Math. Systems Theory 27, 431–452 (1994)

Mathematical Systems Theory

Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems

A. Bar-Noy and S. Kipnis*

Computer Sciences Department, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA

Abstract. In many distributed-memory parallel computers and high-speed communication networks, the exact structure of the underlying communication network may be ignored. These systems assume that the network creates a complete communication graph between the processors, in which passing messages is associated with communication latencies. In this paper we explore the impact of communication latencies on the design of broadcasting algorithms for fully connected message-passing systems. For this purpose, we introduce the *postal model* that incorporates a communication latency parameter $\lambda \geq 1$. This parameter measures the inverse of the ratio between the time it takes an originator of a message to send the message and the time that passes until the recipient of the message receives it. We present an optimal algorithm for broadcasting one message in systems with n processors and communication latency λ, the running time of which is $\Theta((\lambda \log n)/\log(\lambda + 1))$. For broadcasting $m \geq 1$ messages, we first examine several generalizations of the algorithm for broadcasting one message and then analyze a family of broadcasting algorithms based on degree-d trees. All the algorithms described in this paper are practical event-driven algorithms that preserve the order of messages.

1. Introduction

In distributed-memory parallel computers and in data communication networks, processors communicate by sending and receiving messages. This is accomplished by having processors submit messages to and retrieve messages from an underlying communication network, which in turn is responsible for delivering messages from

*Current address: IBM ISRAEL Science and Technology, MATAM Advanced Technology Center, Haifa, Israel 31905. Kipnis@haifasc3.vnet.ibm.com.

their sources to their destinations. As the speed and connectivity of modern communication networks increase, numerous message-passing systems ignore the exact structure of the network and assume that it creates a complete communication graph between the processors. Such systems treat sending a message as a *send-and-forget* event and assume that passing a message between any pair of processors takes roughly the same time. This assumption is incorporated into several distributed-memory parallel computers, such as MIT J-machine [9], TMC CM-5 [12], and IBM Vulcan system [16]. A similar approach was investigated for high-speed communication networks [5], [15] and is incorporated into networks such as PARIS [4] and plaNET [7], [14].

Similar message-passing situations exist in real life. Consider, for example, communication between people in a metropolitan area where the only way for people to contact each other is by sending and receiving letters. The letters, in turn, are picked up regularly by the postal service and are delivered to their destinations after some delay. The letter-communication medium in this environment introduces two important features: first, it transforms a collection of isolated people into a fully connected system, and, second, it creates communication channels of roughly the same delay (since it takes roughly the same time for any letter to arrive at its destination, no matter who its sender and receiver are). The letter-communication medium differs from a telephone-communication medium in that unlike telephone calls, where one party cannot call another before finishing the first conversation, with letters one party can send many letters to many parties before any particular letter arrives.

Traditional models for message-passing systems assume a telephone-like communication medium between processors. In fully connected message-passing systems the *telephone model* assumes that communicating a data item between two processors requires the simultaneous attention of the two processors. This model reflects the situation in systems that use circuit-switching techniques to establish communication between processors. However, numerous parallel and distributed systems use packet-switching techniques at the communication network. The telephone model is not suitable for such packet-switching-based systems since it does not address the issue of communication latencies that arise from the *send-and-forget* nature of passing messages in these systems.

In this paper we introduce the *postal model* which characterizes the situation in message-passing systems that use packet-switching techniques more accurately than the telephone model. (Recently, another model, the *LogP model* [8], was introduced that bears some similarities to our postal model.) The postal model addresses the issue of communication latencies by incorporating a latency parameter, $\lambda \geq 1$ that measures the inverse of the ratio between the time it takes an originator of a message to send the message and the time that passes until the recipient of the message receives it. (For $\lambda = 1$, the postal model reduces to the telephone model in fully connected systems.) More specifically, we assume that an originator p of a message M spends 1 unit of time sending it to its destination. After processor p sends message M, it is free to perform other functions including sending other messages. Message M later arrives at its destination q, and the recipient q then spends 1 unit of time receiving it. The communication latency, λ, is the total time that passes from

the time the originator p starts sending message M until the time the recipient q finishes receiving message M.

We use the framework of the postal model to explore the impact of communication latencies on the design of broadcasting algorithms for fully connected message-passing systems. The topic of global dissemination of information over networks of processors has received much attention over the past few decades (see [3], [10], and [11]). Many variants of problems such as broadcasting, gossiping, combining, and permuting were investigated for both parallel computers and for distributed systems. In particular, an approach similar to the postal model is presented in [6] in the context of combining data in high-speed communication networks.

The remainder of this paper is organized as follows. In Section 2 we describe the postal model for communication in message-passing systems. Section 3 presents an optimal algorithm for broadcasting one message in systems with n processors and communication latency λ. This algorithm is based on a generalized Fibonacci tree and requires $\Theta((\lambda \log n)/\log(\lambda + 1))$ units of time. In Section 4 we address the problem of broadcasting multiple messages in the postal model. We first examine several generalizations of the algorithm for broadcasting one message and then analyze a family of algorithms based on degree-d trees. All the algorithms described are practical event-driven algorithms that preserve the order of messages. Section 5 discusses open problems and directions for further research. We include an appendix that analyzes the behavior of the generalized Fibonacci function and its index function, on which many of our broadcasting algorithms are based.

2. The Postal Model

This section describes the postal model which focuses on three aspects of communicating in message-passing systems: full connectivity, simultaneous I/O, and communication latencies.

Definition 1. A message-passing system with n processors—$\mathcal{MPS}(n)$—consists of a set of processors $\{p_0, p_1, \ldots, p_{n-1}\}$ with the following two properties:

- *Full connectivity*: each processor p in the system can send a point-to-point message to any other processor q in the system.
- *Simultaneous I/O*: each processor p can simultaneously send one message to processor q and receive another message from processor r.

Definition 1 focuses on the processors' perspective of a message-passing system. In such systems processors communicate by sending and receiving messages through a communication network, which creates an abstraction of a fully connected system. In many systems processors can simultaneously send and receive messages using separate input and output ports.

In the postal model we use the term *message* to refer to an atomic piece of data (e.g., a packet) communicated between processors. A message cannot be broken into smaller pieces at the sending processor, at the communication network, or at the

receiving processor. The size of an atomic message is defined as one unit of size, and the time it takes to send or receive an atomic message is defined as one unit of time. Sending large data in the postal model is accomplished by breaking down the data into several atomic messages, any one of which is sent and received separately. This is a standard practice in many packet-switching-based systems.

To model the *send-and-forget* nature of message communication, we separate the event of sending a message from the event of receiving it. We assume that both the sending and the receiving of a message, each takes one unit of time. However, to account for communication latencies, we assume that the receiving event occurs later than the sending event.

Definition 2. A message-passing system with n processors and communication latency λ—$\mathcal{MPS}(n, \lambda)$—is a message-passing system $\mathcal{MPS}(n)$ with the additional property:

- *Communication latency*: if at time t processor p sends a message M to processor q, then processor p is busy sending message M during the time interval $[t, t + 1]$, and processor q is busy receiving message M during the time interval $[t + \lambda - 1, t + \lambda]$.

Definition 2 is used for modeling communication latencies in message-passing systems. The process of door-to-door message delivery in such systems contains various overhead costs, such as message preparation time, output-buffer copying time, output-port submission delays, network propagation delays, input-port absorption delays, input-buffer copying time, and message interpretation time. The communication latency λ is meant to capture both the software and the hardware overhead costs. Formally, the value of the communication latency, λ, measures the ratio between

(i) the amount of time that passes from the time a sender p of a message M starts sending it to the time the receiver q fully receives it, and
(ii) the time that the sender p itself spends sending message M.

We assume that the amount of time that a sender spends preparing and sending a message is equal to the time that a receiver spends receiving and handling the message. The communication latency λ can be thought of as the amount of time required to make a processor q knowledgeable of an atomic message M that is initially held by another processor p, as measured in units of the time it takes a processor to send or receive an atomic message. Although, in reality, the exact value of λ may depend on the particular sender–receiver pair and on the load on the network, in many systems under normal conditions of operation, the value of λ is expected to be fairly uniform across the system and not to fluctuate too much.

3. Broadcasting One Message

This section investigates the problem of broadcasting one message in $\mathcal{MPS}(n, \lambda)$. We describe an optimal-time algorithm for this problem that is based on a

generalized Fibonacci tree. A similar approach for the combining problem is presented in [6].

The problem of broadcasting one message in $\mathcal{MPS}(n, \lambda)$ is defined as follows. At time $t = 0$, processor p_0 has a message M to broadcast to processors $p_1, p_2, \ldots, p_{n-1}$. The goal is to minimize the time at which the last processor receives message M. The running time of an algorithm $\mathcal{A}$ in $\mathcal{MPS}(n, \lambda)$ is denoted by $T_{\mathcal{A}}(n, \lambda)$.

An optimal strategy for broadcasting one message in the postal model is to have each processor, upon receiving message M, send M to a new processor every unit of time. This strategy results in a broadcast tree in which nodes close to the root have higher degrees than nodes farther away. To minimize the height of such a tree, its structure and the degree of the nodes in it should depend on the value of the communication latency λ. For example, Figure 1 illustrates such an optimal-time broadcast tree for $\mathcal{MPS}(14, 2\frac{1}{2})$ that completes the broadcasting process in $t = 7\frac{1}{2}$ units of time. We show that such optimal broadcast trees are based on *generalized Fibonacci numbers*, and we refer to these trees as *generalized Fibonacci trees*. For the cases of $\lambda = 1$ and $\lambda = 2$, for example, these trees are the known binomial and Fibonacci trees, respectively.

Before describing the broadcasting algorithm, we provide some notations and tools. Let G be a right-continuous, nondecreasing, unbounded step function from

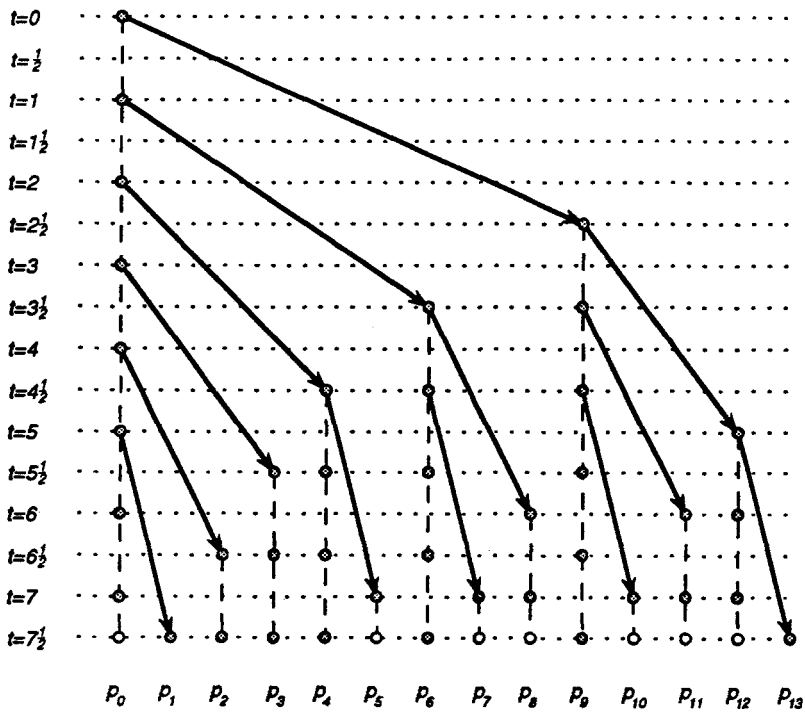

Fig. 1. A generalized Fibonacci broadcast tree for a message-passing system with $n = 14$ processors and communication latency $\lambda = 2\frac{1}{2}$. The height of the tree is $t = 7\frac{1}{2}$ units of time.

the nonnegative real numbers into the positive integers, $G: \Re^+ \to N$. We define an inverse function I_G from the positive integers into the nonnegative real numbers, $I_G: N \to \Re^+$, as follows: $I_G(n) = \min\{t: G(t) \geq n\}$. We call I_G the *index function* of G. The function I_G is well defined for all positive integers $n \geq 1$, since G is right-continuous and unbounded. (For example, consider $G(t) = \lfloor t \rfloor$ and $I_G(n) = n$.) The following two claims provide some properties of $G(t)$ and $I_G(n)$. The claims' proofs are straightforward.

Claim 1. *Let G be a right-continuous, nondecreasing, unbounded step function from the nonnegative real numbers into the positive integers, $G: \Re^+ \to N$.*

(1) *The index function $I_G: N \to \Re^+$ is a nondecreasing, unbounded function.*
(2) *For any nonnegative real number $t \geq 0$, we have $I_G(G(t)) \leq t$.*
(3) *For any positive integer $n \geq 1$, we have $G(I_G(n)) \geq n$.*
(4) *For any $\varepsilon > 0$, if $I_G(n) - \varepsilon \geq 0$, then $G(I_G(n) - \varepsilon) < n$.*

Claim 2. *Let G and H be two right-continuous, nondecreasing, unbounded step functions, $G, H: \Re^+ \to N$. If $G(t) \leq H(t)$ for all $t \geq 0$, then $I_G(n) \geq I_H(n)$ for all integers $n \geq 1$.*

The algorithm for broadcasting one message uses a sequence of numbers that depends on the value of the communication latency λ. For any $\lambda \geq 1$, this sequence is a generalization of the Fibonacci numbers and can be derived from a *generalized Fibonacci function*:

$$F_\lambda(t) = \begin{cases} 1 & \text{if} \quad 0 \leq t < \lambda, \\ F_\lambda(t-1) + F_\lambda(t-\lambda) & \text{if} \quad t \geq \lambda. \end{cases}$$

For any $\lambda \geq 1$, the function $F_\lambda(t)$ is a right-continuous, nondecreasing, unbounded step-function from the nonnegative real numbers into the positive integers. (For $\lambda = 1$, for example, $F_1(t) = 2^{\lfloor t \rfloor}$, and, for $\lambda = 2$, the value of $F_2(t)$ is the Fibonacci number whose index is $\lfloor t \rfloor + 1$.) For any $\lambda \geq 1$, we denote by f_λ the index function of the generalized Fibonacci function F_λ, that is, $f_\lambda = I_{F_\lambda}$. (For $\lambda = 1$, for example, we have $f_1(n) = \lceil \log n \rceil$, and, for $\lambda = 2$, the value of $f_2(n)$ is the index of the last Fibonacci number that is smaller than n.)

We now describe Algorithm BCAST for broadcasting a message M from processor p_0 to the range of processors p_0 through p_{n-1} in $\mathcal{MPS}(n, \lambda)$.

Algorithm BCAST

(a) Processor p_0, at time $t = 0$, checks whether $n = 1$ and if so it finishes its participation in the algorithm. Otherwise, if $n \geq 2$, processor p_0 computes $j = F_\lambda(f_\lambda(n) - 1)$ and sends to processor p_j the message M together with the request that processor p_j broadcast message M to the range of processors p_j through p_{n-1}. Then, at time $t = 1$, processor p_0 recursively applies Algorithm BCAST to the range of processors p_0 through p_{j-1}.
(b) A generic processor p_i, upon receiving message M and the range of processors p_i through p_l, starts applying Algorithm BCAST to broadcast to this range.

(For applying Algorithm BCAST, such a processor p_i treats itself as if it were processor p_0, and it treats the range of processors p_i through p_l as if it were the range of processors p_0 through p_{l-i}.)

For example, consider using Algorithm BCAST in $\mathcal{MPS}(14, 2\frac{1}{2})$. Figure 1 illustrates the corresponding sequence of sending and receiving message M in such a system. At time $t = 0$, processor p_0 computes $j = F_{5/2}(f_{5/2}(14) - 1) = 9$ and sends message M to processor p_9. Then, at time $t = 1$, processor p_0 recursively broadcasts to the range of processors p_0 through p_8, that is, it broadcasts in $\mathcal{MPS}(9, 2\frac{1}{2})$. Processor p_9, on the other hand, receives message M at time $t = 2\frac{1}{2}$ and broadcasts M to the range of processors p_9 through p_{13}, that is, it broadcasts in $\mathcal{MPS}(5, 2\frac{1}{2})$. The algorithm continues recursively in a similar manner.

We now turn to prove the correctness, the performance, and the optimality of Algorithm BCAST in the following series of three lemmas.

Lemma 3 (Correctness). *Algorithm* BCAST *broadcasts message M from processor p_0 to all the processors p_0 through p_{n-1}.*

Proof. By induction on n. For $n = 1$, the lemma is trivially true. Assume the lemma holds for all n' such that $1 \leq n' < n$ and prove it for n. Consider the value of j in item (a) of Algorithm BCAST. From part (4) of Claim 1, we have that $j < n$. Also, since $f_\lambda(n) \geq \lambda \geq 1$ for $n \geq 2$, it follows that $f_\lambda(n) - 1 \geq 0$ and therefore that $j \geq 1$. Consequently, we have $1 \leq j \leq n - 1$ and therefore processor p_0 sends message M to a valid processor p_j. Subsequently, processor p_0 broadcasts M to $j < n$ processors and processor p_j broadcasts M to $n - j < n$ other processors. The lemma now follows from the induction hypothesis. □

Lemma 4 (Performance). *The running time of Algorithm* BCAST *satisfies $T_B(n, \lambda) \leq f_\lambda(n)$.*

Proof. By induction on n. For $n = 1$, we have $T_B(1, \lambda) = 0 = f_\lambda(1)$. Assume that the lemma holds for all n' such that $1 \leq n' < n$ and prove it for n. The running time of the algorithm is the largest of two values. One value, $1 + T_B(j, \lambda)$, is the sum of the time spent by p_0 sending M to p_j and the time it takes p_0 to broadcast M to processors p_0 through p_{j-1} recursively. The other value, $\lambda + T_B(n - j, \lambda)$, is the sum of the time that passes until p_j receives M and the time it takes p_j to broadcast M to processors p_j through p_{n-1}. We then have

$$T_B(n, \lambda) \leq \max\{1 + T_B(j, \lambda), \lambda + T_B(n - j, \lambda)\}.$$

Now since $1 \leq j < n$ (see the proof of Lemma 3), it follows from the induction hypothesis that

$$T_B(n, \lambda) \leq \max\{1 + f_\lambda(j), \lambda + f_\lambda(n - j)\}.$$

We now examine the first argument of the "max" function. The definition of j in item (a) of the algorithm and part (2) of Claim 1 imply that

$$1 + f_\lambda(j) = 1 + f_\lambda(F_\lambda(f_\lambda(n) - 1)) \leq 1 + (f_\lambda(n) - 1) = f_\lambda(n).$$

We next examine the second argument of the "max" function. Part (3) of Claim 1, the definition of F_λ, and the definition of j in item (a) of the algorithm imply that

$$n \leq F_\lambda(f_\lambda(n)) = F_\lambda(f_\lambda(n) - 1) + F_\lambda(f_\lambda(n) - \lambda) = j + F_\lambda(f_\lambda(n) - \lambda),$$

which implies that $n - j \leq F_\lambda(f_\lambda(n) - \lambda)$. Parts (1) and (2) of Claim 1 imply that

$$\lambda + f_\lambda(n - j) \leq \lambda + f_\lambda(F_\lambda(f_\lambda(n) - \lambda)) \leq \lambda + (f_\lambda(n) - \lambda) = f_\lambda(n).$$

The lemma follows since both arguments of the "max" function are bounded by $f_\lambda(n)$. □

Lemma 5 (Optimality). *Let $\mathcal{A}$ be an algorithm for broadcasting one message in $\mathcal{MPS}(n, \lambda)$. The running time of algorithm $\mathcal{A}$ satisfies $T_{\mathcal{A}}(n, \lambda) \geq f_\lambda(n)$.*

Proof. Let $N(t)$, for $t \geq 0$, be the maximal number of processors that can be broadcast to in t units of time by any algorithm for broadcasting one message in $\mathcal{MPS}(n, \lambda)$. From the definition of the postal model, we have $N(t) = 1$ for $0 \leq t < \lambda$. Without loss of generality, in any algorithm that can broadcast to $N(t)$ processors in t units of time, processor p_0 sends message M at time $t_0 = 0$ to some processor p_i, which receives message M at time $t_1 = \lambda$. Processor p_0 then broadcasts M to x other processors, while processor p_i broadcasts M to y other processors, where $x + y + 2 = N(t)$. Since processor p_0 broadcasts M to x other processors in $t - 1$ units of time, we have $x + 1 \leq N(t - 1)$. Similarly, since processor p_i broadcasts M to y other processors in $t - \lambda$ units of time, we have $y + 1 \leq N(t - \lambda)$. Consequently, we have $N(t) \leq N(t - 1) + N(t - \lambda)$ for all $t \geq \lambda$.

We next show that $N(t) \leq F_\lambda(t)$, for any $t \geq 0$. Assume to the contrary, that $t \geq 0$ exists such that $N(t) > F_\lambda(t)$ and let t_0 be the minimal such t. (This minimal t is well defined, since both $N(t)$ and $F_\lambda(t)$ are right-continuous step functions.) Since $N(t) = F_\lambda(t) = 1$ for $0 \leq t < \lambda$, we must have that $t_0 \geq \lambda$. From the property of $N(t)$, from the definition of t_0, and from the definition of $F_\lambda(t)$ we have $N(t_0 - 1) + N(t_0 - \lambda) \geq N(t_0) > F_\lambda(t_0) = F_\lambda(t_0 - 1) + F_\lambda(t_0 - \lambda)$. From the minimality of t_0, we have $N(t_0 - 1) \leq F_\lambda(t_0 - 1)$ and $N(t_0 - \lambda) \leq F_\lambda(t_0 - \lambda)$, which leads to a contradiction. Thus, we have $N(t) \leq F_\lambda(t)$ for all $t \geq 0$.

Finally, let $\mathcal{A}(t)$ denote the number of processors that know message M at time t in algorithm $\mathcal{A}$. The preceding discussion implies that $\mathcal{A}(t) \leq N(t) \leq F_\lambda(t)$. Claim 2 now implies that $I_{\mathcal{A}}(n) \geq f_\lambda(n)$, for all integers $n \geq 1$. However, from the definition of the index function, $I_{\mathcal{A}}$ is the running time of algorithm $\mathcal{A}$, that is, $T_{\mathcal{A}}(n, \lambda) = I_{\mathcal{A}}(n) \geq f_\lambda(n)$. □

The following theorem summarizes Lemmas 3–5.

Theorem 6. *Algorithm* BCAST *is an optimal-time algorithm for broadcasting one message in $\mathcal{MPS}(n,\lambda)$, and its running time is $T_{\mathrm{B}}(n, \lambda) = f_\lambda(n)$.*

The next theorem provides upper and lower bounds on the generalized Fibonacci function $F_\lambda(t)$ and its index function $f_\lambda(n)$. The theorem's proof is given in the appendix.

Theorem 7. *Let $n \geq 1$ be an integer, and let $\lambda \geq 1$ and $t \geq 0$ be reals. Then*

(1) $(\lceil \lambda \rceil + 1)^{\lfloor t/2\lambda \rfloor} \leq F_\lambda(t) \leq (\lceil \lambda \rceil + 1)^{\lfloor t/\lambda \rfloor}$.

(2) $\dfrac{\lambda \log n}{\log(\lceil \lambda \rceil + 1)} \leq f_\lambda(n) \leq 2\lambda + \dfrac{2\lambda \log n}{\log(\lceil \lambda \rceil + 1)}$.

(3) $F_\lambda(t) \geq (\lambda + 1)^{(t/\alpha\lambda)-1}$ *for sufficiently large* λ, *where*

$$\alpha = 1 + \frac{\ln\ln(\lambda+1) + 1}{\ln(\lambda+1) - (\ln\ln(\lambda+1) + 1)}.$$

(4) $f_\lambda(n) \leq (1 + h(\lambda)) \dfrac{\lambda \log n}{\log(\lambda + 1)}$, *for sufficiently large* λ *and* n, *where the function* $h(\lambda) \to 0$ *when* $\lambda \to \infty$ *and* $n \geq 2^\lambda$.

4. Broadcasting Multiple Messages

This section explores the problem of broadcasting $m \geq 1$ messages in $\mathcal{MPS}(n, \lambda)$. We first present lower bounds on the running time of algorithms for broadcasting $m \geq 1$ messages. We then describe several generalizations of Algorithm BCAST to handle $m \geq 1$ messages. Finally, we examine a family of algorithms based on degree-d trees.

The problem of broadcasting $m \geq 1$ messages in $\mathcal{MPS}(n, \lambda)$ is defined as follows. At time $t = 0$, processor p_0 has m messages $M_1, M_2, \ldots, M_m$ to broadcast to processors $p_1, p_2, \ldots, p_{n-1}$. The running time of an algorithm $\mathcal{A}$ in $\mathcal{MPS}(n, \lambda)$ is defined as the arrival time of the last message to the last processor in the algorithm and is denoted by $T_{\mathcal{A}}(n, m, \lambda)$.

4.1. Lower Bounds

The following lemma and corollary bound the running time of any broadcasting algorithm in $\mathcal{MPS}(n, \lambda)$ from below. The lemma follows from the fact that the last message cannot be sent from processor p_0 before time $t = m - 1$ and requires at least $f_\lambda(n)$ units of time to disseminate. The corollary follows from the lemma, part (2) of Theorem 7, and because $f_\lambda(n) \geq \lambda$ for $n \geq 2$.

Lemma 8. *Any algorithm $\mathcal{A}$ for broadcasting m messages in $\mathcal{MPS}(n, \lambda)$ satisfies*

$$T_{\mathcal{A}}(n, m, \lambda) \geq (m - 1) + f_\lambda(n).$$

Corollary 9. *Any algorithm $\mathcal{A}$ for broadcasting m messages in $\mathcal{MPS}(n, \lambda)$ satisfies:*

(1) $T_{\mathcal{A}}(n, m, \lambda) \geq m - 1 + (\lambda \log n)/\log(\lceil \lambda \rceil + 1)$.

(2) $T_{\mathcal{A}}(n, m, \lambda) \geq m - 1 + \lambda$.

4.2. *Generalizations of Algorithm* BCAST

We examine three standard generalizations of Algorithm BCAST to handle $m \geq 1$ messages. The first, Algorithm REPEAT, performs m iterations of Algorithm BCAST, one iteration per message. The second, Algorithm PACK, performs one iteration of Algorithm BCAST on a "long message" comprising the m messages packed together. The third, Algorithm PIPELINE, performs one iteration of Algorithm BCAST on a "stream" of the m messages in a pipeline.

Algorithm REPEAT

This algorithm broadcasts m messages by repeatedly applying Algorithm BCAST m times, once for each of the m messages. Processor p_0 starts the ith iteration of Algorithm BCAST to broadcast message M_i immediately after it sends the last copy of message M_{i-1}. All the other processors follow the steps of Algorithm BCAST for each of the m messages.

A simple analysis of Algorithm REPEAT implies that its running time, $T_R(n, m, \lambda)$, is at most m times the running time of Algorithm BCAST. The following lemma, however, provides an exact analysis of Algorithm REPEAT by considering the overlap between the end of the ith iteration and the beginning of the $(i+1)$st iteration of Algorithm BCAST.

Lemma 10. *The running time of Algorithm* REPEAT *satisfies*

$$\begin{aligned} T_R(n, m, \lambda) &= m \cdot (T_B(n, \lambda) - (\lambda - 1)) + (\lambda + 1) \\ &= m \cdot f_\lambda(n) - (m-1)(\lambda - 1). \end{aligned}$$

Proof. Algorithm REPEAT consists of m iterations of Algorithm BCAST. Theorem 6 implies that each such iteration takes $T_B(n, \lambda) = f_\lambda(n)$ units of time. However, during the last $\lambda - 1$ units of time of the ith iteration, processor p_0 does not send message M_i anymore. (See, for example, Figure 1.) Rather, processor p_0 starts the $(i+1)$st iteration $\lambda - 1$ units of time before the termination of the ith iteration. Now, because the communication latency is λ, the instances of message M_{i+1} that processor p_0 starts sending arrive at other processors after the ith iteration is completed and all the processors are ready for the $(i+1)$st iteration. The running time of the algorithm is, therefore, $m \cdot (T_B(n, \lambda) - (\lambda - 1))$ plus $(\lambda - 1)$ additional units of time required to finish the last iteration. □

The next corollary provides an upper bound on the running time of Algorithm REPEAT. The corollary follows from part (2) of Theorem 7.

Corollary 11. *The running time of Algorithm* REPEAT *satisfies*

$$T_R(n, m, \lambda) \leq \frac{2m\lambda \log n}{\log(\lambda + 1)} + m\lambda + m + \lambda - 1.$$

Algorithm REPEAT does not require any changes to or tuning of Algorithm BCAST. Its running time, however, grows linearly with m, which is not optimal. For

small values of m, the simplicity of Algorithm REPEAT may compensate for the loss in performance.

Algorithm PACK

This algorithm generalizes Algorithm BCAST by treating the m messages as one "long message." In this algorithm processor p_0 packs the m messages into a long message and applies Algorithm BCAST to the long message. Each of the other processors first receives all the m messages in a sequence and then starts forwarding them according to Algorithm BCAST.

Notice that this description of Algorithm PACK does not comply with the requirements of the postal model. In particular, the time it takes a processor to send the long message depends on m and is not a unit of time. To meet the definition of the postal model and thereby to minimize the running time of Algorithm PACK, we change the units of measurement and use Algorithm BCAST with a modified value of the communication latency parmeter. In Algorithm PACK a sender of a long message is busy for m units of time, and a receiver of a long message fully receives it $\lambda + m - 1$ units of time after the sender started sending it. Therefore, we normalize the time scale $t' = t/m$ and the communication latency $\lambda' = (\lambda + m - 1)/m$ to meet the requirements of the postal model. Theorem 6 now indicates that an optimal-time broadcasting algorithm in such a scenario would result by applying Algorithm BCAST with $\lambda' = (\lambda + m - 1)/m = 1 + (\lambda - 1)/m$ as the communication latency parameter. The next lemma gives the running time $T_{\mathsf{PK}}(n, m, \lambda)$ of the resultant Algorithm PACK.

Lemma 12. *The running time of Algorithm* PACK *satisfies*

$$T_{\mathsf{PK}}(n, m, \lambda) = m \cdot T_{\mathsf{B}}\left(n, 1 + \frac{\lambda - 1}{m}\right) = m \cdot f_{1+(\lambda-1)/m}(n).$$

Proof. Algorithm PACK performs one iteration of Algorithm BCAST with a normalized communication latency parameter $\lambda' = 1 + (\lambda - 1)/m$ and with normalized time scale $t' = t/m$. The lemma follows by renormalizing the time scale. □

The next corollary provides an upper bound on the running time of Algorithm PACK. The corollary follows from part (2) of Theorem 7.

Corollary 13. *The running time of Algorithm* PACK *satisfies*

$$T_{\mathsf{PK}}(n, m, \lambda) \leq \frac{2(m + \lambda - 1)\log n}{\log(2 + (\lambda - 1)/m)} + 2(m + \lambda - 1).$$

Algorithm PACK requires only minor changes to and tuning of Algorithm BCAST. For small values of m and large values of λ, the running time of this algorithm is near optimal.

Algorithm PIPELINE

This algorithm generalizes Algorithm BCAST by treating the m messages as a "stream." Like in Algorithm PACK, in Algorithm PIPELINE each processor performs one iteration of Algorithm BCAST as follows: it first sends all the m messages in a sequence to one recipient, and then it recursively broadcasts the stream of m messages to a subrange of the processors. However, unlike in Algorithm PACK, in Algorithm PIPELINE processors start forwarding messages as soon as they arrive. Processor p_0 starts this algorithm by sending the m messages in a sequence to some other processor. Each processor, upon receiving the first message of the stream, forwards it to a new processor and, upon receiving intermediate messages of the stream, forwards them to the same processor to which it sent their predecessors.

To meet the definition of the postal model and thereby to minimize the running time of Algorithm PIPELINE, we need to change the units of measurement and use Algorithm BCAST with a modified value of the communication latency parameter. In Algorithm PIPELINE each sender sequentially sends the stream's m messages to some recipient during m successive units of time. A recipient of a stream receives the first message of the stream λ units of time after the sender started sending it and receives the last message of the stream $m - 1$ units of time later. During this time, the receiver forwards to another processor each new message that it gets. For analysis purposes, we think of a sender of a stream as being busy for m consecutive units of time and a receiver of a stream as being capable of forwarding the whole stream λ units of time after the sender started sending it. The analysis distinguishes between two cases. In the first case, when $m \leq \lambda$, a processor sending the stream completes its task and can begin sending the stream elsewhere before the recipient of the stream can begin forwarding it. In the second case, when $m \geq \lambda$, a processor sending the stream completes its task and can begin sending the stream elsewhere only after the recipient of the stream can begin forwarding it. We call the algorithm PIPELINE-1 when $m \leq \lambda$, and PIPELINE-2 when $m \geq \lambda$.

We first consider Algorithm PIPELINE-1 ($m \leq \lambda$). In this algorithm a stream-sender, upon finishing sending the stream, broadcasts the stream to a larger group of processors, and the stream-recipient, upon receiving the stream, broadcasts the stream to a smaller group of processors. To meet the requirements of the postal model, we normalize the time scale $t' = t/m$ and the communication latency $\lambda' = \lambda/m$. Theorem 6 now indicates that an optimal-time broadcasting algorithm for this situation would result by applying algorithm BCAST with communication latency parameter $\lambda' = \lambda/m$. The following lemma gives the running time $T_{\mathsf{PL1}}(n, m, \lambda)$ of the resultant Algorithm PIPELINE-1.

Lemma 14. *The running time of Algorithm* PIPELINE-1 *satisfies*

$$\begin{aligned} T_{\mathsf{PL1}}(n, m, \lambda) &= m \cdot \left(T_{\mathsf{B}}\left(n, \frac{\lambda}{m}\right) - \frac{\lambda}{m} \right) + (m + \lambda - 1) \\ &= m \cdot f_{\lambda/m}(n) + (m - 1). \end{aligned}$$

Proof. Algorithm PIPELINE-1 performs one iteration of Algorithm BCAST with a normalized communication latency parameter $\lambda' = \lambda/m$ and with a normalized time scale $t' = t/m$. In Algorithm BCAST the last message is sent by some processor in normalized time $T_B(n, \lambda') - \lambda'$. In the actual Algorithm PIPELINE-1 this corresponds to the last instance of starting to send the stream of m messages in time $m \cdot (T_B(n, \lambda/m) - \lambda/m)$. This implies that the last message of the stream is received by the last processor after $(m + \lambda - 1)$ additional units of time, which proves the lemma. □

The next corollary provides an upper bound on the running time of Algorithm PIPELINE-1. The corollary follows from part (2) of Theorem 7.

Corollary 15. *The running time of Algorithm* PIPELINE-1 *satisfies*

$$T_{PL1}(n, m, \lambda) \leq 2\lambda + \frac{2\lambda \log n}{\log(1 + \lambda/m)} + (m - 1).$$

We now consider Algorithm PIPELINE-2 ($m \geq \lambda$). In this algorithm a stream-sender, upon finishing sending its stream, broadcasts the stream to a smaller group of processors, and the stream-recipient, upon receiving the stream, broadcasts the stream to a larger group of processors. Notice that since this process is done recursively, it results in changing the responsibilities of the sender and the receiver in Algorithm BCAST for each sender–receiver pair. To meet the requirements of the postal model, we normalize the time scale $t' = t/\lambda$ and the communication latency parameter $\lambda' = m/\lambda$. Theorem 6 now indicates that an optimal-time broadcasting algorithm for this situation would result by applying Algorithm BCAST with communication latency parameter $\lambda' = m/\lambda$. The following lemma gives the running time $T_{PL2}(n, m, \lambda)$ of the resultant Algorithm PIPELINE-2.

Lemma 16. *The running time of Algorithm* PIPELINE-2 *satisfies*

$$\begin{aligned} T_{PL2}(n, m, \lambda) &= \lambda \cdot \left(T_B\left(n, \frac{m}{\lambda}\right) - \frac{m}{\lambda}\right) + (m + \lambda - 1) \\ &= \lambda \cdot f_{m/\lambda}(n) + (\lambda - 1). \end{aligned}$$

Proof. Algorithm PIPELINE-2 performs one iteration of Algorithm BCAST with a normalized communication latency parameter $\lambda' = m/\lambda$ and with a normalized time scale $t' = t/\lambda$. In Algorithm BCAST the last message is sent by some processor in normalized time $T_B(n, \lambda') - \lambda'$. In the actual Algorithm PIPELINE-2 this corresponds to the last instance of starting to send the stream of m messages in time $\lambda \cdot (T_B(n, m/\lambda) - m/\lambda)$. This implies that the last message of the stream is received by the last processor after $(m + \lambda - 1)$ additional units of time, which proves the lemma. □

The next corollary provides an upper bound on the running time of Algorithm PIPELINE-2. The corollary follows from part (2) of Theorem 7.

Corollary 17. *The running time of Algorithm* PIPELINE-2 *satisfies*

$$T_{\mathsf{PL2}}(n, m, \lambda) \leq \frac{2m \log n}{\log(1 + m/\lambda)} + 2m + \lambda - 1.$$

Algorithm PIPELINE is seemingly similar to Algorithm PACK. However, the fact that Algorithm PIPELINE takes advantage of the nonatomicity of the stream of messages makes it more subtle and more efficient than Algorithm PACK. In particular, Algorithm PIPELINE-2 requires changing the roles of the sender and the receiver in Algorithm BCAST every time a stream is forwarded between a sender–receiver pair. Finally, notice that, for large values of m, none of the generalizations of Algorithm BCAST described in this section, including Algorithm PIPELINE, exhibit near-optimal performance.

4.3. Degree-d Tree Algorithms

We describe a family of algorithms for broadcasting multiple messages that use a broadcast tree of fixed degree d. We denote this family collectively by Algorithm DTREE. For a given value of $1 \leq d \leq n - 1$, Algorithm DTREE uses a *left-to-right, almost-full, degree-d tree* rooted at processor p_0. That is, processor p_0 has d children, ordered from left to right, and each of its children, taken in order from left to right, also has d children. This structure continues down the levels of the tree until n, the number of nodes in the tree, is exhausted.

Algorithm DTREE attempts to balance between two different strategies for broadcasting multiple messages. One strategy, employed for example by Algorithm REPEAT, is first to send one message to as many recipients as possible and only then to start with the other messages. The other strategy, employed for example by Algorithm PIPELINE, is first to send one copy of each message and to pipeline the stream of messages between sender–receiver pairs.

Algorithm DTREE

(a) Processor p_0 sends d copies of message M_1, one after the other, to its d children in the degree-d tree in order from left to right. After processor p_0 finishes sending message M_1, it applies the algorithm to messages M_2 through M_m.
(b) A nonroot processor, upon receiving a message M from its parent in the degree-d tree, sends the message M to its d (or fewer) children in the tree, in order from left to right.

The next lemma bounds the running time of Algorithm DTREE.

Lemma 18. *The running time of Algorithm* DTREE *satisfies*

$$T_{\mathsf{DT}}(n, m, \lambda) \leq d(m - 1) + (d - 1 + \lambda)\lceil \log_d n \rceil.$$

Proof. Assume, for simplicity, that the tree is a full d-ary tree. The last arrival of a message to a processor in the algorithm is the arrival of message M_m to the

rightmost leaf in the tree. Message M_m, which is first sent by processor p_0 to its leftmost child at time $d(m-1)$, is sent by processor p_0 to its rightmost child at time $d(m-1)+(d-1)$. Message M_m, therefore, arrives at the rightmost child of p_0 at time $d(m-1)+(d-1+\lambda)$. In general, if processor p receives message M_m at time t, then message M_m arrives at the rightmost child of p at time $t+(d-1+\lambda)$. Since the height of the tree is at most $\lceil \log_d n \rceil$, message M_m arrives at the rightmost leaf of the tree by time $d(m-1)+(d-1+\lambda)\lceil \log_d n \rceil$. This analysis also bounds the running time of the algorithm when the tree is not a full d-ary tree. □

We next discuss the performance of Algorithm DTREE for several choices of d. When $d=1$, the tree is a line and the performance of the algorithm is near optimal when λ and n are fixed and $m \to \infty$. When $d=n-1$, the tree is a star and the performance of the algorithm is near optimal when m and n are fixed and $\lambda \to \infty$. Both the line and the star algorithms are useful and easy to implement. When $d=2$, the tree is a binary tree and the performance of the algorithm is within a factor of $\max\{2, \log(\lceil \lambda \rceil + 1)\}$ of optimal time. For a fixed λ, this case provides a constant approximation. When $d=\lceil \lambda \rceil + 1$, the performance of the algorithm is within a factor of $\max\{2, \lceil \lambda \rceil + 1\}$ of optimal time. In this case, for $m \leq (\log n)/\log(\lceil \lambda \rceil + 1)$, the running time of the algorithm is within a factor of 3 of the optimal time, independent of λ. Finally, it was recently shown in [13] that the family of algorithms collectively denoted by Algorithm DTREE is within a multiplicative factor of 7 of optimal time, over the whole range of n, m, and λ, when considering order-preserving broadcast. This was shown by considering several ranges of n, m, and λ, choosing an appropriate value of d for each range, and improving the lower bound on order-preserving broadcast for a particular range of n, m, and λ.

5. Further Research

There are many problems left open by our research. Here we list a few that seem to be particularly interesting.

The bounds of Theorem 7 on $F_\lambda(t)$ and $f_\lambda(n)$ are not tight. For $F_\lambda(t)$, the upper bound is roughly the square of the lower bound, and, for $f_\lambda(n)$, there is a gap of a multiplicative factor of 2 and an additive term of 2λ between the upper and lower bounds. We believe that the lower bound on $F_\lambda(t)$ and the upper bound on $f_\lambda(n)$ could be improved but were only able to do so asymptotically when both λ and n are sufficiently large.

This paper leaves a gap between the lower bounds for broadcasting multiple messages and the performance of the algorithms presented in Section 4. We believe that the lower bound of Lemma 8 cannot be substantially improved without changing the model. In fact, we have developed several near-optimal algorithms for broadcasting multiple messages in the postal model [2]. These algorithms, however, are more complicated, make more restrictive assumptions about the level of synchronism of the system, and do not preserve the order of the messages. In addition, it was recently shown [13] that Algorithm DTREE is within a multiplicative factor of optimal time for order-preserving broadcast. It would be

interesting either to develop improved event-driven algorithms that preserve the order of messages or to improve the lower bound for such situations.

This paper explores the broadcasting problem in the postal model. Other problems that involve global communication include gossiping, combining, permuting, and sorting. It would be interesting to explore these and other problems in the postal model.

The postal model assumes that the communication latency is uniform system-wide. It would be interesting to relax this assumption. One direction is to explore time-changing values of λ and design algorithms that adapt to changing λ. Another direction is to investigate hierarchies of latency parameters that may be used to model subsystems within a larger system.

Appendix

This appendix proves Theorem 7 of Section 3 through a series of lemmas that bound the generalized Fibonacci function $F_\lambda(t)$ and its index function $f_\lambda(n)$.

The following lemma provides an upper bound for the generalized Fibonacci function $F_\lambda(t)$.

Lemma 19. *For any real numbers $\lambda \geq 1$ and $t \geq 0$, we have*

$$F_\lambda(t) \leq (\lceil \lambda \rceil + 1)^{\lfloor t/\lambda \rfloor}.$$

Proof. For a given ε, such that $0 \leq \varepsilon < \lambda$, we define a sequence of nonnegative real numbers $t_0, t_1, t_2, \ldots$, as follows: $t_i = i\lambda + \varepsilon$. For any such ε, we prove that $F_\lambda(t_i) \leq (\lceil \lambda \rceil + 1)^{\lfloor t_i/\lambda \rfloor}$ by induction on i. For $i = 0$, from the definitions of $F_\lambda(t_0)$ and ε, we have

$$F_\lambda(t_0) = F_\lambda(\varepsilon) = 1 = (\lceil \lambda \rceil + 1)^0 = (\lceil \lambda \rceil + 1)^{\lfloor \varepsilon/\lambda \rfloor} = (\lceil \lambda \rceil + 1)^{\lfloor t_0/\lambda \rfloor}.$$

Assume now that the claim holds for i, that is,

$$F_\lambda(t_i) = F_\lambda(i\lambda + \varepsilon) \leq (\lceil \lambda \rceil + 1)^i = (\lceil \lambda \rceil + 1)^{\lfloor t_i/\lambda \rfloor},$$

and prove the claim for $i + 1$. By performing $\lceil \lambda \rceil$ expansions of $F_\lambda(t_{i+1}) = F_\lambda((i + 1)\lambda + \varepsilon)$, according to the definition of $F_\lambda(t)$, we get

$$\begin{aligned} F_\lambda((i+1)\lambda + \varepsilon) &= F_\lambda(i\lambda + \varepsilon) + F_\lambda((i+1)\lambda + \varepsilon - 1) \\ &= F_\lambda(i\lambda + \varepsilon) + F_\lambda(i\lambda + \varepsilon - 1) + F_\lambda((i+1)\lambda + \varepsilon - 2) \\ &\vdots \\ &= F_\lambda(i\lambda + \varepsilon) + \cdots + F_\lambda(i\lambda + \varepsilon - (\lceil \lambda \rceil - 1)) \\ &\quad + F_\lambda((i+1)\lambda + \varepsilon - \lceil \lambda \rceil). \end{aligned}$$

Since $F_\lambda(t)$ is nondecreasing, all the $\lceil \lambda \rceil + 1$ terms on the right-hand side of the last

equality are less than or equal to $F_\lambda(i\lambda + \varepsilon)$. The induction hypothesis now gives

$$F_\lambda(t_{i+1}) = F_\lambda((i+1)\lambda + \varepsilon) \leq (\lceil\lambda\rceil + 1)F_\lambda(i\lambda + \varepsilon)$$
$$\leq (\lceil\lambda\rceil + 1)^{i+1} = (\lceil\lambda\rceil + 1)^{\lfloor t_{i+1}/\lambda\rfloor},$$

which completes the inductive proof. Since any $t \geq 0$ can be written as $t = \lfloor t/\lambda\rfloor\lambda + (t - \lfloor t/\lambda\rfloor\lambda)$, where $0 \leq t - \lfloor t/\lambda\rfloor\lambda < \lambda$, the lemma is proved for any value of $t \geq 0$. □

The upper bound on the generalized Fibonacci function $F_\lambda(t)$, provided by Lemma 19, implies the following lower bound on the index function $f_\lambda(n)$.

Lemma 20. *For any real number $\lambda \geq 1$ and integer number $n \geq 1$, we have*

$$f_\lambda(n) \geq \frac{\lambda \log n}{\log(\lceil\lambda\rceil + 1)}.$$

Proof. Given the values of $\lambda \geq 1$ and $n \geq 1$, *let* $0 \leq \varepsilon < \lambda$ be such that $f_\lambda(n) = \lfloor f_\lambda(n)/\lambda\rfloor\lambda + \varepsilon$. Part (3) of Claim 1 implies that

$$n \leq F_\lambda(f_\lambda(n)) = F_\lambda\left(\left\lfloor\frac{f_\lambda(n)}{\lambda}\right\rfloor\lambda + \varepsilon\right).$$

Applying Lemma 19 yields

$$n \leq (\lceil\lambda\rceil + 1)^{\lfloor f_\lambda(n)/\lambda\rfloor}.$$

By taking logarithms and rearranging the resultant inequality, we get

$$\frac{\log n}{\log(\lceil\lambda\rceil + 1)} \leq \left\lfloor\frac{f_\lambda(n)}{\lambda}\right\rfloor \leq \frac{f_\lambda(n)}{\lambda},$$

and the lemma follows. □

We continue by providing a lower bound for the generalized Fibonacci function $F_\lambda(t)$.

Lemma 21. *For any real numbers $\lambda \geq 1$ and $t \geq 0$, we have*

$$F_\lambda(t) \geq (\lceil\lambda\rceil + 1)^{\lfloor t/2\lambda\rfloor}.$$

Proof. For a given ε, such that $0 \leq \varepsilon < 2\lambda$, we define a sequence of nonnegative real numbers $t_0, t_1, t_2, \ldots$, as follows: $t_i = 2i\lambda + \varepsilon$. For any such ε, we prove that $F_\lambda(t_i) \geq (\lceil\lambda\rceil + 1)^{\lfloor t_i/2\lambda\rfloor}$ by induction on i. For $i = 0$, from the definitions of $F_\lambda(t_0)$ and ε, we have

$$F_\lambda(t_0) = F_\lambda(\varepsilon) \geq 1 = (\lceil\lambda\rceil + 1)^0 = (\lceil\lambda\rceil + 1)^{\lfloor\varepsilon/2\lambda\rfloor} = (\lceil\lambda\rceil + 1)^{\lfloor t_0/2\lambda\rfloor}.$$

Assume now that the claim holds for i, that is,

$$F_\lambda(t_i) = F_\lambda(2i\lambda + \varepsilon) \geq (\lceil\lambda\rceil + 1)^i = (\lceil\lambda\rceil + 1)^{\lfloor t_i/2\lambda\rfloor},$$

and prove the claim for $i+1$. By performing $\lceil \lambda \rceil$ expansions of $F_\lambda(t_{i+1}) = F_\lambda(2(i+1)\lambda + \varepsilon)$, according to the definition of $F_\lambda(t)$, we get

$$\begin{aligned} F_\lambda(2(i+1)\lambda+\varepsilon) &= F_\lambda(2i\lambda+\lambda+\varepsilon) + F_\lambda(2(i+1)\lambda+\varepsilon-1) \\ &= F_\lambda(2i\lambda+\lambda+\varepsilon) + F_\lambda(2i\lambda+\lambda+\varepsilon-1) \\ &\quad + F_\lambda(2(i+1)\lambda+\varepsilon-2) \\ &\vdots \\ &= F_\lambda(2i\lambda+\lambda+\varepsilon) + \cdots + F_\lambda(2i\lambda+\lambda+\varepsilon-(\lceil\lambda\rceil-1)) \\ &\quad + F_\lambda(2(i+1)\lambda+\varepsilon-\lceil\lambda\rceil). \end{aligned}$$

Since $F_\lambda(t)$ is nondecreasing, all the $\lceil \lambda \rceil + 1$ terms on the right-hand side of the last equality are greater than or equal to $F_\lambda(2i\lambda + \varepsilon)$. The induction hypothesis now gives

$$\begin{aligned} F_\lambda(t_{i+1}) &= F_\lambda(2(i+1)\lambda+\varepsilon) \geq (\lceil\lambda\rceil+1)F_\lambda(2i\lambda+\varepsilon) \\ &\geq (\lceil\lambda\rceil+1)^{i+1} = (\lceil\lambda\rceil+1)^{\lfloor t_{i+1}/2\lambda \rfloor}, \end{aligned}$$

which completes the inductive proof. Since, for any $t \geq 0$, we have

$$t = \left\lfloor \frac{t}{2\lambda} \right\rfloor 2\lambda + \left(t - \left\lfloor \frac{t}{2\lambda} \right\rfloor 2\lambda\right),$$

where $0 \leq t - \lfloor t/2\lambda \rfloor 2\lambda < 2\lambda$, the lemma is proved for any value of $t \geq 0$. □

The lower bound on the generalized Fibonacci function $F_\lambda(t)$, provided by Lemma 21, implies the following upper bound on the index function $f_\lambda(n)$.

Lemma 22. *For any real number $\lambda \geq 1$ and integer number $n \geq 1$, we have*

$$f_\lambda(n) \leq 2\lambda + \frac{2\lambda \log n}{\log(\lceil\lambda\rceil+1)}.$$

Proof. Given the values of $\lambda \geq 1$ and $n \geq 1$, if $f_\lambda(n) \leq 2\lambda$, then the lemma follows immediately. Assume, otherwise, that $f_\lambda(n) > 2\lambda$ and let $0 \leq \varepsilon < 2\lambda$ be such that $f_\lambda(n) - \varepsilon = \lfloor f_\lambda(n)/2\lambda \rfloor 2\lambda$. We first check the case of $\varepsilon > 0$. Part (4) of Claim 1 implies that

$$n > F_\lambda(f_\lambda(n) - \varepsilon) = F_\lambda\left(\left\lfloor \frac{f_\lambda(n)}{2\lambda} \right\rfloor 2\lambda\right).$$

Applying Lemma 21 yields

$$n > (\lceil\lambda\rceil+1)^{\lfloor f_\lambda(n)/2\lambda \rfloor}.$$

By taking logarithms and rearranging the resultant inequality, we get

$$\left\lfloor \frac{f_\lambda(n)}{2\lambda} \right\rfloor < \frac{\log n}{\log(\lceil\lambda\rceil+1)}.$$

Now since

$$\frac{f_\lambda(n) - 2\lambda}{2\lambda} \leq \left\lfloor \frac{f_\lambda(n)}{2\lambda} \right\rfloor,$$

we get

$$\frac{f_\lambda(n) - 2\lambda}{2\lambda} < \frac{\log n}{\log(\lceil \lambda \rceil + 1)},$$

which implies the lemma for the case of $\varepsilon > 0$. We next check the case of $\varepsilon = 0$, that is, $f_\lambda(n)/2\lambda$ is an integral number. Since $(f_\lambda(n)/2\lambda - 1)2\lambda < f_\lambda(n)$, part (4) of Claim 1 implies that

$$n > F_\lambda\left(\left(\frac{f_\lambda(n)}{2\lambda} - 1\right)2\lambda\right).$$

The right-hand side of the last inequality is well defined, since we assume that $f_\lambda(n) \geq 2\lambda$ and, therefore, $f_\lambda(n)/2\lambda$ is an integral number greater than 1. Applying Lemma 21 now yields

$$n > (\lceil \lambda \rceil + 1)^{(f_\lambda(n)/2\lambda) - 1}.$$

By taking logarithms and rearranging the resultant inequality, the lemma is proved for the case of $\varepsilon = 0$, as well. □

In what follows we show that when the values of λ and n are sufficiently large, an asymptotic lower bound on $F_\lambda(t)$ and an asymptotic upper bound for $f_\lambda(n)$ exist that are very close to the bounds of Lemmas 19 and 20, respectively. In the discussion that follows we assume that λ is large enough. Let

$$\alpha = 1 + \frac{\ln\ln(\lambda + 1) + 1}{\ln(\lambda + 1) - (\ln\ln(\lambda + 1) + 1)}.$$

Notice that $\alpha \geq 1$ for large enough values of λ. We now present two technical claims without proving them.

Claim 23. *For*

$$\alpha = 1 + \frac{\ln\ln(\lambda + 1) + 1}{\ln(\lambda + 1) - (\ln\ln(\lambda + 1) + 1)}$$

and sufficiently large λ, we have

$$\frac{e \ln(\lambda + 1)}{\alpha\lambda} (\lambda + 1)^{(\lambda - 1)\alpha\lambda} \leq 1.$$

Claim 24. *For*

$$\alpha = 1 + \frac{\ln\ln(\lambda + 1) + 1}{\ln(\lambda + 1) - (\ln\ln(\lambda + 1) + 1)}$$

and sufficiently large λ, we have

$$(\lambda+1)^{1/\alpha\lambda}-1 \leq \frac{e\ \ln(\lambda+1)}{\alpha\lambda}.$$

The next lemma provides an asymptotic bound on the generalized Fibonacci function $F_\lambda(t)$.

Lemma 25. *For sufficiently large real number λ and any real number $t \geq 0$, we have*

$$F_\lambda(t) \geq (\lambda+1)^{(t/\alpha\lambda)-1},$$

where

$$\alpha = 1 + \frac{\ln\ln(\lambda+1)+1}{\ln(\lambda+1)-(\ln\ln(\lambda+1)+1)}.$$

Proof. Given λ, assume to the contrary that the lemma does not hold and let t_0 be the minimal t such that $F_\lambda(t) < (\lambda+1)^{(t/\alpha\lambda)-1}$. It must be that $t_0 \geq \lambda$, since for $t < \lambda$ we have $(\lambda+1)^{(t/\alpha\lambda)-1} \leq (\lambda+1)^0 = 1 = F_\lambda(t)$. From the definition of $F_\lambda(t)$ for $t \geq \lambda$ and the minimality of t_0, we have

$$\begin{aligned}(\lambda+1)^{(t_0/\alpha\lambda)-1} > F_\lambda(t_0) &= F_\lambda(t_0-1)+F_\lambda(t_0-\lambda)\\ &\geq (\lambda+1)^{((t_0-1)/\alpha\lambda)-1}+(\lambda+1)^{((t_0-\lambda)/\alpha\lambda)-1}.\end{aligned}$$

Dividing both sides of the inequality by $(\lambda+1)^{((t_0-\lambda)/\alpha\lambda)-1}$ gives

$$(\lambda+1)^{\lambda/\alpha\lambda} > (\lambda+1)^{(\lambda-1)/\alpha\lambda}+1.$$

Rearranging the last inequality gives

$$((\lambda+1)^{1/\alpha\lambda}-1)(\lambda+1)^{(\lambda-1)/\alpha\lambda} > 1.$$

Using Claim 24 to substitute $(e\ \ln(\lambda+1))/\alpha\lambda$ for $(\lambda+1)^{1/\alpha\lambda}-1$ in the last inequality, yields

$$\frac{e\ \ln(\lambda+1)}{\alpha\lambda}\ (\lambda+1)^{(\lambda-1)/\alpha\lambda} > 1.$$

The last inequality contradicts Claim 23 and, thus, the lemma follows. □

Lemma 25 implies the following asymptotic upper bound on the index function $f_\lambda(n)$.

Lemma 26. *For sufficiently large real number λ and integer number n, we have*

$$f_\lambda(n) \leq (1+h(\lambda))\ \cdot\ \frac{\lambda\ \log\ n}{\log(\lambda+1)},$$

where the function $h(\lambda) \to 0$ when $\lambda \to \infty$ and $n \geq 2^\lambda$.

Proof. For any $\varepsilon > 0$, from part (4) of Claim 1 and from Lemma 25, we have

$$n > F_\lambda(f_\lambda(n) - \varepsilon) \geq (\lambda + 1)^{((f_\lambda(n) - \varepsilon)/\alpha\lambda) - 1},$$

where

$$\alpha = 1 + \frac{\ln\ln(\lambda + 1) + 1}{\ln(\lambda + 1) - (\ln\ln(\lambda + 1) + 1)}.$$

By taking logarithms and rearranging, we get

$$\begin{aligned} f_\lambda(n) &< \alpha\lambda \left(\frac{\log n}{\log(\lambda + 1)} + 1 \right) + \varepsilon \\ &\leq \left(\alpha + \frac{\alpha \log(\lambda + 1)}{\log n} + \varepsilon \right) \frac{\lambda \log n}{\log(\lambda + 1)}. \end{aligned}$$

The proof is completed by defining $\alpha + (\alpha \log(\lambda + 1))/\log n + \varepsilon$ to be $1 + h(\lambda)$. Indeed, when $\lambda \to \infty$ and $n \geq 2^\lambda$, the choice of α and the fact that ε can be as small as we wish imply that $h(\lambda) \to 0$. □

Acknowledgments

We thank Shay Kutten of IBM Research for referring us to the results of [6] and for helpful discussions. We thank Don Coopersmith of IBM Research for helping us in the asymptotic analysis of the generalized Fibonacci function. We also thank Marc Snir of IBM Research for several helpful discussions and suggestions.

References

[1] A. Bar-Noy and S. Kipnis, Designing broadcasting algorithms in the postal model for message-passing systems, *Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures*, June 1992, pp. 13–22. Also appeared as IBM Research Report RC-17429, November 1991.

[2] A. Bar-Noy and S. Kipnis, Multiple message broadcasting in the postal model, *Proceedings of the 7th International Parallel Processing Symposium*, Newport Beach, CA, April 1993. Also appeared as IBM Research Report RC-18230, August 1992.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[4] I. Cidon and I. Gopal, PARIS: an approach to integrated high-speed private networks, *International Journal of Digital and Analog Cabled Systems*, Vol. 1, No. 2, April–June 1988, pp. 77–85.

[5] I. Cidon, I. Gopal, and S. Kutten, New models and algorithms for future networks, *Proceedings of the 7th Annual Symposium on Principles of Distributed Computing*, August 1988, pp. 75–89.

[6] I. Cidon, I. Gopal, and S. Kutten, Optimal computation of global sensitive functions in fast networks, in *Distributed Algorithms*, J. van Leeuwen and N. Santoro, eds., Lecture Notes in Computer Science, Vol. 486, Springer-Verlag, New York, 1990, pp. 185-191.

[7] D. Clark, B. Davie, D. Farber, I. Gopal, B. Kadaba, D. Sincoskie, J. Smith, and D. Tennenhouse, The AURORA gigabit testbed, *Computer Networks and ISDN*, 1991.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, LogP: towards a realistic model of parallel computation, *Proceedings of the 4th SIGPLAN Symposium on Principles and Practices of Parallel Programming*, San Diego, CA, May 1993.

[9] W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, M. Larivee, R. Lethin, P. Nuth, S. Wills, P. Carrick, and G. Fyler, The J-Machine: a fine-grain concurrent computer, *Information Processing 89*, Elsevier Science, Amsterdam, 1989, pp. 1147–1153.

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Vol. I, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[11] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, A survey of gossiping and broadcasting in communication networks, *Networks*, Vol. 18, No. 4, 1988, pp. 319–349.

[12] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hills, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, The network architecture of the Connection Machine CM-5, *Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures*, June 1992, pp. 272–285.

[13] P. D. Mackenzie, A lower bound for order preserving broadcast in the postal model, manuscript, University of Texas at Austin, December 1992.

[14] S. Report and R. Special, Gigabit network testbeds, Special Report, *IEEE Computer Magazine*, Vol. 23, No. 9, 1990, pp. 77–80.

[15] J. S. Turner and L. F. Wyatt, A packet network architecture for integrated services, *GLOBECOM 83*, 1983, pp. 2.1.1–2.1.6.

[16] C. B. Stunkel, D. G. Shea, B. Abali, M. M. Denneau, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker, Architecture and implementation of Vulcan, *Proceedings of the 8th International Parallel Processing Symposium,* Cancun, Mexico, April 1994, pp. 268–274.

Received September 21, 1992, and in final form October 22, 1993.