Optimal Broadcast and Summation in the LogP Model

Richard M. Karp[*], Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser

Computer Science Division,
University of California, Berkeley

Abstract

In many distributed-memory parallel computers the only built-in communication primitive is point-to-point message transmission, and more powerful operations such as broadcast and synchronization must be realized using this primitive. Within the LogP model of parallel computation we present algorithms that yield optimal communication schedules for several broadcast and synchronization operations. Most of our algorithms are the absolutely best possible in that not even the constant factors can be improved upon. For one particular broadcast problem, called *continuous broadcast*, the optimality of our algorithm is not yet completely proven, although proofs have been achieved for a certain range of parameters. We also devise an optimal algorithm for summing or, more generally, applying a non-commutative associative binary operator to a set of operands.

1 Introduction

Most models of parallel computation reflect the communication bottlenecks of real parallel machines inadequately. The PRAM [11], for example, allows interprocessor communication at zero cost. Researchers have proposed several variations on the PRAM that address particular aspects of interprocessor communication such as latency, memory contention and synchronization [16, 15, 17, 1, 2], but no single model properly accounts for all these aspects. Valiant's BSP model [18] is more realistic, but imposes a rigid programming style in which a parallel algorithm is expressed as a series of *supersteps*, where each superstep terminates with a barrier synchronization involving all the processors. There have also been a number of studies of parallel communication on specific networks such as the hypercube or mesh.

Recently, Culler et al [9] proposed a general purpose model, the *LogP model* for distributed-memory machines. In this model, processors work asynchronously and communicate by point-to-point messages that travel through a network. The model describes a parallel computer by four parameters:

P, the number of processor-memory pairs;

L, the *latency*, or maximum delay, associated with delivering a message;

g, the *gap*, a lower bound on the time between the transmission of successive messages, or the reception of successive messages, at the same processor;

o, the *overhead*, representing the length of time for which a processor is busy during the transmission or reception of a message.

The parameters L, o and g are measured in units of processor cycles. The model also specifies that the network has a *finite capacity*, such that at most $\lceil L/g \rceil$ messages can be in transit from any processor or to any processor at any time. All algorithms presented in this paper satisfy the capacity constraint of the LogP model, and we do not mention it henceforth. Although the model is asynchronous, in estimating the running time of the algorithms, we assume that that all processors work synchronously and that each message incurs the full latency of L.

Since point-to-point message transmission is the only primitive communication operation provided on some distributed-memory machines, more powerful broadcast and synchronization primitives must be realized in terms of this single operation. We present optimal implementations of several variants of broadcast and synchronization within the LogP model. We also present an optimal algorithm for summing n operands on P processors; this algorithm is derived by viewing the communication pattern of an optimal summation algorithm as the time reversal of an optimal broadcast pattern.

Specifically, we study six fundamental communication problems in the framework of the LogP model:

- The *single-item broadcast problem* in which a data item residing in a source processor is to reach all P processors in minimum time.
- The *k-item broadcast problem* in which k data items residing in a source processor are to reach all P processors in minimum time.
- The *continuous broadcast problem*, in which a source processor generates a new item every g units of time, and all of these items are required to reach all P processors. Associated with any schedule for this problem is its *delay*, defined as the maximum time that can elapse between the creation of a item and its arrival at the last processor to receive it. The problem is to construct a schedule with minimum delay.

[*]Also affiliated with International Computer Science Institute, Berkeley.

- The *all-to-all broadcast problem*, in which each processor is the source of a data item, and all P items are required to reach all processors in minimum time.
- The *combining-broadcast problem* in which each processor has a value, and all processors must learn in minimum time a single reduced value which is computed using an associative and commutative operation on the list of P values.
- The *summing problem*, in which n numbers are to be added on P processors in minimum time, assuming that the initial distribution of the numbers among the processor/memory pairs may be stipulated by the algorithm. (Here, "addition" means the application of any binary associative operator.)

Given the considerable importance of broadcasting problems in parallel and distributed computation, several variations of it have been well studied in the literature[7, 12]. Much of this work has focused on the design of efficient algorithms for broadcasting on specific networks such as hypercubes [13, 14]. For fully connected systems (such as those modeled by LogP) broadcast problems have been studied in several communication models, but without latency. Cockayne and Thomason [8] and Farley [10] gave optimal algorithms for a model where each processor can either send or receive a message in one time step. Alon, Barak and Manber [3] studied reliable broadcast in a model that allows simultaneous send and receive.

Bar-Noy and Kipnis [5] recently introduced the *postal model* which incorporates a latency parameter in addition to stipulating that at most one message be sent or received by a processor per time step. This turns out to be a special case of the *LogP* model (with $g = 1$ and $o = 0$) and is indeed, quite suitable for studying "communication-only" problems. Note that if message events are the only events of interest, the overhead charged in *LogP* for sends and receives can be incorporated into the latency parameter, eliminating o. We can then normalize so that $g = 1$, yielding the postal model. For this model, Bar-Noy and Kipnis have given an optimal algorithm for the single-item broadcast problem as well as a sub-optimal algorithm for the multiple-item broadcast problem [5, 4].

The remainder of the paper is organized as follows. Section 2 defines the single-item broadcast problem and develops an optimal algorithm for it. It also provides an illustration of the roles of the various parameters of the *LogP* model. Section 3 focuses on the much harder problems of k-item broadcast and continuous broadcast, which are analyzed in the postal model. For continuous broadcast, we derive a lower bound on the delay of any schedule. We define *block-cyclic* schedules which admit of a succinct representation and give a general construction of a block-cyclic schedule which is conjectured to satisfy the lower bound for large enough P whenever $L \neq 2$. Thus far the conjecture has been proven for $L \leq 10$ and for infinitely many but not all P. We also prove that when $L = 2$ the lower bound cannot in general be achieved, but we give a construction that comes within one time step of the bound, and thus is optimal. For k-item broadcast, we characterize the structure of optimal schedules and derive lower bounds on their completion time. We also derive lower bounds on the completion time of *single-sending* schedules, in which the source processor sends each item only once. We present a single-sending schedule that is within $L - 1$ steps of the single-sending lower bound (and hence, within $2L - 1$ steps of the general lower bound.) We also show that if the communication model is modified to provide each processor with an input buffer of size 2, then the single-sending lower bound can be achieved. Section 4 discusses the all-to-all broadcast and the combining-broadcast problem and presents their optimal solutions. Finally, Section 5 solves the summing problem.

2 The Single-Item Broadcast Problem

The *single-item broadcast* problem is that of finding a schedule of communication among P processors (numbered $1, \dots, P$) so that a datum initially available at processor 1 is made available to all P processors in the shortest possible time.

Definition 2.1 *Let $\mathcal{A}$ be an algorithm for single-item broadcast. The* delay *of processor i in $\mathcal{A}$, denoted $t_{\mathcal{A}}(i)$, is defined as the time at which the datum is first available at processor i in $\mathcal{A}$. The running time of $\mathcal{A}$, denoted $t_{\mathcal{A}}$, is*

$$t_{\mathcal{A}} = \max_{1 \leq i \leq P} \{t_{\mathcal{A}}(i)\}$$

The complexity of single-item broadcast in the LogP model is defined as

$$B(P; L, o, g) = \min_{\mathcal{A}} t_{\mathcal{A}}$$

$\mathcal{A}$ is optimal for single-item broadcast if $t_{\mathcal{A}} = B(P; L, o, g)$.

Definition 2.2 *Given $t \geq 0$, the number of time steps available, let*

$$P(t; L, o, g) = \max\{i : B(i; L, o, g) \leq t\}$$

denote the maximum number of processors that can be reached by a broadcast algorithm in t steps.

When there is no danger of confusion, we will omit explicit mention of L, o and g and write $B(P)$ instead of $B(P; L, o, g)$ and $P(t)$ instead of $P(t; L, o, g)$. Note that for any broadcast algorithm $\mathcal{A}$, $t_{\mathcal{A}}(1) = 0$;

Our algorithm for single-item broadcast generalizes that of [5] for the postal model. It is based on the simple and intuitive idea that all informed processors should send the datum to uninformed processors as early and as frequently as possible.

Since no processor need receive more than one message in an optimal algorithm, the communication pattern of interest is a tree. A broadcast algorithm $\mathcal{A}$ induces a *broadcast tree* of processors, $T_{\mathcal{A}}$. $T_{\mathcal{A}}$ is a rooted, ordered tree with a node for each processor that participates in the broadcast. The root of $T_{\mathcal{A}}$ is processor 1 (the source of the broadcast) and if processor p sends messages to processors $p_0, \dots, p_k$ (in that order) in $\mathcal{A}$, then in $T_{\mathcal{A}}$ node p has $p_0, \dots, p_k$ as its (ordered) children. If, in addition, we label nodes by the delay of the corresponding processors, the model implies that a parent's label is $L + 2o$ smaller than its oldest child's, while the labels of successive siblings differ by at least g.

Definition 2.3 *The* universal optimal broadcast tree, *denoted $\mathcal{B}$, is defined to be the infinite labeled ordered tree in which the root has label 0 and a node with label t has children labeled $t + ig + L + 2o$, $i \geq 0$.*

Definition 2.4 *Let $\mathcal{B}(P)$ be the rooted subtree of $\mathcal{B}$ consisting of the P nodes with smallest labels (ties being broken arbitrarily.)*

Theorem 2.1 *$\mathcal{B}(P)$ is optimal for single-item broadcast.*

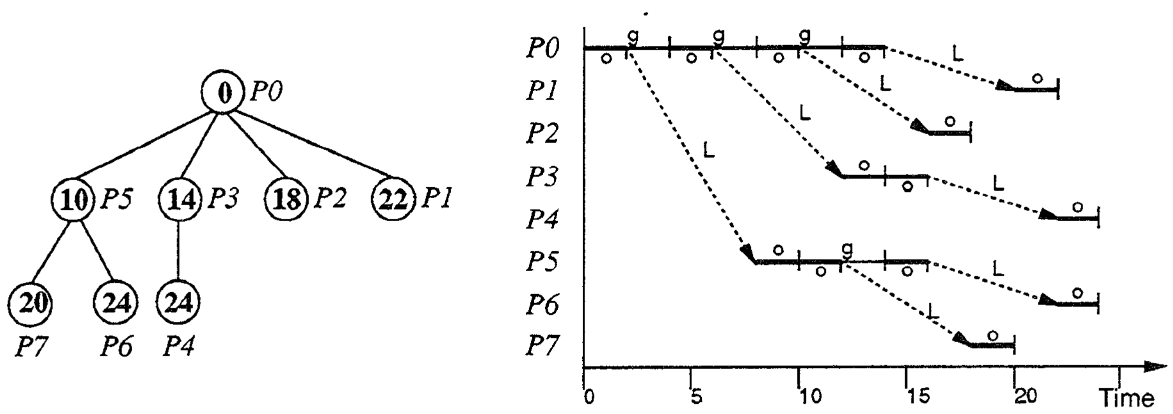


Figure 1: *Optimal broadcast tree for $P = 8, L = 6, g = 4, o = 2$ (left) and the activity of each processor over time (right). The number shown for each node is the time at which it has received the data and can begin sending it on.*

As an illustration, the optimal broadcast tree for 8 processors for $L = 6, g = 4$, and $o = 2$, is given in Figure 1.

The universal tree can also be used to determine the maximum number of processors that can be reached by a single-item broadcast in t steps. This quantity, $P(t)$, can be be computed using a generalized Fibonacci recurrence.

Definition 2.5 *Let $L > 0$ be a fixed integer. Define the sequence $\{f_i\}$ by:*

1. $f_i = 1$ *for* $0 \leq i < L$.
2. $f_i = f_{i-1} + f_{i-L}$ *otherwise.*

Fact 2.1 *For each t, $1 + \Sigma_{i=0}^{i=t} f_i = f_{t+L}$.*

The following can be easily proved. (See [5] for a proof and for bounds on f_t.)

Theorem 2.2 *For $t \geq 0$ and $L > 0$, $P(t; L, 0, 1) = f_t$*

3 The k-item Broadcast Problem

In this section, we consider the *k-item broadcast problem* in which $k > 1$ data items initially residing at a source processor are to be broadcast to all of the other $P - 1$ processors in minimum time. We will study this problem in the postal model. The algorithm of Bar-Noy and Kipnis [6] for k-item broadcast is sub-optimal except for the case $L = 1$ [4]. Its running time of $2B(P) + k + O(L)$ is considerably larger than the lower bound derived below.

Our lower bound will be given in terms of the $\{f_i\}$ sequence defined in Section 2. Let n be the index such that $f_n < P - 1 \leq f_{n+1}$, so that by Theorem 2.2 $B(P - 1) = n + 1$. Let $k^* = \lfloor \sum_{i=0}^{n} f_i/(P - 1) \rfloor$.

Theorem 3.1 *If $g = 1$ and $o = 0$ any algorithm for broadcasting k items from a single source requires at least $B(P - 1) + L + (k - 1) - k^*$ steps.*

Proof: Since $P - 1$ messages must be received per item and at most $\min(f_k, P - 1)$ messages can be received at time step $L + k$, the total number of items that can be broadcast in time $L + t$ is no more than $\lfloor \sum_{j=0}^{t} \min(f_j, P - 1)/(P - 1) \rfloor = k^* + t - n$.□

Theorem 3.2 *If an algorithm broadcasts $k \geq k^*$ items in time $t + L$ with $t = B(P - 1) + (k - 1) - k^*$, then it must have the source send distinct items in the first $(k - k^*)$ steps.*

Proof: If not, there are at least $k^* + 1$ items that have to be broadcast by the source in the $n + 1$ time steps $k - k^* = t - n, \ldots, t$. But a message sent by the source at time $t - i$ can propagate to at most f_i processors by time $t + L$.□

Thus, an algorithm that meets the lower bound must consist of two phases: a *continuous phase* during which the source sends only one copy of the first $k - k^*$ items (each of which is broadcast optimally among $P - 1$ processors by the recipient of the source's copy) and an *endgame* lasting $B(P - 1)$ steps during which the last k^* items are broadcast by having the source send multiple copies of each.

3.1 Continuous Broadcast

We define a variation of the the multiple-item broadcast problem called *continuous broadcast.* Consider a situation in which items are continuously generated at a source processor at intervals of g and each item is to be communicated to each of the other $P - 1$ processors. Given any algorithm for this problem, define the *delay of item i* to be the difference between the time it is generated and the earliest time when it has been received by every processor. Our goal is to minimize the maximum delay of an item. The continuous broadcast problem is of interest in itself, and its analysis sheds light on the continuous phase of the k-item broadcast problem.

In this scenario, it is clear that the source cannot afford to send multiple copies of a message and hence a lower bound on the delay of an item is $L + B(P - 1)$. This lower bound can only be achieved if for each i, processor P_i, the recipient of item i from the source, initiates an optimal $(P - 1)$-way broadcast at time $L + i$. Such a solution to the continuous broadcast problem implies a solution to the k-item broadcast problem in time $L + B(P - 1) + k - 1$. Since it can be shown that $k^* \leq L$, this comes within L steps of the lower bound for k-item broadcast.

We turn now to the problem of constructing a continuous broadcast schedule with delay $L + B(P - 1)$. The difficulty of the problem lies in ensuring that the staggered broadcasts do not interfere with one another by requiring some processor to send or receive multiple items in a time step. As a simple example of a pitfall to be avoided, suppose the root of the optimal $(P - 1)$-way broadcast tree has r children. (In this section, we shall only consider values of $P - 1$ for which the tree is unique, i.e. those for which $P - 1 = P(t)$ for some time t.) Then, for any i, the processors $P_{i+1}, \ldots, P_{i+r-1}$ must each be distinct from P_i since P_i is required to send item i for r consecutive steps starting at $L + i$ and hence cannot be used as a sender in the broadcast of the next $r - 1$ items.

More generally, let T_{P-1} denote the tree for optimal single-item broadcast among $(P-1)$ processors and let each node of the tree be labeled with the delay of the corresponding processor as in Section 2. A node with delay d in tree i denotes a reception of value i at time $i+d$ and will be said to be a node with *time* $d+i$. Given a copy of T_{P-1} for each data item i, our task is to assign processors to nodes of the trees satisfying :

Correctness: No processor is assigned to two nodes of the same tree.

Non-interference in receives: No processor is assigned to two nodes with the same time.

Non-interference in sends: If a processor is assigned to an internal node with $r \geq 1$ children and time t, it is not assigned to any internal node with time $t+1, \ldots, t+(r-1)$.

It is obvious that any processor assignment that satisfies these criteria will achieve minimum possible delay but it is not clear *a priori* that interference can be avoided altogether. Indeed, we will show later that for $L=2$, the lower bound of $L+B(P-1)$ on an item's delay cannot be achieved.

3.2 Block-cyclic Processor Assignments

For $L \geq 3$, the constraints imposed by the non-interference requirements are not as stringent as for $L=2$ and we can continue to use optimal broadcast trees for each item. Moreover, our processor assignment schemes, which we call *block-cyclic* are highly-structured and have compact descriptions.

We shall use the case $L=3$, $P-1=9$ as a running example to explain our construction. The optimal broadcast tree T_9 for this case is shown in Figure 2. From the tree we can infer that, since there are three leaves with delay 7, three processors will receive the first data item (i.e., the item which is generated by the source at time 0, and whose broadcast tree is initiated at time L) at time $L+7$. Similarly, by examining the broadcast tree for the second data item, we see that two processors will receive it at time $L+7$, corresponding to the two nodes with delay 6 in the optimal broadcast tree. Letting S_i denote the multiset of items received at time $L+i$, and denoting the first data item by a, the second, by b etc., we can infer that $S_7 = \{a,a,a,b,b,c,d,e,h\}$. We can elaborate this notation by indicating, for each item received at time $L+7$, whether it corresponds to an internal node of the item's broadcast tree; such an item is denoted by an upper-case letter, subscripted by the number of children of the corresponding internal node, while a leaf continues to be represented by a lower-case letter. Thus, E_2 represents a copy of item e which must be transmitted to two other processors; it corresponds to the node of delay 3 in the broadcast tree for e, the fifth item. In this expanded representation $S_7 = \{a,a,a,b,b,c,D_1,E_2,H_5\}$ Moreover, the multiset S_{7+i} is isomorphic to S_7 except that the identity of the items is translated by i. For example, $S_8 = \{b,b,b,c,c,d,E_1,F_2,I_5\}$.

To better exhibit the similarities among these multisets we adopt the *relative addressing* convention that, at every step t, a denotes the item whose broadcast terminates at time t, b, the item whose broadcast terminates at $t+1$ etc.. In this notation, an item i can be an H_5 (at time $L+i$), an E_2 (at $L+i+3$), a D_1 (at $L+i+4$), a c (at $L+i+5$), etc. With this convention, the multiset of items received at any time $i \geq L+7$ is $\{a,a,a,b,b,c,D_1,E_2,H_5\}$. The relative addressing convention is used throughout the following discussion of continuous broadcast.

We now specify the reception schedules for the upper case letters. Certain constraints arise because a processor cannot send two items in the same time step. For example, a processor receiving H_5 in a time step will be busy sending that item to other processors for that time step and the four subsequent ones; thus, it cannot receive another item corresponding to an upper case letter until 5 steps later. To ensure that this condition is met we assign a block of 5 processors to receive H_5's in a cyclic fashion, a block of 2 processors to receive E_2's and a single processor to receive the D_1s. This leaves one *receive-only processor* which will receive the same lower-case letter at every time step. By construction, this schedule has no interference between sends.

Let us now consider the reception of lower case letters. Within a block of n processors, each processor's reception schedule will be periodic with period n, and the schedule of the j^{th} processor in the block will be the same as that of the first processor, but offset from it by $j-1$ time steps. For the block of H_5 recipients the period will be 5, and the receptions between successive occurrences of H_5 will correspond to a word of length 4 over the alphabet $\{a,b,c\}$. The choice of words is, however, restricted by two considerations. The first is that since our schedules are cyclic within blocks, the multiset of items that any processor receives in 5 consecutive steps is the same as that received by the entire block of processors in a single step, and thus must be contained in the multiset $\{a,a,a,b,b,c,D_1,E_2,H_5\}$ of items received at a single time step. Thus, for example, the word $cccc$ is disallowed.

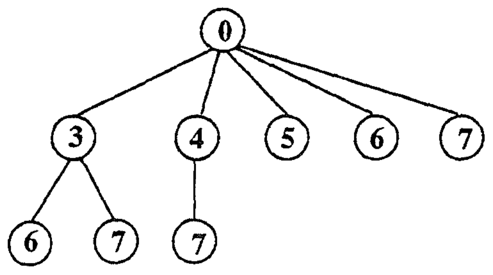
The second restriction derives from the correctness requirement, which states that no processor may receive the same item twice. By our translation rules, an H at time i is the same as a c at $(i+5)$, a b at $(i+6)$ or an a at $(i+7)$, so each of these receptions is disallowed. Since the patterns are cyclic, this disallows receptions of b at $i+1$ or a at $i+2$, ruling out any word that starts with b or has a in the second position. If we were to choose c as the first letter, b would be disallowed in the second position as well, since a b at $i+2$ is the same as a c at $i+1$.

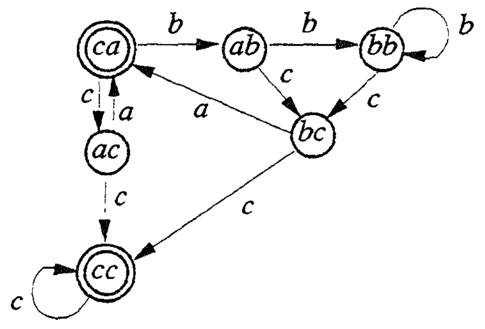
We observe that the constraints on the word that may occur in a block headed by H_5 are identical to those that would arise if we were choosing a 5-letter word and had chosen c as the first letter. Moreover, the initial constraints turn out to be independent of block size. This allows us to describe the set of legal words compactly by the automaton shown in Figure 2. It can be shown that the following recipe gives precisely those words of length $r-1$ (corresponding to a block of size r) that satisfy the second restriction.

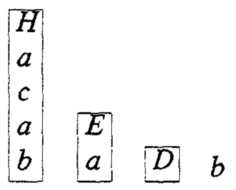
- Start at one of the start states of the automaton (marked with double circles in the figure).
- Follow a directed path with r edges that ends in the same state. This yields a word of length $r+2$, including the two letters of the start state.
- Delete the first letter and the last two letters of this word to obtain a word of length $r-1$.

For the H_5 block, this procedure yields the set $\{cccc, acab, abca, abbb\}$. Of these four words, the first restriction excludes $cccc$ and $abbb$, so the word for the H_5 block must be either H_5acab or H_5abca.

To complete our example, we can choose the word $acab$ for the H_5 block, a for the E_2 block, the empty word for the D_1 block and the lower-case letter b for the receive-only processor. This gives a correct schedule, since the multiset $\{a,a,a,b,b,c,D_1,E_2,H_5\}$ of items received at a time step is the union of the multisets


Optimal Broadcast Tree L=3, P = 9

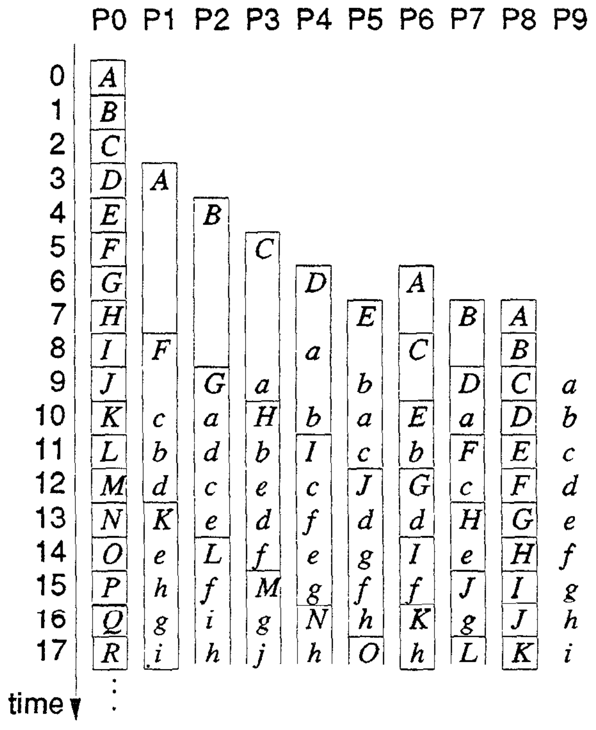

Automaton L = 3


Receiving Pattern

legal receive patterns:

ca((ca)(bb*ca)*)**

*cc**


Continuous Broadcast Schedule

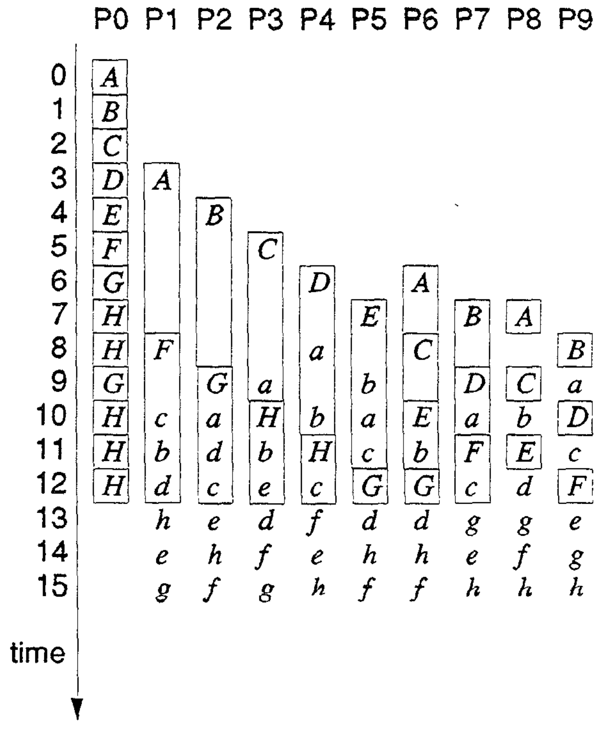

Broadcast Schedule for 8 Values

Figure 2: *Example of k-broadcast and continuous broadcast* $P = 10, L = 3, g = 1, o = 0, k = 8$.

$\{H_5, a, c, a, b\}$, $\{E_2, a\}$, $\{D_1\}$ and $\{b\}$. The resulting continuous broadcast schedule is shown in Figure 2, where we revert to the "absolute addressing"notation in which, for example, a in time step t means the first data item and *not* the item whose broadcast terminates at time t.

In general, a block-cyclic assignment stipulates a block of r processors for each internal node of T_{P-1} that has r children. These blocks use up all processors but one, which is designated the receive-only processor. The scheme is completely specified by assigning words of length $r-1$ to blocks of size r, the words being chosen from an alphabet of L lower-case letters corresponding to the L distinct delays on the leaves of T_{P-1}, and assigning an arbitrary word of length 1 to the receive-only processor. The assigned words must exactly use up all the letters occurring in the multiset of items received at a general time step. In addition, each word must respect the correctness criterion as illustrated by the second restriction in the preceding example. In our solution we choose words of the following forms, all of which are easily verified to satisfy correctness.

Lemma 3.1 *Let $a, b, c, \ldots$ denote the L letters available for constructing words, with the letter a at time i denoting the item whose broadcast terminates at time i; b, the item whose broadcast terminates at time $i+1$; etc.. Then the following words are legal:*

- $a^{L-2}(ca)^*b^*$
- $b^{L-3}c^*$, $c^{L-4}d^*$, $d^{L-5}e^*, \ldots$

3.3 Minimum Delay Algorithm

We now have all the machinery required to sketch our algorithm. For any fixed value of L, we consider problem instances parametrized by t, the maximum delay of a single-item broadcast. The t^{th} problem instance $I(t)$ consists of a multiset of block sizes and letters determined by the broadcast tree T_{P-1}, where $P-1 = P(t)$. It turns out that $I(t)$ is the disjoint union of $I(t-1)$ and $I(t-L)$, with the exception that in taking the union, the largest block size of $I(t-1)$ is augmented by one. This decomposition suggests the following inductive algorithm.

Suppose that we have solved problem instances $I(t-L)$ through $I(t-1)$ in such a way that the receive-only processor receives a b in each of the solutions and that the largest block of $I(t-1)$ is assigned a word of the form $a^{L-2}(ca)^*b^*$. To construct a solution for $I(t)$, we simply string together the solutions for $I(t-1)$ and $I(t-L)$. Of the two b's from the receive-only processors, we append one to the end of the largest word in $I(t-1)$ and use the other for the receive-only processor in $I(t)$, so that the induction can be continued. Thus, for each value of L, we need only solve L consecutive " base cases" in order to guarantee solutions for all larger values of t. Using a computer program, we have been able to obtain such solutions for all $L \leq 10$. The algorithm for solving the base cases is outlined below.

We use up one letter at a time, starting with the highest (a being the lowest) and stopping when all except a, b, c have been used up. In using up a letter, we only use the (unique) word from Lemma 3.1 that contains it and we use it on the smallest unused block. Finally, we use a combination of $b^{L-3}c^*$ and $a^{L-2}(ca)^*b^*$ to consume the remaining letters.

Theorem 3.3 *For $3 \leq L \leq 10$ there exists $t(L)$ such that whenever $P-1 = P(t)$ for some $t \geq t(L)$, continuous broadcast with a delay of $L + B(P-1)$ can be achieved.*

Corollary 3.1 *For $3 \leq L \leq 10$ there exists $t(L)$ such that, for all k and all P such that $P-1 = P(t)$ for some $t \geq t(L)$, the k-item broadcast problem can be solved in time $L + B(P-1) + k - 1$.*

We remark that although the values $t(L)$ in the preceding theorem are small, block-cyclic schedules cannot always achieve minimum delay. For example, when $L = 4$ and $t = 8$ no block-cyclic schedule can achieve a delay of $L + t$ for $P - 1 = P(t)$.

Recall that optimal algorithms for the k-item broadcast problem have an endgame in addition to a continuous phase. Our optimal solutions for continuous broadcast might not be amenable to optimal endgame solutions, resulting in sub-optimal solutions to k-item broadcast. This is illustrated in Figure 2 which shows the complete broadcast schedule for $k = 8$, $L = 3$, $P - 1 = 9$ (using absolute addressing). From time step 10 onwards, every non-source processor ($P1 - P9$) receives an item at every time step. For an optimal solution, every processor must have received $k^* = 2$ items by time step 9. This can be achieved only by having the last two processors alternately receive the item that corresponds to D_1 under relative addressing. If only one processor were to receive it all the time, then after time step 9 that processor would have received 3 items, while the "receive-only" processor would have received only 1 item, forcing the complete schedule to take one step more than the lower bound of Theorem 3.1.

As mentioned earlier, the lower bound of Theorem 3.1 is not tight for the case $L = 2$.

Theorem 3.4 *For $L = 2$, there are infinitely many values of P for which the delay of an item in continuous broadcast is at least $L + B(P-1) + 1$.*

Proof: (Sketch) A delay of $L + B(P-1)$ is achievable only by using an optimal broadcast tree for each item. Using the relative addressing notation, the only two lower-case letters when $L = 2$ are a and b, and the optimal broadcast tree determines the number of copies of a and of b to be received at each time step. The correctness requirement implies the following: if a processor is sending an item X at a given time step, then it may receive an a at that time step only if, at the step just prior to receiving X, it did not send any item. On the other hand, at any given time step, at most one processor is not sending an item. When $t \geq 7$ these requirements lead to a contradiction, as it is not possible to receive the required number of a's at each time step.□

Corollary 3.2 *For $L = 2$, there exist values of P and k such that the time required for k-item broadcast exceeds the lower bound of Theorem 3.1.*

Proof: (Sketch) Decompose any algorithm into a continuous phase and an endgame as indicated by Theorem 3.2 and apply Theorem 3.4 to the continuous phase.□

However, if a single extra time step is allowed, we can find (non-optimal) broadcast trees for which non-interfering schedules can be constructed.

Theorem 3.5 *For continuous broadcast with $L = 2$, a delay of $L + B(P-1) + 1$ can be achieved whenever $P - 1 = P(t)$ for some t.*

Proof: (Sketch) Prune the optimal tree for $P(t+1)$ processors by removing both leaves from a fraction f of the nodes with 3 children, both leaves from all nodes which have 4 or more children, and the leaf with larger delay from a fraction g of the nodes with a single child as well as from nodes which have 2 children. For suitably chosen f and g, the resulting trees result in block sizes and letters that yield block-cyclic solutions.□

3.4 Broadcasting a Finite Number of Items

In this section we return to the problem of broadcasting k items from the source processor to all other processors. Our goal is to minimize the total time required for the entire broadcast, whereas in Section 3.1 we tried to minimize the delay of each individual item. Theorem 3.1 gives a lower bound of $B(P-1)+L+(k-1)-k^*$ steps, where $k^* \leq L$. Corollary 3.1 shows that, for $3 \leq L \leq 10$ and $P-1 = P(t)$ for a sufficiently large t, the time to broadcast k items is bounded above by $B(P-1)+L+(k-1)$. The main result of this section is the following.

Theorem 3.6 *For all k, L and P, the k-item broadcast problem can be solved in time $B(P-1)+2L+k-2$.*

Define a *single-sending* schedule as one where the source processor transmits each item exactly once. Clearly, the execution time of any single-sending schedule is at least $B(P-1)+L+k-1$, since some item must leave the source processor at time $k-1$ or later, experience a delay of L while being transmitted to its first destination, and then experience a further delay of at least $B(P-1)$ before reaching all processors. Theorem 3.6 follows directly from the following theorem.

Theorem 3.7 *For all k, L and P, there is a single-sending schedule for the k-item broadcast problem with execution time $B(P-1)+2L+k-2$.*

Let the items to be broadcast be denoted $1,2,\cdots,k$. The processing of item i is divided into three parts: the initial transmission, the optimal broadcast phase, and the end game.

Initial transmission: At time $i-1$ item i is transmitted from the source processor to another processor called the *the root processor for item i*.

Optimal broadcast phase: Starting from the root processor at time $i-1+L$, item i is transmitted according to an optimal $B(P-1)-L$-step broadcast tree.

End game: For items $i<k$, the remaining processors receive i in at most $2L$ further steps; for item k, the remaining processors receive k in at most $2L-1$ further steps.

The complete proof of Theorem 3.7 will be given in the final paper. Here we content ourselves with exhibiting the promised single-sending schedule in the case where L is odd and $P-1=P(t)$ for some t.

A processor that receives item i during the optimal broadcast phase is called a *sender for item i*. If it receives item i at time $i+t-r$ (corresponding to an internal node with r children in the optimal broadcast tree for item i) then it is called an *r-sender for item i*. An r-sender for item i sends that item to $r-L$ processors during the optimal broadcast phase, in a manner dictated by the optimal $t-L$-step broadcast tree; it sends item i to L additional processors during the end game. A processor which receives item i during the end game is called a *receiver for item i*. Call item i *active* for processor p if p is a sender for item i, and *inactive* for p if p is a receiver for item i. If a processor is an r-sender for item i then it is also an r-sender for all items $j \equiv i \pmod r$; it is a receiver for all other items.

An r-block consists of r processors which are, respectively, r-senders for items congruent to $1,2,\cdots,r \pmod r$. Thus a block contains exactly one sender for each item. Note that a block of size r must receive $r-1$ inactive copies of i. Processor $p_{i \bmod r}$ in a block of size r can transmit r copies of i in consecutive time steps. Thus each block of size r sends $\min(L,r)$ inactive transmissions of i. The receive-only processor is placed in a block by itself.

The reception of the inactive items during the endgame requires that certain blocks of size $r \leq L$ send items to other blocks, while a processor in a block of size $r \geq L$ will spend the endgame sending the item to processors within its own block.

In order to better understand the reception pattern of item i, we define a digraph called a *block transmission digraph* which represents the transmission pattern of item i between the blocks. For each block of size r, there is a vertex labeled r. The block containing the receive-only processor has a vertex labeled 0. A thick edge from vertex r_1 to r_2 denotes an active transmission of item i between the corresponding blocks. A normal edge with weight w denotes w inactive transmissions of item i between the corresponding blocks. Active transmissions have an implicit edge weight of 1. Also, the largest block receives an active transmission from the source processor. Thus, the weights of all edges into a vertex with label $r>0$ must sum to r as must those of all edges out of it. For $r=0$, i.e. the block containing just the receive-only processor, there is only one edge into r and it has weight one; there are no edges out of r. Each item has the same block transmission digraph. Figure 3 shows a block transmission digraph for $L=3$ and $P-1=P(11)=41$.

Let the processors in an r-block be designated $p_1, p_2, \cdots, p_r$, where, for $i=1,2,\cdots,r$, p_i is an r-sender for item i, and the other $r-1$ processors are receivers for item i. Then p_i receives item i during the optimal broadcast phase for that item. We now describe the schedule according to which of the remaining $r-1$ processors in the block receive item i. There are several cases.

1. If $r \geq 2L$ then, during the first L time steps after $t+i-1$, L processors in the block receive i from processor $p_{i \bmod r}$ in the block. Another $r-2L$ processors receive i from processors in $r-2L$ blocks of size 1. The remaining $L-1$ processors receive i from processor $p_{i \bmod (L-1)}$ in a block of size $L-1$(which we denote by $p_{i \bmod (L-1), L-1}$). The items are received by the processors in the block as follows :
   (a) the L transmissions of item 1 from processor p_1 in the block is received by processor p_{r-i+1} in the block at time step $t+i$ for $1 \leq i \leq L$.
   (b) the $r-2L$ processors which receive item 1 from blocks of 1 are $p_{L+1}, p_{L+2}, \cdots p_{r-L}$ and they receive 1 at time step $t+L$.
   (c) the $L-1$ processors that receive item 1 from processor $p_{1,L-1}$ are $p_2, \cdots p_L$ where p_i receives 1 at time step $t+2L-i+2$.
   (d) if processor p_i received item j at time $\bar{t}$ then processor $p_{(i+1) \bmod r}$ receives item $j+1$ at time $\bar{t}+1$.

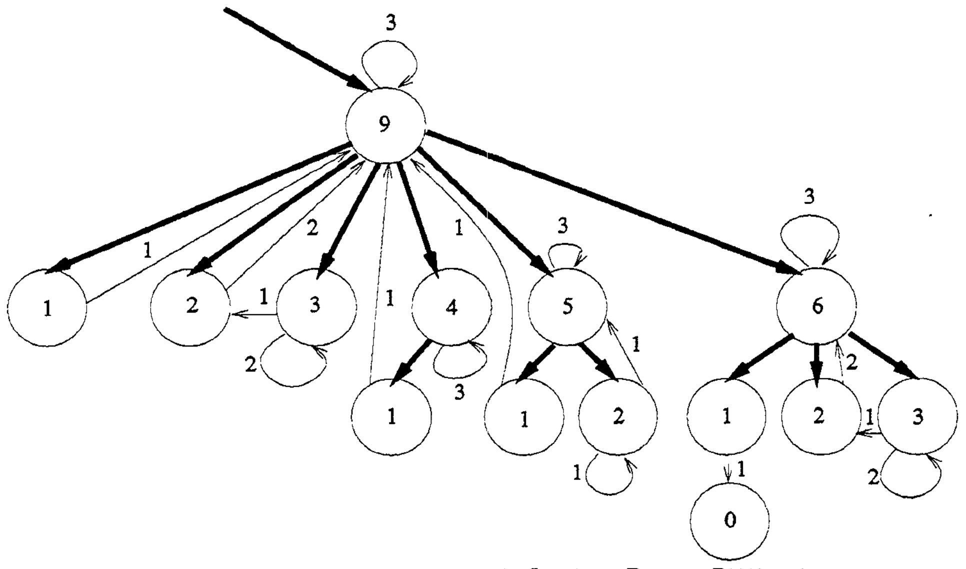


Figure 3: *The block transmission digraph for* $L = 3$ *and* $P - 1 = P(11) = 41$.

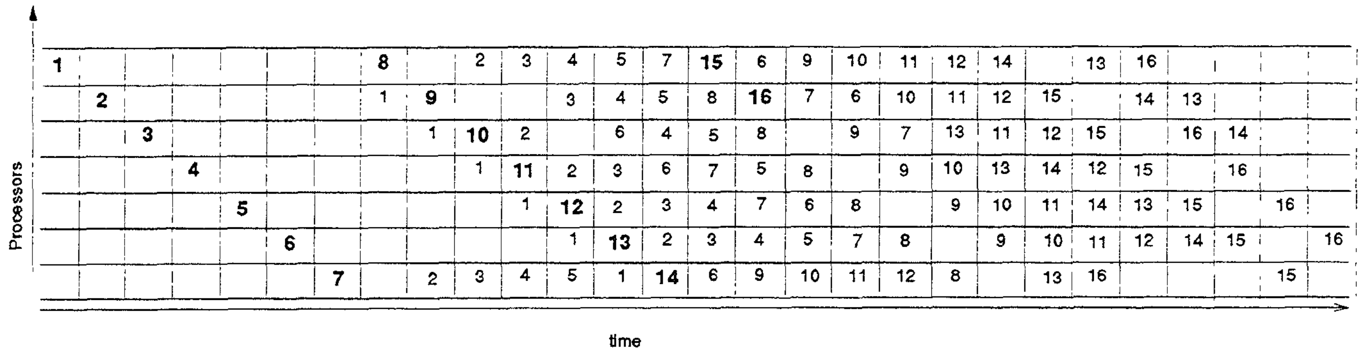
As we can see, this scheme insures that all processors in the block receive all k items with only an extra delay of L time steps. We see that only one processor, call it p, is receiving an item in time step $B(P-1) + 2L + k - 1$ and that the item is k. Since $p_{k \bmod r}$ in the block is available to send at time $B(P-1) + L + k - 2$ and p is available to receive at $B(P-1) + 2L + k - 2$, thus p will receive item k one step earlier. Thus the extra delay is at most $L - 1$.

2. If $L + 1 < r < 2L$ then again L processors receive item i from processor $p_{i \bmod r}$ in the block. The remaining $r - L - 1$ receive the item from processor $p_{i \bmod (r-L)}$ in a block of size $r - L$(which we denote as $p_{i \bmod (r-L), r-L}$). For the receiving scheme, we have processor $p_{i \bmod r}$ in the block sending i at time steps $t + i$ to $t + i + L - 1$. Processor $p_{i \bmod (r-L), r-L}$ sends i at time steps $t+i+L$ to $t+i+r-2$. The items are received as follows :
   (a) Consider item i. If $i \bmod r \equiv 1$ then processor p_j receives i at time $t + i + j - 2$ for $2 \leq j \leq r$.
   (b) If $1 < i \bmod r \leq r - 2$ then i is received by the processors in the block from time step $t+i$ to $t+i+r-2$ in the following order : $p_r, p_1, p_2, p_3, \cdots p_{(i-1) \bmod r}, p_{(i+1) \bmod r}, \cdots p_{r-1}$.
   (c) If $i \bmod r \equiv r - 1$ then consider r.
      i. If r is odd then the first 4 processors to receive i are the following (listed in order of reception) : $p_{\lfloor \frac{r}{2} \rfloor}, p_{\lceil \frac{r}{2} \rceil}, p_r, p_1$. The order of the rest of the receptions is as follows :
         A. p_j receives i at time $t + i + 5 + 2(j-2)$ if $j < \lfloor \frac{r}{2} \rfloor$
         B. p_j receives i at time $t+i+4+2(j - \lceil \frac{r}{2} \rceil - 1)$ if $j > \lceil \frac{r}{2} \rceil$ and $j \neq r - 1$
      ii. If r is even then the first 3 processors to receive i are the following (listed in order of reception) : $p_{\frac{r}{2}}, p_1, p_r$. The order of the rest of the receptions is as follows :
         A. p_j receives i at time $t + i + 4 + 2(j-2)$ if $j < \frac{r}{2}$
         B. p_j receives i at time $t + i + 3 + 2(j - \frac{r}{2} - 1)$ if $j > \frac{r}{2}$ and $j \neq r - 1$
   (d) If $i \bmod r \equiv 0$ then consider r.
      i. If r is odd then
         A. p_1 receives i at time $t + i$
         B. $p_{\lfloor \frac{r}{2} \rfloor}$ receives i at time $t + i + L$
         C. $p_{\lceil \frac{r}{2} \rceil}$ receives i one time step after it receives $i - 1$
         D. p_{r-1} receives i at time $t + i + L - 1$
         E. the rest of the processors p_j receive i one time step before they receive $i - 1$
      ii. If r is even then
         A. p_1 receives i at time $t + i + 2$
         B. $p_{\frac{r}{2}}$ receives i one time step after it received $i - 1$
         C. p_{r-1} receives i at time $t + i + L$
         D. p_j receives i the time step before it receives $i - 1$ if $\frac{r}{2} < j < r - 1$
         E. p_j receives i the time step after it receives $i - 1$ if $1 < j < \frac{r}{2}$

   From the construction, we see that no processor has two items which arrive in the same time step. Since we are delaying sending by at most $r - L - 1$ steps thus we are adding at most $L - 1$ steps beyond the lower bound given. Figure 4 shows the reception table for a block of size 7 with $L = 5$ and $k = 16$.

3. If $L \leq r \leq L + 1$ then either $L - 1$ or L processors receiving i will receive it from processor $p_{i \bmod r}$ in the block. The receiving scheme is the same as the one for $L + 1 < r < 2L$. Obviously we are not delaying any sending. Thus blocks of size L and $L + 1$ receive all the items by time $B(P-1) + L + k - 1$.

4. If $2 < r < L$ then the $r - 1$ processors receiving i receive it from processor $p_{i \bmod (r-1)}$ in a block of size $r - 1$. If $r > 3$ then we delay sending until item 1 can first be received in the


Processors
time

1							**8**		2	3	4	5	7	**15**	6	9	10	11	12	14		13	16				
	2						1	**9**			3	4	5	8	**16**	7	6	10	11	12	15		14	13			
		3						1	**10**	2		6	4	5	8		9	7	13	11	12	15		16	14		
			4						1	**11**	2	3	6	7	5	8		9	10	13	14	12	15		16		
				5						1	**12**	2	3	4	7	6	8		9	10	11	14	13	15		16	
					6						1	**13**	2	3	4	5	7	8		9	10	11	12	14	15		16
						7		2	3	4	5	1	**14**	6	9	10	11	12	8		13	16				15	

Figure 4: *The reception table for $L = 5$, $r = 7$ and $k = 16$. The items received as active items are denotes by bold numbers.*

same time step as some active item i is being received by a processor in the block where $i \bmod r \equiv 1$. We then use the same scheme as the one for $L+1 < r < 2L$. Obviously, we are delaying sending by at most $r - 1 < L$. For $r = 3$ we have a slightly different reception scheme: we delay sending item 1 until it can be first received at the same time as some active item i is received by a processor in the block where $i \bmod r \equiv 2$. We have p_3 receiving item 1 before p_2. For the rest of the items j, j is received by processor p_m at time $\bar{t}$ iff $j - 1$ was received by processor p_{m-1} at time $\bar{t} - 1$. Again, we are delaying sending by at most $r - 1 = 2$ steps.

5. If $r = 2$, we see that we have used up all but one of the blocks of size 1, but the blocks of size $L + 2 \leq r < 2L$ and $r = L$ all use blocks which are one size larger than needed. Thus one of the transmissions of i is not needed per each block. So, these transmissions are instead sent to processors in each block of 2 which needs to receive it. Since each of these "extra" transmissions come from blocks whose transmissions are first being received at time $t + L + 1$ (except for the case $L = 3$ where blocks of size 3 have their transmissions first being received in time step $t + 2$) thus we have all the transmissions to the blocks of 2 arriving at time $t + L$ (or for $L = 3$ and $r = 3$, at time $t + 1$). Since we are assuming L is odd, the reception of these items will not interfere with the receptions of the active items for these blocks.
6. If $r = 1$ then there is only one processor in the block and it would have received all k items as active items.
7. The *receive-only* processor receives all its items from the processor in the remaining free block of 1.

3.5 Optimal Broadcasting of k Items

We have given an algorithm for broadcasting k items that needs only $L-1$ time steps more than the lower bound of $B(P-1)+L+k-1$. In this section we show that this lower bound is achievable if we modify the model slightly by assuming that, at any time step,

- each processor p has a buffer containing all items sent to p at least L time steps earlier which have not yet been received by p, and
- each processor p can examine the items in its buffer and determine which one to receive during the time step.

More than one item may enter a processor's buffer at a given time step, but only one item can be received at each time step. [1]

Theorem 3.8 *For all k, L and P, there is a single-sending schedule for k-item broadcast which is optimal on the modified model.*

In the optimal broadcast scheme under the modified assumptions, item i is transmitted from the source processor at time $i-1$, reaching its root processor at time $i - 1 + L$. It is then transmitted to the other processors according to an optimal t-step broadcast tree T_t, except that the reception of the item at some leaves of the broadcast tree will be delayed, as described below. The one-to-one assignment of non-source processors to the nodes of T_t is the same as the assignment used in the algorithm given in the previous section. Thus, if a processor is an r-sender for item i, then it corresponds to a node of out-degree r in T_t; if a processor is a receiver for item i then it corresponds to a leaf of T_t.

We now describe the behavior of a processor at a typical time step. Just prior to each time step, processor p's buffer will consist entirely of inactive items (for p). At the beginning of the time step, at most one active item and one inactive item will arrive at p's buffer. If an active item i has arrived, then p will receive it and commence sending it to other processors. If p is assigned to a node u of T_t then p sends the item to those processors that are assigned to the children of T_t. An inactive item that reaches p at the same time as some active item is said to be *delayed*. It will remain in the buffer until it is received at some later time step in which no active item arrives. We omit the further details of the processor assignment and schedule of receptions and transmissions. It can be shown that the entire process will be completed by time $L + B(P-1) + k - 1$.

Figure 5 gives an optimal schedule where $L = 3$, $P - 1 = 13$ and $k = 14$. The processors are grouped into blocks exactly as in the previous section. In the figure, a table entry of i associated with processor p_j at time t indicates that, at time t, p_j receives the i^{th} item. If item i is circled then i is an active item that causes an inactive item to be delayed. If an item i is enclosed in a square then i is an inactive item that was delayed by some active item.

4 Other Broadcast Problems

In this section, we present two other generalizations of the broadcast problem and give optimal algorithms for these problems.

[1]There is a scheme which achieves the lower bound and needs a buffer size of only 2.

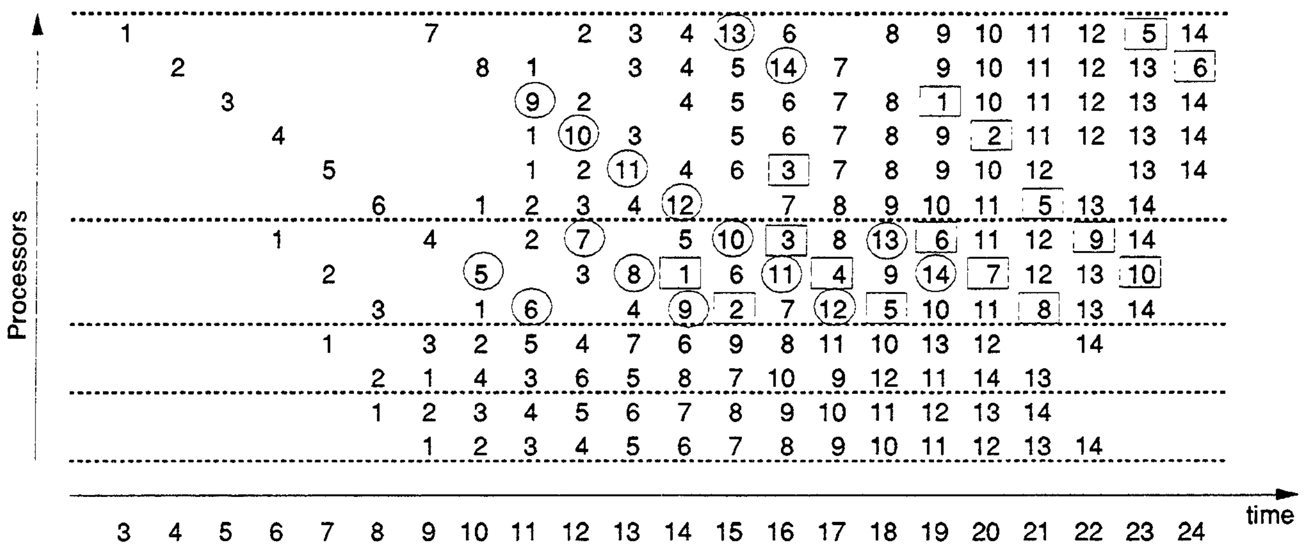


Figure 5: $L = 3, P - 1 = 13, k = 14$.

4.1 All-to-All Broadcast

In the P-way all-to-all broadcast problem, each of P processors has a data item that is to be made available to every processor. Of course, one could solve this problem using P one-to-all broadcasts but a more efficient (and simpler) solution is possible.

Observe that since each processor must receive $P - 1$ items and the first one cannot be received until time $L + 2o$, a lower bound on the time for P-way all-to-all broadcast is $L + 2o + (P - 2)g$. On the other hand, if processor i sends out its data item to processors $i + 1 \pmod P), \ldots, i + (P - 1) \pmod P$ (in that order) at time $0, g, \ldots, (P - 2)g$, each processor would receive messages at time $L + 2o, L + 2o + g, \ldots, L + 2o + (P - 2)g$. We thus have a simple optimal algorithm for all-to-all broadcast. The same communication schedule is optimal for the problem of *all-to-all personalized communication* in which each processor has a distinct item to send to every other processor.

The lower bound argument and our algorithm extend easily to the situation where each of the P processors has k items to broadcast: since a processor must receive $k(P - 1)$ items and the first one cannot be received until $L + 2o$, we have a lower bound of $L + 2o + g(k(P - 1) - 1)$. This bound is matched by k repetitions of the previous algorithm.

We remark that the order of transmission for processor i does not have to be the one indicated above. Any collection of permutations of the set $S = \{1, \ldots, P\}$, one for each processor, such that no processor is the target of two messages at the same time, will yield an optimal algorithm. (For k-item broadcast, analogous permutations of the multiset consisting of k copies of S will lead to optimality.)

4.2 The Combining-Broadcast Problem

In parallel computation, one frequently encounters situations where each processor holds a value and the P values are to be combined into one (or *reduced*) using some operation (such as max or +). Reduction can be viewed as "all-to-one" broadcast (with a slight change in model parameters) and is thus solved optimally by simply reversing the directions of messages in optimal broadcast.

If the reduced value is to be made available to all processors, we get a problem which we may think of as an *all-to-all broadcast with combining*. Clearly, this problem can be solved by a reduction followed by broadcast, which is optimal to within a factor of 2. However, we show below that all-to-all broadcast with combining takes no longer than all-to-one reduction.

Let x_i be a value initially available at processor i, $i = 0, \ldots, P - 1$. The problem is to make $x_0 + \ldots + x_{P-1}$ available to each processor in the shortest possible time. The '+' operation is assumed to be commutative and associative. In the sequel, all arithmetic on processor indices will be modulo P and for indices i, j, $x[i : j]$ will denote the quantity $x_j + x_{j-1} + \ldots + x_i$. Thus, for example, if $P = 5, x[1 : 3] = x_3 + x_2 + x_1$ while $x[3 : 1] = x_1 + x_0 + x_4 + x_3$.

For simplicity, we shall work with the postal model and assume that the combining operation takes zero time. With these assumptions, it is clear that if a message is sent to processor i at time t, it can be received at time $t + L$, combined with x_i and the result transmitted to another processor at time $t + L$ (arriving at its destination at time $t + 2L$.)

Our algorithm will be described in terms of the sequence $\{f_i\}$ defined in Section 2. Let T, the amount of time for the algorithm, be fixed, and let $P = p(T; L, 0, 1)$. Our algorithm has the following simple description: at time $j = 0, 1, \ldots, T - L$, processor i, $i = 0, \ldots, P - 1$ sends its current value to processor $i + f_{j+L-1}$. (As indicated earlier, the values sent at time j arrive at their destinations at time $j + L$ and are instantaneously combined into the current value at the destination processor before transmission at time $j + L$.)

Theorem 4.1 *The algorithm presented above leaves the value $x[0 : P - 1]$ at each of the $P = p(T; L, 0, 1)$ processors at time T.*

Proof: We inductively prove that at time j processor i has the value $x[i - f_j + 1 : i]$. Observe that this corresponds to a maximal set of processors whose values can be combined by time j. □

5 Optimal Summation

In this section, we consider the problem of computing the sum of n operands in the LogP model. The input to the problem is a set of operands $X_1, \ldots, X_n$ and we assume that an algorithm can choose how these are initially distributed among the P processors. We assume that the addition operation is commutative [2] and that each addition operation takes unit time. Our proof of optimality is based on the following inversion of the problem: instead of finding an algorithm to add n operands in minimum time, find an algorithm that adds the maximum number of operands in t units of time.

5.1 Summation Trees

Without loss of generality, a summing algorithm $\mathcal{A}$ can be viewed as a rooted binary tree $T_{\mathcal{A}}$ with a leaf for each operand and an internal node for each addition operation. Each operation is carried out on one of the P processors and hence, operands are of two types: those available in the processor's local memory and intermediate results received from other processors. A summation algorithm can be represented by the computation schedules for the various processors, together with a tree giving the pattern of communication among them.

An example is shown in Figure 6. The initial work for each processor is represented by a linear chain of input-summing nodes. Unless the processor is a leaf of the communication tree, it then repeatedly receives a value, adds it to its partial sum and performs a chain of $g - o - 1$ input-summing nodes. Observe that local computations overlap the delivery of incoming messages and the processor reception overhead begins as soon as the message arrives.

We will establish that the communication pattern induced by optimal summation is the time-reversal of an optimal single-item broadcast tree: a processor that receives the item at time d in optimal broadcast sends a (single) message at time $t - d$. Suppose that some processor sends messages $M_1, \ldots, M_k$ at time $T_1, \ldots, T_k$. Identifying messages with the corresponding nodes in the binary tree, it is clear that M_i and M_j represent sums of disjoint sets of input operands. Thus, the processor could add M_i to some leaf of M_k at time T_i without changing the total number of leaves in the tree. This gives an algorithm in which each processor (except one) sends exactly one message. For uniformity of discussion, we will assume that the processor responsible for the final addition (at time t) also sends a message at time t.

Suppose that in a summation algorithm a processor initiates message receptions at times $R_1 < \ldots < R_k$ and a message transmission at time S. Then $R_j = S - (o+1) - (k-j)g$ for otherwise one of the receptions could be delayed allowing its sender to add more input operands into its partial sum. We will call such an algorithm *lazy*. If we reverse the direction and timing of all messages in a lazy algorithm, i.e. a message sent at time S from processor i to j is replaced by one that is received by i from j at time $t - S$, we get a broadcast algorithm in which sending processors "wait" for one time step before transmissions. (This corresponds in $\mathcal{A}$ to the steps that receiving processors spend just after receptions in adding the received partial sums.) Thus, lazy summation algorithms for a machine with parameters L, o, g, P are in one-to-one correspondence with broadcast algorithms for a machine with parameters $L+1, o, g, P$.

Lemma 5.1 *Let $\mathcal{A}$ be a lazy summation algorithm that adds n operands in t steps and in which the processors initiate their message transmissions at times $S_1 \le S_2 \le \ldots \le S_P = t$. Then* $n = \Sigma_{i=1}^{P} S_i - op + 1$

[2] Our optimal algorithm for commutative summation can be used for non-commutative summation with an appropriate renumbering of the operands.

Proof: Let k_i be the number of messages received by processor i. Then, the number of input operands that are directly summed by processor i is exactly $S_i - (o+1)k_i + 1$. Summing over the processors yields the result. □

We see that the broadcast pattern corresponding to optimal lazy summation must be that which minimizes $\Sigma(t - S_i)$, which is precisely an optimal broadcast pattern.

6 Conclusion

In this paper, we have considered different broadcast problems including single-item, k-item, all-to-all, and combining broadcast. We also considered the problem of summation, which, as we have shown, can be considered as a "reverse" broadcast. We have presented optimal algorithms for single-item, all-to-all and combining broadcasting as well as for summation.

For the k-item broadcast problem, we considered the case where the root was *single-sending*, and presented an algorithm which required only $L - 1$ additional steps above the lower bound for single-sending and hence, at most $2L - 1$ steps above the general lower bound. In fact, by relaxing certain restrictions on the $LogP$ model, we were able to present an algorithm which achieved the single-sending lower bound. We introduced the *continuous broadcast* problem to further examine whether the lower bound is achievable. We have shown that for $3 \le L \le 10$ and $P - 1 = P(t)$ for large enough t, we can broadcast each item in optimal time using a *block-cyclic* processor assignment scheme. We conjecture that the same is true for all $L > 2$. We also showed that optimal continuous broadcast used for k items yields a schedule that is within L steps of optimal.

Acknowledgements

We thank Amotz Bar-Noy, Shlomo Kipnis and Ramesh Subramonian for valuable discussions.

Computational support was provided by the NSF Infrastructure Grant number CDA-8722788. Richard Karp and Abhijit Sahay are supported by NSF/DARPA Grant CCR-9005448. Eunice Santos is supported by a DOD-NDSEG Graduate Fellowship. Klaus Erik Schauser is supported by an IBM Graduate Fellowship.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir. On Communication Latency in PRAM Computation. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 11–21. ACM, June 1989.

[2] A. Aggarwal, A. K. Chandra, and M. Snir. Communication Complexity of PRAMs. *Theoretical Computer Science*, pages 3–28, March 1990.

[3] N. Alon, A. Barak, and U. Manber. On disseminating infomation reliably without broadcasting. In *Proceedings of the International Conference on Distributed Computing Systems*, 1987.

[4] A. Bar-Noy and S. Kipnis. Broadcasting Multiple Messages in Simultaneous Send/Receive Systems. Technical Report NO. RC 18352, IBM Research Division, September 1992. Also to appear in *Discrete Applied Mathematics*.

[5] A. Bar-Noy and S. Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 11–22, June 1992.

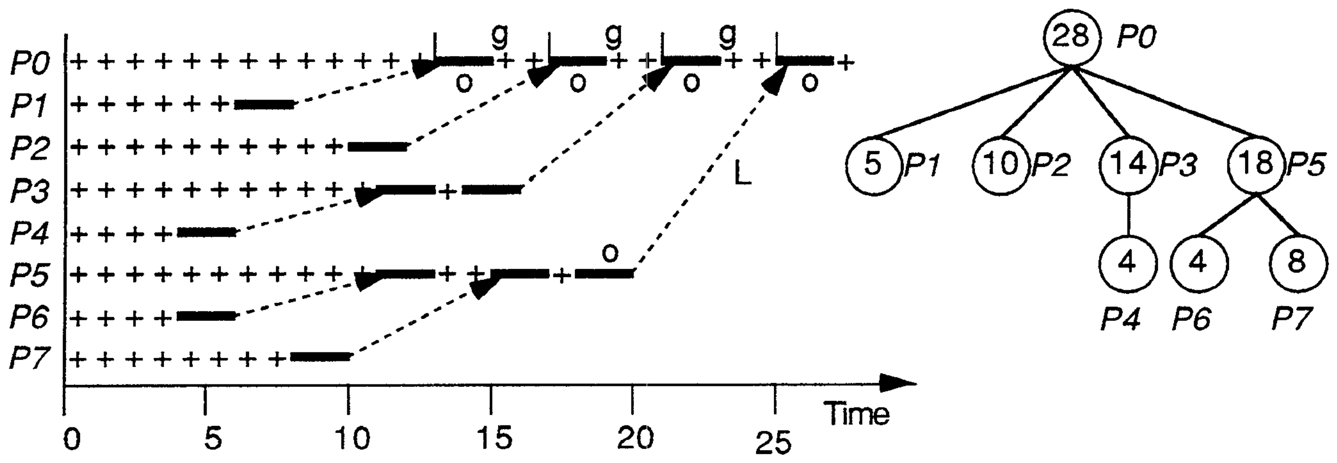


Figure 6: *Computation schedule (left) and communication tree (right) for optimal summation with* $t = 28, P = 8, L = 5, g = 4, o = 2$.

[6] A. Bar-Noy and S. Kipnis. Multiple Message Broadcasting in the Postal Model. In *Proceedings of the Seventh International Parallel Processing Symposium*, April 1993.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[8] E. Cockayne and A. Thomason. Optimal multi-message broadcasting in complete graphs. In *Proceedings of the 11th SE Conference on Combinatorics, Graph Theory, and Computing*, pages 181–199, 1980.

[9] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993. Also appears as TR No. UCB/CS/92 713.

[10] A. M. Farley. Broadcast time in communication networks. *SIAM Journal on Applied Mathematics*, 39(2):385–390, October 1980.

[11] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, pages 114–118, 1978.

[12] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18(4):319–349, 1988.

[13] C-T. Ho. Optimal Comunication Primitives and Graph Embeddings on Hypercubes. Ph.D. Thesis, Yale University 1990.

[14] C-T. Ho and S.L. Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648. IEEE Computer Society, 1986.

[15] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM Simulation on a Distributed Memory Machine. In *Proceedings of the Twenty-Fourth Annual ACM Symposium of the Theory of Computing*, pages 318–326. ACM, ACM, May 1992.

[16] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21:339–374, 1984.

[17] C. H. Papadimitriou and M. Yannakakis. Towards an Architecture-Independent Analysis of Parallel Algorithms. In *Proceedings of the Twentieth Annual ACM Symposium of the Theory of Computing*, pages 510–513. ACM, 1988.

[18] L. G. Valiant. A Bridging Model for Parallel Computation. *Communications of the Association for Computing Machinery*, 33(8):103–11, August 1990.