Technical Report CS-93-210, The University of Tennessee, Knoxville, USA, 1993.

Successive Broadcasts on Hypercube

Frédéric Desprez[*] Pierre Fraigniaud[†]
LIP, CNRS URA 1398
Ecole Normale Supérieure de Lyon
69364 Lyon Cedex 07, France
and
Bernard Tourancheau [‡§]
The University of Tennessee
Computer Science Department
107, Ayres Hall
Knoxville, TN 37996-1301
USA
e-mail: btouranc@cs.utk.edu

Abstract

Broadcasting is an information dissemination problem in which information originating at one node of a communication network must be transmitted to all the other nodes as quickly as possible [FLar, HHL86]. In this paper, we consider the problem in which all the nodes of a network must, by turns, broadcast a distinct message. We call this problem the *successive broadcasts* problem. Successive broadcasts is a communication pattern that appears in several parallel implementations of linear algebra algorithms on distributed memory multicomputers. Note that the successive broadcasts problem is different from the *gossip* problem [HHL86] in which all the nodes must perform a broadcast in any order, even simultaneously. We present an algorithm solving the successive broadcasts problem on hypercubes. We derive a lower bound on the time of any successive broadcasts algorithms that shows that our algorithm is within a factor of 2 of the optimality.

Key words: Broadcasting, hypercube, successive broadcasts problem.

1 Introduction

Many algorithms – particularly in linear algebra – have the following form. The data structure is a set S of n data $\mu_1, \ldots, \mu_n$. The algorithms perform in n steps. Let $S^{(0)} = S$, that is $\mu_j^{(0)} = \mu_j, j =$

[*]This work was supported by MRE grant No.974, the CNRS-NSF grant No.950.22 and the research program C3
[†]Supported by the research programs ANM and C3.
[‡]On leave from LIP, CNRS URA 1398, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.
[§]This work was supported in part by the National Science Foundation under grant ASC-8715728, the National Science Foundation Science and Technology Center Cooperative Agreement CCR-8809615, the DARPA and ARO under contract DAAL03-91-C-0047, CNRS-NSF grant 950.223/07, Archipel SA and MRE under grant 974, research program PRC C^3, and DRET.

$1, \ldots, n$. At step i, $1 \leq i \leq n$, the algorithms construct a set $S^{(i)} = \{\mu_j^{(i)}, j = 1, \ldots, n\}$ by

$$\mu_j^{(i)} = F(\mu_j^{(i-1)}, \mu_i^{(i-1)}), j = 1, \ldots, n,$$

where F is an *updating* function characterizing a particular algorithm. Let $t_{comp}(F)$ be the arithmetic time necessary to apply the function F. Such an algorithm has a cost $n^2 t_{comp}(F)$.

Now, such an algorithm can be implemented on a distributed memory multicomputer as follows. The set S is divided in p blocks, where p is the number of processors, such that each processor holds approximatively $\frac{n}{p}$ data. This allocation being fixed, each processor, by turns, computes a single value from the block it holds, and broadcasts this value to all the other processors. Then, each processor updates its data, another processor computes the next value and broadcast it, and so on. This process completes after n broadcasts. Thus, if $t_{broad}(p)$ is the communication time necessary to broadcast a single data from one processor to the $p-1$ other ones, then communications and computations can be easily scheduled in a time $n(t_{broad}(p) + \lceil \frac{n}{p} \rceil t_{comp}(F))$ when the broadcast are performed in distinct phases. The *arithmetic* speedup is then close to p. However, the communication cost implies that the efficiency of the algorithm will decrease as the number of processors increases. Indeed, for instance on hypercube, and it may be worse on other topologies, $t_{broad}(p) \geq \log_2(p)$, therefore the global communication time is $O(n \log_2(p))$.

In this paper, we present a communication scheduling that allows broadcasts initiated by successive sources to be pipelined on a hypercube so that the global communication time falls to $O(n + \log_2(p))$. Our result is based on the fact that it is possible to initiate different broadcasts by turns from different nodes of a hypercube, with no conflicts and insuring the correct march of the algorithm (each value is received at the due time).

Section 2 below formally states the *successive broadcasts problem*, and Section 3 describes how to solve it on a hypercube. Section 4 contains some concluding remarks.

2 Statement of the problem

Let us consider a distributed memory multicomputer with p processors labeled from 1 to p. Let S be a set of p data $\mu_1, \ldots, \mu_p$ distributed among the processors, each processor holding one data. The distribution depends on a function "alloc" that specifies the processor holding a given data: the processor alloc(j) holds μ_j. We consider the following algorithm to be scheduled on the multicomputer. This algorithm is given in term of instructions to be executed by any processor (the function "mynode" returns the label of the current processor).

Successive Broadcasts Algorithm

```
Begin
    For $i = 1$ to $p$ do
        If mynode $=$ alloc($i$) then $x \leftarrow \mu_i$;
        broadcast(alloc($i$), $x$);                          (1)
        store($x$);                                          (2)
End.
```

The Successive Broadcasts Algorithm consists in p iterations. Each iteration deals with a different data $\mu_i, i = 1, \ldots, p$. At iteration i, Processor alloc(i), holding μ_i, broadcasts μ_i to all the other processors (Instruction (1)). When a processor different than alloc(i) encounters the instruction "Broadcast(alloc(i), x)", it means that he will wait upon reception of a value x from one of its neighboring processors and, eventually, he will continue the broadcast by forwarding this value to some of its neighbors following the protocol adapted to a broadcast from node alloc(i). After having received, and eventually forwarded, the most recent value x, each processor stores this value (Instruction (2)).

Our problem is to organize the communications with respect to the task order of the Successive Broadcasts Algorithm. For instance two values μ_{i_0} and μ_{i_1} should not arrive in a processor in a reverse order (eventually, if this situation appends, we should be able to know in advance the order or reception so that every processor can correctly schedule the storage of the data). Note that organizing the communications depends on finding a adapted allocation of the element μ_i and, this allocation being fixed, finding an adapted scheduling of the broadcast operations.

There is an easy, but not efficient, way to implement the Successive Broadcasts Algorithm, by performing p separated broadcasts. Now, the global bandwidth of the network will certainly be inefficiently used during each broadcast (especially if the size of the data is small). Therefore it should be faster to interleave several broadcasts, still up to the respect of the task order of the algorithm and the communication constraints. This is what we will do on any d-dimensional hypercube Q_d.

Notations

Q_d is the graph of $p = 2^d$ vertices labeled from 0 to $2^d - 1$ such that there is an edge between two vertices x and y if and only if their binary representations differ in exactly 1 bit. For a vertex x of binary representation on d bits $x_{d-1}x_{d-2}\ldots x_1x_0$, each subscript i denotes a different *dimension*. For two vertices x and y, $\delta(x, y)$ denotes the Hamming distance between x and y, that is the number of bits from which they differ. If $x \oplus y$ denotes the bitwise xoring of x and y ($(x \oplus y)_i = x_i + y_i \pmod 2, i = 0, \ldots, d-1$), and $|x|$ denotes the number of bit 1 in x, then $\delta(x, y) = |x \oplus y|$. For any bit value x_i, $\overline{x_i}$ denotes its complement.

Communication constraints

We assume the following communication constraints. Each processor is supposed to be able to deal with only one atomic message at any given time. In particular, a processor cannot simultaneously receive more than one message. However, we assume that any processor can send a message simultaneously to all its neighbors. The communication process is thus *whispering* for the reception, and *shouting* for the emission [FLar]. Moreover, a processor cannot simultaneously send and receive. Finally, we count 1 for the time to send an atomic message from any processor to its neighbors.

3 Successive broadcasts on hypercube

Our goal is to find an allocating function "alloc" and to organize the communications so that, for each processor, the storages of Instruction (2) will be performed in the correct order, that is $\mu_1, \mu_2, \ldots, \mu_p$. An implementation of the Successive Broadcasts Algorithm satisfying this condition on the order of the storages is said *valid*. Of course we want also our implementation to respect the communication constraints.

There exists a valid implementation of the Successive Broadcasts Algorithm performing in time $p \log_2 p$ by completing each broadcast before initiating the next one. Indeed, a broadcast in an hypercube with p processors can be done in $\log_2 p$ time under the communication constraints stated before. However, it is possible to do much better:

Proposition 1
There exists a valid implementation of the Successive Broadcasts Algorithm performing in time $2p + \log_2 p - 2$ under the specified communication constraints.

To prove this proposition, we need to recall the definition of a *spanning binomial tree* of an hypercube, and its different rotations (see [JH89]).

Definition. *The* spanning binomial tree *rooted at a vertex x of Q_d is denoted $SBT(x)$. For any vertex $u = u_{d-1} \dots u_1 u_0$ of Q_d, let k be the dimension satisfying $(u \oplus x)_k = 1$ and $(u \oplus x)_i = 0, \forall i < k$ ($k = -1$ if $u \oplus x = 0$). Let $M_u = \{k-1, \dots, 0\}$ (eventually empty if $k \leq 0$). Then the father of u in $SBT(x)$ is*

$$\begin{cases} u_{d-1} \dots u_{k+1} 0 u_{k-1} \dots u_0 & \text{if } k \neq -1; \\ \emptyset & \text{if } k = -1; \end{cases}$$

and his k children are

$$\begin{cases} u_{d-1} \dots u_{k+1} 1 u_{k-1} \dots \overline{u_j} \dots u_0, \forall j \in M_u & \text{if } k \neq -1; \\ u_{d-1} \dots \overline{u_j} \dots u_0, \forall j \in \{0, \dots, d-1\} & \text{if } k = -1. \end{cases}$$

Note that $SBT(x) = x \oplus SBT(0)$ where the xoring applies to each vertex of $SBT(0)$.

For any vertex x of Q_d, $R(x)$ denotes its left bit rotation, that is

$$R(x_{d-1} x_{d-2} \dots x_1 x_0) = x_{d-2} \dots x_1 x_0 x_{d-1}.$$

For $j > 1$, the j-th left rotation is defined as $R^j = R \circ R^{j-1}$ ($R^0 = R^d = Id$). This operator can be applied on spanning binomial trees on all the vertices since it preserves the adjacency. Note that a vertex u has no child in $R^j(SBT(x)), j \in \{0, \dots, d-1\}$ if and only if $u_j = \overline{x_j}$. This property is the key of the proof of Proposition 1 as one will see later in the proof of Lemma 3. Figure 1 shows $SBT(000), SBT(011)$ and $R^1(SBT(011))$ in Q_3.

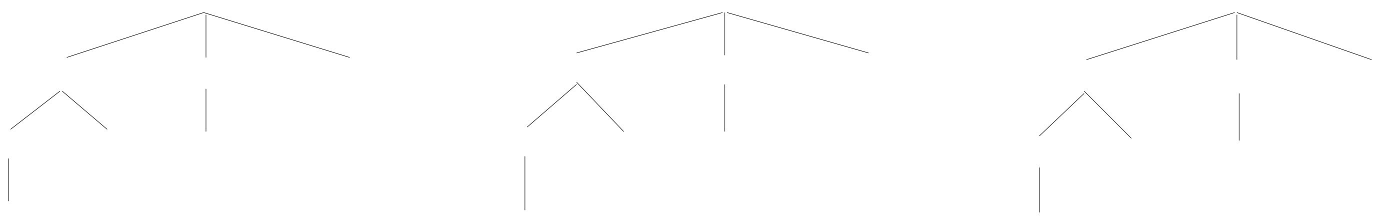


Figure 1: $SBT(000), SBT(011)$ and $R^1(SBT(011))$ in Q_3.

Any of the trees $R^j(SBT(x)), j = 0, \dots, d-1$ can be used for broadcasting a message from x [JH89]: at each step, the message is sent from all the vertices at the same level in the tree simultaneously to all their children in the tree. The levels are labeled from 0 to d, the root being the only vertex of level 0. Such a broadcast takes d steps since the depth of these trees is d. More precisely, if u is at distance $\delta(u, x)$ from the source x, u will receive the message after $\delta(u, x)$ steps since these trees contains only shortest path between any vertex and his children in the trees.

We are now ready to give our allocation and broadcasting strategies.

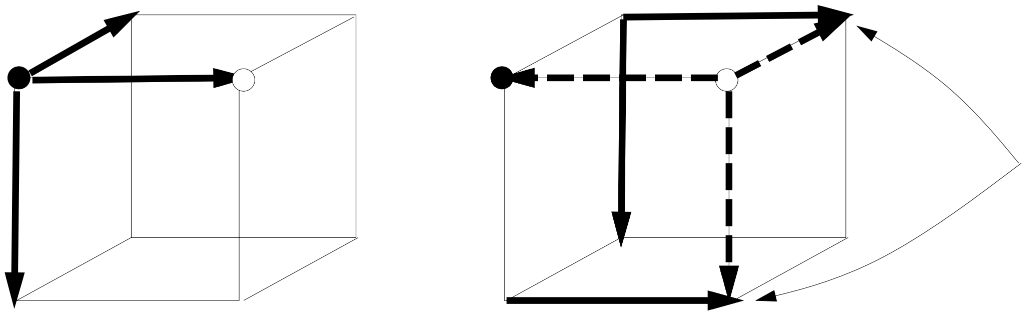


Figure 2: A conflict occurs when two SBT's are initiated at each step.

Allocation strategy

Our allocation uses the Binary Reflected Gray Code sequence $\mathrm{BRGC}(d) = \{x^{(1)}, x^{(2)}, \ldots, x^{(2^d)}\}$ (see for instance [Joh87]). Recall that $\mathrm{BRGC}(1) = \{0, 1\}$ and $\mathrm{BRGC}(d) = \{0\mathrm{BRGC}(d-1), 1\overline{\mathrm{BRGC}(d-1)}\}$ where $\overline{\mathrm{BRGC}(d)}$ denotes $\mathrm{BRGC}(d)$ in the reverse order. For instance $\mathrm{BRGC}(2) = \{00, 01, 11, 10\}$ and $\mathrm{BRGC}(3) = \{000, 001, 011, 010, 110, 111, 101, 100\}$. We define $\mathrm{alloc}(j) = x^{(j)}$, $1 \leq j \leq 2^d$ that is the j-th element of the BRGC sequence.

Broadcasting strategy

First we describe the broadcasting algorithm used by a processor $x^{(j)}, j \in \{1, \ldots, 2^d\}$. Let ν be such that $x^{(j)} \oplus x^{(j+1)} = 2^\nu$ (with $x^{(2^d+1)} = x^{(1)}$) that is $x^{(j)}$ and $x^{(j+1)}$ differs in dimension ν. Then, $x^{(j)}$ uses $R^\nu(SBT(x^{(j)}))$ to broadcast μ_j. Note that when $x^{(j)}$ performs its broadcasts, $x^{(j+1)}$ will receive μ_j at the first step. Moreover, by construction, $x^{(j+1)}$ is a leaf in $R^\nu(SBT(x^{(j)}))$.

Now we have to specify how these different broadcasts can be scheduled successively. Since $x^{(j+1)}$ is a leaf in $R^\nu(SBT(x^{(j)}))$, after reception of μ_j, and assuming that all the previous values have been received, $x^{(j+1)}$ is ready to begin the broadcast of μ_{j+1}. However, if $x^{(j+1)}$ performs the first step of its broadcast when $x^{(j)}$ performs the second step of its broadcast, a conflict will occur (see for instance Figure 2). Our broadcast scheduling is thus as follows: after reception of μ_j, Processor $x^{(j+1)}$ waits one top, and then begins the broadcast of μ_{j+1}. There is no modification of the broadcasts which are still performed level by level as specified before. Only the time when the different broadcast will be initiated is specified here: every two steps, a new broadcast begins.

Lemma 1 *The allocation strategy and the broadcast scheduling specified above produce a* valid *implementation of the Successive Broadcast Algorithm.*

Proof: We have to show that the order of reception of the sequence $\mu_j, j = 1, \ldots, p$ is correct. Assume it is not, and let i be the smallest integer such that $x^{(i)}$ receives two values in a wrong order, that is receives $\mu_{j''}$ before $\mu_{j'}$ for some j' and j'' satisfying $j'' > j'$.

Let $t_{j'}$ and $t_{j''}$ be respectively the emission time of $\mu_{j'}$ by $x^{(j')}$ and the emission time of $\mu_{j''}$ by $x^{(j'')}$. Since there are $j'' - j' - 1$ vertices between $x^{(j')}$ and $x^{(j'')}$ in the Gray code sequence, $t_{j''} \geq t_{j'} + 2(j'' - j')$, and $\delta(x^{(j')}, x^{(j'')}) \leq j'' - j'$.

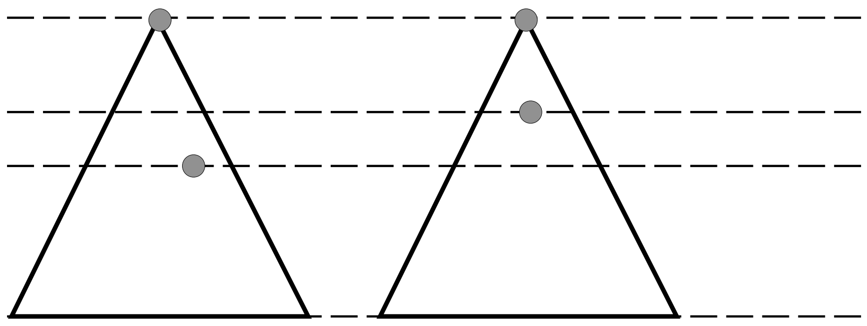
Now, since $x^{(i)}$ receives $\mu_{j''}$ before $\mu_{j'}$, $t_{j''} + \delta(x^{(j'')}, x^{(i)}) \leq t_{j'} + \delta(x^{(j')}, x^{(i)})$ Thus $2(j'' - j') + \delta(x^{(j'')}, x^{(i)}) \leq \delta(x^{(j')}, x^{(j'')}) + \delta(x^{(j'')}, x^{(i)})$. Therefore $\delta(x^{(j')}, x^{(j'')}) \geq 2(j'' - j')$, and then $j' = j''$, a contradiction. $\square$

Lemma 2 *Assuming there is no conflicts, the broadcast scheduling specified above allows to perform the Successive Broadcasts Algorithm in time* $2p + \log_2 p - 2$.

Proof: A new broadcast is initiated every 2 tops. After $2(p-1)$ tops, the last broadcast begins and completes in $\log_2 p$ tops. $\square$

Lemma 3 *The broadcast scheduling specified above performs the Successive Broadcasts Algorithm without any conflicts.*

Proof: Let $T(x^{(i)})$ and $T(x^{(j)})$ be the broadcasts trees, as specified before, of $x^{(i)}$ and $x^{(j)}$ respectively, $i < j$. Let u be a vertex of Q_d at level l_i in $T(x^{(i)})$, and v another vertex of Q_d at level l_j in $T(x^{(j)})$. Assume that, for some time t, u is active in the broadcast of $x^{(i)}$ and v is active in the broadcast of $x^{(j)}$. Note that this assumption implies $l_j < l_i$. The question is: is it possible that $u = v$?



There are mainly two cases:

1. At time t, u and v are *sending* μ_i and μ_j respectively. In that case, $l_i - l_j = 2(j - i)$. On the other hand, $x^{(i)}$ and $x^{(j)}$ differs in at most $j - i$ bits. Now, by construction of the broadcast trees,

$$\begin{aligned} & u = x^{(i)} \oplus \hat{u} \quad \text{where } \hat{u} \in R^r(SBT(0)) \text{ at level } l_i \\ \text{and} \quad & v = x^{(j)} \oplus \hat{v} \quad \text{where } \hat{v} \in R^s(SBT(0)) \text{ at level } l_j \end{aligned}$$

for some integers r and s. Therefore

$$u = v \Rightarrow \hat{u} \oplus \hat{v} = x^{(i)} \oplus x^{(j)}.$$

Again by construction of the trees, $|\hat{u} \oplus \hat{v}| \geq l_i - l_j = 2(j - i)$ and $|x^{(i)} \oplus x^{(j)}| \leq j - i$. Thus $u \neq v$, otherwise $i = j$.

The case where, at time t, u and v are *receiving* μ_i and μ_j respectively, can be treated similarly.

2. At time t, u is *sending* μ_i and v is *receiving* μ_j. In that case, $l_i - l_j = 2(j-i) - 1$. Thus if $j - i > 1$, a similar argument as in case 1 shows that $u \neq v$.

   Assume that $j = i + 1$ and denote ν the dimension in which $x^{(i)}$ and $x^{(i+1)}$ differ. Node u is at level l_i in $T(x^{(i)})$ and v is at level $l_i - 1$ in $T(x^{(i+1)})$. Thus, if $u = v$ then $u \oplus x^{(i)}$ has exactly l_i bits 1, and $u \oplus x^{(i+1)}$ has exactly $l_i - 1$ bits 1. Therefore $u_\nu = \overline{(x^{(i)})_\nu}$, and u is a leaf in $T(x^{(i)}) = x^{(i)} \oplus R^\nu(SBT(0))$: a contradiction with the fact that u is *sending* μ_i. Therefore $u \neq v$. □

Proof of Proposition 1: directly follows from Lemmas 1, 2 and 3. □

Lower Bound. Concerning a lower bound of our problem, we state the following results:

Proposition 2 *Let $\alpha(d)$ be the maximum, over all the spanning trees T of Q_d, of the number of leaves of T. Any implementation of the Successive Broadcasts Algorithm with no conflict performs in a time at least $2p - \alpha(d) - 1$ where $p = 2^d$.*

Proof: In any implementation of the Successive Broadcasts Algorithm with no conflict, any processor has to receive $p-1$ messages, and to send its own message. Moreover, let $T(x)$ be the broadcast tree of processor $x, x \in \{0, \ldots, p-1\}$. The number of leaves of $T(x)$ is smaller than $\alpha(d)$. Therefore, during the broadcast from x, at least $p - \alpha(d) - 1$ vertices have to forward the message after reception. Thus during the whole implementation of the Algorithm, there are at least $p(p - \alpha(d) - 1)$ forwarding operations. Therefore, there exists a vertex x_0 who forwards at least $p - \alpha(d) - 1$ messages. Thus any implementation of the Successive Broadcasts Algorithm needs a time of at least $(p-1) + 1 + (p - \alpha(d) - 1)$. □

Note that $\alpha(d) \geq \frac{p}{2}$. Indeed, this bound is reached by the Spanning Binomial Tree. However, for $d \geq 4$, it is possible to do better "by hand": $\alpha(4) \geq 10$. Therefore, it may be possible to perform faster than $\frac{3}{2}p + o(p)$. On the other hand, it is impossible to perform in $p + o(p)$.

Simulations. We performed some simulations of different successive broadcasts implementations and generated the corresponding time-stamped traces. We obtain the Figures with a re-execution of the generated data on the ParaGraph[1] tool.

The successive broadcasts are started following a Gray code in the hypercube. Figure 3 presents a simulation of the successive broadcasts using a classical spanning binomial trees in a synchronised routine. Figure 4 presents a simulation of the interleaved spanning binomial trees and Figure 5 presents a simulation of our implementation that shows how the rotated spanning binomial broadcast trees are interleaved in a 4-cube. Remark the effective gain in time as the time scale is the same on each Figures and that only 11 trees are represented in Figure 3, while the 16 trees are shown in Figure 4 and Figure 5.

4 Conclusion

We studied in this paper the *successive broadcasts* problem in which the nodes of a network have to perform by turns a broadcast in a specified order. This problem comes from parallel implementation of linear algebra algorithms on distributed memory multicomputers.

[1]Available via internet : send an email message containing "send index from paragraph" to netlib@ornl.gov.

A straightforward scheduling performs in time $O(p \times T_{broadcast}(p))$ in any network. We show that, in hypercubes, it is possible to perform successive broadcasts in time $O(p + \log_2 p)$ by using individual broadcasting based on the rotations of the spanning binomial tree. We showed that our approach is within a factor of 2 of the optimality. An open problem stays to find the exact complexity of the successive broadcasts problem in hypercube. We derived a lower bound based on counting the maximum number of leaves $\alpha(d)$ of any spanning tree of the d-cube. However, we did not succeed in finding the explicit value of $\alpha(d)$, and this value may be not sufficient to give a tight lower bound.

Besides its many applications in linear algebra, the successive broadcasts problem is interesting by itself. In particular it could be of first interest to study this problem on other topologies as grids and meshes, and under other communication constraints as specified in [FLar].

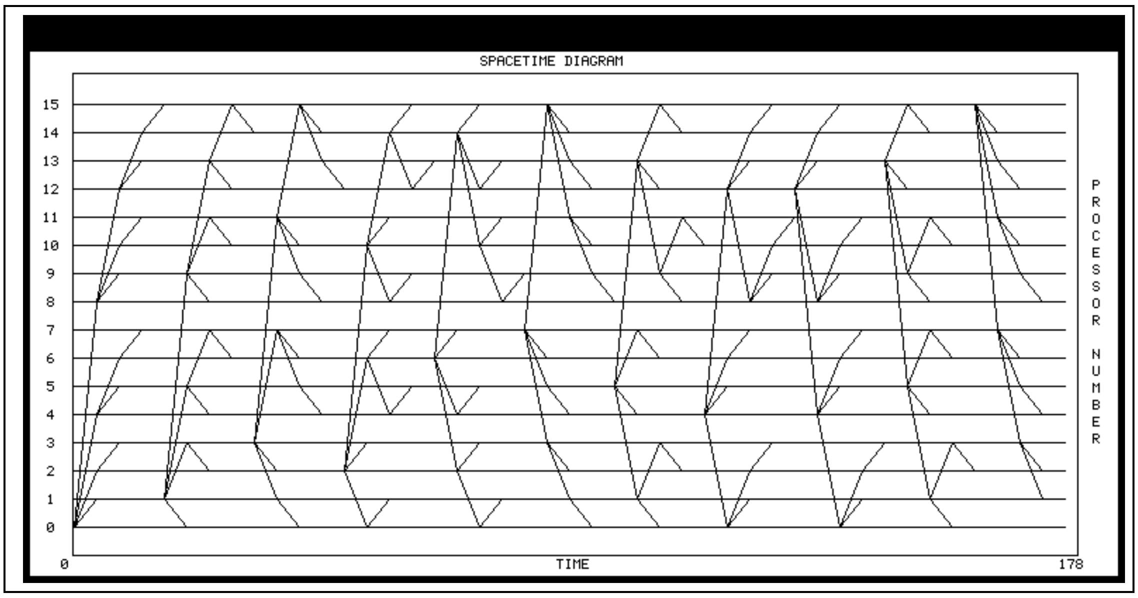


Figure 3: Several spanning binomial broadcast trees synchronised during the Successive Broadcasts Algorithm on a 4-cube.

References

[FLar] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Maths (special issue on broadcasting)*, (to appear).

[HHL86] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1986.

[JH89] S.L. Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transaction on Computers*, 38(9):1249–1268, 1989.

[Joh87] S.L. Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Paralle and Distributed Computing*, 4:133–172, 1987.

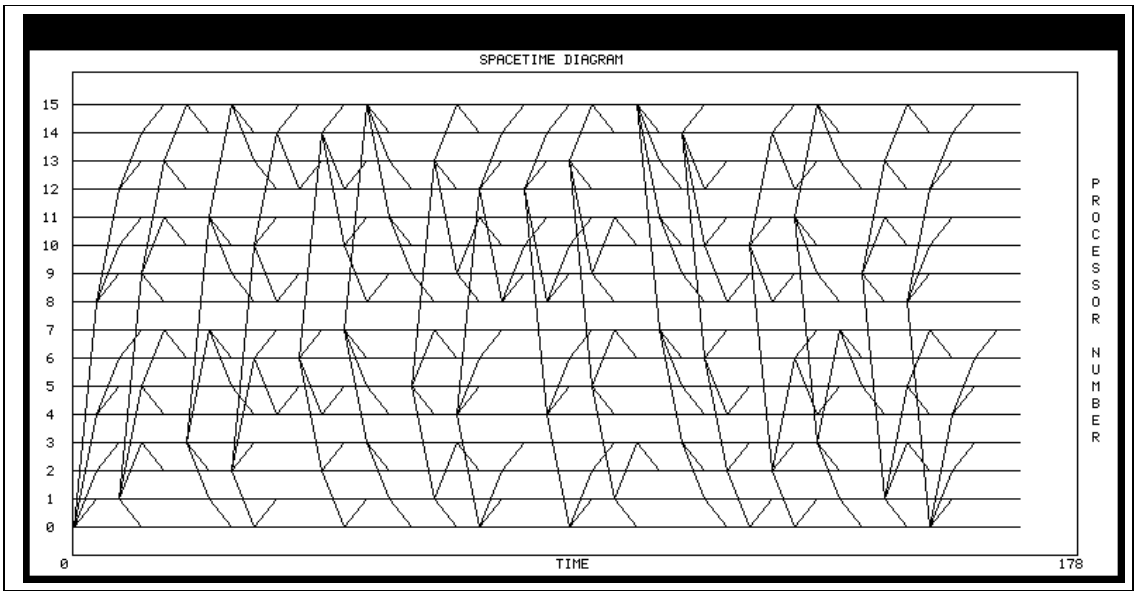


Figure 4: Interleaving of the spanning binomial broadcast trees during the Successive Broadcasts Algorithm on a 4-cube.

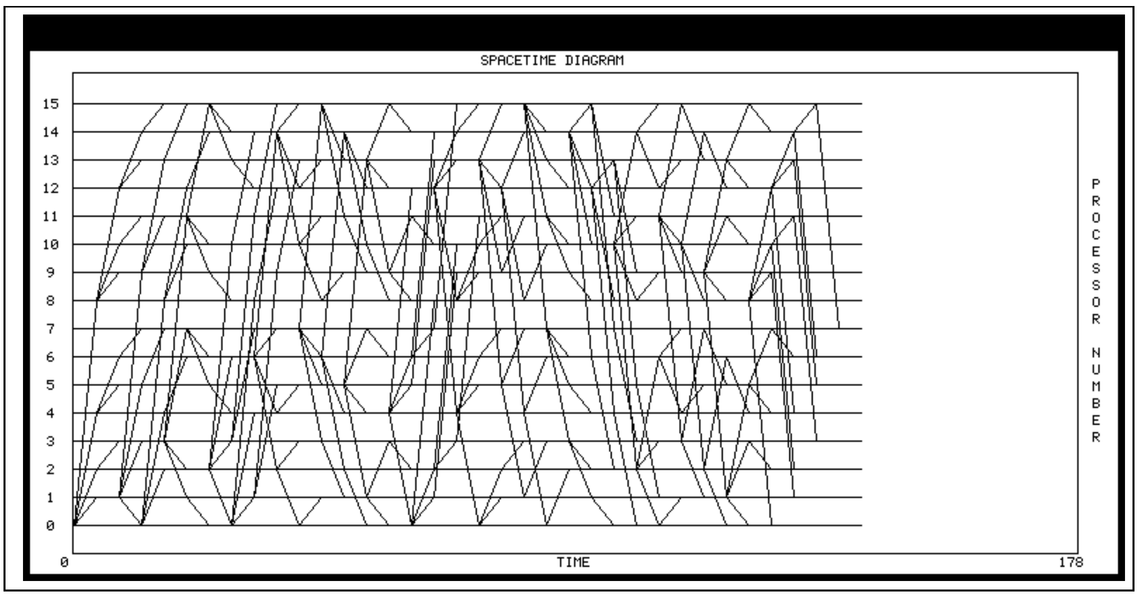


Figure 5: Interleaving of the different rotated spanning binomial broadcast trees during our implementation of the Successive Broadcasts Algorithm on a 4-cube.