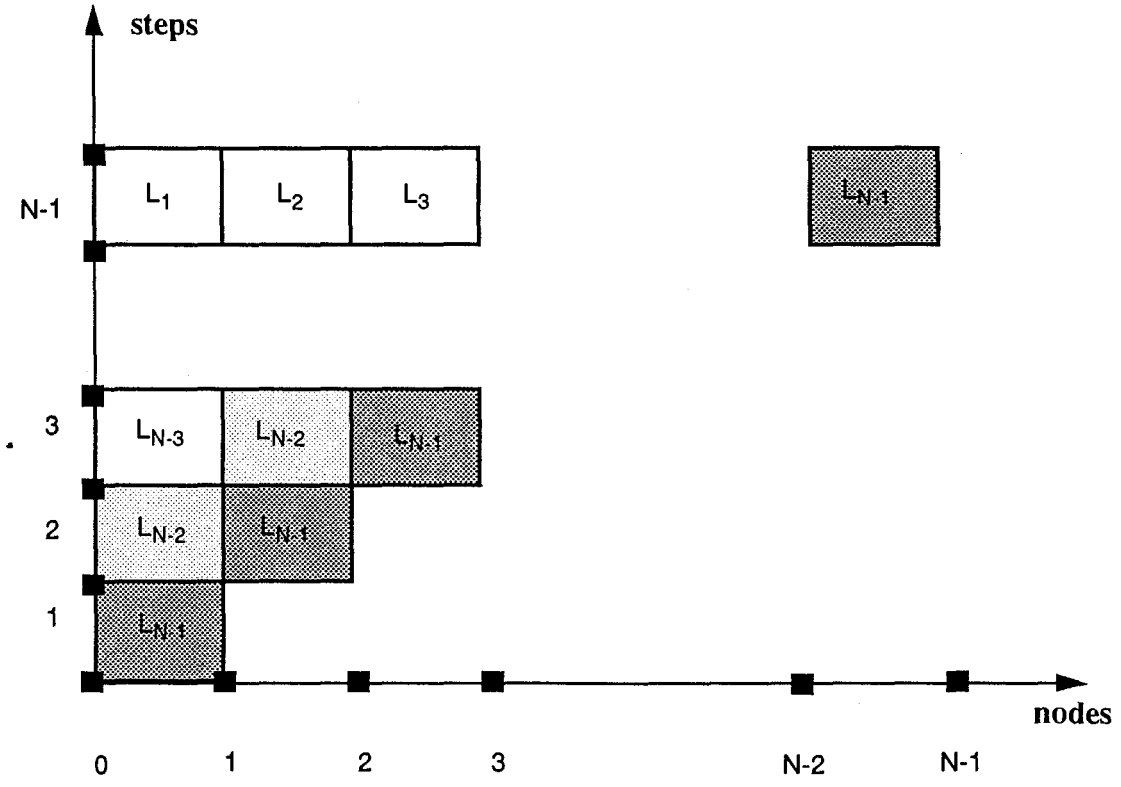


fig 1.2. Scattering in array of N processors.

It is not as easy to extend this approach to spanning trees as it was for broadcasting. The reason is that we have to make sure that at every step the information sent from every node does not exceed the information received by the same node. Otherwise some delay will be introduced at each step and significant changes in the algorithm will be needed. In the broadcasting algorithm every node sends exactly the same amount of information to its children of the spanning tree as it receives from its parent at every step -- a packet of size B. Unfortunately, during scattering every node in a spanning tree potentially has to send different amounts of information to its children. Thus, we introduce an approach which is limited to certain specific spanning trees.

Assume that r is a node of a graph G and there exists a perfectly balanced k-ary spanning tree T of G rooted at r. The height of T is h. Then we can modify the scattering algorithm for arrays

as follows.

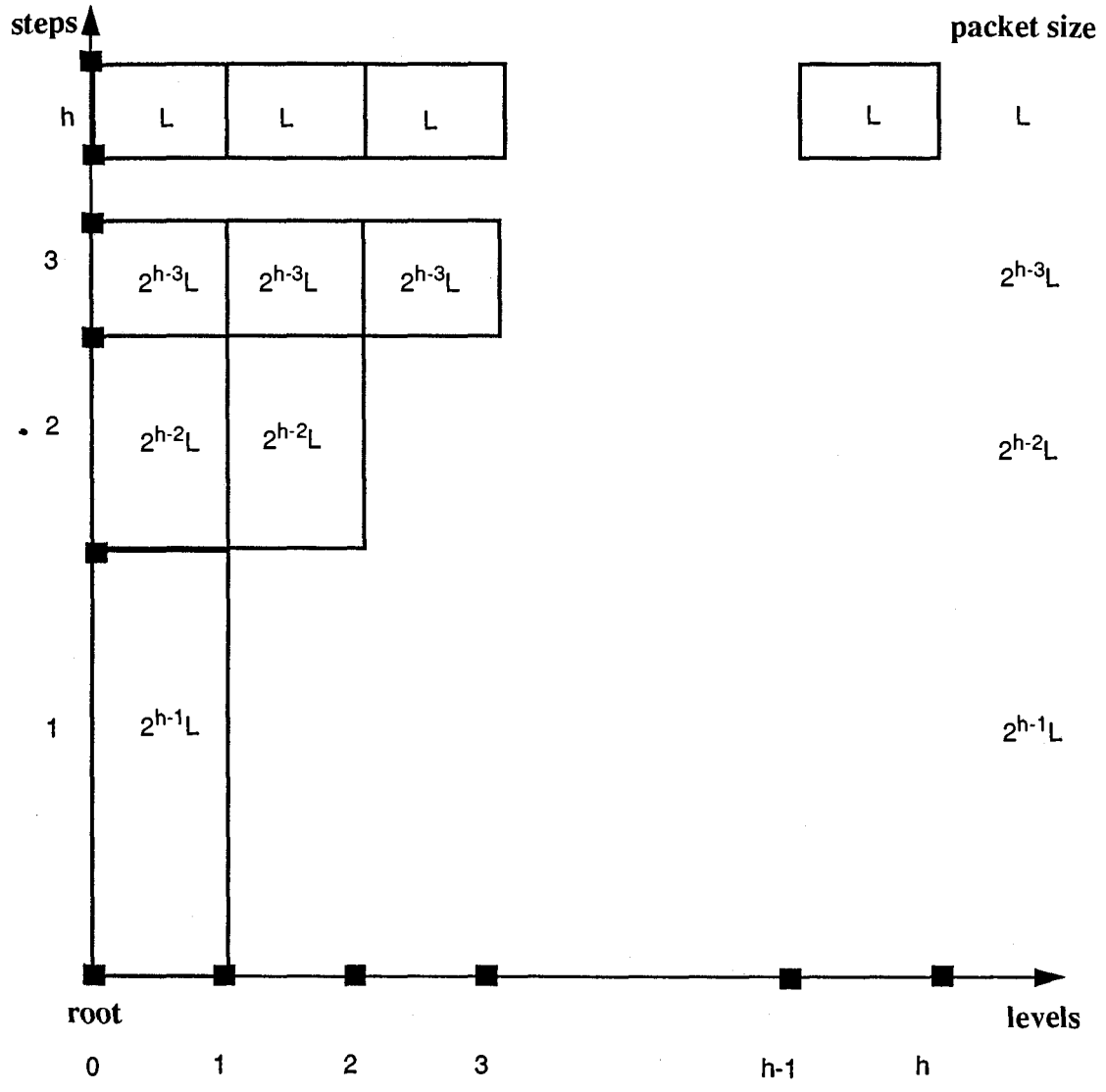


fig. 1.3. Scattering in perfectly balanced binary tree of height h.

In the first step we send from r to its children packets consisting of the messages for all leaves (i.e. nodes at distance h from r) in the corresponding subtree. Since T is a perfectly balanced k-ary tree, it follows that the number of leaves in every subtree of r is k^{h-1}. At every subsequent step i, $2 \leq i \leq h$, we send from r to all its children packets of size $P_i = k^{h-i}L$ containing the messages for all nodes at distance $h+1-i$ from r. At the same time every node which has received a

packet of size P_{i-1} at step i-1 splits the packet into k parts and during step i sends to each of its children a packet of size $\frac{1}{k}P_{i-1}$ containing the information for their subtree. During step i the node receives a packet of the same size from its parent (fig 1.3 shows scattering in a perfectly balanced binary tree).

Thus every step i, $1 \le i \le h$, takes time $k^{h-i}L\tau + \beta$. At the h-th step all nodes receive their messages. The time required to complete the scattering is therefore $\frac{k^h - 1}{k-1}L\tau + h\beta$. The total number of nodes in the tree is $\frac{k^{h+1}-1}{k-1}$ and therefore the scattering time is $\frac{N-1}{k}L\tau + h\beta$. This is stated in the following lemma.

Lemma 1.4:

Let h be the height of a perfectly balanced k-ary spanning tree of a graph G rooted at node r (the originator). Then there exists a protocol for link-bound scattering from r and the time to complete the protocol is $\frac{N-1}{k}L\tau + h\beta$.

Usually, it is very difficult to find such kind of spanning trees in a graph. We can generalize the lemma for partial scattering in any graph G:

Lemma 1.5:

Let h be the height of a perfectly balanced k-ary tree T rooted at node r (the originator) and let this tree be a subgraph of a graph G. Then there exists a protocol for link-bound scattering from r to all nodes in T and the time to complete the protocol is $\frac{M-1}{k}L\tau + h\beta$, where $M = \frac{k^{h+1}-1}{k-1}$ is the number of nodes in T.

Using this lemma we can construct scattering algorithms for arbitrary graphs in two phases. First we have to construct a perfectly balanced k-ary tree rooted at node r and this tree must be a subgraph of G. Each of the nodes in the tree will collect the messages for a set of nodes of G in such way that every node of G is in exactly one set. During phase one we scatter in the tree. During phase two all nodes from the tree scatter in parallel within their sets. This approach will be further illustrated in Chapter 4.

1.2.2 Disjoint spanning trees

The time needed to pipeline a message along a spanning tree of height h, even if it is possible to construct the tree in such a way that h is equal to D (the diameter of the graph), is still not too close to the broadcasting lower bound presented in Lemma 1.1. This is because we pipeline the whole message along the tree and thus the propagation time is proportionate to the length of the entire message. The lower bound suggests that it might be possible to split the message into packets and then pipeline the packets in parallel. In the best case, we might be able to split the message into Δ even parts, where Δ is the degree of the graph. To do so we need to find other spanning trees which will be used to pipeline the other parts of the message. However, during the pipelining, all links of the tree are busy for most of the time. Therefore, we have to find edge-disjoint spanning trees to be able to send different parts of the message in parallel.

It is also easy to observe that in case of a full-duplex model it is sufficient that the trees are arc-disjoint and in the case of a half-duplex model they have to be edge-disjoint, because every edge could be used only in one direction at a time.

To simplify the notation we shall further use the abbreviation ADST for arc-disjoint spanning trees, EDST for edge-disjoint spanning trees and ST for spanning tree.

Thus under the link-bound model giving p ADST or EDST rooted at the originator and using them to pipeline simultaneously different parts of the message defines a broadcasting algorithm:

Lemma 1.6: [BeFr91]

> Let h(p) be the maximum height of p ADST (EDST) rooted at node r of a graph G. Then there exists a protocol for link-bound broadcasting under the F^* (H^*) model from node r which completes its execution in time

$$\left(\sqrt{\frac{L\tau}{p}} + \sqrt{(h(p)-1)\beta}\right)^2$$

In this thesis we consider two Δ-regular graphs. There are no more than Δ ADST or EDST in any graph, because every node must be reached through all the trees and there are no more than Δ different links to any node. Therefore, our goal will be to find Δ ADST (EDST) of minimum possible height in CCCd and BFd.

Similarly, we can try to construct several arc(edge)-disjoint perfectly balanced k-ary trees to perform scattering. The exact construction heavily depends on the particular structure of the graph and, as it is shown in Chapter 4, some modifications might be necessary.

1.3 General upper bounds

Our description of pipelining and disjoint spanning trees implies the usage of link-bound communication. We can extend the algorithms to processor-bound networks using labeling of the links to avoid the usage of more than one link by a node at a time. In [FrLa91] Fraigniaud and Lazard use the notion of processor-bound disjoint-labeling of the edges (arcs) of p EDST (ADST) of a given graph. We briefly describe their approach here.

Definition:

A *processor-bound labeling* of the edges (arcs) of a spanning tree T of a graph G is a labeling $lab: E(T) \rightarrow N^+$ satisfying $lab(u, v) \neq lab(u, w)$ for any two distinct children v and w of u in T, and $lab(t, u) < lab(u, w)$, where t is the father of u in T and w is any child of u in T.

Definition:

A *processor-bound disjoint-labeling* of the edges (arcs) of p EDST (ADST) of a graph G is a processor-bound labeling lab_i of each tree T_i, $0 \leq i \leq p-1$, such that for every node u $lab_i(u, v) \neq lab_j(u, w)$, $lab_i(v, u) \neq lab_j(w, u)$ and $lab_i(u, v) \neq lab_j(w, u)$, $\forall v \neq w$ and $0 \leq i, j \leq p-1$.

Definition:

The *delay* of a processor-bound disjoint-labeling of p EDST (ADST) is the smallest integer ρ, such that for every node u and for any pair of edges (pair of non-symmetrical arcs) of extremity u these edges (non-symmetrical arcs) have a distinct label modulo ρ.

Using this labeling they perform the link-bound broadcasting algorithm in G under the processor-bound model sending the packets every ρ steps to insure the use of one link by any processor at any given time (see [FrLa91]). Since these labelings are strictly specific for every graph to give a general upper bound on the broadcasting under the processor-bound model Fraigniaud and Lazard use the notion of chromatic index:

Definition:

The *chromatic index* of graph G is the minimum number q, such that it is possible to label all edges of E(G) with numbers in {0,1,..,q-1} in such a way that each vertex has all its edges labeled with different numbers.

Since for any graph G ρ is at most equal to q(G) and the maximum labeling is at most h(p)(q(G)-1), the broadcasting time under the processor-bound model in G can be obtained by:

Lemma 1.7: [FrLa91]

Let q(G) be the chromatic index of G, and h(p) be the maximum height of p ADST (EDST) of G rooted at some node r. There exists a protocol for a processor-bound broadcasting from node r which takes time

$$\left(\sqrt{\frac{q(G)L\tau}{p}} + \sqrt{h(p)\,(q(G)-1)\,\beta}\right)^2.$$

Thus, the broadcasting paradigm for both link- and processor-bound models can be solved by finding as many ADST (EDST) of small heights as possible.

Constructing EDST for the half-duplex model may not be easy. Therefore, we need some tools to construct upper bounds on the broadcasting and scattering times under the half-duplex model given the results in the full-duplex one. Here we prove an original lemma which could be used for deriving algorithms for information dissemination under the half-duplex model from the corresponding algorithms in full-duplex model.

Lemma 1.8:

> If an information dissemination protocol P for a graph G takes time T(P) under the full-duplex model of communication and consists of k steps which start at the same time at all nodes involved then there exists a protocol Q under the half-duplex model for G such that $T(Q) \leq 2T(P)$.

Proof: P can be split into k steps which start at the same time at all nodes involved. The amount of information transmitted during step i at any node does not exceed some L_i, since P finishes in some finite time T(P). We transform every step i of P into two steps of Q (i_1 and i_2) in the following manner. For every link used during step i of P we have two cases. (1) the link is used in one direction and (2) the link is used in both directions. In both cases the time to complete step i at any node is $T_i \leq \beta + L_i\tau$. In step i_1 of Q we send at most L_i information units from one of the nodes to the one on the other side of the link (for both case (1) and case (2)) and in step i_2 of Q -- vice versa (for case (2)). Thus after those two steps every node has received exactly the same information it receives after step i of P. Every link is used in one direction at a time. Also, each of the steps i_1 and i_2 takes time $T_i \leq \beta + L_i\tau$ and therefore the time to complete Q is at most twice the time to complete P. □

As it is easy to see this lemma can only be used in special cases, for which the above conditions apply. Fortunately, all broadcasting algorithms deal with packets of fixed size and therefore every step starts at the same time at every node which transmits at that step. Therefore, Lemma 1.8 can be used to derive broadcasting upper bounds under for half-duplex communication. The lemma can also be applied to some scattering algorithms as we show in Chapter 4.

CHAPTER 2

BINARY SUBTREES OF CCCd AND BFd

In this chapter we introduce two structures (ascending and descending binary subtrees of CCCd and BFd) which are extensively used in constructing our broadcasting and scattering algorithms. We start with some general notation. Unless otherwise stated any reference to a number n should be understood to be n mod d, where d is the dimension of CCCd or BFd. Any reference to an integer interval (a, b), such that $b < a$, should be understood to be $(a, b + d)$. For example, when $d > 4$, $i \in (d-2, 2)$ means that i is d - 1, d or d + 1. Since, d mod d = 0 and d + 1 mod d = 1, it follows that i is either d - 1, 0 or 1.

Gd we use Gd to denote that certain statement applies to both CCCd and BFd.

X a cycle label, which is a binary string of length d -- $x_0x_1...x_{d-1}$; we also refer to some cycles using their labels, like 'cycle X' instead of 'cycle with label X'. Notice that the left-most bit of the label has the lowest index.

$\bar{0}$ this will stand for the label 00...0.

|X| the number of 1's in X.

X[i] the i-th bit of X, where i varies from 0 to d-1.

X_i X with inverted i-th bit, i.e. $X_i = x_0x_1...\bar{x}_i...x_{d-1}$.

i-node the i-th node of a cycle X, i.e. (i, X); for example, *0*-node is any node (0, X). Also, a node (i, X) is said to be *at level i* in X. Thus, *i*-node and node at level i are synonyms.

Gd[i = j] The set of all cycles of the graph Gd such that their labels X have j in bit i; for example CCCd[0 = 0] is the set of cycles X of CCCd such that X[0]=0, also CCCd[0,1,..,k = 0,0,..,0] is the set of all cycles X of CCCd such that the first k bits of their labels are 0's.

$\alpha(X)$ $\alpha(X) = \max \{i \mid X[i] = 1\}$, for $X \neq \bar{0}$ and

$\alpha(\bar{0}) = 0$. In other words, $\alpha(X)$ is the position of the last non-zero bit of X.

$\beta(X)$ $\beta(X) = \min \{i \mid X[i] = 1\}$, for $X \neq \bar{0}$ and

$\beta(\bar{0}) = 0$ (we may use $\beta(\bar{0}) = d$ as well). This is the position of the first non-zero bit of X.

$\alpha 1(X)$ $\alpha 1(X) = \alpha(X) + 1$ (thus if $\alpha(X) = d\text{-}1$ then $\alpha 1(X) = d$, which is $\alpha 1(X) = 0$).

$\beta 1(X)$ $\beta 1(X) = \beta(X) - 1$ (thus if $\beta(X) = 0$, which is $\beta(X) = d$, then $\beta 1(X) = d\text{-}1$).

Here is an important property of $\alpha(X)$ and $\beta(X)$ which we use in our proofs:

- **Property 2.1**:

$$\forall (X)\, \alpha(X) \geq \beta(X)$$

Proof: $\alpha(\bar{0}) = \beta(\bar{0}) = 0$ by definition and $\forall (X \neq \bar{0}) \mid X \mid > 0$, therefore

$\max \{i \mid X[i] = 1\} \geq \min \{i \mid X[i] = 1\}$, therefore

$\alpha(X) \geq \beta(X)$. □

We define several types of arcs in Gd (CCCd and BFd).

Definition:

We call an arc $(i, X) \rightarrow (j, Y)$ *cross arc* if $X \neq Y$ and *straight arc* otherwise.

Definition:

A *compound arc* $(i, X) \Rightarrow (j, Y)$ is the pair of arcs $(i, X) \rightarrow (k, Z)$ and $(k, Z) \rightarrow (j, Y)$, where X and Y are two different cycles of Gd and Z is identical to either X or Y. We also denote the compound arc as

$$(i, X) \rightarrow (k, Z) \rightarrow (j, Y).$$

Apparently, each compound arc consists of one cross arc and one straight arc. We call the intermediate node (k, Z) of this compound arc a *hop* or *hop node.*

Definition:

All arcs $(i, X) \rightarrow (i+1, Y)$, where i+1 stands for i+1 mod d, we call *ascending arcs* and all arcs $(i, X) \rightarrow (i-1, Y)$, where i-1 stands for i-1 mod d, we call *descending arcs.*

The above definition can be extended to compound arcs as well:

Definition:

All arcs $(i, X) \Rightarrow (i+1, Y)$, where i+1 stands for i+1 mod d, we call *ascending compound arcs* and all arcs $(i, X) \Rightarrow (i-1, Y)$, where i-1 stands for i-1 mod d, we call *descending compound arcs.*

For simplicity we further use the term ascending (descending) arc to refer to ascending (descending) compound arcs as well. In addition, there is a property of CCCd and BFd which will allow us to use similar constructions in both graphs:

Property 2.2:

Every cross arc of BFd can be mapped to a compound arc of CCCd.

Proof: There are two types of cross arcs in BFd.

$(i, X) \rightarrow (i+1, X_i)$:

Corresponds to $(i, X) \rightarrow (i, X_i) \rightarrow (i+1, X_i)$ of CCCd;

$(i, X) \rightarrow (i-1, X_{i-1})$:

Corresponds to $(i, X) \rightarrow (i-1, X) \rightarrow (i-1, X_{i-1})$ of CCCd. □

At this point we define what we refer to as ascending binary subtree of Gd (CCCd or BFd). We use one definition for both CCCd and BFd. This is achieved by using the mapping given in the proof of Property 2.2.

Definition:

We call *ascending binary subtree* of Gd (BFd or CCCd) of *height* $h \leq d$ rooted at node (r, Y) the subgraph T of Gd constructed as

follows:

step 0: make (r, Y) the root of T;

step j for each (i, X) added to T during step j-1

add $(i, X) \rightarrow (i+1, X)$ to T;

add $(i, X) \rightarrow (i+1, X_i)$ (for BFd) or

$(i, X) \Rightarrow (i+1, X_i)$ (for CCCd), that is

$(i, X) \rightarrow (i, X_i) \rightarrow (i+1, X_i)$, to T;

repeat the above step for all $1 \leq j \leq h$. [1]

Notice that an ascending binary subtree of CCCd includes compound arcs. In fact, each non-leaf node of such a tree is connected via a compound arc to exactly one of its children. Clearly, only ascending arcs (BFd) and ascending compound arcs (CCCd) are used to build the tree and at each step j we add *r+j*-nodes to the tree. In addition, all nodes added at step j belong to cycles X, such that $\forall (r+j \leq i \leq r-1)\, X[i] = Y[i]$. The proof of this important property is given bellow.

Property 2.3:

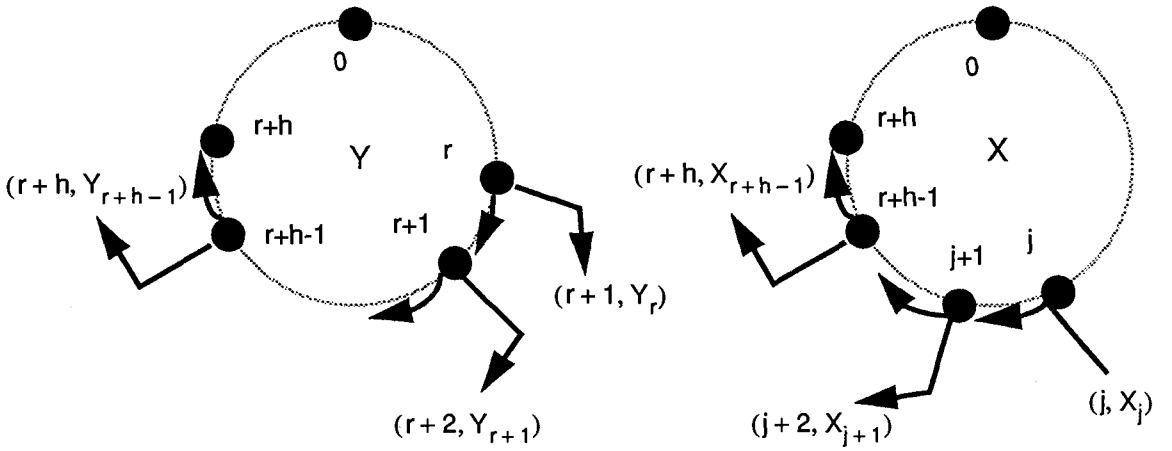
For every ascending binary subtree T of Gd rooted at (r, Y),

for every node $(r+j, X) \in T$, $\forall (r+j \leq i \leq r-1)\, X[i] = Y[i]$.

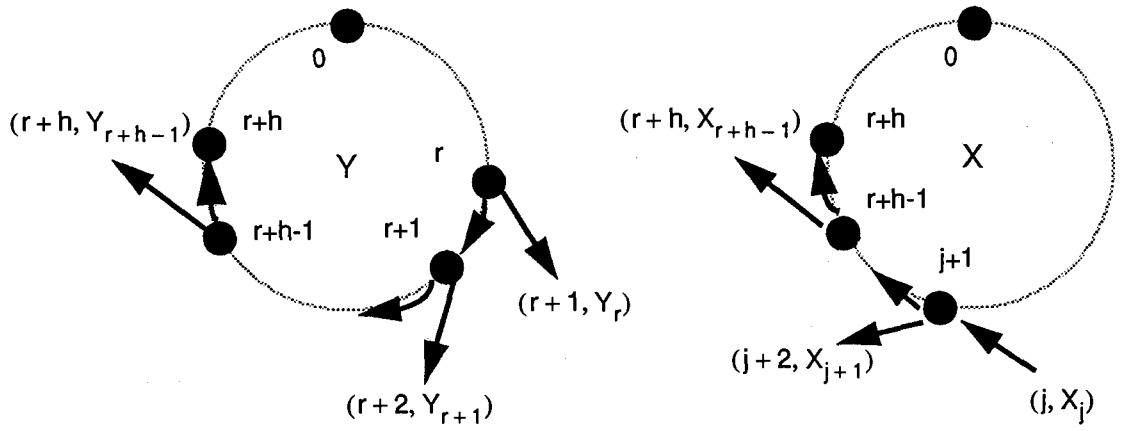
Proof: We denote the statement 'for every node $(r+j, X) \in T$, $\forall (r+j \leq i \leq r-1)\, X[i] = Y[i]$' by Ψ. As it was mentioned above, it is clear that at each step j we add *r+j*-nodes to T. This means that each node $(r+j, X) \in T$ was added to the tree during step j. We prove by induction that Ψ applies to all nodes added to T during any step j.

j = 1: We add two new nodes during step 1. One of them is $(r+1, Y)$, i.e. it belongs to the cycle of the root node. The other one is $(r+1, Y_r)$. Y_r has the same bits as Y except for $Y_r[r] = \overline{Y[r]}$. Therefore, $\forall (r+1 \leq i \leq r-1)\, Y_r[i] = Y[i]$ and Ψ holds for j=1.

1. If h = d then view the leaf (r + d, Y) as a node different from (r, Y) for symmetry.



Two cycles of an ascending binary subtree of CCCd rooted at (r, Y)



Two cycles of an ascending binary subtree of BFd rooted at (r, Y)

fig. 2.1.

$j = k$: Assume that Ψ holds for all nodes added to T during step k, i.e. for any $r+k$-node from T.

$j = k+1$: Let consider any of the nodes added to T during step k+1 -- $(r + k + 1, X)$. Since only ascending arcs are used to build T, the

node's parent could either be $(r+k, X_{r+k})$ (if the node was reached via a cross arc or compound arc) or $(r+k, X)$. Those are both $r+k$-nodes and therefore Ψ holds for either of them. This means that $\forall (r+k \le i \le r-1)\, X_{r+k}[i] = Y[i]$ or $\forall (r+k \le i \le r-1)\, X[i] = Y[i]$. In the latter case clearly Ψ holds for $(r+k+1, X)$. The only difference between the labels of X and X_{r+k} is in the $r+k$-th bit. Therefore, if the parent was $(r+k, X_{r+k})$, again we can conclude that $\forall (r+k+1 \le i \le r-1)\, X[i] = Y[i]$. □

For now, we do not consider the hop nodes of the compound arcs (for CCCd) to be in the tree. When our particular algorithms are described the hops will be taken into consideration.

Property 2.4:

Every ascending binary subtree of Gd is a perfectly balanced binary tree.

Proof: We show that any ascending binary subtree T of Gd is a tree by proving that each node in such a tree has exactly one parent. Assume that there exists a node $(r+j+1, X)$ from T that has two parents. Since only ascending arcs are used to build T those two parents should be $(r+j, X)$ and $(r+j, X_{r+j})$. These are $r+j$-nodes of T, an ascending binary subtree of Gd, and according to Property 2.3 $X[r+j] = Y[r+j]$ and $X_{r+j}[r+j] = Y[r+j]$, thus $X[r+j] = X_{r+j}[r+j]$. However, $X[r+j] \neq X_{r+j}[r+j]$ by definition. Therefore, our assumption was wrong and $(r+j+1, X)$ has only one parent. In addition, every non-leaf node of T has exactly two children. □

There are some properties of the ascending binary subtrees of Gd, which we use in our proofs later in this thesis. The first one tells us that given a node (i, X) of an ascending binary subtree T of Gd, rooted at some node (r, Y) and of height h, all nodes of cycle X with levels higher than i and not exceeding r+h are straight descendants (definition follows) of (i, X) in T.

Definition:

Node (j, X) is a *straight descendant* of another node (i, X) in a subtree T of Gd iff (j, X) can be reached from (i, X) by only using

straight arcs of T.

Property 2.5:

For any ascending binary subtree T of Gd of height h rooted at (r, Y), for each $(i, X) \in T$ $\forall (i < m \leq r+h)$ (m, X) is a straight descendant of (i, X) in T.

Proof: We use induction on m.

m = i + 1: Let $(i, X) \in T$ and $i \leq r+h-1$. Then this node was added to T at step $j = i - r \leq h - 1$. Therefore, during step $j+1 \leq h$ node $(i+1, X)$ was also added to T via a straight arc (see the definition of ascending binary subtree) and (i, X) is its parent. Thus $(i+1, X)$ is a straight descendant of (i, X) in T.

m = i + k: We assume that $(i+k, X)$ is a straight descendant of (i, X) in T, where $i+k \leq r+h-1$.

m = i+k+1: Since $(i+k, X) \in T$ and $i+k \leq r+h-1$ then this node was added to T during some step $j = i+k-r$, where $j \leq r+h-1-r = h-1$. Therefore, at step $j+1 \leq h$ node $(i+k+1, X)$ was added via a straight arc to T and its parent is $(i+k, X)$, which is a straight descendant of (i, X). Thus $(r+k+1, X)$ is also a straight descendant of (i, X) in T. □

This means that once a node (i, X) from a cycle X is added to the tree all nodes from X at levels between i and r+h are included in the tree and could be reached from (i, X) by only using straight arcs of the tree.

Definition:

Given a subtree T of Gd and a cycle X of Gd (CCCd or BFd) we say that $X \in T$ (*X is represented in T*) iff $\exists i ((i, X) \in T)$.

Now, imagine that a cycle X represented in an ascending binary subtree T of Gd of height h, rooted at (r, Y), has more than one node reached from another cycle represented in T. Let two of those nodes be (i, X) and (j, X), $i \neq j$. If $r + h > i > j$ then according to Property 2.5 (i, X) is a

straight descendant of (j, X) in T. Therefore, (i, X) is reached via straight arc from its parent in T. This contradicts to the assumption that (i, X) was reached from another cycle. Therefore, $i < j < r + h$, but then following the same reasoning (j, X) is a straight descendent of (i, X) and must be reached from its parent via a straight arc, which is another contradiction. Therefore, we can conclude that each cycle represented in T is reached via a cross arc (BFd) or compound arc (CCCd) exactly once.

Every non-leaf node of T has exactly one child in another cycle. Each cross arc (compound arc) adds a new cycle to the tree, since no cycle is reached via cross arcs (compound arcs) more than once. Therefore, there is a 1-1 correspondence between the non-leaf nodes of T and the cycles (excluding the cycle of the root Y) represented in T. There are $2^h - 1$ non-leaf nodes in T (according to Property 2.4 T is a perfectly balanced binary tree) and hence, there are $2^h - 1$ different cycles represented in T and Y is not among them. Therefore,

Property 2.6:

There are 2^h different cycles represented in any ascending binary subtree of Gd of height h.

Our algorithms use ascending binary subtrees having a specific property. The root (r, Y) is always chosen in such a way that $r \geq \alpha 1(Y)$. Given a tree with this property we try to determine which is the first node of each cycle added to the tree.

Lemma 2.1:

Given an ascending binary subtree T of Gd of height h rooted at (r, Y), such that $r \geq \alpha 1(Y)$, the first node of every cycle $X \in T$ added to T is its $\alpha 1$-node.

Proof: Every cycle $X \in T$ is reached exactly once via a cross arc (BFd) or compound arc (CCCd) from another cycle represented in T. Let assume that the first node of X added to T is $(r+j, X)$. Then its parent is $(r+j-1, X_{r+j-1})$ and according to Property 2.3 $\forall (r+j-1 \leq i \leq r-1) X_{r+j-1}[i] = Y[i]$. Since $r \geq \alpha 1(Y)$ then all bits of Y between r and d-1 must be 0's. Therefore, $\forall (r+j-1 \leq i \leq d-1) X_{r+j-1}[i] = 0$ and consequently

$\forall (r+j \le i \le d-1)\, X[i] = 0$, because X has the same bits as X_{r+j-1} except for the r+j-1-th bit, i.e. X[r+j-1] = 1. Therefore, according to the definition of $\alpha 1$, $\alpha 1(X) = r+j$ and it follows that the first node of X added to T is its $\alpha 1$-node. □

It is interesting to find some general way to describe all cycles represented in a given ascending binary subtree T of Gd of height h rooted at node (r, Y). If h = d then 2^d (i.e. all) cycles are represented in T. If h < d then all nodes of T are reached at steps $j \le h < d$ and thus according to Property 2.3 for all cycles X represented in T $\forall (r+h \le i \le r-1)\, X[i] = Y[i]$, where the addition is modulo d. This means that the labels of all cycles $X \in T$ differ from the label of Y in at most h bits. There are 2^{i-1}, $1 \le i \le h$, labels that differ from Y in i bits. Therefore, including Y, there are exactly 2^h possible labels which differ from the label of Y in at most h bits. Also, there are exactly 2^h different cycles represented in T (Property 2.6). Therefore, if h < d then $X \in T$ iff $\forall (r+h \le i \le r-1)\, X[i] = Y[i]$. We can formulate this as:

Property 2.7:

For any ascending binary subtree T of Gd of height h < d rooted at node (r, Y) a cycle X is represented in T iff $\forall (r+h \le i \le r-1)\, X[i] = Y[i]$ and also all cycles of Gd are represented in any ascending binary subtree T of Gd of height d.

If the height h of an ascending binary subtree of Gd rooted at (r, Y) is such that r + h = d then we can conclude:

Lemma 2.2:

If T is an ascending binary subtree of Gd rooted at node (r, Y) of height h = d - r then all cycles X in

Gd[0,1,..,r-1 = Y[0],Y[1],..,Y[r-1]]

are represented in T.

Proof: T is such that r + h = d, thus for each cycle X represented in T $\forall (d \le i \le r-1)\, X[i] = Y[i]$ and vice versa (Property 2.7). The addition is modulo d, i.e. d stands for d mod d, which is 0. Therefore, each cycle X, such that $\forall (0 \le i \le r-1)\, X[i] = Y[i]$, is repre-

sented in T and Gd[0,1,..,r-1 = Y[0],Y[1],..,Y[r-1]] is exactly the set of cycles of Gd, such that their labels match the first r-1 bits of Y. □

As we already mentioned, only ascending arcs are used to construct ascending binary subtrees of Gd. We can use the descending arcs to define descending binary subtrees with similar properties. It is possible to define those subtrees in a uniform manner for both CCCd and BFd using the mapping of cross arcs of BFd to compound arcs of CCCd (Property 2.2). This, however, will create unnecessary complications for describing our scattering algorithm for CCCd. Since this is the only algorithm for which we need descending binary subtrees of CCCd, we prefer to give separate definitions.

Definition:

We call *descending binary subtree* of BFd of *height* $h \le d$ rooted at node (r, Y) the subgraph T of BFd constructed as follows:

step 0: make (r, Y) the root of the tree;

step j for each node (i, X) added to T at step j-1

add $(i, X) \rightarrow (i-1, X)$ to T;

add $(i, X) \rightarrow (i-1, X_{i-1})$ to T;

repeat the above step for all $1 \le j \le h$.[1]

Observe that if we used compound arcs $(i, X) \Rightarrow (i-1, X_{i-1})$, which is $(i, X) \rightarrow (i-1, X) \rightarrow (i-1, X_{i-1})$, to build descending binary subtrees of CCCd then the hop-node of such an arc would coincide with the other child of (i, X). To simplify our presentation and avoid artificial substitutions and other tricks we use another type of compound arcs to define descending binary subtrees in CCCd:

Definition:

We call *descending binary subtree* of CCCd of *height* $h \le d$ rooted at node (r, Y) the subgraph T of CCCd constructed as follows:

1. If h = d then view the leaf (r - d, Y) as a node different from (r, Y).

step 0: make (r, Y) the root of the tree;

step j for each node (i, X) added to T at step j-1

add $(i, X) \rightarrow (i-1, X)$ to the tree;

add $(i, X) \Rightarrow (i-1, X_i)$, that is

$(i, X) \rightarrow (i, X_i) \rightarrow (i-1, X_i)$, to T;

repeat the above step for all $1 \leq j \leq h$.[1]

For now, we do not consider the hop-nodes of the compound arcs of CCCd to be in the tree. Those are taken into account when our particular algorithms are presented.

For both CCCd and BFd the descending binary subtrees have structures similar to the structure of the ascending binary subtrees of Gd. We can show properties of the descending binary subtrees corresponding to the results we already have for the ascending binary subtrees using similar proofs.

Observe that at each step j nodes of the type (r - j, X) are added to the descending binary subtrees for both CCCd and BFd. Following the proof for Property 2.3 we can conclude:

Property 2.8:

For every descending binary subtree T of CCCd rooted at (r, Y),

for each node $(r-j, X) \in T$, $\forall (r+1 \leq i \leq r-j)\, X[i] = Y[i]$.

For every descending binary subtree T of BFd rooted at (r, Y),

for each node $(r-j, X) \in T$, $\forall (r \leq i \leq r-j-1)\, X[i] = Y[i]$.

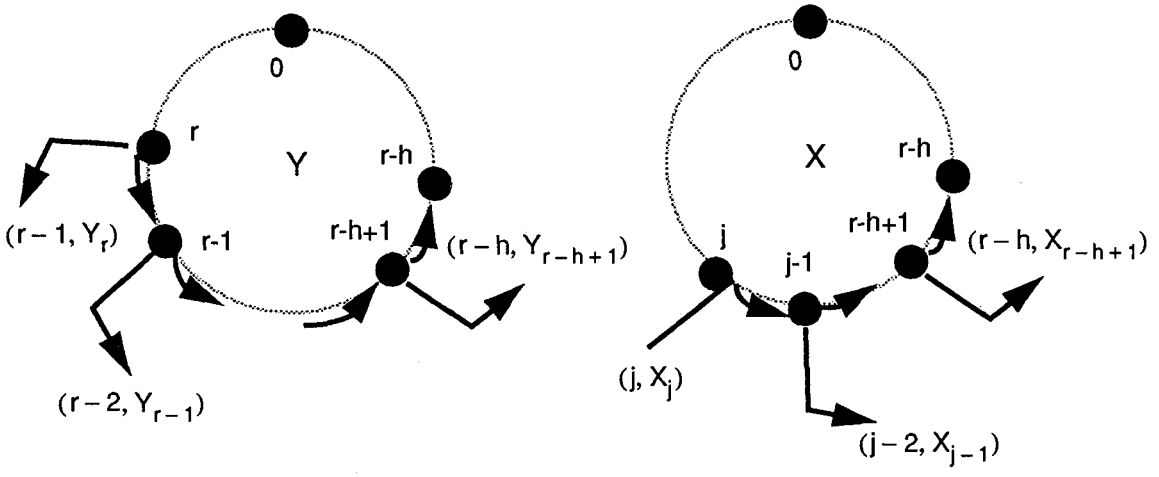
Using Property 2.8 and applying the approach used to prove Property 2.4 we can derive:

Property 2.9:

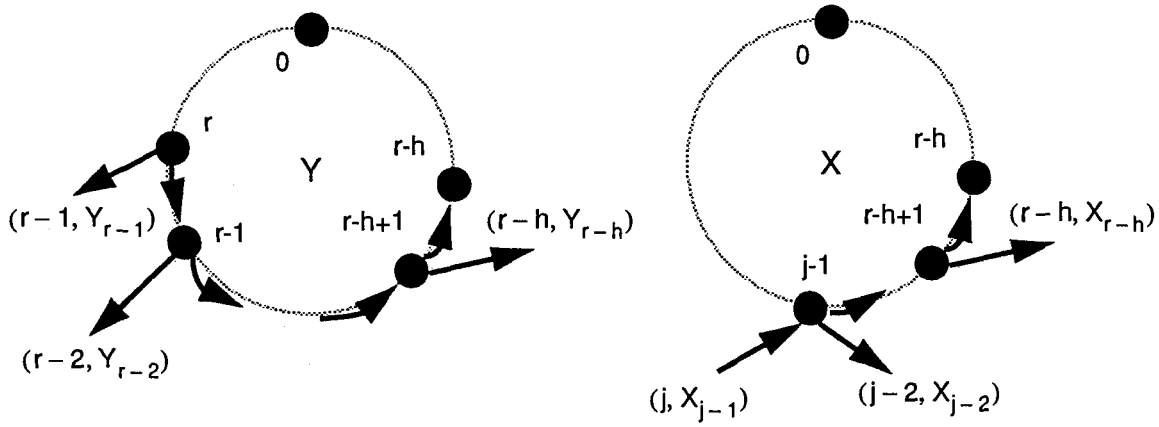
Every descending binary subtree of Gd is a perfectly balanced binary tree.

It is also easy to modify the proof of Property 2.5 and Property 2.6 to show that:

1. If h = d then view the leaf (r - d, Y) as a node different from (r, Y).



Two cycles of a descending binary subtree of CCCd rooted at (r, Y)



Two cycles of a descending binary subtree of BFd rooted at (r, Y)

fig.2.2.

Property 2.10:

For any descending binary subtree T of Gd of height h rooted at (r, Y), for each $(i, X) \in T$ $\forall (r - h \leq m < i)$ (m, X) is a straight descendant of (i, X) in T.

and

Property 2.11:

There are 2^h different cycles represented in every descending binary subtree of Gd of height h.

For each cycle represented in a descending binary subtree of Gd we want to know which is the first node from the cycle added to the tree. All descending binary subtree which we use for our broadcasting and scattering algorithms are such that $r \leq \beta 1(Y)$ (for CCCd) and $r \leq \beta(Y)$ (for BFd). For these special cases we can prove:

- **Lemma 2.3:**

  Given a descending binary subtree T of CCCd (BFd) of height h rooted at (r, Y), such that $r \leq \beta 1(Y)$ ($r \leq \beta(Y)$ for BFd), the first node of every cycle $X \in T$ added to T is its $\beta 1$-node (β-node).

Proof: Every cycle $X \in T$ is reached exactly once via a compound arc of CCCd (cross arc of BFd) from another cycle represented in T (according to Property 2.11 there are 2^h different cycles represented in T and this is also the number of cross/compound arcs in T). Let assume that the first node of X added to T is $(r-j, X)$. Then its parent is $(r-j+1, X_{r-j+1})$ ($(r-j+1, X_{r-j})$ for BFd) and according to Property 2.8

$$\forall (r+1 \leq i \leq r-j+1)\, X_{r-j+1}[i] = Y[i] \text{ (and for BFd}$$
$$\forall (r \leq i \leq r-j)\, X_{r-j}[i] = Y[i]).$$

Since $r \leq \beta 1(Y)$ ($r \leq \beta(Y)$ for BFd) then all bits of Y between 0 and r (r-1) must be 0's. Therefore, $\forall (0 \leq i \leq r-j+1)\, X_{r-j+1}[i] = 0$ (and for BFd $\forall (0 \leq i \leq r-j)\, X_{r-j}[i] = 0$). The only difference between the label of X and the label of its parent is the r-j+1-th bit (r-j-th bit for BFd). Therefore, $X[r-j+1] = 1$ ($X[r-j] = 1$ for BFd) and all other bits of X are the same as the bits of its parent, i.e.

$$\forall (0 \leq i \leq r-j)\, X[i] = 0 \text{ (and for BFd}$$
$$\forall (0 \leq i \leq r-j-1)\, X[i] = 0).$$

Therefore, $\beta 1(X) = \beta(X)-1 = r-j+1-1 = r-j$ ($\beta(X) = r-j$ for BFd). Therefore, the first node of X added to T is its $\beta 1$-node (β-node for BFd). □

In the end, we can state the counterpart of Property 2.7:

Property 2.12:

For any descending binary subtree T of Gd of height $h < d$ rooted at node (r, Y) a cycle X is represented in T iff

$\forall (r+1 \le i \le r-h)\ X[i] = Y[i]$ (Gd is CCCd) or

$\forall (r \le i \le r-h-1)\ X[i] = Y[i]$ (Gd is BFd).

All cycles of Gd are represented in any descending binary subtree T of Gd of height d.

If the height h of a descending binary subtree of Gd rooted at (r, Y) is such that $r - h \equiv 0$ (which could be considered as r - h = d modulo d) then we can conclude:

Lemma 2.4:

If T is an descending binary subtree of CCCd rooted at node (r, Y) of height h = r then all cycles X in

CCCd[r+1,r+2,...,d = Y[r+1],Y[r+2],...,Y[d]]

are represented in T.

If T is an descending binary subtree of BFd rooted at node (r, Y) of height h = r then all cycles X in

BFd[r,r+1,..,d-1 = Y[r],Y{r+1],..,Y[d-1]]

are represented in T.

Proof: T is such that $r - h = 0$ (d mod d), thus for each cycle X represented in T we simply substitute d for r - h in the result stated in Property 2.12. □

CHAPTER 3

BROADCASTING IN CCCd AND BFd

In this chapter we describe our algorithms for broadcasting under the linear model of communication in CCCd and BFd networks. Our main approach is to construct Δ arc-disjoint spanning trees of minimal height in both graphs, where Δ is the degree of the graph. Then by pipelining along the ADST we give upper bounds on the broadcasting time using the results presented in Chapter 1. Both graphs are vertex transitive and therefore it is sufficient to give the ADST rooted at any vertex r. The optimal solution would be to find Δ ADST of height D (the diameter of Gd). For both graphs we give Δ ADST of height close to D and also 2 ADST of height almost D, which proved to yield faster algorithms for short messages.

We first construct the ADST and analyze the algorithms under the full-duplex link-bound model (F*) and then analyze the other models of communication (F1, H*, H1) using the general upper bounds described in Chapter 1.

3.1 Three Arc-Disjoint Spanning Trees in the Cube-Connected Cycles graph

Our goal is to construct 3 ADST of CCCd -- T0, T1 and T2. We assume the originator to be $(1, \bar{0})$. In each step and substep of the following algorithm we use arcs which have not been used during the preceding steps to ensure that the trees are arc-disjoint. We analyze the height of each tree after every step.

step 0:

Add $(1, \bar{0}) \rightarrow (0, \bar{0})$, $(0, \bar{0}) \rightarrow (0, \bar{0}_0)$, $(0, \bar{0}_0) \rightarrow (1, \bar{0}_0)$ to T0;

add $(1, \bar{0}_0) \rightarrow (2, \bar{0}_0)$ to T0

(the height of T0 is 4);

add $(1, \bar{0}) \rightarrow (1, \bar{0}_1)$, $(1, \bar{0}_1) \rightarrow (2, \bar{0}_1)$ to T1;

add $(1, \bar{0}_1) \rightarrow (0, \bar{0}_1)$, $(0, \bar{0}_1) \rightarrow (0, \bar{0}_{01})$, $(0, \bar{0}_{01}) \rightarrow (1, \bar{0}_{01})$ to T1;

add $(1, \bar{0}_{01}) \rightarrow (2, \bar{0}_{01})$ to T1

(the height of T1 is 5);

add $(1, \bar{0}) \rightarrow (2, \bar{0})$ to T2

(the height if T2 is 1).

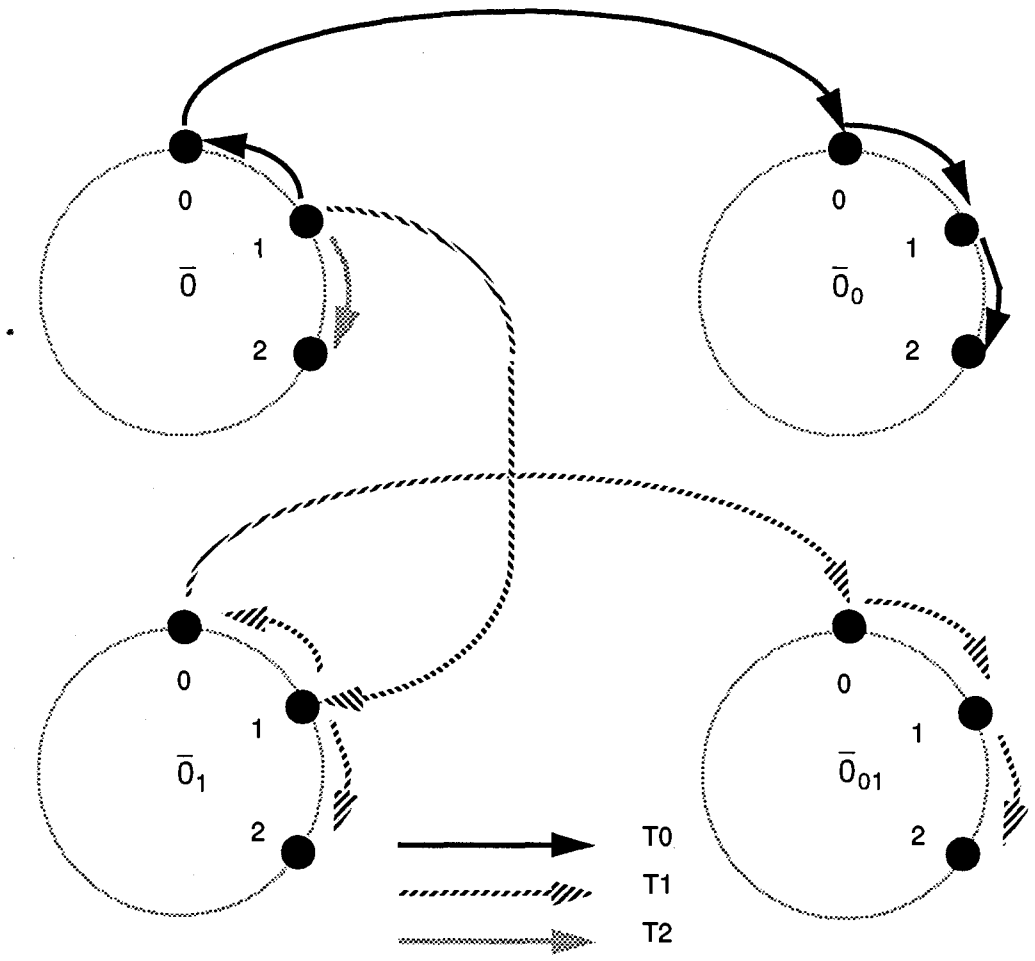


fig. 3.1. CCCd after step 0.

During the next step we build 4 trees -- Ta, Tb, Tc and Td, rooted at nodes $(2, \bar{0})$, $(2, \bar{0}_1)$, $(2, \bar{0}_0)$ and $(2, \bar{0}_{01})$ respectively. Those four trees are built in a uniform manner. We show how to build any of them, denoted by T and rooted at (2, Y). Notice that $r \geq \alpha 1\,(Y)$ for all root nodes $(2, \bar{0})$, $(2, \bar{0}_1)$, $(2, \bar{0}_0)$ and $(2, \bar{0}_{01})$. In this way we can use Lemma 2.1 to determine the first node reached from each cycle represented in either of the trees.

step 1:

step 1.1: Build an ascending binary subtree oh height d-2 rooted at $(2, Y)$ and call this tree T.[1]

step 1.2: For every $(0, X), X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,[2] added to T during step 1.1 add $(0, X) \rightarrow (1, X)$ to T.

step 1.3: For every node (1, X) added to T at step 1.2 add $(1, X) \rightarrow (1, X_1)$ to T.

step 1.4: For every node (1, X) added to T at step 1.3 $\forall (1 \leq j \leq \alpha(X) - 1)$ add $(j, X) \rightarrow (j+1, X)$ to T.

step 1.5 For every node $(\alpha(X), X)$ added at step 1.4 to T add $(\alpha(X), X) \rightarrow (\alpha(X), X_{\alpha(X)})$ to T.

Since $r = 2$ and $h = d - 2$, according to Lemma 2.2, after step 1.1 all nodes of Ta are in CCCd[0,1 = 0,0], all nodes of Tb are in CCCd[0,1 = 0,1], all nodes of Tc are in CCCd[0,1 = 1,0] and all nodes of Td are in CCCd[0,1 = 1,1]. Those four sets of cycles are non-intersecting and therefore, the four trees are vertex disjoint and consequently arc disjoint. According to Lemma 2.1, all cycles in any of the trees are reached via their $\alpha 1$-nodes first. Since those $\alpha 1$-nodes are added to the trees via a compound arc, we have to consider the hop-nodes of the compound arcs as well, and in that case the first node of each cycle added to a tree is the cycle's α-node. According to Property 2.5, once a cycle X is reached through its α-node, all nodes of X at levels higher than $\alpha(X)$ and not exceeding d are straight descendants of $(\alpha(X), X)$. Thus, the only straight arcs that are added during step 1.1 are those between nodes $(\alpha(X), X)$ and (d, X), and the only cross arcs added are the ones pointing to α-nodes at levels greater than 1.

Consider any of the four trees T, rooted at $(2, Y)$. The shortest path from $(2, Y)$ to any leaf of T consists of d - 2 arcs of T. In the worst case, all these arcs are compound arcs, i.e. each of them is composed of two arcs of CCCd. Therefore, in the worst case, the shortest path from the

1. We remove arcs $(d-1, \bar{0}_1) \rightarrow (0, \bar{0}_1)$ from Tb, $(d-1, \bar{0}_0) \rightarrow (0, \bar{0}_0)$ from Tc and $(d-1, \bar{0}_{01}) \rightarrow (0, \bar{0}_{01})$ from Td, because they add nodes which are in the trees already.

2. The cycles containing the originators are treated separately in step 3.

root $(2, Y)$ to a leaf of T is:

$$(2, Y) \Rightarrow (3, Y_2) \Rightarrow \ldots \Rightarrow (d, Y_{2\ldots d-1}).$$

Therefore, the furthermost leaf $(0, X)$ of T is such that $\forall (2 \le i \le d-1)\, X[i] \neq Y[i]$ and the height of T is $2(d-2) = 2d-4$.

Definition:

$z(X) = |\{i | X[i] = 0, 2 \le i \le d-1\}|$, i.e. it is the number of zero bits between positions 2 and d-1 of X.

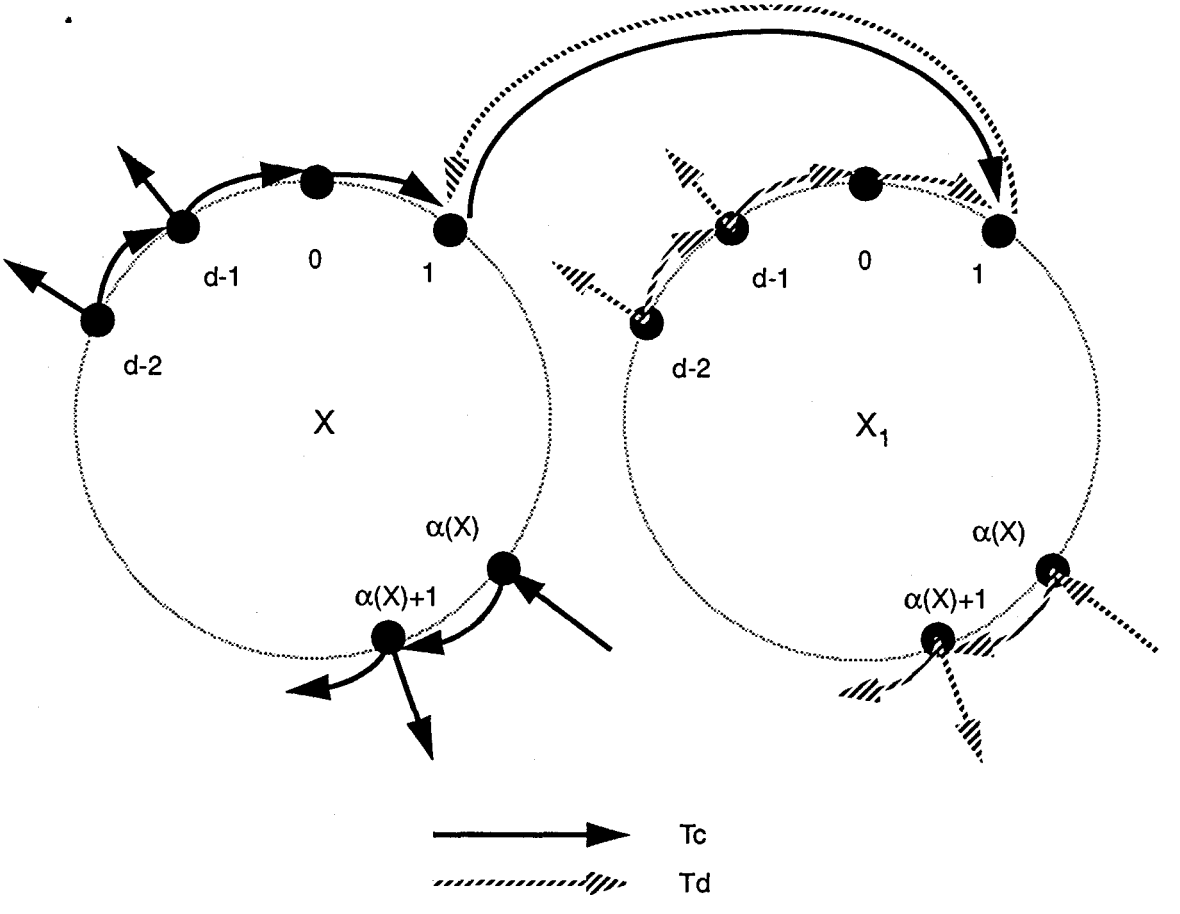


fig. 3.2. Two cycles of CCCd after step 1.3; $X \in CCCd[0, 1 = 1, 0]$.

Since $\alpha 1(Y) \ge 2$, it follows that the furthermost leaf $(0, X)$ is such that $z(X) = 0$. It is also

clear that any node $(i, X) \in T$, such that $z(X) > 0$, is of distance *less* that $2d - 4$ from the root of T.

In step 1.2 we add one more level to each tree by using arcs not included in any tree before. Ta, Tb, Tc and Td are still vertex and arc-disjoint and of height $2d - 3$.

Assume that after step 1.2 $(1, X)$ is in some tree T and X is in $CCCd[0,1 = x,x]$. Therefore, X_1 is in $CCCd[0,1 = x,\bar{x}]$ and its *1*-node $(1, X_1)$ is in some other tree -- $\bar{T}$. During step 1.3 node $(1, X)$ is added to $\bar{T}$ and $(1, X_1)$ -- to T. Thus those two nodes are represented in two different trees after step 1.3. The arcs used during the step are cross arcs to nodes of level 1 and therefore were not used before (remember that during this step we do not consider cycles $\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}$), i.e. all four trees remain arc-disjoint. The height of the trees increases by one and is $2d - 2$ after the step. More precisely, in step 1.3, all nodes $(1, X)$, such that X is in $CCCd[0,1 = 0,1]$ are reached via a cross arc from Ta, all nodes $(1, X)$, where X is in $CCCd[0,1 = 0,0]$ -- from Tb, all nodes $(1, X)$, X is in $CCCd[0,1 = 1,1]$ -- from Tc and all nodes $(1, X)$, X is in $CCCd[0,1 = 1,0]$ -- from Td.

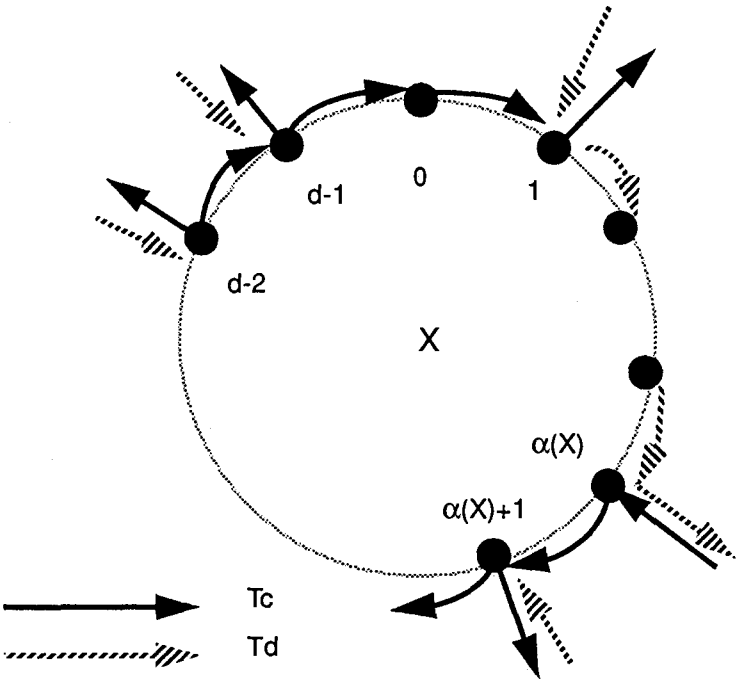


fig. 3.3. A cycle $X \in CCCd[0, 1 = 1, 0]$ after step 1.

During step 1.4, starting from $(1, X)$, in all cycles $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$ we add to some

tree $\bar{T}$ straight arcs which were not used in the preceding steps until we reach $(\alpha(X), X)$. Node $(1, X)$ was added to $\bar{T}$ during step 1.3. All nodes of X at levels between $\alpha(X)$ and d are represented in some other tree -- T. Therefore, in step 1.4, node $(\alpha(X), X)$ is added to a second tree.

All four trees remain arc-disjoint. No more than $\max\{\alpha(X) - 2 | X \in CCCd, X \neq \bar{0}\}$ levels were added to any of the trees, i.e. their height is $2d - 2 + d - 3 = 3d - 5$ after step 1.4. During the last substep, one more level is added to each tree, i.e. their height is 3d - 4.

Lemma 3.1:

After step 1 Ta, Tb, Tc and Td are 4 arc-disjoint trees of height $3d - 4$ and all nodes (i, X), such that $\alpha(X) \leq i \leq d - 1$ or $i = 1$, are represented in two different trees.

Proof: We already know that all *1*-nodes and α-nodes are included in two different trees -- in one tree during step 1.1 (tree T) and in another one during steps 1.3 and 1.4 (tree $\bar{T}$). In the last substep 1.5 we add the cross neighbours of all α-nodes to the tree $\bar{T}$. Those cross neighbours are in fact the parents of the α-nodes in the other tree T. Since all nodes (i, X), such that $\alpha(X) < i \leq d - 1$ are parents of α-nodes in some tree, it follows that they are all added to a second tree during the last step. The trees remain arc-disjoint, because the arcs added during step 1.5 were not used before and also every arc is added to one tree only. □

We attach Ta to T2, Tb and Td to T1, Tc to T0. Thus, T0, T1 and T2 are arc-disjoint trees and each cycle is represented in two of them. More precisely,

Lemma 3.2:

After step 1, for every cycle $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $X \in CCCd[1 = 1]$ then $\forall(\alpha(X) \leq i \leq d, 1)$ $(i, X) \in T1$,

if $X \in CCCd[1 = 0]$ then $\forall(1 \leq i \leq d - 1)$ $(i, X) \in T1$;

and

if $X \in CCCd[0, 1 = 1, 0]$ then $\forall(\alpha(X) \leq i \leq d, 1)$ $(i, X) \in T0$,

if $X \in CCCd[0, 1 = 1, 1]$ then $\forall(1 \leq i \leq d - 1)$ $(i, X) \in T0$;

and

if $X \in CCCd[0, 1 = 0, 0]$ then $\forall(\alpha(X) \le i \le d, 1)$ $(i, X) \in T2$,

if $X \in CCCd[0, 1 = 0, 1]$ then $\forall(1 \le i \le d-1)$ $(i, X) \in T2$.

After we take into account the distance between the roots of Ta, Tb, Tc, Td and the originator $(1, \bar{0})$, we can conclude that the height of T0 is $3d$, the height of T1 is $3d+1$ and the height of T2 is $3d-3$. Notice that so far only ascending straight arcs were used. Therefore, we can use the descending straight arcs, as well as some of the remaining cross arcs to make all cycles represented in all trees.

The purpose of the next step -- step 2, is to make sure that all nodes of cycles $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$ are included in all trees.

During the description of step 2, any reference to a (node from) cycle X *should be treated as a reference to a (node from) cycle* X, *such that* $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$.

In an attempt to simplify the presentation of step 2, we describe the trees separately and analyze their heights after each substep. At each substep, only free arcs are used and no arc is added to more than one of the trees. In this way, the trees are preserved arc-disjoint.

First, we want to make all *0*-nodes and *d-1*-nodes represented in all trees.

Tree T1 contains all nodes from $X \in CCCd[1 = 0]$, except for the *0*-nodes. The descending arcs $(1, X) \rightarrow (0, X)$ have not been used yet. Therefore, we can perform step 2.1:

step 2.1:

For every cycle $X \in CCCd[1 = 0]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,
add $(1, X) \rightarrow (0, X)$ to T1.

This substep adds new arcs to non-leaf nodes of T1, and therefore its height remains unchanged.

Tree T0 contains all nodes from $X \in CCCd[0, 1 = 1, 1]$, except for the *0*-nodes. We can use the same approach:

step 2.2:

For every cycle $X \in CCCd[0, 1 = 1, 1]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

add $(1, X) \rightarrow (0, X)$ to T0.

Any new node $(0, X)$ is added to a node $(1, X)$, $X \in CCCd[0, 1 = 1, 1]$. Therefore, node $(1, X)$ was included in T0 during step 1.3. Its parent is $(1, X_1)$, $X_1 \in CCCd[0, 1 = 1, 0]$. Node $(1, X_1)$ was added to T0 in step 1.2. Its parent is $(0, X_1)$. The distance from $(0, X_1)$ to node $(2, \bar{0}_0)$ is at most $2d - 4$, and if it is exactly $2d - 4$ then $z(X_1) = 0$ (see the analysis after step 1). The distance from $(2, \bar{0}_0)$ to $(1, \bar{0})$ is 4.

Therefore, the maximal distance in T0 from any new node $(0, X)$ to the root $(1, \bar{0})$ is $4 + (2d - 4) + 3$, i.e. $2d + 3$, and in the worst case $z(X) = z(X_1) = 0$.

After this substep, all *0*-nodes and *d-1*-nodes from cycles $X \in CCCd[0 = 1]$ are included in T0. The cross arcs between nodes of level 0 are still free and we can use them to add all *0*-nodes and *d-1*-nodes from the remaining cycles to T0:

step 2.3:

For every cycle $X \in CCCd[0 = 1]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

add $(0, X) \rightarrow (0, X_0)$

and $(0, X_0) \rightarrow (d - 1, X_0)$ to T0.

Two more levels are added to the *0*-nodes from cycles $X \in CCCd[0 = 1]$, i.e. to nodes included in T0 either in step 2.2 or in step 1.1. Therefore, the maximal distance from a node $(d - 1, X_0)$, $X_0 \in CCCd[0 = 0]$, to $(1, \bar{0})$ is $2d + 3 + 2$, i.e. $2d + 5$, in T0, and in the worst case $z(X_0) = z(X) = 0$.

We can expand T2 in a similar way:

step 2.4:

For every cycle $X \in CCCd[0, 1 = 0, 1]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

add $(1, X) \rightarrow (0, X)$ to T2.

For every cycle $X \in CCCd[0 = 0]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

add $(0, X) \rightarrow (0, X_0)$

and $(0, X_0) \rightarrow (d-1, X_0)$ to T2.

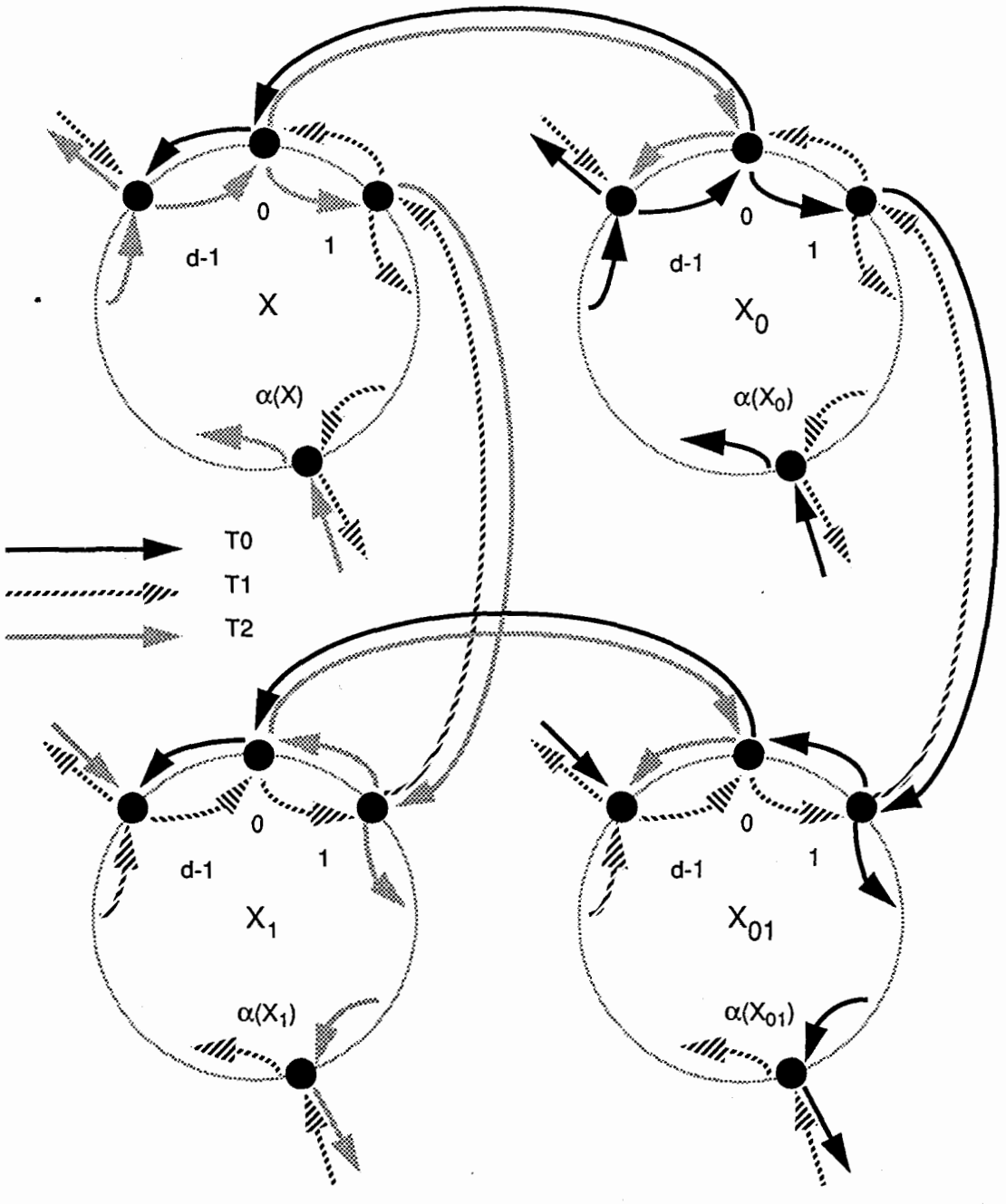


fig. 3.4. Four cycles of CCCd after step 2.4; $X \in CCCd[0, 1 = 0, 0]$.

Following the analysis for steps 2.2 and 2.3, we can conclude that the maximal distance from a node $(d-1, X_0)$, $X_0 \in CCCd[0=1]$, to $(1, \bar{0})$ is $2d+2$ in T2 (because the distance from the root $(1, \bar{0})$ to $(2, \bar{0})$ is only 1), and in the worst case $z(X_0) = 0$.

After step 2.4 all *0*-nodes and *d-1*-nodes are represented in all trees. In the following substeps of step 2 we treat separately cycles X, such that z(X) is even and cycles X, for which z(X) is odd.

step 2.5

For every cycle $X \in CCCd[0=0]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $z(X)$ is *odd* then

$\forall (d-1 \geq i \geq 2)$ add $(i, X) \rightarrow (i-1, X)$ to T0,

$\forall (\alpha(X)-1 \geq i \geq 2)$ add $(i, X) \rightarrow (i, X_i)$ to T0.

After this substep, all nodes of cycles $X \in CCCd[0=0]$, z(X) is odd, are in T0. Since z(X) is odd, i.e. $z(X) > 0$, it follows that, in T0, the maximal distance from the *d-1*-nodes of those cycles to $(1, \bar{0})$ is $2d+4$. We add $d-2$ levels to these *d-1*-nodes. Therefore, in T0, the maximal distance from any of their descendants to the root $(1, \bar{0})$ is $3d+2$.

Also, all nodes (i, X_i), $(\alpha(X)-1 \geq i \geq 2)$, $X \in CCCd[0=0]$ and $z(X)$ is odd, are included in T0. However, if $z(X)$ is odd then $z(X_i)$, $(\alpha(X)-1 \geq i \geq 2)$, is even, and X_i is in $CCCd[0=0]$ as well. Therefore, we only need to add to T0 the remaining nodes from those cycles for which z(X) is even:

step 2.6

For every cycle $X \in CCCd[0=0]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $z(X)$ is *even* then

$\forall (d-1 \geq i \geq \alpha(X)+1)$ add $(i, X) \rightarrow (i-1, X)$

and $(2, X) \rightarrow (1, X)$ to T0.

In this step, we only add one more level to nodes $(2, X)$, $X \in CCCd[0=0]$ and z(X) is even, i.e. we add one level to nodes included in T0 during step 2.5. We also add $d-1-\alpha(X)$ lev-

els to the *d-1*-nodes of the cycles. $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$ and thus $\alpha(X) > 1$. Therefore, the height of T0 after step 2.6 is $3d + 3$.

T2 is expanded in a similar way:

step 2.7

For every cycle $X \in CCCd[0 = 1]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $z(X)$ is *odd* then

$\forall (d-1 \geq i \geq 2)$ add $(i, X) \rightarrow (i-1, X)$ to T2,

$\forall (\alpha(X)-1 \geq i \geq 2)$ add $(i, X) \rightarrow (i, X_i)$ to T2.

For every cycle $X \in CCCd[0 = 1]$, $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $z(X)$ is *even* then

$\forall (d-1 \geq i \geq \alpha(X)+1)$ add $(i, X) \rightarrow (i-1, X)$

and $(2, X) \rightarrow (1, X)$ to T2.

Following the analysis for T0 and taking into account the fact that the distance from $(1, \bar{0})$ to $(2, \bar{0})$ is only 1, we can conclude that the height of T2 does not exceed the height of T0.

Lemma 3.3:

After step 2.7, for every cycle $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

if $X \in CCCd[1 = 1]$ then $\forall (\alpha(X) \leq i \leq d, 1)$ $(i, X) \in T1$,

if $X \in CCCd[1 = 0]$ then $\forall (0 \leq i \leq d-1)$ $(i, X) \in T1$;

and

if $X \in CCCd[0, 1 = 1, 0]$ then $\forall (\alpha(X) \leq i \leq d, 1)$ $(i, X) \in T0$,

if $X \in CCCd[0, 1 = 1, 1]$ then $\forall (0 \leq i \leq d-1)$ $(i, X) \in T0$,

if $X \in CCCd[0 = 0]$ then $\forall (0 \leq i \leq d-1)$ $(i, X) \in T0$;

and

if $X \in CCCd[0, 1 = 0, 0]$ then $\forall (\alpha(X) \leq i \leq d, 1)$ $(i, X) \in T2$,

if $X \in CCCd[0, 1 = 0, 1]$ then $\forall (0 \leq i \leq d-1)$ $(i, X) \in T2$,

if $X \in CCCd[0 = 1]$ then $\forall (0 \leq i \leq d-1)$ $(i, X) \in T2$.

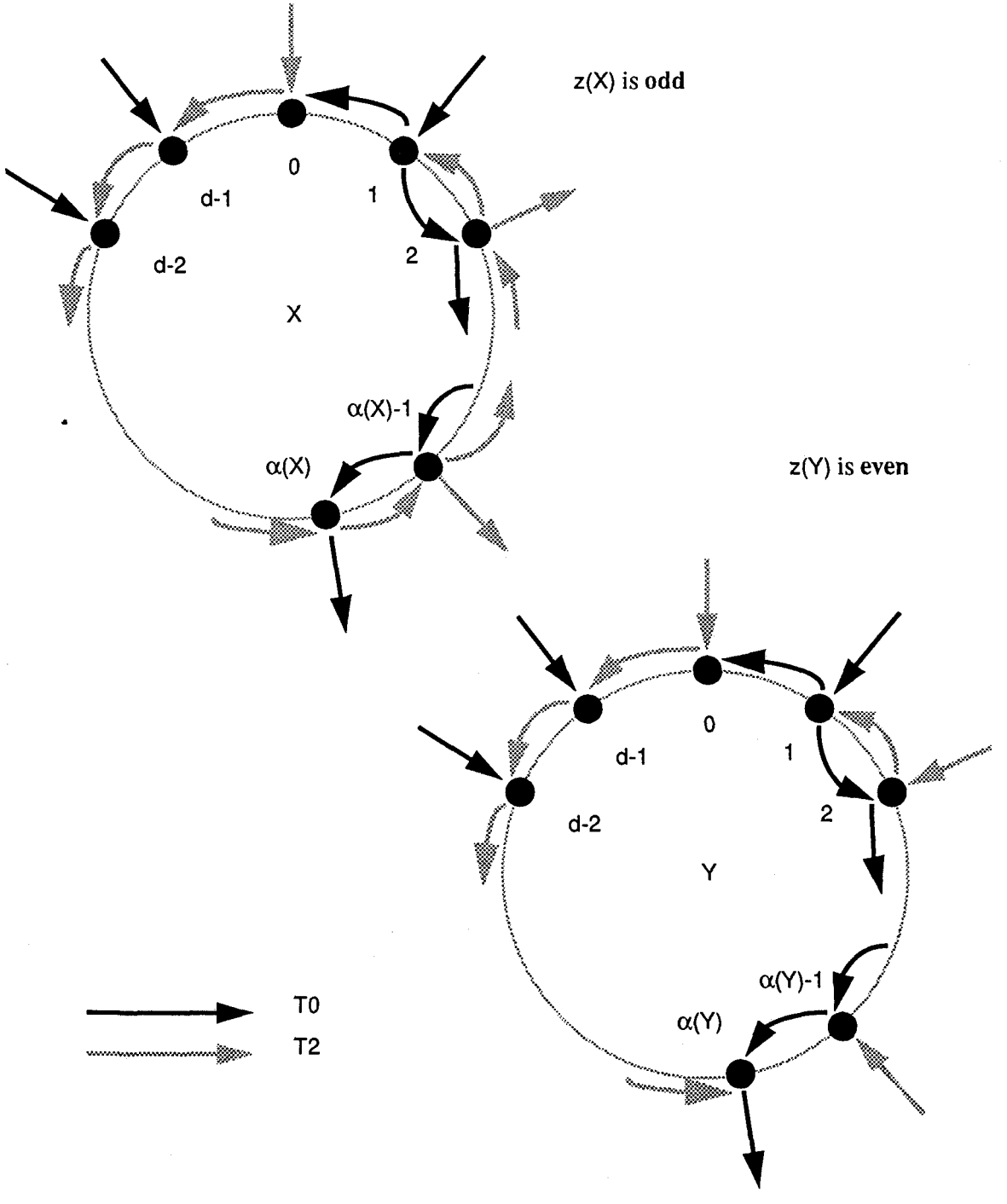


fig. 3.5. T0 and T2 in two cycles of CCCd[0,1 = 1,1] after step 2.7.

Proof: The proof follows from Lemma 3.2 and the fact that in substeps 2.1 through 2.7

tree T1 was expanded to cover all *0*-nodes of cycles $X \in CCCd[1 = 0]$, tree T0 was expanded to cover the *0*-nodes of cycles $X \in CCCd[0, 1 = 1, 1]$ and all nodes of cycles $X \in CCCd[0 = 0]$, and tree T2 was expanded to cover all *0*-nodes of cycles $X \in CCCd[0, 1 = 0, 1]$ and all nodes of cycles $X \in CCCd[0 = 1]$. □

Observe that in steps 2.5 and 2.7 not all descending and cross arcs from cycles X, z(X) is even, were used. In particular, the arcs between levels $\alpha(X)$ and 2 are still free. Therefore, we can use them now:

step 2.8

For every cycle $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$,

z(X) is even,

$\forall (\alpha(X) - 1 \geq i \geq 2)$

add $(i+1, X) \rightarrow (i, X)$

and $(i, X) \rightarrow (i, X_i)$ to T1 if $X \in CCCd[1 = 1]$ or

to T0 if $X \in CCCd[0, 1 = 1, 0]$ or

to T2 if $X \in CCCd[0, 1 = 0, 0]$.

After this substep, all nodes of all cycles $X \notin \{\bar{0}, \bar{0}_0, \bar{0}_1, \bar{0}_{01}\}$ are included in all trees. We add $\alpha(X)$ - 2 levels to nodes included to the their corresponding trees in step 1.1. Therefore, the heights of all trees remain the same.

Finally, we have to take care of the cycles containing the originators of Ta, Tb, Tc and Td to make the trees ADST. Observe that all nodes (i, Y) of those cycles, where $2 \leq i \leq d-1$, are already in two different trees after step 1. For any of the four cycles we call the trees T and $\bar{T}$. T is the tree which contains the ascending binary subtree built from node 2, Y in step 1.1 and $\bar{T}$ is the other tree.

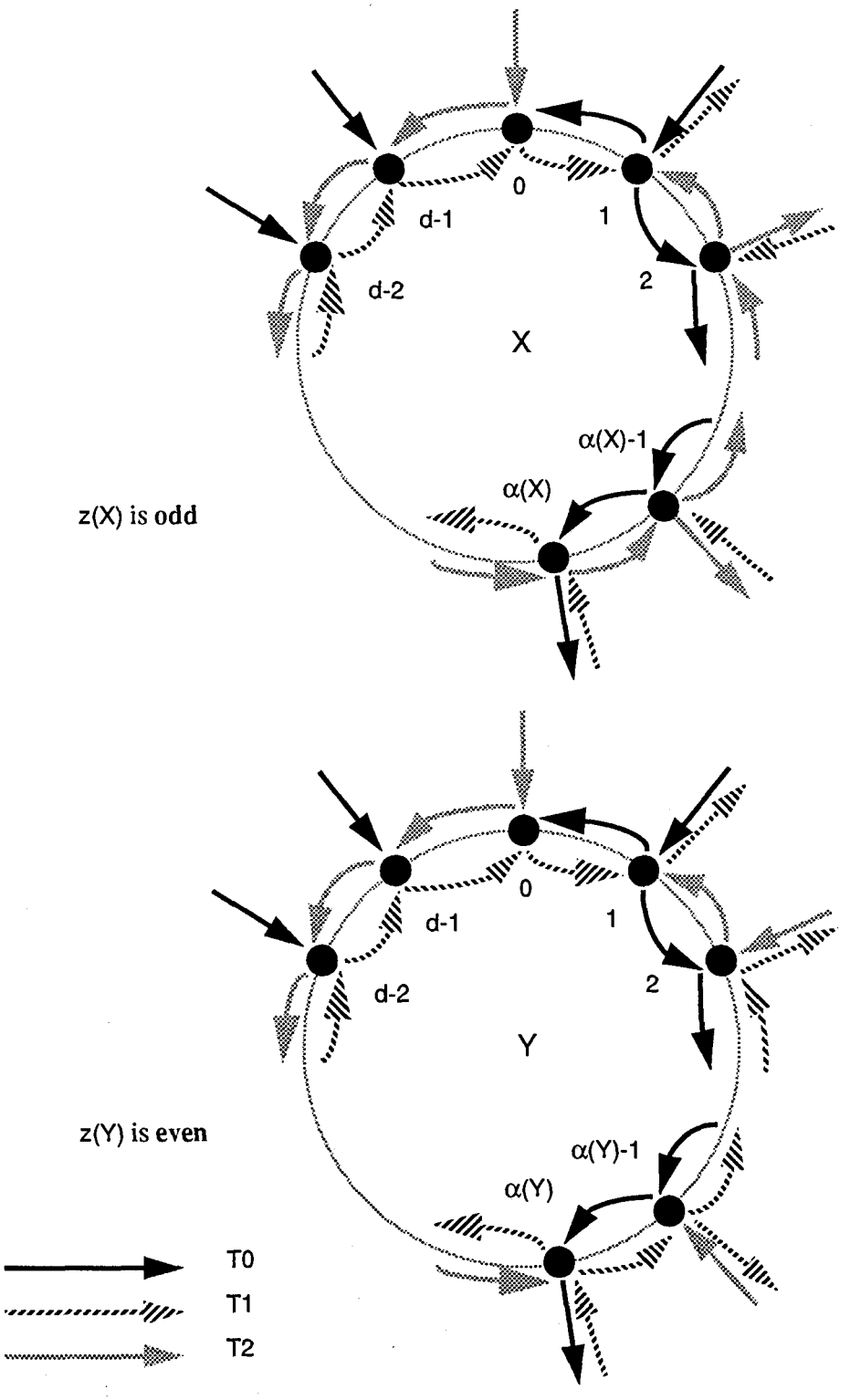


fig. 3.6. T0, T1 and T2 in two cycles of CCCd[0,1 = 1,1] after step 2.

step 3:

step 3.1 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$

add $(1, Y) \rightarrow (1, Y_1)$ to T.

step 3.2 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$

add $(1, Y) \rightarrow (0, Y)$ to $\bar{T}$.

step 3.3 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$ $\forall (d-1 \geq i \geq 2)$

add $(i+1, Y) \rightarrow (i, Y)$ to $\bar{T}$.

step 3.4 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$, $\forall (d-1 \geq i \geq 2)$

remove $(i, Y_i) \rightarrow (i, Y)$ from $\bar{T}$ and

add $(i, Y_i) \rightarrow (i, Y)$ to T2.

step 3.5 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$

add $(d-1, Y) \rightarrow (0, Y)$ and

$(2, Y) \rightarrow (1, Y)$ to T2.

step 3.6 For $Y \in \{\bar{0}_0, \bar{0}_{01}\}$

add $(0, Y) \rightarrow (0, Y_0)$ to $\bar{T}$.

step 3.7 Add $(d-1, \bar{0}_1) \rightarrow (0, \bar{0}_1)$ and $(2, \bar{0}_1) \rightarrow (1, \bar{0}_1)$ to T2.

step 3.8 For $Y \in \{\bar{0}, \bar{0}_1\}$, $\forall (d-1 \geq i \geq 2)$

add $(i+1, Y) \rightarrow (i, Y)$ to T0.

step 3.9 Add $(0, \bar{0}_1) \rightarrow (1, \bar{0}_1)$ to T0.

The purpose of the first three substeps is to ensure that all nodes from cycles $\bar{0}_0$ and $\bar{0}_{01}$ are represented in both trees -- T0 and T1. After substep 3.5, all nodes of those cycles are included in T2 as well. Substep 3.6 extends T0 to reach the cycles $\bar{0}$ and $\bar{0}_1$. The next substep is to ensure that all nodes of $\bar{0}_1$ are in T2. The last two substeps include all nodes of cycles $\bar{0}_0$ and $\bar{0}_{01}$ to T0.

In all four cycles we build branches of T0 or T1 of height d-2 starting from nodes of small distances from $(1, \bar{0})$ (6 at the most -- the extension of T0 in $\bar{0}_1$). We add at most two levels to T2 to nodes reached during step 1.1 or step 2, and since its height after step 2 was $3d-1$, it follows that the height of the trees remains at most $3d+3$. This construction does not interfere with the arcs used at the previous steps, therefore:

Theorem 3.1:

There exist 3 ADST in CCCd of maximum height $3d+3$.

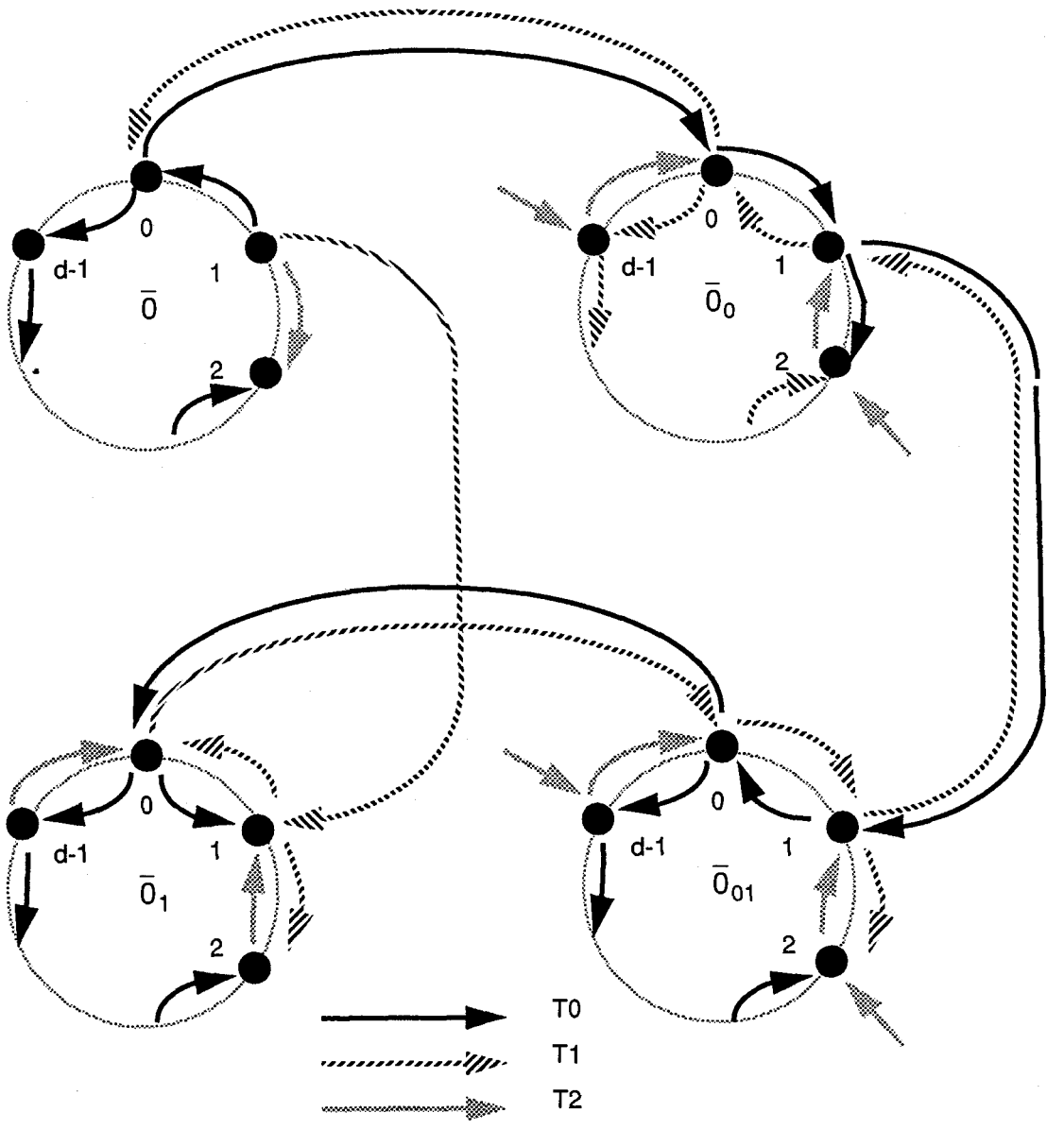


fig. 3.7. Arcs added to T0, T1 and T2 during steps 0 and 3.

3.2 Four Arc-Disjoint Spanning Trees in the Butterfly graph

In this section we introduce an algorithm for building four ADST of BFd of height 2d+1,

rooted at node $(0, \bar{0})$. We call the four trees T0, T1, T2 and T3. They are constructed in two phases. During the first phase we build the trees in such a way that each of them covers half of the nodes in BFd. During the next phase we expand the trees to include all nodes of BFd.

3.2.1 Phase 1.

First we build two ascending binary subtrees of BFd of height d-1 rooted at $(1, \bar{0})$ (T0) and at $(1, \bar{0}_0)$ (T1). Their leaves are *d*-nodes, i.e. *0*-nodes. All cycles of BFd[0 = 0] are represented in T0 and all cycles of BFd[0 = 1] are represented in T1 (Lemma 2.2). The two sets of cycles are disjoint and therefore T0 and T1 are vertex disjoint and therefore arc disjoint.

We connect both trees to the originator $(0, \bar{0})$ (fig.3.7a). The two trees in BF4 are shown on fig.3.7a. According to Lemma 2.1 each cycle of BFd is first reached through its $\alpha 1$-node in either T0 or T1. Thus, following Property 2.4, we can state that T0 and T1 are two arc-disjoint trees rooted at $(0, \bar{0})$, such that:

$$\forall (X \in BFd[0 = 0])\, \forall (\alpha 1(X) \le i \le d)\ ((i, X) \in T0) \text{ and}$$

$$\forall (X \in BFd[0 = 1])\, \forall (\alpha 1(X) \le i \le d)\ ((i, X) \in T1).$$

We extend T0 and T1 to cover all nodes in their corresponding cycles, using the following algorithm:

$$\forall (X \in BFd[0 = 0])$$

$$\forall (0 \le i \le \alpha 1(X) - 2) \text{ add } (i, X) \rightarrow (i+1, X) \text{ to T0 and}$$

$$\forall (X \in BFd[0 = 1])$$

$$\forall (0 \le i \le \alpha 1(X) - 2) \text{ add } (i, X) \rightarrow (i+1, X) \text{ to T1.}$$

Notice that arcs are added only within the cycles. The new arcs were not used before. Therefore, T0 and T1 are still arc-disjoint and also they cover all nodes in their cycles:

Lemma 3.4:

After phase 1 T0 and T1 are two arc-disjoint trees rooted at $(0, \bar{0})$ and:

$$\forall ((i, X), X \in BFd[0 = 0])\ (i, X) \in T0 \text{ and}$$

$$\forall ((i, X), X \in BFd[0 = 1])\ (i, X) \in T1.$$

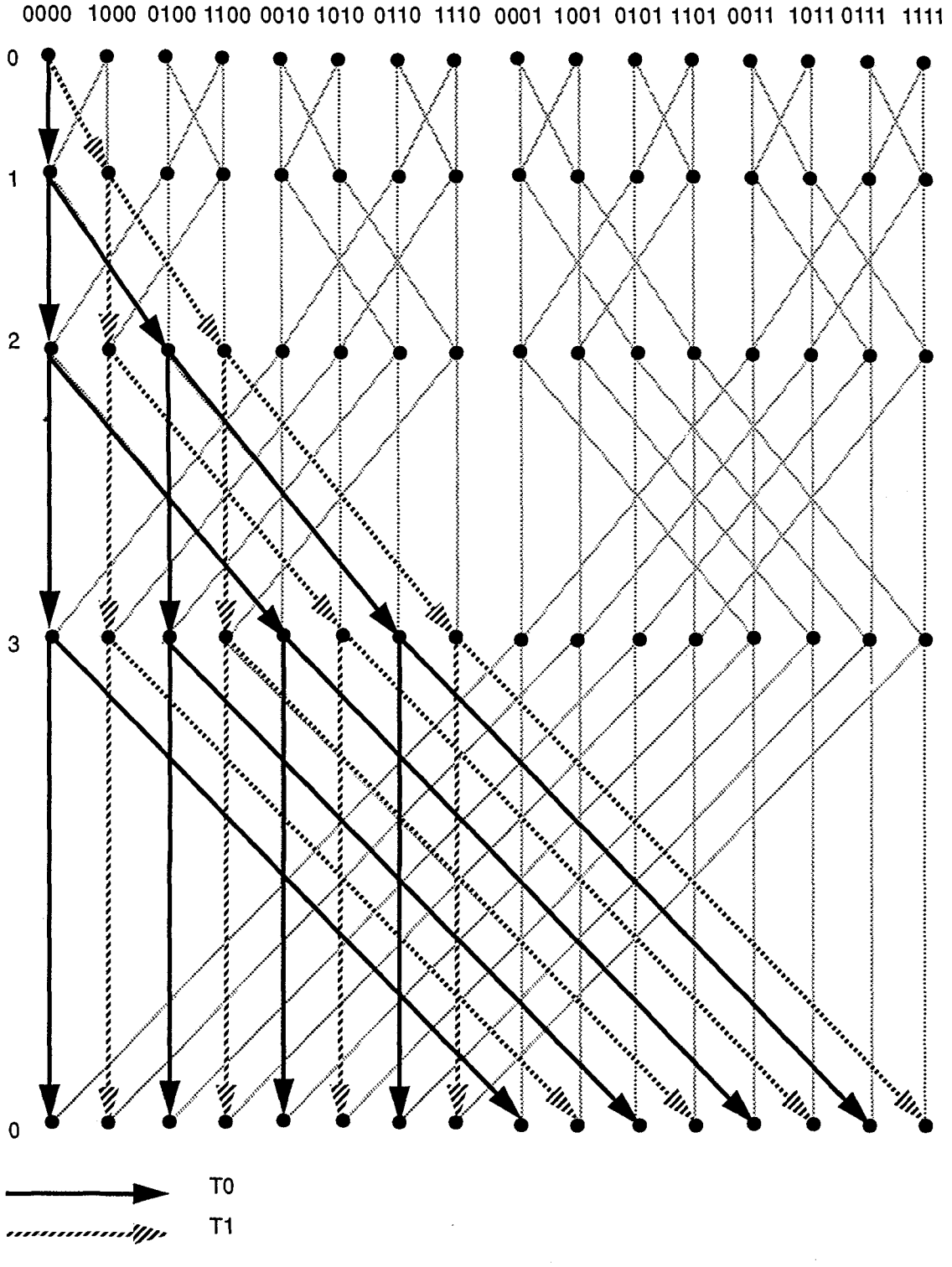


fig.3.7a. T0 and T1 in BF4.

During phase 1 only ascending arcs were used to build the two trees. Here is an important property of the arcs of BFd:

Property 3.1:

All arcs of BFd are either ascending or descending and each arc is of exactly one kind.

Proof: Let consider an arbitrary node (i, X). According to the definition of BFd it has four arcs: to (i+1, X), (i+1, X_i), (i-1, X) and (i-1, X_{i-1}), where i+1 stands for i+1 mod d and i-1 stands for i-1 mod d. Apparently, the first two arcs are ascending and the second two arcs are descending.

Let suppose that there exists a descending arc $(j, X) \rightarrow (j-1, X)$ which is identical to $(i, X) \rightarrow (i+1, X)$. Then j = i and (j-1 mod d) = (i+1 mod d), i.e. (i-1 mod d) = (i+1 mod d), which is only true for $d = 2$. However, in our presentation we consider BFd's such that $d \geq 3$, therefore the arc $(i, X) \rightarrow (i+1, X)$ is only ascending.

Suppose that there exists a descending arc $(j, X) \rightarrow (j-1, X_{j-1})$ identical to $(i, X) \rightarrow (i+1, X_i)$. Then again i = j and (j-1 mod d) = (i+1 mod d), which is impossible for $d \geq 3$. Therefore, the arc $(i, X) \rightarrow (i+1, X_i)$ is only ascending. Similarly, we can show that the other two arcs of (i, X) are only descending. □

Since we only used ascending arcs so far it is possible to build two other arc-disjoint trees with similar properties using only descending arcs. We construct two descending binary subtrees of BFd of height d-1 rooted at $(d-1, \bar{0})$ (T2) and at $(d-1, \bar{0}_{d-1})$ (T3). Again, those two trees are arc disjoint (consequence of Lemma 2.4). The first node added to each cycle represented in those trees is the cycle's β-node (Lemma 2.3). We connect the trees to the originator $(0, \bar{0})$ (fig.3.7b) and extend them to cover all nodes in their cycles:

$\forall (X \in BFd\,[d-1=0])$

$\forall (d \geq i \geq \beta(X)+2)$ add $(i, X) \rightarrow (i-1, X)$ to T2 and

$\forall (X \in BFd\,[d-1=1])$

$\forall (d \geq i \geq \beta(X)+2)$ add $(i, X) \rightarrow (i-1, X)$ to T3.

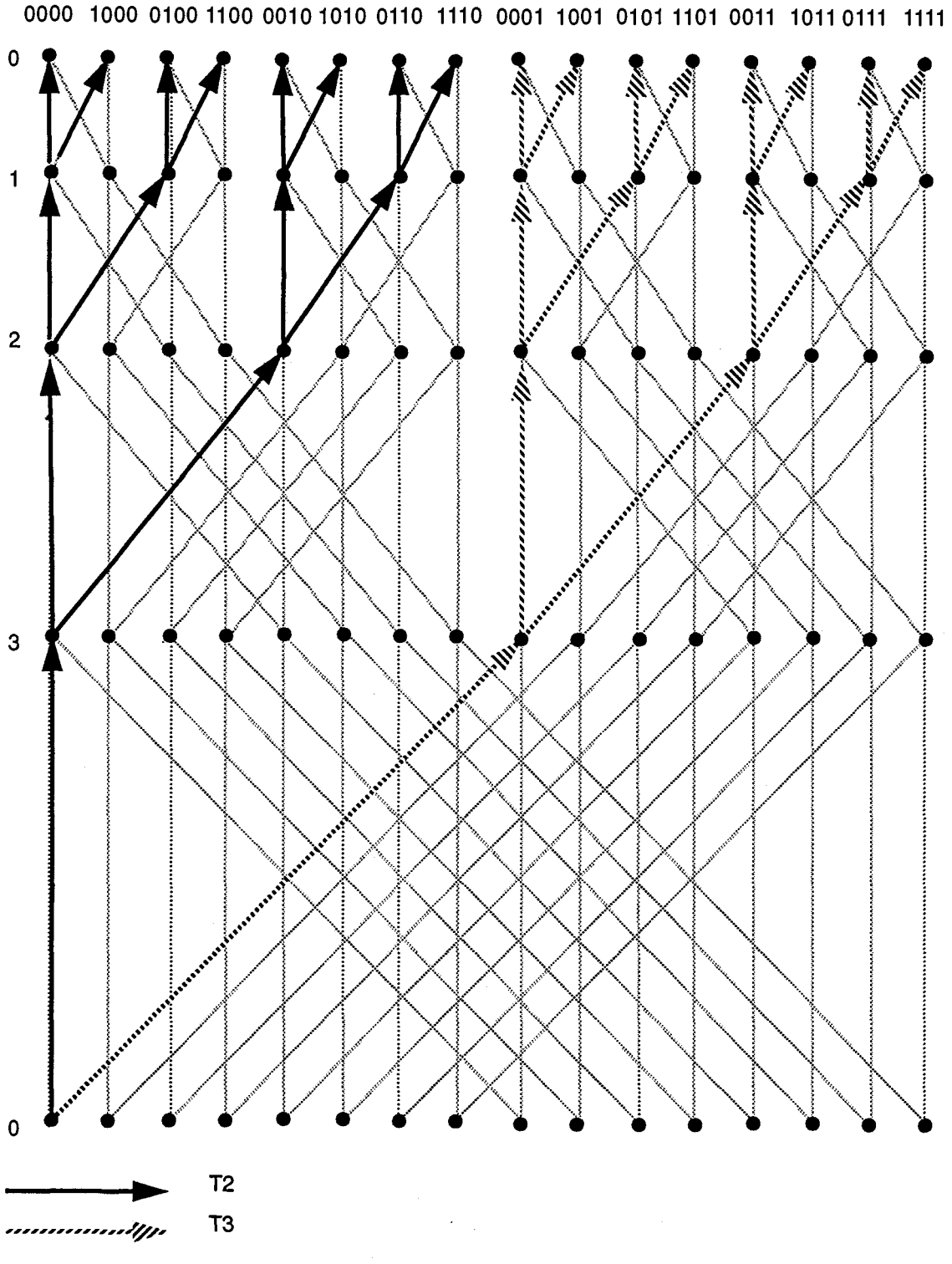


*fig. 3.7b.*T2 and T3 in BF4.

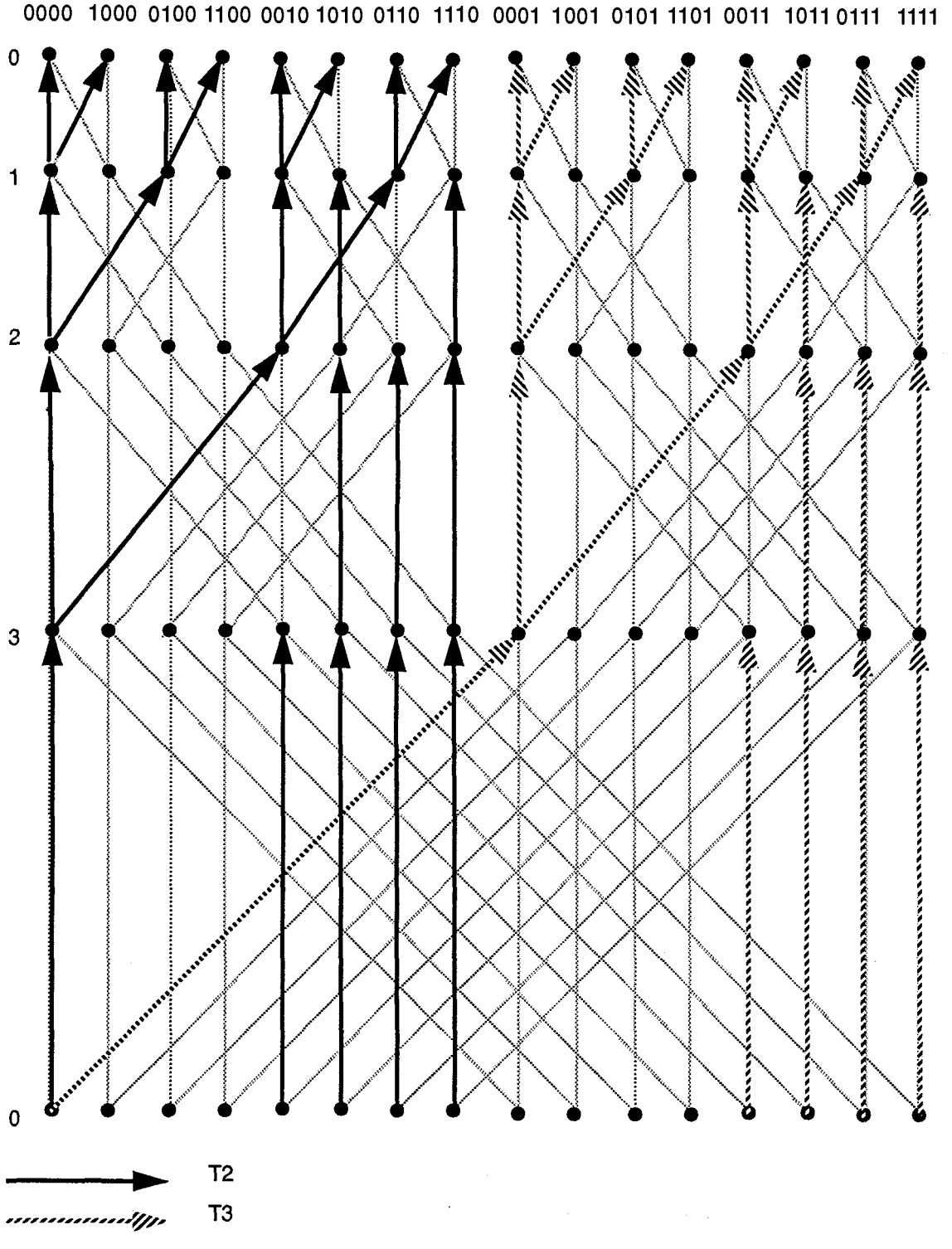


fig. 3.8. T2 and T3 in BF4 after phase 1. The extensions covering all nodes in a cycle are only shown for four cycles.

Only descending arcs are used and the trees are preserved arc-disjoint. Therefore, T0, T1, T2 and T3 are mutually arc-disjoint after phase 1 and

Lemma 3.5:

After phase 1 T2 and T3 are two arc-disjoint trees, such that:

$\forall((i, X), X \in BFd[d-1=0])\ ((i, X) \in S2)$ and

$\forall((i, X), X \in BFd[d-1=1])\ ((i, X) \in S3)$.

3.2.2 Phase 2

During this phase we extend all trees to cover all nodes of BFd. We use only ascending arcs for T0 and T1 and only descending arcs for T2 and T3. Again, we first show how to extend T0 and T1 and then apply the same approach to T2 and T3.

Let us look at T0 and T1 after phase 1 (fig. 3.8 shows the other two trees after phase 1). Each of them covers half of the cycles of BFd. All straight ascending arcs are used except for the ones pointing to $\alpha 1$-nodes, since all $\alpha 1$-nodes were reached via cross arcs. Also, we showed that only few of the ascending cross arcs were used during phase 1 -- those pointing to the $\alpha 1$-nodes. But the $\alpha 1$-nodes have a straight in-coming ascending arc free and thus we can inform them via that straight arc.

Our algorithm starts from all *0*-nodes, which are leaves of a tree T (T0 or T1). For every *0*-node, we add its ascending cross arc to T if the arc was not used before. Thus, all nodes at level 1 of the opposite tree but the $\alpha 1$-nodes are included in T. In fact, the only $\alpha 1$-node at level 1 with already used in-coming ascending cross arc is $(1, \bar{0}_0)$. At the end of the algorithm we show how to add this node to T0.

Now for every node (1, X) included to T in the previous step we add its out-going ascending cross arc to T if the arc was not used before. After that all nodes at level 2 but the $\alpha 1$-nodes are included in both trees. Since we have already included all *1*-nodes, except $(1, \bar{0}_0)$, in both trees, we can reach the $\alpha 1$-nodes of level 2 using straight arcs of the type $(1, X) \rightarrow (2, X)$. Therefore, after this step all nodes of level 2 are informed in both trees. We continue with this procedure level

by level until all nodes of level d are reached. Then we add the arc $(0, \bar{0}_0) \rightarrow (1, \bar{0}_0)$ to T0.

We formally describe the algorithm and prove that after phase 2 T0 and T1 are two ADST of BFd.

step 0: For every node $(0, X)$, $X \neq \bar{0}$, from tree T (T0 or T1), add $(0, X) \rightarrow (1, X_0)$ to the same tree T.

Notice that all these arcs are not used yet, since ascending cross arcs were only used so far to add $\alpha 1$-nodes. $\alpha 1(X) = 1$ only for $\bar{0}_0$ and the arc $(0, 0) \rightarrow (1, \bar{0}_0)$ is not used during this step.

step i: For every node (i, X), $\alpha 1 (X_i) \neq i + 1$, added to some tree T (T0 or T1) during step i-1, add
$(i, X) \rightarrow (i+1, X_i)$ to T.
For every node (i, X), $\alpha 1(X) = i + 1$, added to T during step i - 1, add
$(i, X) \rightarrow (i+1, X)$ to T;

repeat step i for all $1 \leq i \leq d-1$.

Those arcs are not used yet, since ascending cross arcs from nodes (i, X) of T were used so far to add nodes of the type $(\alpha(X), X)$ to T and the arcs of that type are not used during these steps.

step d: Add $(0, \bar{0}_0) \rightarrow (1, \bar{0}_0)$ to T0.

First we have to show that after this algorithm is performed both T0 and T1 are still trees, i.e. every node from T0 (from T1) is reached exactly once. We already know that after phase 1 T0 (T1) is a tree. After phase 1 T0 (T1) covers nodes (i, X) such that X[0] = 0 (X[0] = 1 for T1) (Lemma 3.4).

During step 0 of phase 2 nodes (0, X) of tree T0 (tree T1) are connected to nodes $(1, X_0)$. According to the definition, $X_0[0] = \overline{X[0]} = 1$ ($X_0[0] = 0$ for T1). Therefore, the new nodes added to T0 (T1) belong to cycles never reached in T0 (T1) before. In each subsequent step j of phase 2 we add to T0 (T1) nodes of levels j+1, therefore they were not included in T0 (T1) during the previous steps of phase 2. Also, the new nodes belong to cycles with labels different from the labels

of the parents' cycles in at most one bit, which is at position different from 0. Therefore, all new nodes added to T0 are from cycles X, such that X[0] = 1 (all new nodes added to T1 are from cycles X, such that X[0] = 0). Therefore, those new nodes were not included in T0 (T1) during phase 1. Therefore, all new nodes are added to T0 (T1) exactly once. ❐

Lemma 3.6:

T0 and T1 are arc-disjoint spanning trees of BFd.

Proof: T0 and T1 are arc-disjoint after phase 1 (Lemma 3.4). In phase 2 only unused arcs are added to the trees and no arc is added to both of them. Therefore the trees remain arc-disjoint. Also, nodes informed during phase 1 in T0 (T1) are never reached again in the same tree. T0 (T1) covers all nodes in BFd[0 = 0] (BFd[0 = 1 for T1) after phase 1 (Lemma 3.4), i.e. each of the trees had $d2^{d-1}$ nodes. During phase 2 $d2^{d-1}$ new nodes are added to each tree ($2^{d-1}-1$ to T0 and 2^{d-1} to T1 in step 0, $(d-1)$ (2^{d-1}) to both trees in steps 1 through d-1 and 1 to T0 at step d). Therefore, T0 and T1 contain $d2^d$ different nodes and thus they are spanning trees of BFd. ❐

Only ascending arcs were used to build the trees. Therefore, we can extend T2 and T3 in a similar manner using the remaining descending arcs:

step 0: For every node $(0, X)$, $X \neq \bar{0}$, from tree T (T0 or T1), add $(0, X) \rightarrow (d-1, X_{d-1})$ to the same tree T.

All these arcs are not used yet, since descending cross arcs were only added so far to nodes of the type $(\beta(X), X)$. $\beta(X) = d\text{-}1$ only for $\bar{0}_{d-1}$ and the arc $(0, 0) \rightarrow (d-1, \bar{0}_{d-1})$ is not used during this step.

step i: For every node (i, X), $\beta(X_{i-1}) \neq i-1$, added to some tree T during step i-1, add
$(i, X) \rightarrow (i-1, X_{i-1})$ to T.
For every node (i, X), $\beta(X) = i\text{-}1$, added to T during step i-1, add
$(i, X) \rightarrow (i-1, X)$ to T as well;

repeat step i for all $1 \leq i \leq d-1$.

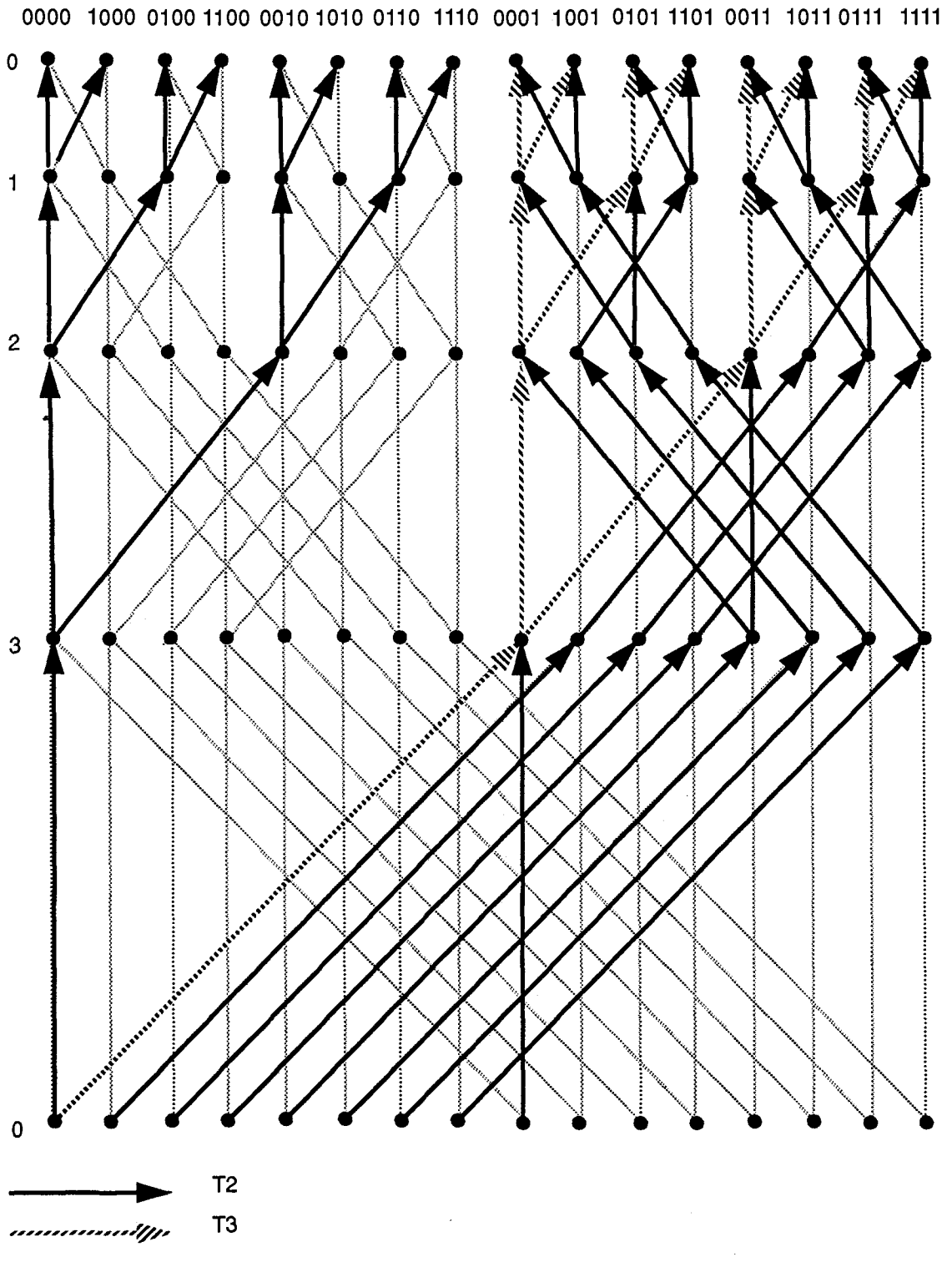


fig. 3.9. Arcs added to T2 in BF4 during phase 2. Arcs added to T2 and T3 in the beginning of phase 1 are also shown.

All these arcs are not used yet, since descending cross arcs originating from (i, X) were used so far to add nodes of the type ($\beta(X)$, X) and the arcs of that type are not used during these steps.

step d: Add $(0, \bar{0}_{d-1}) \rightarrow (d-1, \bar{0}_{d-1})$ to T2.

Clearly, all four trees are arc-disjoint and T2 and T3 are 2 ADST of BFd.

Theorem 3.2:

There exist 4 ADST of BFd rooted at $(0, \bar{0})$ of maximum height 2d+1.

Proof: During phase 1 we build binary subtrees of height d-1 and add a single arc from $(0, \bar{0})$ to their roots. Then we add at most d-1 levels to their leaves. Therefore, all four trees are of height $2d-1$ after phase 1. Phase 2 takes d+1 steps, i.e. at most d+1 new levels are added to nodes (0, X), which are of distance d from $(0, \bar{0})$ in any of the trees. Therefore, the height of any of the four trees is at most $2d+1$. □

3.3 Two Arc-Disjoint Spanning Trees in the Butterfly graph

We build an ascending binary subtree of BFd of height d rooted at $(0, \bar{0})$ and call it T0. We build a descending binary subtree of BFd of height d rooted at $(0, \bar{0})$ and call it T1. According to Lemmas 2.1 and 2.2 all cycles of BFd are represented in both trees and are first reached through their $\alpha 1$-nodes in T0 and their β-nodes in T1. Therefore, according to Property 2.5 we can state:

Lemma 3.7:

For every cycle $X \in BFd$ $\forall(\alpha 1(X) \le i \le d)$ $(i, X) \in T0$ and

$\forall(0 \le i \le \beta(X))$ $(i, X) \in T1$.

We remove arc $(d-1, \bar{0}) \rightarrow (0, \bar{0})$ from T0 and arc $(1, \bar{0}) \rightarrow (0, \bar{0})$ from T1, since there is no need to send the message to the root $(0, \bar{0})$. We consider two types of cycles of CCCd.

We expand the trees in each cycle X for which $\alpha 1(X) - \beta(X) \ge \left\lfloor \frac{d}{2} \right\rfloor$ as follows:

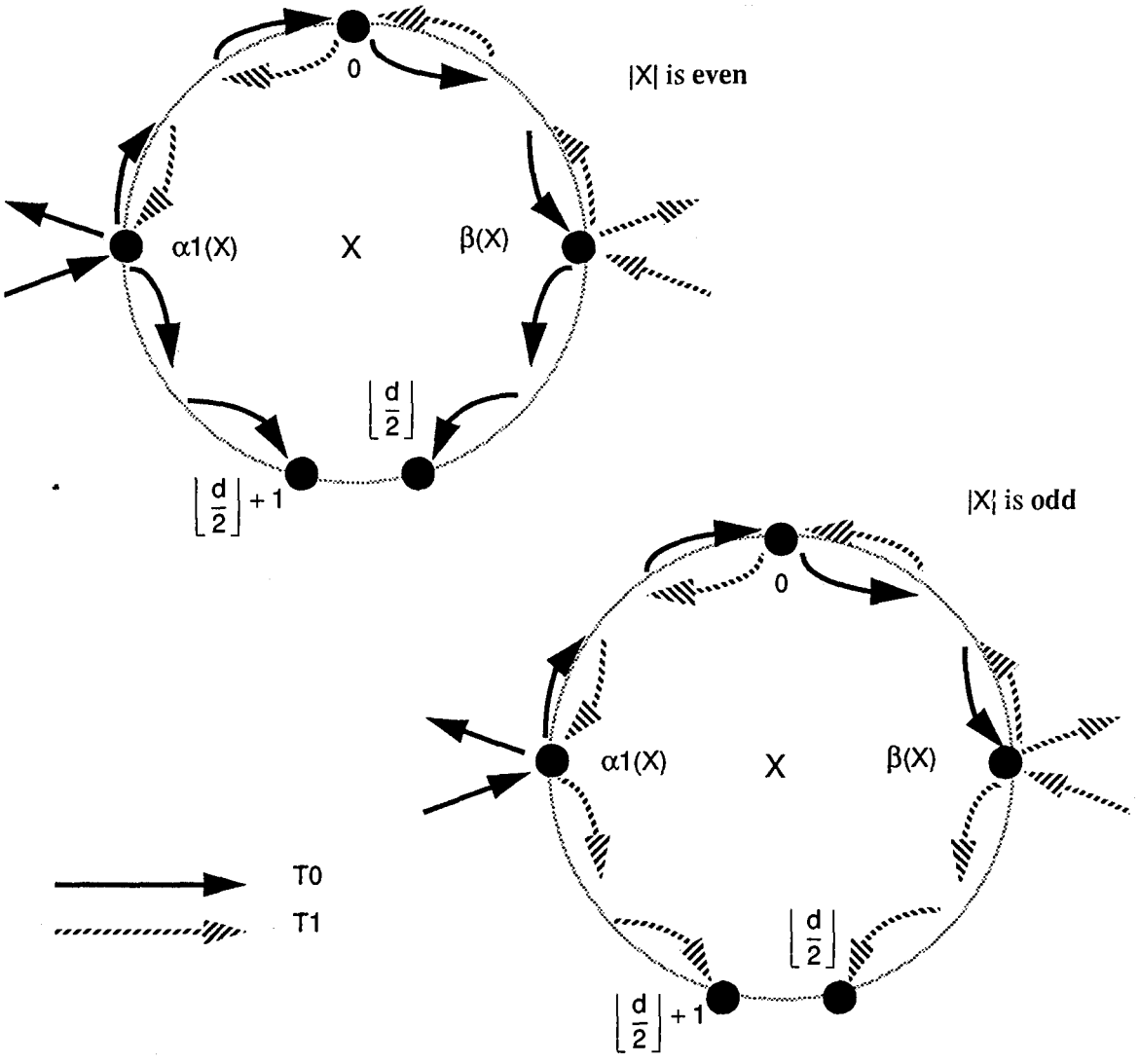


fig. 3.10. The new arcs are shown in the cycles.

|X| is even.

$\forall (0 \le i \le \lfloor d/2 \rfloor - 1)$ add $(i, X) \rightarrow (i+1, X)$ to T0;

$\forall (\alpha 1(X) \ge i \ge \lfloor d/2 \rfloor + 2)$ add $(i, X) \rightarrow (i-1, X)$ to T0 and

$\forall (d \ge i \ge \alpha 1(X) + 1)$ add $(i, X) \rightarrow (i-1, X)$ to T1.

Since $\alpha 1(X) - \beta(X) \ge \lfloor d/2 \rfloor$ it follows that $\alpha 1(X) > \lfloor d/2 \rfloor$ and therefore no ascending straight arcs of levels lower than $\lfloor d/2 \rfloor$ were added to T0 before. It also follows that

$\beta(X) < \lfloor d/2 \rfloor$ and therefore no descending arcs between levels higher than $\lfloor d/2 \rfloor$ were added to T0 before. Therefore, it is possible to perform the step.

We add at most $\lfloor d/2 \rfloor$ new levels to either tree in those cycles. The trees remain arc-disjoint and all nodes of cycles X, $\alpha 1(X) - \beta(X) \geq \lfloor d/2 \rfloor$ and $|X|$ is even, are represented in T0. However, the nodes (i, X), where $\beta(X) + 1 \leq i \leq \alpha 1(X) - 1$ are only in T0, but not in T1.

$|X|$ is odd.

$\forall (d \geq i \geq \lfloor d/2 \rfloor + 2)$ add $(i, X) \rightarrow (i-1, X)$ to T1;

$\forall (\beta(X) \leq i \leq \lfloor d/2 \rfloor - 1)$ add $(i, X) \rightarrow (i+1, X)$ to T1 and

$\forall (0 \leq i \leq \beta(X) - 1)$ add $(i, X) \rightarrow (i+1, X)$ to T0.

We add at most $\lfloor d/2 \rfloor$ new levels to both trees, while preserving them arc-disjoint. All nodes of those cycles are in T1. The nodes (i, X), where $\beta(X) + 1 \leq i \leq \alpha 1(X) - 1$ are only in T1, but not in T0. However, their cross neighbours are in a cycle Y such that $\alpha 1(Y) = \alpha 1(X)$, $\beta(Y) = \beta(X)$ and $||X| - |Y|| = 1$. Therefore $\beta(Y) + 1 \leq i \leq \alpha 1(Y) - 1$ and $|Y|$ is even and thus all nodes of Y are in T0. Also, the cross links of the nodes (i, X), $\beta(X) + 1 \leq i \leq \alpha 1(X) - 1$, are not used yet. Therefore, we can use these cross links to add the cross neighbour of (i, X) to T1 and to add (i, X) to T0. This means that the height of the trees increases by at most 1 and is therefore at most $d + \lfloor d/2 \rfloor + 1$. All together $\lfloor d/2 \rfloor + 1$ levels are added to the trees in those cycles.

For the rest of the cycles X, such that $\alpha 1(X) - \beta(X) < \lfloor d/2 \rfloor$, do the following:

$\forall (\beta(X) \leq i \leq \alpha 1(X) - 1)$ add $(i, X) \rightarrow (i+1, X)$ to T1 and
add $(i+1, X) \rightarrow (i, X)$ to T0.

Since $\alpha 1(X) - \beta(X) < \lfloor d/2 \rfloor$ it follows that at most $\lfloor d/2 \rfloor$ new levels are added to the leaves to T0 and T1, which are of distance from $(0, \bar{0})$ not exceeding d, and all nodes (i, X), $\beta(X) \leq i \leq \alpha 1(X)$, are in both trees while the trees are still arc-disjoint because those arcs were not used before. Notice that every other node (i, X) from those cycles has a cross arc in T0/T1 which connects it to a node $(\alpha(Y), Y)$ (resp. $(\beta(Y), Y)$) in some cycle Y, such that $\alpha 1(X) - \beta(X) < \lfloor d/2 \rfloor$. Therefore, this node of Y is already represented in both trees. The

reverse arc of the cross link connecting the two nodes has not been used yet. We add (i, X) to T1 (T0) using the reverse arc, thus increasing the distance from the root of the trees by at most 1.

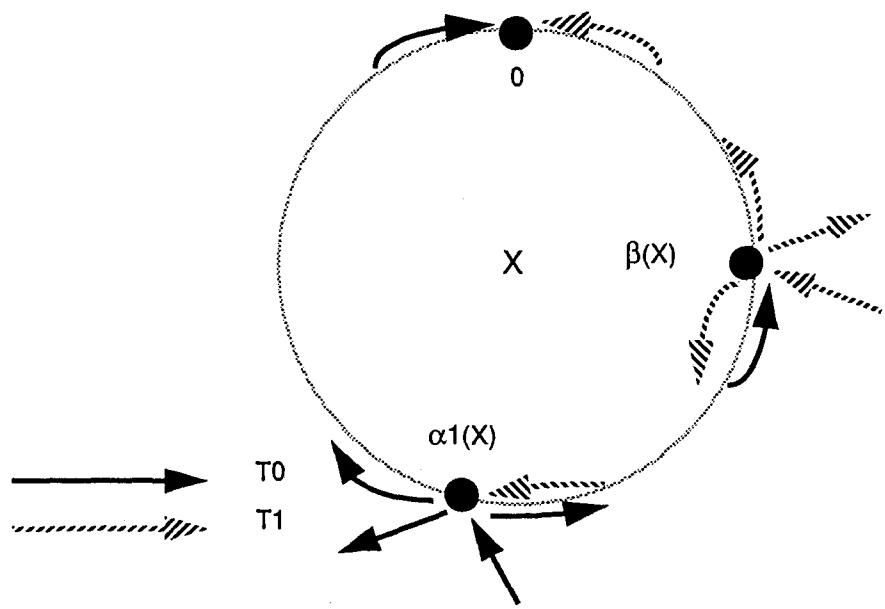


fig. 3.11.

All nodes of BFd are represented in both trees. Therefore,

Theorem 3.3:

There exist two ADST of BFd of height $d + \left\lfloor \frac{d}{2} \right\rfloor + 1$.

3.4 Two Arc-Disjoint Spanning Trees in the Cube-Connected Cycles graph

We build an ascending binary subtree of height d-1 rooted at $(1, \bar{0})$ and call it T0. We build another ascending binary subtree T1 rooted at $(1, \bar{0}_0)$ of height d-1. T0 and T1 are arc-disjoint and according to Lemma 2.2 all cycles of CCCd[0 = 0] are represented in T0 and all cycles of CCCd[0 = 1] are represented in T1. The height of the trees is $2(d-1)$, since every non-leaf node has a compound arc to one of its children. We add $(0, \bar{0}) \rightarrow (1, \bar{0})$ to T0 and $(0, \bar{0}) \rightarrow (0, \bar{0}_0) \rightarrow (1, \bar{0})$ to T1 and thus their height increases to $2d-1$ and $2d$ respectively.

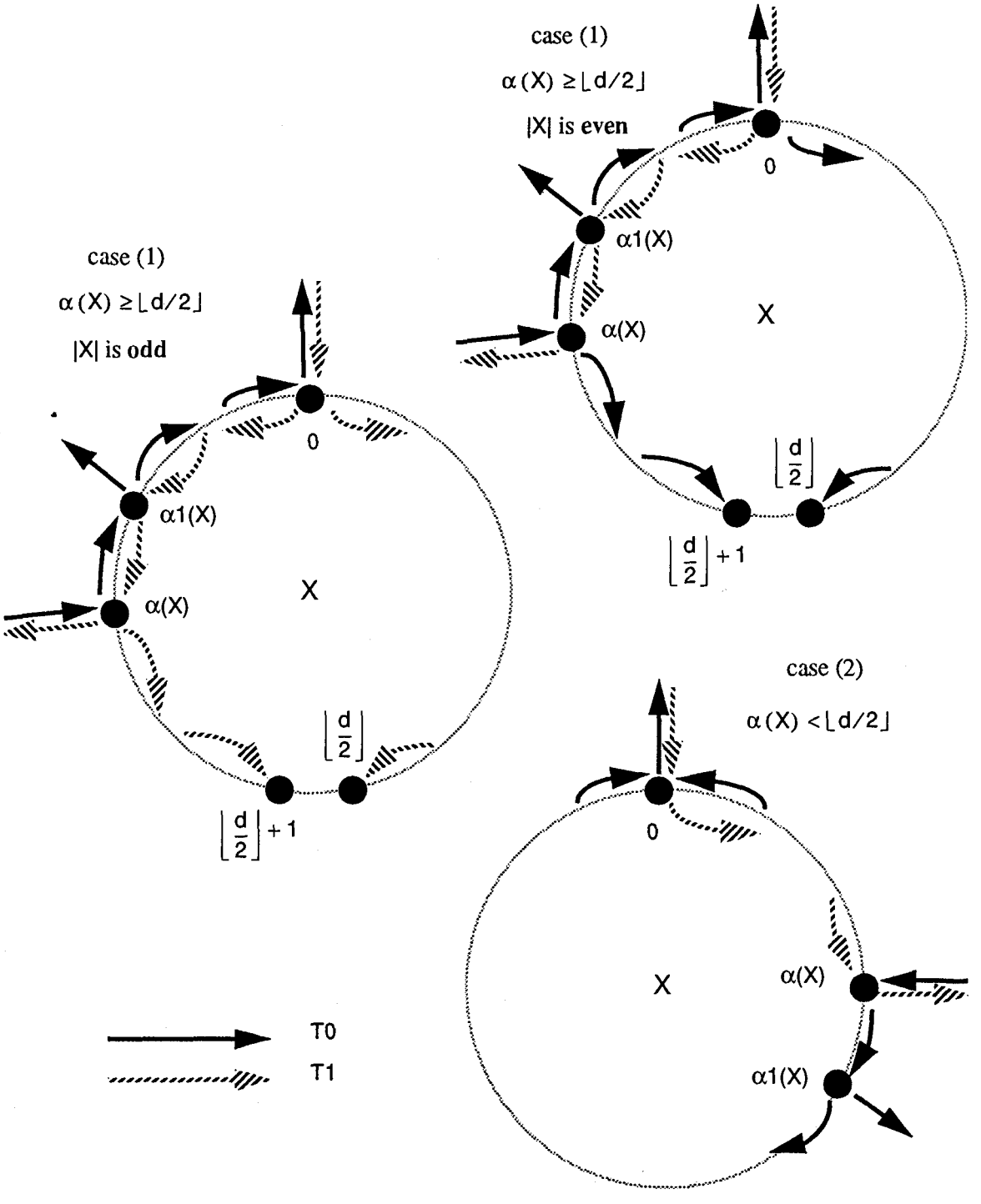


fig. 3.11a. Using the algorithms from section 3.3 for CCCd. All cycles shown are from CCCd [0 = 0].

All leaves of T0 and T1 are *0*-nodes. For each X, such that $(0, X)$ is in some tree T we add $(0, X) \rightarrow (0, X_0)$ to T. The height of both trees increases by 1 and is 2d for T0 and 2d + 1 for T1.

In both trees the first node reached of each cycle $X \in T$ is the cycle's α-node. This is due to Lemma 2.1 and the fact that compound arcs are used to build the trees and their hop-nodes are the α-nodes of the cycles. Thus, in every cycle X all nodes (i, X), $\alpha(X) \leq i \leq d-1$ are in some tree T and node $(0, X)$ is in both trees.

This is very similar to what occurs when 2 ADST in BFd are built (see section 3.3) and therefore, we can extend the trees using the same approach. We consider again two types of cycles: (1) cycles X, such that $\alpha(X) \geq \lfloor d/2 \rfloor$ and (2) cycles X, such that $\alpha(X) < \lfloor d/2 \rfloor$.

For case (1) we apply the algorithm used for case (1) in section 3.3. The only difference is that we use $\alpha(X)$ instead of $\alpha 1(X)$ and 0 instead of $\beta(X)$. Again, the algorithm adds at most $\lfloor d/2 \rfloor + 1$ new levels to the trees and preserves them arc-disjoint. Similarly, for case (2) we use the algorithm for case (2) from section 3.3. The trees are again arc-disjoint and they cover all nodes of CCCd. Therefore:

Theorem 3.4:

There exist two ADST in CCCd of height $2d + \left\lfloor \frac{d}{2} \right\rfloor + 2$.

3.5 Upper bounds on broadcasting in CCCd and BFd under the various models

Given the arc-disjoint trees described in this chapter and the broadcasting algorithms shown in Chapter 1 we can prove several upper bounds on the broadcasting time under various models.

3.5.1 Full-duplex links

We use directly the results from this chapter to show upper bounds on the broadcasting time in CCCd and BFd for the link-bound and the processor-bound communication.

Lemma 3.8:

$$B_{F^*}(CCCd) \le \left(\sqrt{\frac{L\tau}{2}} + \sqrt{(2d + \left\lfloor \frac{d}{2} \right\rfloor + 1)\beta}\right)^2 \quad \text{(2 ADST)},$$

$$B_{F^*}(CCCd) \le \left(\sqrt{\frac{L\tau}{3}} + \sqrt{(3d+2)\beta}\right)^2 \quad \text{(3 ADST)},$$

$$B_{F^*}(BFd) \le \left(\sqrt{\frac{L\tau}{2}} + \sqrt{(d + \left\lfloor \frac{d}{2} \right\rfloor)\beta}\right)^2 \quad \text{(2 ADST)},$$

$$B_{F^*}(BFd) \le \left(\sqrt{\frac{L\tau}{4}} + \sqrt{2d\beta}\right)^2 \quad \text{(4 ADST)}.$$

• **Proof:** All bounds are derived from Lemma 1.6 by substituting the values for 2 and Δ ADST of CCCd and BFd respectively (Theorems 3.4, 3.1, 3.3 and 3.2). □

The chromatic index of CCCd is q(CCCd) = 3 and q(BFd) = 4 (see fig. 3.12). Therefore, we can use theorems 3.4, 3.1, 3.3, 3.2 and apply Lemma 1.7 to obtain upper bounds on the broadcasting time in CCCd and BFd for processor-bound communication:

Lemma 3.9:

$$B_{F_1}(CCCd) \le \left(\sqrt{\frac{3L\tau}{2}} + \sqrt{2(2d + \left\lfloor \frac{d}{2} \right\rfloor + 2)\beta}\right)^2 \quad \text{(2 ADST)},$$

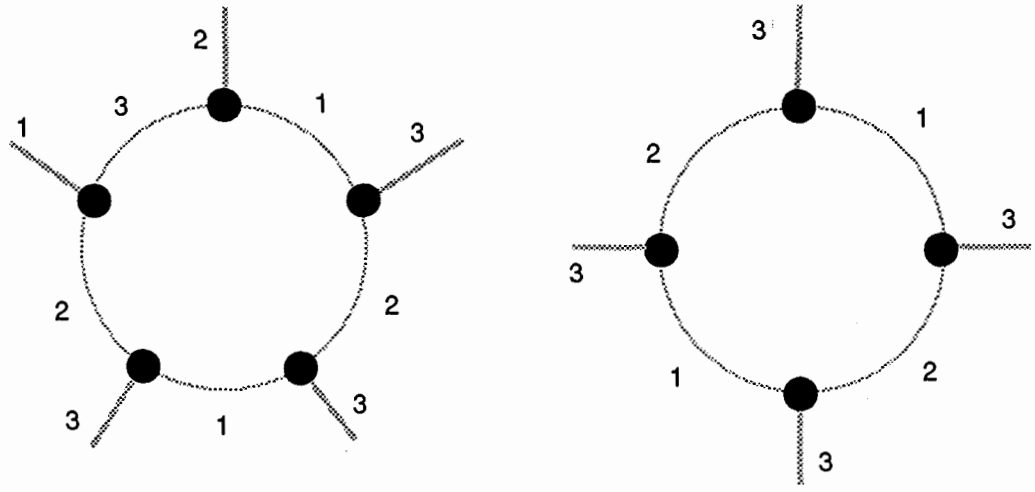
$$B_{F_1}(CCCd) \le \left(\sqrt{L\tau} + \sqrt{2(3d+3)\beta}\right)^2 \quad \text{(3 ADST)},$$

$$B_{F_1}(BFd) \le \left(\sqrt{2L\tau} + \sqrt{3(d + \left\lfloor \frac{d}{2} \right\rfloor + 1)\beta}\right)^2 \quad \text{(2 ADST)},$$

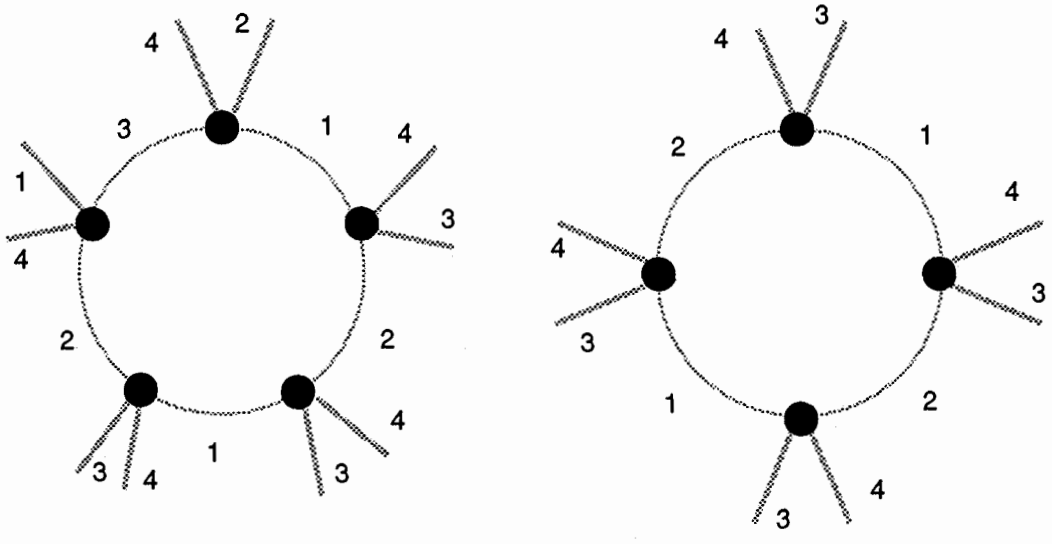
$$B_{F_1}(BFd) \le \left(\sqrt{L\tau} + \sqrt{3(2d+1)\beta}\right)^2 \quad \text{(4 ADST)}.$$

3.5.2 Half-duplex links

For CCCd we use Lemma 1.8 multiplying by two the results from lemmas 3.8 and 3.9:



Labeling of CCCd for odd and even values of d



Labeling of BFd for odd and even values of d

fig. 3.12.

Lemma 3.10:

$$B_{H^*}(CCCd) \le \left(\sqrt{L\tau} + \sqrt{2(2d + \left\lfloor \frac{d}{2} \right\rfloor + 1)\beta}\right)^2 \qquad \text{(2 ADST)},$$

$$B_{H^*}(CCCd) \le \left(\sqrt{\frac{2L\tau}{3}} + \sqrt{2(3d+2)\beta}\right)^2 \qquad \text{(3 ADST)},$$

$$B_{H1}(CCCd) \le \left(\sqrt{3L\tau} + \sqrt{4\left(2d + \left\lfloor \frac{d}{2} \right\rfloor + 2\right)\beta}\right)^2 \quad \text{(2 ADST)},$$

$$B_{H1}(CCCd) \le \left(\sqrt{2L\tau} + \sqrt{4(3d+3)\beta}\right)^2 \quad \text{(4 ADST)}.$$

We can use the same approach for BFd as well. Notice, however, that two of the four ADST shown in section 3.3 (T0 and T1) are exclusively built out of ascending arcs and each arc in BFd cannot be both ascending and descending (Property 3.1). Therefore, those two trees are not only arc-disjoint but also edge-disjoint spanning trees of BFd. Therefore, we can broadcast in parallel along the two trees (of height 2d+1) under the H* model (using Lemma 1.6). Given those two trees and the fact that q(BFd) = 4 and Lemma 1.7 we can derive an upper bound under H1 as well:

Lemma 3.11:

$$B_{H^*}(BFd) \le \left(\sqrt{\frac{L\tau}{2}} + \sqrt{2d\beta}\right)^2$$

$$B_{H1}(BFd) \le \left(\sqrt{2L\tau} + \sqrt{3(2d+1)\beta}\right)^2.$$

3.5.3 Comparison between the bounds obtained using different number of ADST

For each graph and each model we have two upper bounds depending on how many ADST are used. We want to compare the bounds and determine which one is better for any given length L of the broadcasted message. Clearly, this depends on the size of the message given all other parameters are fixed.

We define $\varepsilon_M(Gd)$ to be the difference between the bound derived from 2 ADST for graph Gd under model M and the bound derived from Δ ADST for the same graph and model. For instance,

$$\varepsilon_{F^*}(CCCd) = \left(\sqrt{\frac{L\tau}{2}} + \sqrt{\left(2d + \left\lfloor \frac{d}{2} \right\rfloor + 1\right)\beta}\right)^2 - \left(\sqrt{\frac{L\tau}{3}} + \sqrt{(3d+2)\beta}\right)^2.$$

This is obviously a growing function of L, since the left term of the above expression

grows faster than the right one. Also, for $L = 0$ the function is negative. Therefore, if we can find some value of L, L_0, for which $\varepsilon_{F^*}(CCCd) = 0$, than for $0 < L < L_0$ using 2 ADST gives tighter upper bound and for $L > L_0$ using 3 ADST gives better upper bound.

We substitute $L\tau$:

$L\tau = l\beta$,

i.e. we use variable $l = \frac{\tau}{\beta}L$ instead of L.

It is our goal to find some $l_0 = \frac{\tau}{\beta}L_0$, for which $\varepsilon_{F^*}(CCCd) = 0$.

$$\varepsilon_{F^*}(CCCd) = \left(\left(\sqrt{\frac{l}{2}} + \sqrt{2d + \left\lfloor \frac{d}{2} \right\rfloor + 1}\right)^2 - \left(\sqrt{\frac{l}{3}} + \sqrt{3d+2}\right)^2\right)\beta$$

and since $\sqrt{\frac{l}{2}} + \sqrt{2d + \left\lfloor \frac{d}{2} \right\rfloor + 1} > 0$ and $\sqrt{\frac{l}{3}} + \sqrt{3d+2} > 0$ we can conclude that $\varepsilon_{F^*}(CCCd) = 0$ for some $l = l_0$, such that

$$\sqrt{\frac{l_0}{2}} + \sqrt{2d + \left\lfloor \frac{d}{2} \right\rfloor + 1} = \sqrt{\frac{l_0}{3}} + \sqrt{3d+2}.$$

We denote l_0 for this function as $l_{0F^*}(CCCd)$:

$$l_{0F^*}(CCCd) = \left(\frac{\sqrt{3d+2} - \sqrt{2d + \lfloor d/2 \rfloor + 1}}{\sqrt{1/2} - \sqrt{1/3}}\right)^2.$$

Table 3.1 displays some values of $l_{0F^*}(CCCd)$ for different values of d. For example, for $d = 10$, if the length of the message $L < L_0$, i.e. $L\tau < 18.482\beta$, it is faster to broadcast under F^* in CCCd using 2 ADST instead of 3 ADST of greater height.

The same reasoning applies to CCCd under F1 and BFd under both F^* and F1. Similarly, to $l_{0F^*}(CCCd)$ we define

$$l_{0F1}(CCCd) = \left(\frac{\sqrt{2(3d+3)} - \sqrt{2(2d + \lfloor d/2 \rfloor + 2)}}{\sqrt{3/2} - 1}\right)^2$$

and

$$l_{0F^*}(BFd) = \left(\frac{\sqrt{2d} - \sqrt{(d + \lfloor d/2 \rfloor)}}{\sqrt{1/2} - \sqrt{1/4}}\right)^2$$

$$l_{0F1}(BFd) = \left(\frac{\sqrt{3(2d+1)} - \sqrt{3(d+\lfloor d/2 \rfloor + 1)}}{\sqrt{2}-1}\right)^2.$$

Under the half-duplex model we simply double the time for all bounds for CCCd and therefore

$$l_{0H^*}(CCCd) = l_{0F^*}(CCCd) \text{ and}$$
$$l_{0H1}(CCCd) = l_{0F1}(CCCd).$$

For BFd we only have one bound for each model and therefore $\varepsilon_{H^*}(BFd)$ and $\varepsilon_{H1}(BFd)$ are simply not defined.

Table 3.1: Values of l_0 for various models

d	CCCd, F*	CCCd, F1	BFd, F*	BFd, F1
3	14.156	8.529	4.710	2.935
4	10.730	6.620	3.348	2.194
5	15.909	9.938	6.220	4.167
6	13.240	8.359	5.022	3.436
7	18.176	11.550	7.826	5.412
8	15.841	10.127	6.695	4.684
9	20.628	13.240	9.463	6.662
10	18.482	11.909	8.369	5.934
11	23.169	14.968	11.113	7.913
12	21.144	13.697	10.043	7.187
13	25.759	16.718	12.771	9.166
14	23.817	15.489	11.717	8.440
15	28.379	18.482	14.433	10.419
16	26.498	17.284	13.391	9.693

CHAPTER 4

SCATTERING IN CCCd AND BFd

In this chapter we describe our algorithms for scattering under the linear model of communication for both CCCd and BFd networks. Our main approach is to use two phases of scattering. In the first phase we scatter along perfectly balanced binary subtrees of CCCd and BFd. We assign a cycle to every node from each subtree. In the end of the phase, every node of the trees has part of the information to be further scattered within its cycle. To minimize the time we use several disjoint trees and pipeline different parts of the information for every cycle along different trees. In phase 2 we scatter in all cycles of CCCd and BFd in parallel using multiple originators.

4.1 Cube-Connected Cycles

The scattering algorithm described in Chapter 1 is modified for CCCd. CCCd is vertex transitive and therefore we can assume the originator to be $(0, \bar{0})$.

4.1.1 Phase 1

In this phase we build three binary subtrees of CCCd and scatter messages of size $M = \frac{dL}{3}$ along each of the trees. At the end of the phase each node of the trees contains a third of the information needed for its assigned cycle. Cycles are assigned in such way that each cycle receives the information for its nodes from three different originators.

First we show how to build and scatter along one of those trees (T0), and then we describe the algorithms for the other two trees -- T1 and T2.

We build an ascending binary subtree of height d rooted at $(0, \bar{0})$. It is a perfectly balanced binary tree and therefore we can scatter in the tree using the scattering algorithm described in Section 1.2.1.

However, we only want to use the part of the tree, which is in CCCd[0 = 0]. We remove

node $(0, \bar{0}_0)$, the subtree rooted at $(0, \bar{0}_0)$ and the arc $(0, \bar{0}) \rightarrow (0, \bar{0}_0)$ from the tree. The resulting tree is called T0. The height of T0 (we ignore the hop-nodes for now) is d. Every non-leaf node of T0 (except for the root) is connected via a straight arc to one of its children and via a compound arc to its other child.

We can scatter in T0 in the same way we would scatter in a perfectly balanced binary subtree of height d, rooted at $(0, \bar{0})$. This means that at each step i, $1 \leq i \leq d$, node $(0, \bar{0})$ sends to node $(1, \bar{0})$ a packet of size $2^{h-i}M$, comprised of the messages for all nodes of distance $d - i + 1$ from $(0, \bar{0})$ in T0. During every step i, $2 \leq i \leq d$, each node (j, X), which has received during step $i - 1$ a packet from its parent in T0, splits the packet into two equal parts. Each of the parts contains the messages for the nodes in one of the branches of (j, X). Node (j, X) sends the two packets in parallel to its children in T0.

Definition:

$P_i(j, X)$ denotes the packet which node (j, X) has to receive during step i of the scattering algorithm in T0.

Every non-leaf and non-root node of T0 has a compound arc to its cross child. It is not possible to send a packet over a compound arc in a single communication. For that reason, we split each step of the above described algorithm into 2 substeps, called substep 1 and substep 2.

Every packet $P_i(j, X)$, $1 \leq i \leq d-1$, contains the information for packets $P_{i+1}(j+1, X_j)$ and $P_{i+1}(j+1, X)$. We refer to those two packets as $P'_i(j, X)$ and $P''_i(j, X)$, respectively. For simplicity, we say that a packet 'contains other packets' instead of 'contains the information for other packets'.

Every node (j, X) of T0, which has to receive a packet $P_i(j, X)$, $1 \leq i \leq d-1$, receives the packet in two substeps of step i. In substep 1, node (j, X) receives $P'_i(j, X)$ and in substep 2, it receives $P''_i(j, X)$. This way, it is possible for (j, X) to forward $P'_i(j, X)$, which is $P_{i+1}(j+1, X_j)$, to its cross child during substep 2 of step i. $P_{i+1}(j+1, X_j)$ is sent to the hop-node -- (j, X_j). Thus, the hop-node can send $P_{i+1}(j+1, X_j)$ to $(j+1, X_j)$ during the next step -- step $i+1$. During step $i+1$ node (j, X) forwards $P''_i(j, X)$, which is $P_{i+1}(j+1, X)$, to its straight child -- $(j+1, X)$.

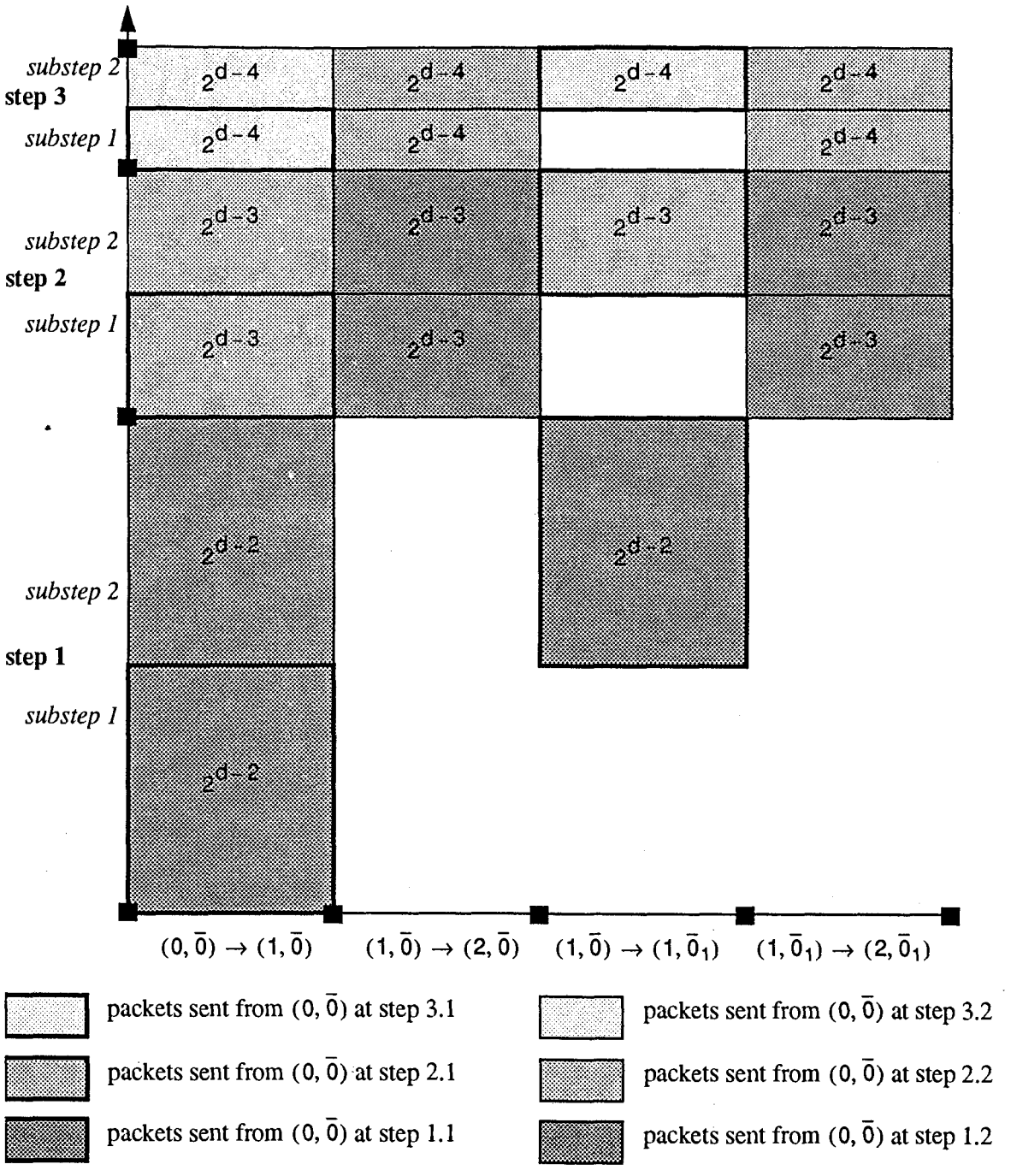


fig.4.1. Scattering in T0. Observe that both nodes at level 2 -- $(2, \bar{0})$ and $(2, \bar{0}_1)$ receive their packets at the same time.

More formally, the algorithm for scattering in T0 is:

step i, $1 \le i \le d-1$ for the originator $(0, \bar{0})$:

substep 1 $(0, \bar{0})$ sends $P'_i(1, \bar{0})$ to $(1, \bar{0})$.

substep 2 $(0, \bar{0})$ sends $P''_i(1, \bar{0})$ to $(1, \bar{0})$ and
$(1, \bar{0})$ sends $P'_i(1, \bar{0})$, which is $P_{i+1}(2, \bar{0}_1)$, to $(1, \bar{0}_1)$ (hop-node of the compound arc to the cross child of $(1, \bar{0})$).

step i, $2 \le i \le d-1$ for every node (j, X) that during substep 2 of step i - 1 has received a packet $P''_{i-1}(j, X)$, i.e. $P_i(j+1, X)$:

substep 1 (j, X) sends $P'_i(j+1, X)$ to $(j+1, X)$.

substep 2 (j, X) sends $P''_i(j+1, X)$ to $(j+1, X)$ and
$(j+1, X)$ sends $P'_i(j+1, X)$, i.e. $P_{i+1}(j+2, X_{j+1})$, to $(j+1, X_{j+1})$ (hop-node of the compound arc to the cross child of $(j+1, X)$).

step d *substep 1* $(0, \bar{0})$ sends $P_d(1, \bar{0})$ to $(1, \bar{0})$ and
every node (j, X) that during substep 2 of step d - 1 has received a packet $P''_{d-1}(j, X)$, which is $P_d(j+1, X)$, forwards this packet to $(j+1, X)$.

Lemma 4.1:

Every packet $P_i(j, X)$, $1 \le i \le d$, is received at node (j, X) during step i of the scattering algorithm.

Proof: We use induction to prove the statement for every step i.

i = 1 There is only one packet to be received during step 1 -- $P_1(1, \bar{0})$. According to the above algorithm this packet is received at $(1, \bar{0})$ in step 1.

i = k Assume that all packets $P_k(j, X)$ are received at nodes (j, X) in step k.

i = k+1 Consider any packet $P_{k+1}(j, X)$, which has to be received at a node (j, X) during step k+1. There are two possible cases for (j, X):

case 1 The parent of (j, X) is $(j-1, X)$.

Therefore, $P_{k+1}(j, X)$ is $P''_k(j-1, X)$. According to our assumption, $P''_k(j-1, X)$ is received at $(j-1, X)$ in step k. During the entire algorithm packets destined for a node's straight child are always sent to the node in substep 2 of some step. Therefore, $P''_k(j-1, X)$ was received at $(j-1, X)$ during substep 2 of step k. Therefore, node $(j-1, X)$ sends $P''_k(j-1, X)$, which is $P_{k+1}(j, X)$, to (j, X) in step k+1 (see step k+1 of the algorithm).

case 2 The parent of (j, X) is $(j-1, X_{j-1})$.

Therefore, $P_{k+1}(j, X)$ is $P'_k(j-1, X_{j-1})$. According to our assumption, $P''_k(j-1, X_{j-1})$ is received at node $(j-1, X_{j-1})$ during step k. During the entire algorithm packets destined for a node's cross child are always sent to the node in substep 1 of some step. In substep 2 of that step, those packets are sent to the hop-node of the compound arc to the cross child. Therefore, $P'_k(j-1, X_{j-1})$ was forwarded to $(j-1, X)$ in substep 2 of step k. Therefore, node $(j-1, X)$ forwards $P'_k(j-1, X_{j-1})$, which is $P_{k+1}(j, X)$ to (j, X) in step k+1 (see step k+1 of the algorithm). □

Therefore, after d steps, each node of T0 receives its packet of size M.

Lemma 4.2:

Phase 1 in T0 takes time $(2^d - 1)M\tau + (2d-1)\beta$.

Proof: The size of each packet $P_i(j, X)$ is $2^{d-i}M$. Therefore, all steps, except for the last one, take time $2\beta + 2^{d-i}M\tau$. The last step takes a time of only $\beta + M\tau$. □

Now, similarly to T0 we construct T1, building a descending binary subtree of height d rooted at $(0, \bar{0})$ and removing the subtree pointed to by arc $(0, \bar{0}) \rightarrow (0, \bar{0}_0)$. T1 has exactly the same structure as T0 -- the root has only one child, every other non-leaf node has a child reached via straight arc and a child reached via compound arc. The hop-node of the compound arc is different from both children. Therefore, we can scatter in T1 the same way we scatter in T0. The time to complete the algorithm is again $(2^d - 1)M\tau + (2d-1)\beta$.

It is our goal, however, to scatter in parallel along both trees. We denote the packets used during the scattering in T1 by $Q_i(j, X)$.

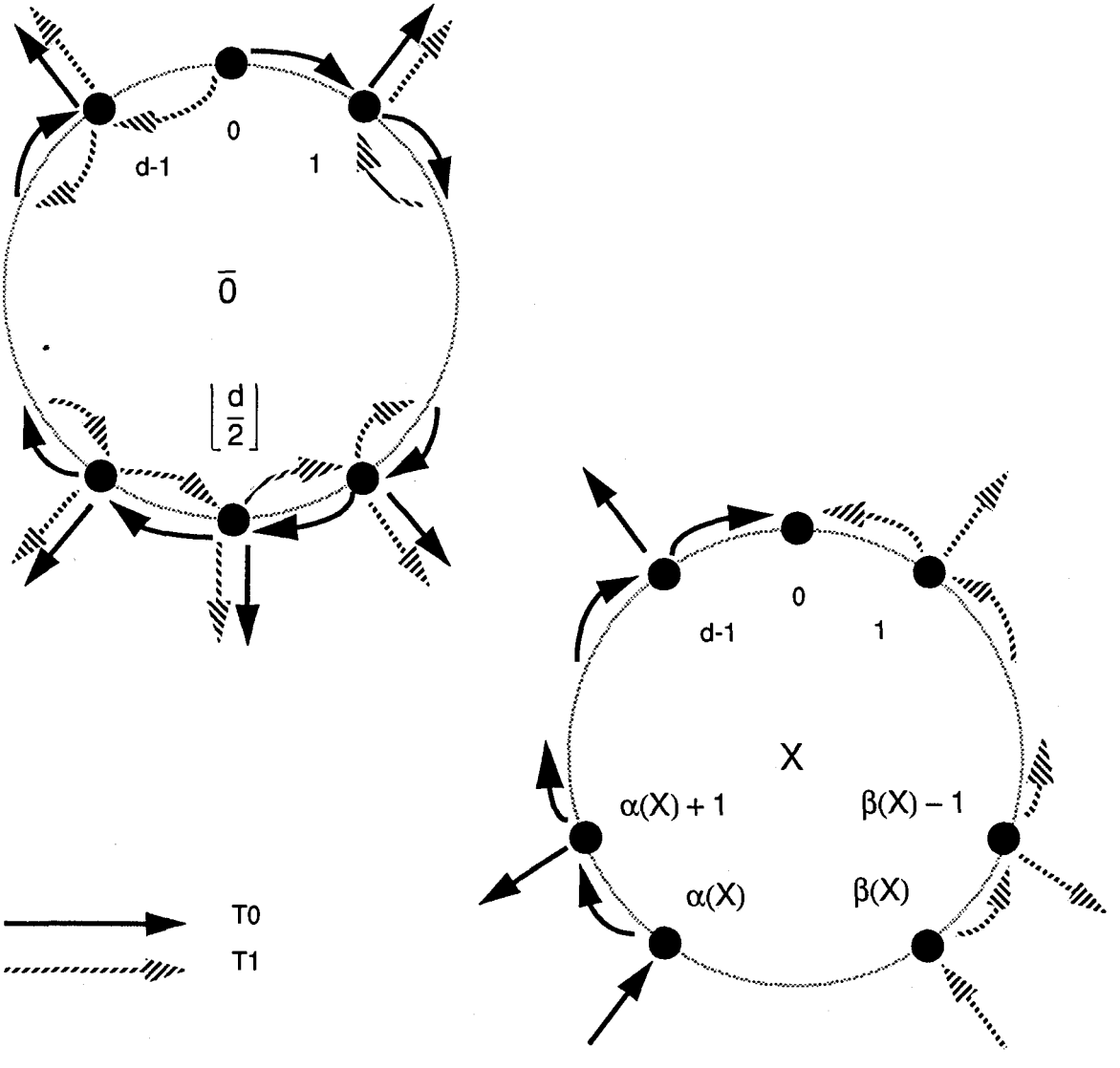


fig.4.2. T0 and T1 in two cycles of CCCd [0, 1 = 0, 0] .

All cycles of CCCd[0 = 0] are represented in T0 (Lemma 2.2). They are reached first through their α-nodes (Lemma 2.1 and taking into account the hop nodes of the compound arcs) and in every cycle X all nodes with levels higher than α(X) are in T0 (Property 2.4). Similarly, all cycles of CCCd[0 = 0] are represented in T1 (Lemma 2.4) and they are reached via their β-nodes

(Lemma 2.3 and taking into account the hop-nodes of each compound arc). All nodes of X with levels lower than $\beta(X)$ are in T1 (Property 2.10). We know that $\forall (X \in CCCd)\, \alpha(X) \geq \beta(X)$ (Property 2.1). Also, for all cycles, except for $\bar{0}_i$ and $\bar{0}$, $\alpha(X) \neq \beta(X)$. Therefore,

$$\forall (X \in CCCd, X \neq \bar{0}_i, X \neq \bar{0})\ (\alpha(X) > \beta(X)).$$

Therefore, for all cycles, except $\bar{0}$ and $\bar{0}_i$, the two trees are arc-disjoint. In fact, the only common arcs used by both trees are the ones of the type $(i, \bar{0}) \rightarrow (i, \bar{0}_i)$.

Assume that **d is odd**. Notice that the common arcs are used only after step $\lfloor d/2 \rfloor$. For example, in T0, every node $(j, \bar{0})$, $\lfloor d/2 \rfloor + 1 \leq j \leq d-1$ uses its cross arc during substep 2 of steps i, $j \leq i \leq d-1$. The amount of information sent during step i over the cross arc is $2^{d-i-1}M$. During those substeps the same arcs are used by T1 to send the same amount of information. Therefore, our goal is to have the information from T0 for those nodes received before step $\lfloor d/2 \rfloor + 1$. Then, we can use some idle first substeps of steps 1 through $\lfloor d/2 \rfloor + 1$ to send the packets from T0 over the cross arcs, accumulate them at the hop-node and continue the normal scattering at the appropriate step.

Consider any packet $P_i(j, \bar{0})$, $1 \leq j \leq \left\lfloor \frac{d}{2} \right\rfloor$, $j \leq i \leq j + \left\lfloor \frac{d}{2} \right\rfloor$. This packet contains several other packets, among them $P_q(d-j, \bar{0})$, where $q = i + d - 2j$. This is the packet received at $(d-j, \bar{0})$ in step q. Half of $P_q(d-j, \bar{0})$ -- $P'_q(d-j, \bar{0})$, has to be sent to $(d-j, \bar{0}_{d-j})$ during substep 2 of step q.

During the same substep, node $(d-j, \bar{0})$ has to send to $(d-j, \bar{0}_{d-j})$ a packet received from T1 -- $Q'_q(d-j, \bar{0})$.

Now, consider the packet $Q_i(d-j, \bar{0})$, which node $(d-j, \bar{0})$ receives along T1 at step i. This packet contains $Q_q(j, \bar{0})$, which node $(j, \bar{0})$ receives along T1 in step q. Part of this packet -- $Q'_q(j, \bar{0})$, has to be sent to $(j, \bar{0}_j)$ during substep 2 of step q.

At the same time, $P'_q(j, \bar{0})$ has to be sent to $(j, \bar{0}_j)$ as well.

To summarize, in substep 2 of step q, node $(j, \bar{0})$ has to send to $(j, \bar{0}_j)$ both $P'_q(j, \bar{0})$ and $Q'_q(j, \bar{0})$. During the same substep, node $(d-j, \bar{0})$ has to send to $(d-j, \bar{0}_{d-j})$ both $P'_q(d-j, \bar{0})$

and $Q'_q(d-j, \bar{0})$.

The size of all those four packets is the same. Also, $P'_q(d-j, \bar{0})$ is contained in $P_i(j, \bar{0})$ and $Q'_q(j, \bar{0})$ is contained in $Q_i(d-j, \bar{0})$. Therefore, we can swap the packets as follows:

1. Remove from packet $Q_i(d-j, \bar{0})$ the information for packet $Q_q(j, \bar{0})$ and make this information part of $P_i(j, \bar{0})$.

2. Remove from packet $P_i(j, \bar{0})$ the information for packet $P_q(d-j, \bar{0})$ and make this information part of $Q_i(d-j, \bar{0})$.

In this way, the packet size remains the same; however, $P_q(d-j, \bar{0})$ is received at $(d-j, \bar{0})$ during step i, as part of packet $Q_i(d-j, \bar{0})$. Similarly, $Q_q(j, \bar{0})$ is received at $(j, \bar{0})$ before step q.

In all substeps 1, all cross arcs are idle. Substep 1 of step i+1 takes time $2^{d-i-2}M\tau + \beta$. The size of $P'_q(d-j, \bar{0})$ and $Q'_q(j, \bar{0})$ is $2^{d-q-1}M$. Since $q = d + i - 2j$, it follows that

$d-(d+i-2j)-1 = 2j-i-1$. Then we can conclude that
$2j-i-1 \le (d-1)-i-1$, because $j \le \lfloor d/2 \rfloor$. However,
$(d-1)-i-1 = d-i-2$, and therefore
$d-q-1 \le d-i-2$.

Therefore, there is enough time during substep 1 of step i+1 to send $P'_q(d-j, \bar{0})$ and $Q'_q(j, \bar{0})$ over the cross arcs of $(d-j, \bar{0})$ and $(j, \bar{0})$, respectively. In this way, those packets are received at the hop-nodes before the end of step q, because $q = d+i-2j \le i+1$, since $j \le \lfloor d/2 \rfloor$.

We can apply this approach to all couples of packets $P_q(d-j, \bar{0})$ and $Q_q(j, \bar{0})$, $q = d+i-2j \le i+1$, $1 \le i \le \left\lfloor \frac{d}{2} \right\rfloor$, $j \le i \le j + \left\lfloor \frac{d}{2} \right\rfloor$.

Thus, none of the nodes in $\bar{0}$ receives packets from both trees after step $\left\lfloor \frac{d}{2} \right\rfloor$. We don't have to use the cross arcs simultaneously, and the hop-nodes receive their packets from both trees on time. Therefore, the parallel scattering in T0 and T1 finishes as fast as the single scattering in T0 and T1.

Now, let us consider the other case -- **d is even.** The approach used above was based on the idea of having all nodes of $\bar{0}$ informed earlier along one of the trees. When d is even, we can use the same technique for all couples of packets $P_q(d-j, \bar{0})$ and $Q_q(j, \bar{0})$, $q = d+i-2j \le i+1$, $1 \le j \le \frac{d}{2}-1$, $j \le i \le j+\frac{d}{2}$; however, special care must be taken of the packets for node $(\frac{d}{2}, \bar{0})$, since it is reached at the same time along both trees and therefore, swapping the massages for that node will not accelerate their receiving.

First, let us analyze the packets received by node $(\frac{d}{2}, \bar{0})$ along T0 -- packets $P_i(\frac{d}{2}, \bar{0})$, $\frac{d}{2} \le i \le d$. Each of those packets, except for the packet received during step d, consists of two parts -- one part, which has to be forwarded to the straight neighbour of $(\frac{d}{2}, \bar{0})$, and a second part, which has to be forwarded to the cross neighbour of $(\frac{d}{2}, \bar{0})$.

Observe that it is not really necessary to forward any packets to the straight neighbour of $(\frac{d}{2}, \bar{0})$, because all nodes $(j, \bar{0})$, $\frac{d}{2}+1 \le j \le d$, receive their packets $P_i(j, \bar{0})$, $j \le i \le d$, along T1. Therefore, it is sufficient to send to $(\frac{d}{2}, \bar{0})$ only packets $P'_i(\frac{d}{2}, \bar{0})$, $\frac{d}{2} \le i \le d-1$, and packet $P_d(\frac{d}{2}, \bar{0})$. The total amount of information contained in these packets is

$$M + \sum_{i=d/2}^{d-1} 2^{d-i-1}M = 2^{d/2}M.$$

The size of packet $P_{d/2}(\frac{d}{2}, \bar{0})$ is $2^{d/2}M$ as well. Therefore, in this packet, we can replace the information for packet $P''_{d/2}(\frac{d}{2}, \bar{0})$ (we showed already that there is no need to send this packet to $(\frac{d}{2}, \bar{0})$) with the information for all packets $P'_i(\frac{d}{2}, \bar{0})$, $\frac{d}{2} \le i \le d-1$, and packet $P_d(\frac{d}{2}, \bar{0})$. In this way, by the end of step $\frac{d}{2}$, node $(\frac{d}{2}, \bar{0})$ receives all packets from T0 which it needs for the entire algorithm. Therefore, we can use substeps 1 of steps i, $\frac{d}{2}+1 \le i \le d-1$, to forward to the hop-node $(\frac{d}{2}, \bar{0}_{d/2})$ all these packets, except for $P'_i(\frac{d}{2}, \bar{0})$, instead of sending them during substeps 2 of the same steps. Thus, substeps 2 can be used by T1.

Now we only have to arrange for $P'_i(\frac{d}{2}, \bar{0})$ to be received at $(\frac{d}{2}, \bar{0})$ before step $\frac{d}{2}$, so that we can send this packet to the hop-node during substep 1 of step $\frac{d}{2}$, which will allow T1 to use the arc during substep 2 of step $\frac{d}{2}$.

For that purpose, we have to speed up the information flow along cycle $\bar{0}$ and make it pos-

sible for $(\frac{d}{2}, \bar{0})$ to receive the packet earlier.

Observe that during step 1 node $(0, \bar{0})$ sends as part of $P''_1(1, \bar{0})$ a piece of information of size M, which is supposed to be received by node $(d, \bar{0})$, i.e. the root itself, during step d. Obviously, there is no need to include this piece of information in $P''_1(1, \bar{0})$ and we can simply remove it from the packet.

Packet $P'_i(\frac{d}{2}, \bar{0})$, of size $2^{d/2-1}M$, is contained in $P''_1(1, \bar{0})$.

Packet $P'_1(1, \bar{0})$, which is $P_2(2, \bar{0}_1)$, consists of $P'_2(2, \bar{0}_1)$ and $P''_2(2, \bar{0}_1)$, each of size $2^{d-3}M$, i.e. larger than $2^{d/2-1}M$ for all $d \geq 4$. Those two packets have to be sent along $(1, \bar{0}_1) \rightarrow (2, \bar{0}_1)$ during substeps 1 and 2 of step 2.

We rearrange the packets and transmit them as follows:

1. In packet $P'_i(\frac{d}{2}, \bar{0})$, replace a piece of information S, of size $2^{d/2-1}M$, from the information for $P''_2(2, \bar{0}_1)$ with the information for packet $P'_i(\frac{d}{2}, \bar{0})$. Execute substep 1 of step 1 as usual. After this substep, S is still not sent to node $(1, \bar{0})$. Instead, $P'_i(\frac{d}{2}, \bar{0})$ is already at that node.

2. In substep 2 of step 1 send $P''_1(1, \bar{0})$ along $(0, \bar{0}) \rightarrow (1, \bar{0})$ and
send $P'_2(2, \bar{0}_1)$ along $(1, \bar{0}) \rightarrow (1, \bar{0}_1)$ and
send $P'_i(\frac{d}{2}, \bar{0})$ along $(1, \bar{0}) \rightarrow (2, \bar{0})$.

This substep finishes $M\tau$ earlier for arc $(0, \bar{0}) \rightarrow (1, \bar{0})$, because the size of $P''_1(1, \bar{0})$ is now only $2^{d-2}M - M$. It also finishes earlier for arc $(1, \bar{0}) \rightarrow (1, \bar{0}_1)$, because, instead of the entire packet $P_2(2, \bar{0}_1)$, only $P'_2(2, \bar{0}_1)$ was sent along the arc. Therefore, substep 1 of step 2 can start $M\tau$ earlier along $(1, \bar{0}) \rightarrow (1, \bar{0}_1)$. This introduces some asynchronism in the scattering algorithm.

Packet $Q_{d-1}(1, \bar{0})$ is received at $(2, \bar{0})$ during step 1 in place of packet $P_{d-1}(d-1, \bar{0})$ as described earlier in this section.

3. During substep 1, $P'_2(2, \bar{0}_1)$ is forwarded along $(1, \bar{0}_1) \rightarrow (2, \bar{0}_1)$ and $P'_2(2, \bar{0})$ is sent along $(1, \bar{0}) \rightarrow (2, \bar{0})$. Also, a packet comprised of $Q_{d-1}(1, \bar{0})$ and S is sent over the arc

$(1,\bar{0}) \rightarrow (1,\bar{0}_1)$. The size of this packet is at most $2^{d-3}M+M$ and the substep starts $M\tau$ earlier than it starts for all other arcs. Therefore, the packet is received at the hop-node before the beginning of substep 2.

4. Packet $P'_i(\frac{d}{2},\bar{0})$ is received at node $(2,\bar{0})$ by the end of step 1. If $d = 4$ then $\frac{d}{2} = 2$, and therefore we have achieved our goal -- receiving $P'_i(\frac{d}{2},\bar{0})$ at $(\frac{d}{2},\bar{0})$ before the beginning of step $\frac{d}{2}$. In all other cases, we forward $P'_i(\frac{d}{2},\bar{0})$ to its destination during the next $\frac{d}{2}-2$ substeps. Each of those substeps lasts long enough to send the packet over a single arc. Therefore, after $\frac{d}{2}$ substeps, $P'_i(\frac{d}{2},\bar{0})$ reaches $(\frac{d}{2},\bar{0})$. This is again before step $\frac{d}{2}$. Therefore, we can forward $P'_i(\frac{d}{2},\bar{0})$ to the hop-node $(\frac{d}{2},\bar{0}_{d/2})$ during substep 1 of step $\frac{d}{2}$ and use the second substep of step $\frac{d}{2}$ to send $Q'(\frac{d}{2},\bar{0})$ to the hop-node.

Now, similarly to T0 and T1, we build two trees from $(0,\bar{0}_0)$ in CCCd[0 = 1]. We remove the last level of those trees and call them T0' and T1', respectively. Their height is d-1. Then we merge T0' and T1' into one tree and add $(0,\bar{0}) \rightarrow (0,\bar{0}_0)$ to it. This tree is called T2. The height of T2 is d.

During substep 1 of steps i, $1 \le i \le d-1$, $(0,\bar{0})$ sends packets of size $2^{d-i-1}M$ to $(0,\bar{0}_0)$. The node scatters in T0' using the scheme which node $(1,\bar{0}_1)$ employs after step 1. Thus, the scattering in T0' is always one substep ahead of the scattering in T0.

During substep 2 of steps i, $1 \le i \le d-1$, $(0,\bar{0})$ sends packet of size $2^{d-i-1}M$ to $(0,\bar{0}_0)$. The node scatters in T1' using the scheme employed by node $(d-1,\bar{0}_{d-1})$ after step 1. Thus, the scattering in T1' finishes at the time the scattering in T1 is finished. Since the scattering in tree T0' is one substep ahead, the common cross arcs are used by T0' and T1' during different substeps. During substep 1 of step d, a message of size M is sent from $(0,\bar{0})$ to $(0,\bar{0}_0)$.

In the end of the phase, all nodes from T0, T1 and T2 receive a message of size M, $M = \frac{dL}{3}$. Also, no link is used in both directions, and therefore we can conclude:

Lemma 4.3:

Phase 1 in CCCd finishes in time $(2^d-1)\frac{dL}{3}\tau + (2d-1)\beta$ under F^* and H^*.

Observe that each node $(i, \bar{0})$, $1 \le i \le d-1$, receives two messages by the end of step d -- one message from $T0$ and one message from $T1$. One of these two messages is received before step d and the other message is received during step d. This is caused by the modifications we made in order to be able to scatter in parallel along $T0$ and $T1$. Also, arcs $(i, \bar{0}) \rightarrow (i, \bar{0}_i)$, $1 \le i \le d-1$, are not used in step d. Therefore, in step d, nodes $(i, \bar{0})$, $1 \le i \le d-1$, can send one of their messages along $(i, \bar{0}) \rightarrow (i, \bar{0}_i)$.

Similarly, each node $(i, \bar{0}_0)$, $1 \le i \le d-1$, receives two messages by the end of step d -- one along $T0'$ and one along $T1'$. The scattering in $T0'$ is always one substep ahead of the scattering in $T1'$. Arcs $(i, \bar{0}_0) \rightarrow (i, \bar{0}_{0i})$, $1 \le i \le d-1$, are not used during step d. Therefore, in the last step, nodes $(i, \bar{0}_0)$, $1 \le i \le d-1$, can send one of their messages along $(i, \bar{0}_0) \rightarrow (i, \bar{0}_{0i})$.

The time required to complete the algorithm remains the same. In phase 2, we use the fact, that one of the messages is at the hop-node after phase 1.

4.1.2 Phase 2

We show how to scatter in cycles $\bar{0}$ and $\bar{0}_0$.

Scattering in $\bar{0}$ and $\bar{0}_0$

We scatter in $\bar{0}$ from one originator -- the root $(0, \bar{0})$, using an algorithm similar to the one proposed in [FrLa91].

Case 1: d odd.

We consider two subgraphs of $\bar{0}$ -- the array $(0, \bar{0})$, $(1, \bar{0})$, ..., $(\lfloor d/2 \rfloor, \bar{0})$ and the array $(0, \bar{0})$, $(d-1, \bar{0})$, ..., $(d - \lfloor d/2 \rfloor, \bar{0})$. Each of them consists of $\lfloor d/2 \rfloor + 1$ nodes. We scatter in both arrays in parallel using our algorithm for scattering in arrays presented in Section 1.2.1. The time to complete the algorithm is therefore $\lfloor d/2 \rfloor (L\tau + \beta)$. Since d is odd this equals $\frac{d-1}{2} L\tau + \left\lfloor \frac{d}{2} \right\rfloor \beta$.

Case 2: d even.

In this case we split $\bar{0}$ into the same two subgraphs; this time, however, node $(d/2, \bar{0})$ is represented in both arrays, since $\lfloor d/2 \rfloor = d - \lfloor d/2 \rfloor = d/2$. Therefore, it is sufficient to send a half of the message for $(d/2, \bar{0})$ along each array. Thus, the first step of the scattering algorithm only takes time $\frac{L}{2}\tau + \beta$ instead of $L\tau + \beta$ and therefore the scattering in both arrays finishes $\frac{L}{2}\tau$ earlier, i.e. in time $\frac{d}{2}(L\tau + \beta) - \frac{L}{2}\tau = \frac{d-1}{2}L\tau + \frac{d}{2}\beta$, which is again $\frac{d-1}{2}L\tau + \left\lfloor \frac{d}{2} \right\rfloor \beta$.

Since every link is used in only one direction, we can conclude that the scattering time for cycle $\bar{0}$ is:

$$s_{F^*}(\bar{0}) \le s_{H^*}(\bar{0}) \le \frac{d-1}{2}L\tau + \left\lfloor \frac{d}{2} \right\rfloor \beta.$$

A similar idea is used for the scattering in $\bar{0}_0$. There are two originators -- $(0, \bar{0})$, which contains two thirds of the information for the cycle, and $(0, \bar{0}_0)$, which contains one third of the information for the cycle.

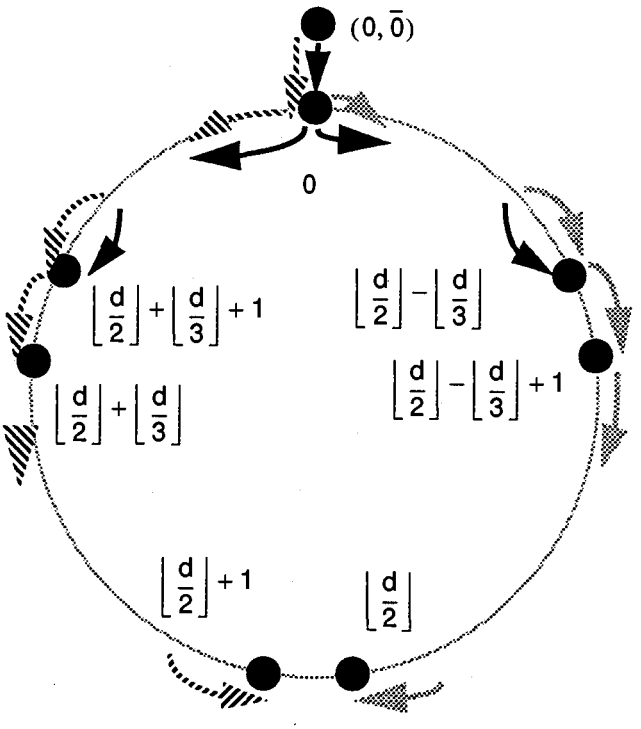


fig.4.3. Scattering in cycle $\bar{0}_0$.

Node $(0, \bar{0}_0)$ contains the entire messages for $\left\lfloor \frac{d}{3} \right\rfloor$ nodes of $\bar{0}_0$. If $d \bmod 3 = k$, $k \neq 0$,

then this node also contains a message of size $\frac{k}{3}L$ for itself. Therefore, in any case, node $(0,\bar{0}_0)$ has to scatter $\left\lfloor\frac{d}{3}\right\rfloor$ messages of size L to $\left\lfloor\frac{d}{3}\right\rfloor$ nodes of the cycle. We choose them to be nodes $(i,\bar{0}_0)$, $\left\lfloor\frac{d}{2}\right\rfloor+1-\left\lfloor\frac{d}{3}\right\rfloor\le i\le\left\lfloor\frac{d}{2}\right\rfloor$. Node $(0,\bar{0}_0)$ scatters by using the algorithm for scattering in arrays. The algorithm requires $\left\lfloor\frac{d}{2}\right\rfloor$ steps. After step $\left\lfloor\frac{d}{3}\right\rfloor$, all messages have been sent from $(0,\bar{0}_0)$.

Node $(0,\bar{0})$ contains the messages for the rest of the nodes -- nodes $(i,\bar{0}_0)$, where $\left\lfloor\frac{d}{2}\right\rfloor+1\le i\le d-1$ or $1\le i\le\left\lfloor\frac{d}{2}\right\rfloor-\left\lfloor\frac{d}{3}\right\rfloor$. Node $(0,\bar{0})$ also contains a message of size $\frac{3-k}{3}L$, $k = d \bmod 3$, for node $(0,\bar{0}_0)$.

- Node $(0,\bar{0})$ uses the algorithm for scattering in arrays to scatter in nodes $(i,\bar{0}_0)$, $\left\lfloor\frac{d}{2}\right\rfloor+1\le i\le d$. This scattering requires $\left\lceil\frac{d}{2}\right\rceil$ steps.

After step $\left\lfloor\frac{d}{3}\right\rfloor$, node $(0,\bar{0})$ originates scattering in nodes $(i,\bar{0}_0)$, $1\le i\le\left\lfloor\frac{d}{2}\right\rfloor-\left\lfloor\frac{d}{3}\right\rfloor$ by using the algorithm for scattering in arrays. Therefore, after step $\left\lfloor\frac{d}{3}\right\rfloor$, arc $(0,\bar{0})\to(0,\bar{0}_0)$ is used for both scatterings. The packets are batched into one packet of double size. Therefore, each step after step $\left\lfloor\frac{d}{3}\right\rfloor$, except for the last one, takes a time of $2L\tau+\beta$. The first $\left\lfloor\frac{d}{3}\right\rfloor$ steps and the last step take time of $L\tau+\beta$.

Each link is used in one direction only, and therefore the time required to complete the scattering in $\bar{0}_0$ is

$$s_{F^*}(\bar{0}_0)\le s_{H^*}(\bar{0}_0)\le\left\lceil\frac{d}{2}\right\rceil L\tau+\left(\left\lceil\frac{d}{2}\right\rceil-\left\lfloor\frac{d}{3}\right\rfloor-1\right)L\tau+\left\lceil\frac{d}{2}\right\rceil\beta.$$

Scattering in cycles $X\neq\{\bar{0},\bar{0}_0\}$

Our goal is to assign 3 originators to every cycle $X\notin\{\bar{0},\bar{0}_0\}$. Each of the three originators has to be a node from T0, T1 or T2 and contain one third of the information which has to be scattered within the cycle.

We assign nodes $(\alpha(X),X_{\alpha(X)})$ and $(\beta(X),X_{\beta(X)})$ to every cycle $X\notin\{\bar{0},\bar{0}_0\}$.

According to lemmas 2.1 and 2.3, all cycles of CCCd[0 = 0] are represented in T0 and

T1. According to lemmas 2.2 and 2.4, all these cycles, except for $\bar{0}$, are reached from nodes $(\alpha(X), X_{\alpha(X)})$ in T0 and from nodes $(\beta(X), X_{\beta(X)})$ in T1. $\alpha(X) \neq 0$ and $\beta(X) \neq 0$ for any of them. Therefore, every cycle from $CCCd[0=0]$, except for $\bar{0}$, is assigned to one node from T0 and to one node from T1. There are $2^{d-1}-1$ cycles in $CCCd[0=0]$ (excluding $\bar{0}$) and there are $2^{d-1}-1$ non-leaf and non-root nodes in T0 and $2^{d-1}-1$ non-leaf and non-root nodes in T1. All leaves of T0 and T1 are *0*-nodes. Therefore, each non-root and non-leaf node of T0 and T1 is assigned to exactly one cycle.

Similarly, every non-root node of T0' and T1' is assigned to exactly one cycle of $CCCd[0=1]$ (excluding $\bar{0}_0$).

We assign every leaf $(0, X)$ of T0 to cycle X_0 and every leaf $(0, X)$ of T1 to cycle X.

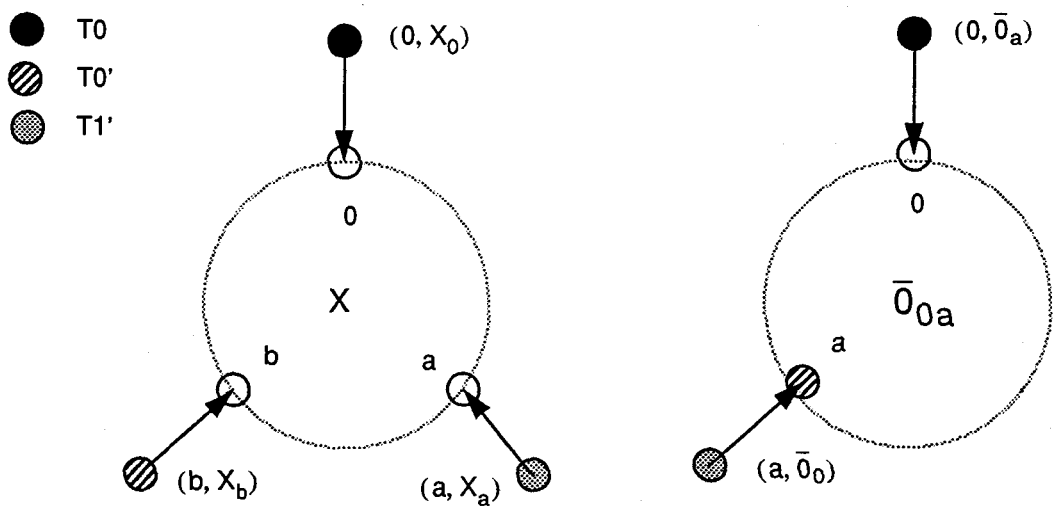


fig.4.4. Originators for phase 2 in $CCCd[0=1]$.

All leaves of T0 and T1 are *0*-nodes (i.e. nodes $(0, X)$) in cycles of $X \in CCCd[0=0]$. Observe that $X_0 \in CCCd[0=1]$. There are $2^{d-1}-1$ leaves in each tree T0 and T1 (excluding $(0, \bar{0})$) and there are $2^{d-1}-1$ cycles in each subset $CCCd[0=0]$ and $CCCd[0=1]$ (exclud-

ing $\bar{0}$ and $\bar{0}_0$). Therefore, each leaf of T0 and T1 is assigned to exactly one cycle, and each cycle is assigned to a leaf from either T0 or T1. Therefore, every cycle $X \notin \{\bar{0}, \bar{0}_0\}$ is assigned to three originators, and each originator is a node from T0, T1 or T2.

We first examine the cycles X from CCCd[0 = 1]. We define k = d mod 3.

case 1. Cycles $X \notin \{\bar{0}_{0i} \mid 1 \le i \le d-1\}$.

The three originators are adjacent to three different nodes of the cycle. Each originator contains one third of the information for the cycle. The originators are denoted by $(0, X_0)$, (a, X_a), and (b, X_b), where $0 < a < b < d$. There are three possible cases:

(a) $a \le \left\lfloor \frac{d}{2} \right\rfloor < b$

Node (a, X_a) contains a message of size $\frac{k}{3}L$ for node $(\left\lfloor \frac{d}{2} \right\rfloor - \left\lfloor \frac{d}{3} \right\rfloor, X)$ and a message of size L for each (i, X), $\left\lfloor \frac{d}{2} \right\rfloor - \left\lfloor \frac{d}{3} \right\rfloor + 1 \le i \le \left\lfloor \frac{d}{2} \right\rfloor$. We denote the set of nodes informed from (a, X_a) by S_a.

Node (b, X_b) contains a message of size $\frac{k}{3}L$ for node $(\left\lfloor \frac{d}{2} \right\rfloor + \left\lfloor \frac{d}{3} \right\rfloor + 1, X)$ and a message of size L for each (i, X), $\left\lfloor \frac{d}{2} \right\rfloor + 1 \le i \le \left\lfloor \frac{d}{2} \right\rfloor + \left\lfloor \frac{d}{3} \right\rfloor$. We denote the set of nodes informed from (b, X_b) by S_b.

Node $(0, X_0)$ contains messages for nodes (i, X), $\left\lfloor \frac{d}{2} \right\rfloor + \left\lfloor \frac{d}{3} \right\rfloor + 1 \le i \le \left\lfloor \frac{d}{2} \right\rfloor - \left\lfloor \frac{d}{3} \right\rfloor$, where the addition is modulo d. It has a message of size $\frac{3-k}{3}L$ for the first and last nodes from this set and a message of size L for all other nodes from the set.

During the first $\left\lfloor \frac{d}{3} \right\rfloor + 1$ steps, node (a, X_a) forwards to (a, X) the messages for nodes

$$(\left\lfloor \frac{d}{2} \right\rfloor, X), (\left\lfloor \frac{d}{2} \right\rfloor - 1, X), \ldots, (\left\lfloor \frac{d}{2} \right\rfloor - \left\lfloor \frac{d}{3} \right\rfloor, X)$$

in that order. Every message, received at any node of X during some step, is forwarded during the next step to its destination along the shortest path.

Similarly, during the first $\left\lfloor \frac{d}{3} \right\rfloor + 1$ steps, (b, X_b) forwards to (b, X) the messages for

$$(\lfloor \frac{d}{2} \rfloor + 1, X),\ (\lfloor \frac{d}{2} \rfloor + 2, X),\ ...,\ (\lfloor \frac{d}{2} \rfloor + \lfloor \frac{d}{3} \rfloor + 1, X)$$

in that order. Every message, received at any node of X during some step, is forwarded during the next step to its destination along the shortest path.

The distance from (a, X_a) to any node $(i, X) \in S_a$ is at most $\lfloor \frac{d}{2} \rfloor$. The distance from (b, X_b) to any node $(i, X) \in S_b$ is also at most $\lfloor \frac{d}{2} \rfloor$, and $S_a \cap S_b = \varnothing$. Therefore, in $\lfloor \frac{d}{2} \rfloor$ steps, each of which lasts at most $L\tau + \beta$, those two originators complete their scattering.

During the first $\lfloor \frac{d}{3} \rfloor$ steps of the scattering in X, node $(0, X_0)$ forwards to node $(0, X)$ all information it contains, except for the message for node $(0, X)$. Since this information does not exceed $\lfloor \frac{d}{3} \rfloor L$, it is sufficient to send a packet of size L during each step. In step $\lfloor \frac{d}{3} \rfloor + 1$, node $(0, X_0)$ forwards the message for $(0, X)$ to node $(0, X)$.

During the last $\lfloor \frac{d}{2} \rfloor - \lfloor \frac{d}{3} \rfloor$ steps, node $(0, X)$ scatters simultaneously in two arrays:

$$(0, X),\ (1, X),\ ...,\ (\lfloor \frac{d}{2} \rfloor - \lfloor \frac{d}{3} \rfloor, X) \text{ and } (0, X),\ (d-1, X),\ ...,\ (\lfloor \frac{d}{2} \rfloor + \lfloor \frac{d}{3} \rfloor + 1, X).$$

Notice that the first packet sent along the first of those arrays reaches (a, X) during step $\lfloor \frac{d}{3} \rfloor + 1$ at the earliest, because $0 < a$. This packet is for node $(\lfloor \frac{d}{2} \rfloor - \lfloor \frac{d}{3} \rfloor, X)$. Its size is $\frac{3-k}{3}L$. Node (a, X) may still have a packet to be transmitted during the next step. This could only be the packet of size $\frac{k}{3}L$ for node $(\lfloor \frac{d}{2} \rfloor - \lfloor \frac{d}{3} \rfloor, X)$. In this case we simply batch the two packets into one packet of size L and continue the scattering.

Similarly, we may have to batch packets for node $(\lfloor \frac{d}{2} \rfloor + \lfloor \frac{d}{3} \rfloor + 1, X)$. In any case, the entire algorithm takes $\lfloor \frac{d}{2} \rfloor$ steps and each step takes time $L\tau + \beta$. Each link is used in one direction only, and therefore the scattering time is

$$s_{F^*}(X) \le s_{H^*}(X) \le \lfloor \frac{d}{2} \rfloor (L\tau + \beta).$$

(b) $\lfloor \frac{d}{2} \rfloor \le a < b$.

In this case we assign a second label to each node from the cycle:

node (i, X), $0 \le i \le d-1$, is assigned a second label $(d + i - b \bmod d, X)$.

Therefore, the second labels of nodes (a, X), (b, X) and $(0, X)$ are $(d + a - b \bmod d, X)$, $(0, X)$ and $(d - b, X)$, respectively.

Since $\left\lfloor \frac{d}{2} \right\rfloor \le a < b$, it follows that $d - b \le \left\lfloor \frac{d}{2} \right\rfloor < d + a - b \bmod d$.

We can scatter in X using the second label of each node. The three originators are adjacent to three different nodes of the cycle and their second labels are correlated as in case 1(a). Therefore, we can use the same algorithm.

(c) $a < b \le \left\lfloor \frac{d}{2} \right\rfloor$.

The second label assigned to each node (i, X), $0 \le i \le d-1$, is $(d + i - a \bmod d, X)$. Nodes (a, X), (b, X) and $(0, X)$ are assigned second labels $(0, X)$, $(b - a, X)$ and $(d - a, X)$, respectively.

Since $a < b \le \left\lfloor \frac{d}{2} \right\rfloor$, it follows that $(d - a, X) \le \left\lfloor \frac{d}{2} \right\rfloor < b - a$.

Therefore, we can scatter in X as in case 1(a), using the second label of each node.

case 2. Cycles $X = \bar{0}_{0i}$, $1 \le i \le d-1$.

One of the originators is adjacent to node $(0, X)$, and the other originator is adjacent to node (a, X). Originator (a, X_a) contains two thirds of the information for the cycle. Notice, however, that in substep 1 of step d, a message of size $M = \frac{dL}{3}$ was sent to (a, X). Therefore, we can scatter from three originators -- $(0, X_0)$, (a, X_a) and (a, X), each containing a third of the information for the cycle. There exist two cases:

(a) $a < d - \left\lfloor \frac{d}{2} \right\rfloor$

We assign a second label to each node from the cycle:

node (i, X), $0 \le i \le d-1$, is assigned a label $(d + i - a \bmod d, X)$.

Therefore, the second labels of nodes (a, X), $(a+1, X)$ and $(0, X)$ are $(0, X)$, $(1, 0)$

and $(d-a, X)$, respectively.

Node (a, X) (original label) is adjacent to node $(a+1, X)$ (original label). Therefore, the three originators are adjacent to three different nodes of the cycle -- nodes with second labels $(0, X)$, $(1, X)$ and $(d-a, X)$.

Since $a < d - \left\lfloor \frac{d}{2} \right\rfloor$, it follows that $1 \le \left\lfloor \frac{d}{2} \right\rfloor < d - a$.

Therefore, we can scatter in X as in case 1(a), using the second label of each node. The only difference is that we use arc $(0, X) \rightarrow (1, X)$ instead of $(1, X_1) \rightarrow (1, X)$ to scatter from the originator adjacent to $(1, X)$. However, the first packet originating from node $(0, X_0)$ is sent over that arc in step $\left\lfloor \frac{d}{2} \right\rfloor + 1$. The size of the packet is $\frac{3-k}{3}L$. The last packet originating from node $(0, X)$ is sent over the arc in the same step and its size is $\frac{k}{3}L$. Therefore, we can batch those two packets into one packet of size L and continue the algorithm as usual.

(b) $a \ge d - \left\lfloor \frac{d}{2} \right\rfloor$

We use the same approach as in case 2(a). We assign the same second labels. Node (a, X) (original label) is adjacent to node $(a-1, X)$ (original label). Therefore, the three originators are adjacent to three different nodes of the cycle -- nodes with second labels $(0, X)$, $(d-1, X)$ and $(d-a, X)$. This time, $a \ge d - \left\lfloor \frac{d}{2} \right\rfloor$ and therefore $d - a \le \left\lfloor \frac{d}{2} \right\rfloor < d - 1$. Therefore, we can scatter in X using the second labels.

In CCCd[0 = 0] we have similar cases; however, one of the originators is always at $(0, X)$ instead of $(0, X_0)$. Therefore, we can apply the same techniques as in cases 1 and 2 above and keep that originator inactive during the first step of the corresponding algorithm.

The above-described scattering algorithms can be performed in parallel in all cycles. Therefore, the time required to complete phase 2 is the time required to finish the scattering in the worst case for all cycles.

Lemma 4.4:

Phase 2 in CCCd takes time

$$\left\lceil \frac{d}{2} \right\rceil L\tau + \left(\left\lceil \frac{d}{2} \right\rceil - \left\lfloor \frac{d}{3} \right\rfloor - 1 \right) L\tau + \left\lceil \frac{d}{2} \right\rceil \beta \text{ under } F^* \text{ and } H^*.$$

4.2 Butterflies

4.2.1 Phase 1

During the first phase we scatter packets of size $M = \frac{dL}{2}$, each of which contains half of the information of the d messages for a cycle. We use 2 arc-disjoint perfectly balanced binary trees. In the end of the phase, every node of each tree contains a message of size M.

We call the trees T0 and T1. T0 is an ascending binary subtree of height d-1 rooted at $(0, \bar{0})$. T1 is a descending binary subtree of height d-1 rooted at $(0, \bar{0})$. Therefore, they are 2 arc-disjoint perfectly balanced binary trees. Therefore, we can scatter along T0 and T1 in parallel using the algorithm described in Chapter 1.

Each non-leaf node of both trees has a cross neighbour from another cycle represented in its corresponding tree. Every cycle represented in T0 and T1 is reached only once through each tree (Lemma 2.1 and 2.3). Therefore, every non-leaf node of T0 and T1 has a unique cycle adjacent to it. If the trees were of height d rather than d-1, then their current leaves would have been non-leaf nodes and therefore they have unique cycles adjacent to them as well.

Lemma 4.5:

Phase 1 in BFd takes time $(2^{d-1} - 1) M\tau + (d-1)\beta$ under F^* and H^*.

Proof: The description of phase 1 assumed full-duplex links. A careful look at T0 and T1 will reveal that, except for the cycle $\bar{0}$, they are not only arc-disjoint but also edge-disjoint, since the only common edges used are the ones from cycle $\bar{0}$. This means that we can slightly modify the algorithm used in phase 1 to ensure the usage of each link in one direction at any given time. Obviously, we are concerned with the links of $\bar{0}$ only. The situation is almost identical to the scattering in $\bar{0}$ of CCCd during phase 1 (see section 4.1.1). We can apply exactly the same approach we used for CCCd -- swapping packets from T0 and T1 in the way described in section 4.1.1.

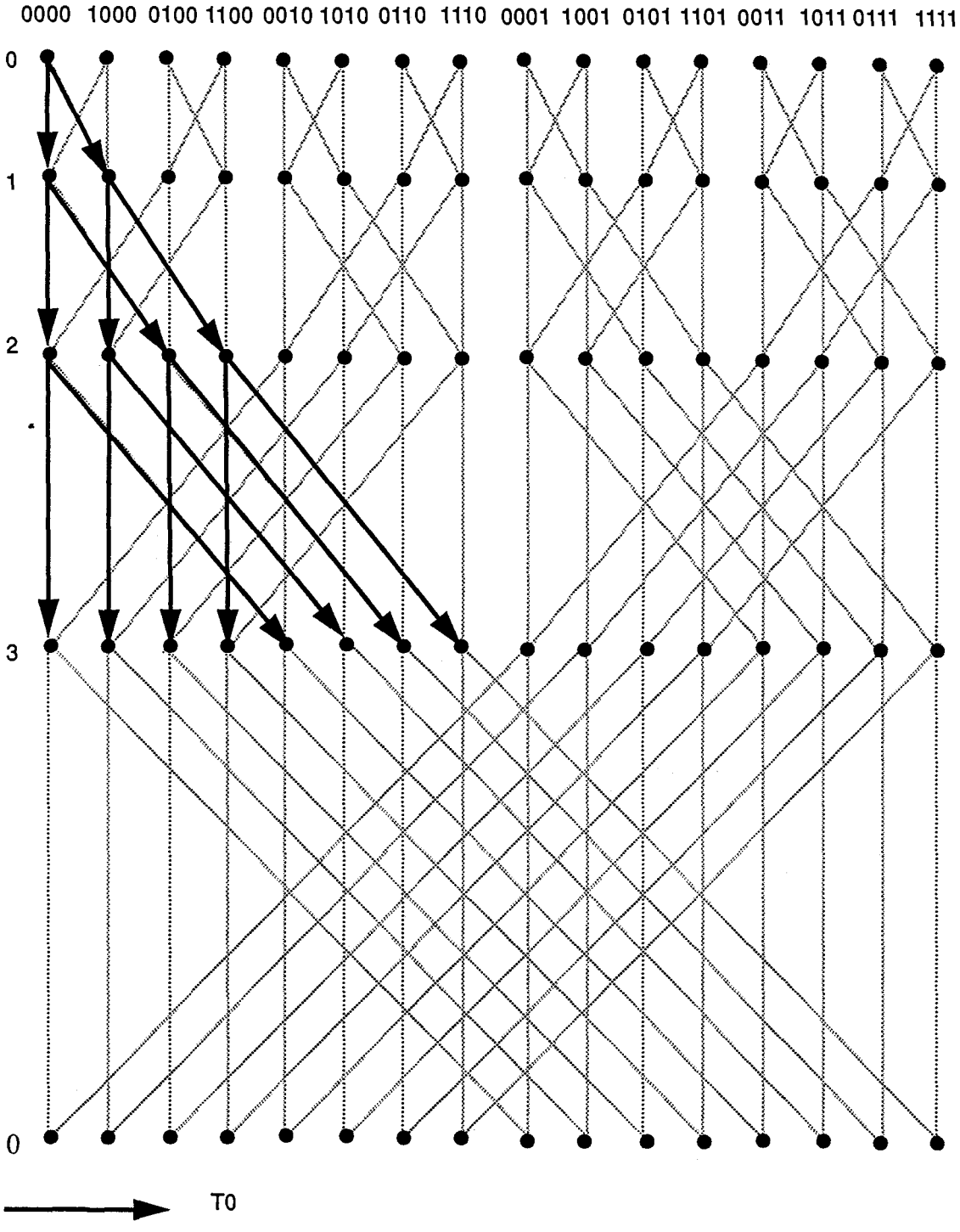


fig.4.5. T0 in BF4 after phase 1.

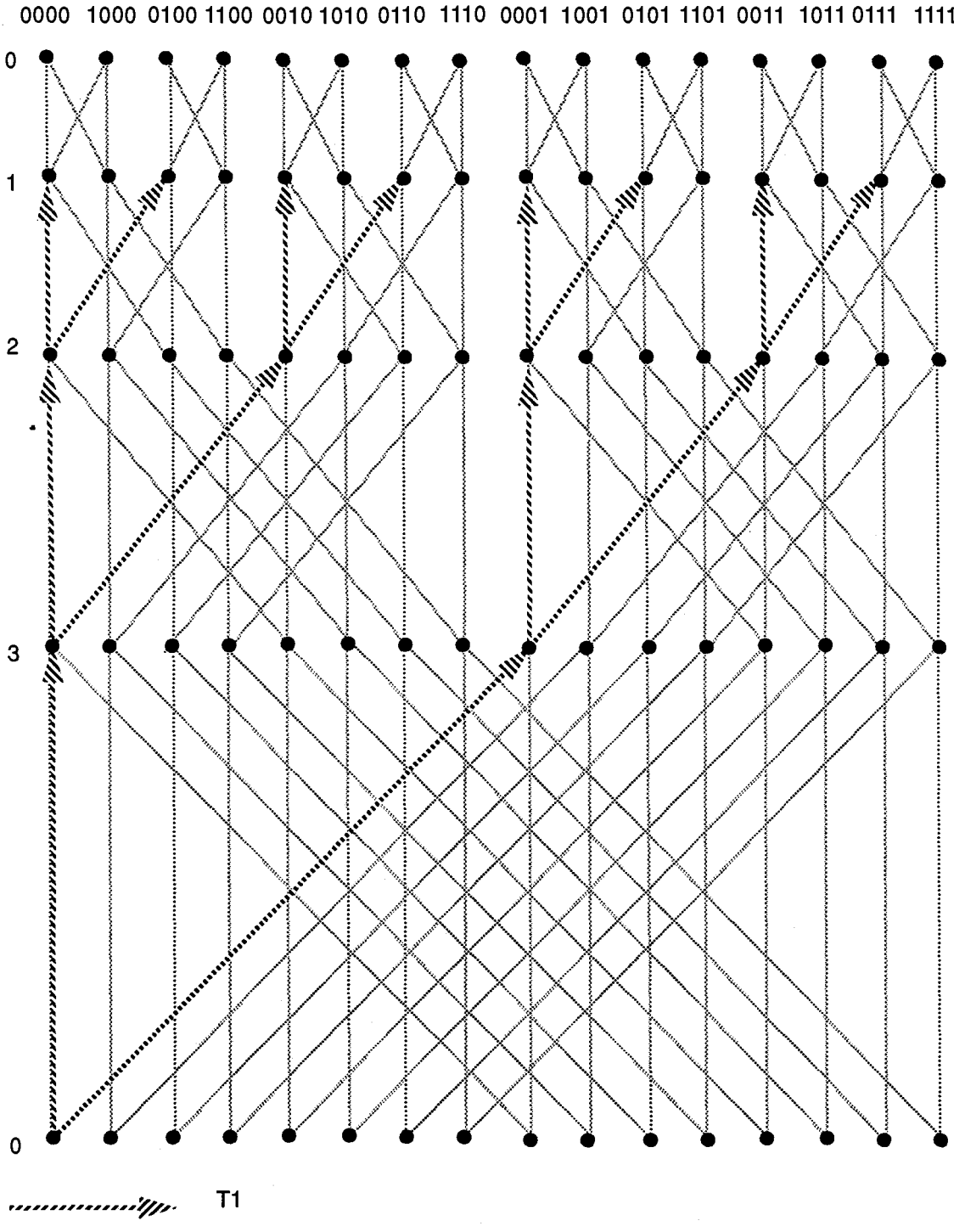


fig.4.6. T1 in BF4 after phase 1.

Thus, we make sure that tree T0 does not use the links $(i, \bar{0}) \rightarrow (i+1, \bar{0})$, where

$\left\lfloor \frac{d}{2} \right\rfloor \le i \le d-1$ and T1 does not need the rest of the links of $\bar{0}$. Consequently, the scattering in T0 and T1 finishes in the same time and no common links are used. Therefore, we can conclude that phase 1 takes the same time under both F^* and H^*. □

4.2.2 Phase 2

We scatter in $\bar{0}$ from one originator -- the root $(0, \bar{0})$, using the algorithm described in section 4.1.2.1 Therefore, $s_{F^*}(\bar{0}) \le s_{H^*}(\bar{0}) \le \frac{d-1}{2}L\tau + \left\lfloor \frac{d}{2} \right\rfloor \beta$.

After phase 1, for all cycles $X \neq \bar{0}$, there exist two nodes adjacent to X, such that each of them contains half of the information for the cycle. Those nodes are $(\alpha 1(X)-1, X_{\alpha 1(X)})$ and $(\beta(X)+1, X_{\beta(X)})$, and they are adjacent to $(\alpha 1(X), X)$ and $(\beta(X), X)$, respectively. According to Property 2.1, those two nodes are different for each cycle $X \neq \bar{0}$, and therefore we have two originators adjacent to two different nodes of the ring, each containing half of the information for the ring, i.e. $\frac{d}{2}L$ information units.

Assume without loss of generality that the originators are adjacent to nodes (a, X) and (b, X), and that $0 \le a < b \le d-1$. We separately analyze two cases for d.

Case 1: d is *even*.

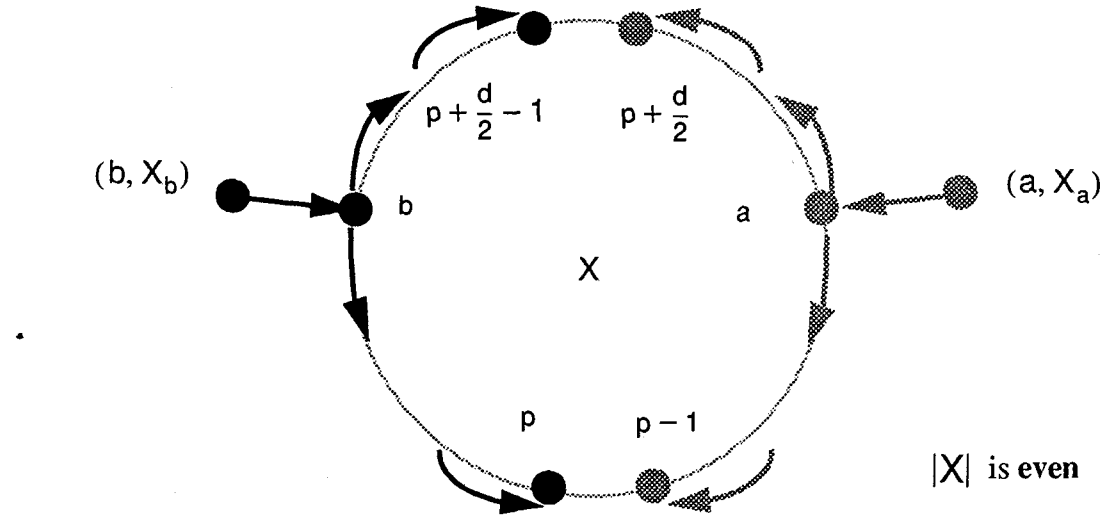
We know that $a < b$. Therefore, $b-a > 0$. We group all nodes $(i, X) \in X$, such that

$$b - \left\lfloor \frac{b-a}{2} \right\rfloor \le i < b - \left\lfloor \frac{b-a}{2} \right\rfloor + \frac{d}{2}$$

(the addition is modulo d), in one set called S_b, and the rest of the nodes -- in another set called S_a. Both sets are of cardinality $\frac{d}{2}$. Also, $(a, X) \in S_a$ and $(b, X) \in S_b$. Each of the originators contains $\frac{d}{2}$ messages of length L, which are to be sent to the nodes from their sets.

We consider two sets in order to scatter in S_b -- $B1 = \{(i, X) | i \ge b\} \cup \{(b, X_b)\}$ and $B2 = \{(i, X) | i \le b\} \cup \{(b, X_b)\}$. Observe that $B1 \cap B2 = \{(b, X), (b, X_b)\}$ and therefore $|B1 \cap B2| = 2$. Also, $B1 \cup B2 = S_b \cup \{(b, X_b)\}$ and, since $(b, X_b) \notin S_b$, it follows that $|B1 \cup B2| = \frac{d}{2} + 1$.

We scatter from (b, X_b) in B1 using the algorithm for scattering in arrays described in section 1.2.1. The algorithm finishes in $|B1| - 1$ steps. During the last step, node (b, X) receives its message M_b.



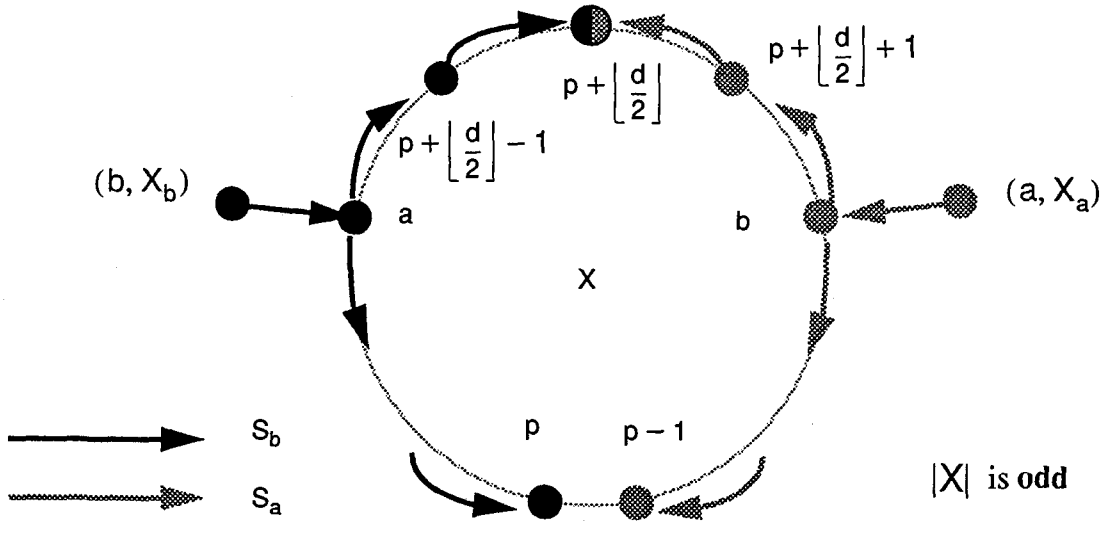


fig. 4.7. S_a and S_b in ring (cycle) X; $p = b - \left\lfloor \frac{b-a}{2} \right\rfloor$.

Then we scatter from (b, X_b) in B2 using the same algorithm, which takes another $|B2| - 1$

steps. Again, during the last step, node (b, X) receives M_b. Therefore, instead of sending M_b during the last step of the scattering in B1, the originator (b, X) could start scattering in B2 one step earlier. Thus, the total number of steps required to send the messages from (b, X_b) to all nodes of S_b is $|B1| + |B2| - 3$. But $|B1| + |B2| = |B1 \cup B2| + |B1 \cap B2| = \frac{d}{2} + 3$. Therefore, the total number of steps is d/2.

Similarly, we scatter in S_a, in d/2 steps, using two subsets of S_a -- . $\{(i, X) | i \geq a\} \cup \{(a, S_a)\}$ and $\{(i, X) | i \leq a\} \cup \{(a, S_a)\}$. A message of length L is sent over every link used at each step. No link is used in two directions. Therefore, the scattering time is:

$$s_{F^*}(X) \leq s_{H^*}(X) \leq \frac{d}{2}L\tau + \frac{d}{2}\beta.$$

Case 2: d is *odd.*

Again we split the nodes into two sets -- S_a and S_b.

$$S_b = \{(i, X) | (b - \left\lfloor \frac{b-a}{2} \right\rfloor) \leq i \leq b - \left\lfloor \frac{b-a}{2} \right\rfloor + \left\lfloor \frac{d}{2} \right\rfloor\}.$$

S_a contains the remaining nodes of X and the last element of S_b -- node $(b - \left\lfloor \frac{b-a}{2} \right\rfloor + \left\lfloor \frac{d}{2} \right\rfloor, X)$. Thus, $|S_a| = |S_b| = \lceil d/2 \rceil$ and one node belongs to both sets. Each of the originators contains the entire message for all nodes in its corresponding set but half of the message for the common node. Again, we scatter as in case 1; however, the first message sent from the originators is of length L/2 and it is destined for the common node. We need $\lceil d/2 \rceil$ steps to perform the algorithm, because S_a and S_b have $\lceil d/2 \rceil$ elements. The first step takes time $\beta + \frac{L}{2}\tau$ and all other steps take time $\beta + L\tau$. Each link is used in one direction only, and therefore the total time needed to complete the scattering is

$$s_{F^*}(X) \leq s_{H^*}(X) \leq \frac{d}{2}L\tau + \left\lceil \frac{d}{2} \right\rceil \beta$$

Since we scatter in all cycles of BFd in parallel, the time to complete phase 2 is given by:

Lemma 4.6

Phase 2 in BFd takes time $\frac{d}{2}L\tau + \left\lceil \frac{d}{2} \right\rceil \beta$ under F^* and H^*.

4.3 Upper bounds on scattering in CCCd and BFd under the various models

According to lemmas 4.3 and 4.4, the scattering time in CCCd under the F* and H* models is at most

$$(2^d-1)\frac{dL}{3}\tau+(2d-1)\beta+\left\lceil\frac{d}{2}\right\rceil L\tau+(\left\lceil\frac{d}{2}\right\rceil-\left\lfloor\frac{d}{3}\right\rfloor-1)L\tau+\left\lceil\frac{d}{2}\right\rceil\beta.$$

Since $\Delta(CCCd)=3$ we can divide each step of the scattering algorithm under F* (H*) in three substeps, and in each of those substeps use only one link at each node. Thus, the time needed under F1 (H1) is three times the scattering time under F* (H*). We denote d mod 3 by k.

Theorem 4.1:

If d is even then

$$s_{F^*}(CCCd)\le s_{H^*}(CCCd)\le (d2^d+d+k-3)\frac{L\tau}{3}+(2d+\frac{d}{2}-1)\beta,$$

$$s_{F1}(CCCd)\le s_{H1}(CCCd)\le (d2^d+d+k-3)L\tau+3(2d+\frac{d}{2}-1)\beta;$$

If d is odd then

$$s_{F^*}(CCCd)\le s_{H^*}(CCCd)\le (d2^d+d+k)\frac{L\tau}{3}+(2d+\left\lfloor\frac{d}{2}\right\rfloor)\beta,$$

$$s_{F1}(CCCd)\le s_{H1}(CCCd)\le (d2^d+d+k)L\tau+3(2d+\left\lfloor\frac{d}{2}\right\rfloor)\beta.$$

Both phases of the scattering in BFd take the same time under F* and H*. According to lemmas 4.5 and 4.6, this time is

$$(2^{d-1}-1)\frac{d}{2}L\tau+(d-1)\beta+\frac{d}{2}L\tau+\left\lceil\frac{d}{2}\right\rceil\beta.$$

To scatter under F1 and H1 we can divide each step of the algorithm into $\Delta(BFd)=4$ substeps and use only one link associated to a processor at every substep. Therefore,

Theorem 4.2:

$$s_{F^*}(BFd)\le s_{H^*}(BFd)\le 2^d d\frac{L\tau}{4}+(d+\left\lceil\frac{d}{2}\right\rceil-1)\beta$$

$$s_{F1}(BFd)\le s_{H1}(BFd)\le 4s_{F^*}(BFd)$$

CONCLUSION

In this thesis we have presented several algorithms for broadcasting and scattering in CCCd and BFd networks under the linear model of communication.

We have shown 3 arc-disjoint spanning trees of CCCd of height $3d+3$, while the shortest existing construction was of height $4d$ [FrHo91r]. We have given an explicit construction of 4 arc-disjoint spanning trees of BFd of height $2d+1$. To the best of our knowledge no other scheme for 4 ADST of BFd of shorter height has been proposed yet. Both constructions yield trees of a heights close to the diameter of the graphs. In both cases the difference is of order $\left\lfloor \frac{d}{2} \right\rfloor$. In addition, we have shown how to build 2 ADST of even smaller heights for both graphs. The heights of those trees differ from the graph diameter by a constant and thus they proved to allow faster broadcasting for short messages. The results were extended to both full- and half-duplex links with processor- and link-bound communication.

Even though our research brings the upper bounds on the broadcasting time in the studied networks very close to the existing lower bounds it still remains to be shown if it is possible to construct Δ ADST of smaller heights.

The problem of scattering under the linear cost model of communication has not been studied for CCCd and BFd before. In our work we have presented algorithms for scattering in those graphs. While the scattering time for CCCd differs from the existing lower bound by a term of order $dL\tau$, the scattering time for BFd is even closer to the lower bound. In addition, our algorithm for BFd applies to the half-duplex model as well.

To facilitate our proofs we have introduced the notion of compound arcs, ascending and descending binary subtrees. Those constructions have given us the opportunity to describe our algorithms in a uniform manner. It is our belief that the expressiveness of the constructions goes beyond the studied networks. They may be applied to other graphs derived from the hypercube to generate arc-disjoint trees of small depths and scattering algorithms under the linear cost model.

We have shown how to overcome the problem of scattering along a perfectly balanced binary tree containing compound arcs by splitting each step into two substeps in such way that after every step all nodes involved are informed on time. Our technique could be further exploited for other hypercube-derived networks as well.

In addition, more research has to be done in improving the existing lower bounds. They are all based on general considerations and thus do not capture the specific features of the particular graph. In fact, the start-up time and the propagation time do not appear together in any lower bound on the scattering under the linear model of communication.

BIBLIOGRAPHY

[AkKr86] S.Akers and B.Krishnamurthy. A group theoretic model for symmetric interconnection networks. *1986 International Conference on Parallel Processing*, 216-233, 1986.

[AnBaRo90] F.Annexstein, M.Baumslag and A.Rosenberg. Group action graphs and parallel architectures. *SIAM Journal of Computing*, 19, 544-569, 1990.

[BeFr91] J.C.Bermond and P.Fraigniaud. Communications in interconnection networks. *Proceedings Combinatorial Optimization in Science and Technology'91*, 1992.

[BeFrPe92] J.C.Bermond, P.Fraigniaud and J.Peters. Antepenultimate broadcasting. *Technical report, CMPT TR 92-03, School of Computing Science, Simon Fraser University*, 1992.

[BeHeLiPe92] J.C.Bermond, P.Hell, A.Liestman and J.Peters. Broadcasting in bounded degree graphs. *SIAM Journal of Discrete Mathematics*, 5, 10-24, 1992.

[BrCyHo91] J.Bruck, R.Cypher and C.T.Ho. On the construction of fault-tolerant cube-connected cycles networks. *1991 International Conference on Parallel Processing*, I692-I694, 1991.

[Fr90] P.Fraigniaud. Communications intensives dans les architectures a memoire distribuee et algorithmes paralleles de recherche de racines de polynomes. *Ph.D. thesis, Ecole Normale Superieure de Lyon, France*, LIP-IMAG, URA CNRS #1398, December 1990.

[FrHo91] P.Fraigniaud and C-T.Ho. Arc-disjoint spanning trees on cube-connected cycles networks. *1991 International Conference on Parallel Processing*, I225-I229, 1991.

[FrHo91r] P.Fraigniaud and C-T.Ho. Arc-disjoint spanning trees on cube-connected cycles networks. *Research Report RJ 7931 (72914), IBM*, January 1991.

[FrLa91] P.Fraigniaud and E.Lazard. Methods and problems of communication in usual networks. *Research Report #91-33, Lab. de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, France*, 1991, to appear in *Discrete Applied Mathematics*.

[FrMiRo90] P.Fraigniaud, S.Miguet and Y.Robert. Scattering on a ring of processors. *Parallel Computing*, 13, 377-383, 1990.

[HeHeLi88] S.D.Hedetniemi, S.T.Hedetniemi and A.Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18, 319-349, 1988.

[Ho90] C-T.Ho. Optimal communication primitives and graphs embeddings on hypercubes. *Ph.D. thesis, Yale University*, 1990.

[HrJeMo90] J.Hromkovic, C-D.Jeschke and B.Monien. Optimal algorithms for dissemination of information in some interconnection networks. *Proceedings of the 25th MFCS'90, Lecture Notes in Computer Science 452, 337-346*, 1990.

[JoHo89] S.L.Johnsson and C-T.Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38, 1249-1268, 1989.

[KlMoPei93] R.Klasing, B.Monien, R.Peine and E.Stohr. Broadcasting in butterfly and DeBruijn networks. *Report #113, Department of Mathematics and Computer Science, University of Paderborn, 1993*. To appear in *Discrete Applied Mathematics*, special issue on gossiping and broadcasting.

[LiPe88] A.Liestman and J.Peters. Broadcast networks of bounded degree. *SIAM Journal of Discrete Mathematics*, 1, 531-540, 1988.

[LiPe92] A.Liestman and J.Peters. Minimum broadcast digraphs. *Discrete Applied Mathematics*, 37/38, 401-419, 1992.

[MeCh90] D.Meliksetian and C.Y.R.Chen. Communication aspects of the cube-connected cycles. *1990 International Conference on Parallel Processing*, I579-I580, 1990.

[Pe77] M.C.Pease. The indirect binary *n*-cube microprocessor array. *IEEE Transactions on Computing C-26*, 5, 2-7, 1990.

[PeSy93] J.Peters and M.Syska. Circuit-Switched broadcasting in torus networks. *Technical Report CMPT TR 93-04, School of Computing Science, Simon Fraser University*, 1993.

[PrVu81] F.Preparata and J.Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24, 300-309, 1981.

[SaSch88] Y.Saad and M.Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 35, 867-872, 1988.

[SaSch89] Y.Saad and M.Schultz. Data communication in hypercubes. *Journal of Parallel and Distributed Computing*, 6, 115-135, 1989.

[Sh79] Y.Shiloach. Edge-disjoint branching in directed multigraphs. *Information Processing Letters*, 8, 24-27, 1979.

[Stohr91] E.Stohr. Broadcasting in the butterfly network. *Information Processing Letters*, 39, 41-43, 1991.

[StWa90] Q.Stout and B.Wager. Intensive hypercube communication, prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10, 167-181, 1990.