Scattering and Gathering Messages in Networks of Processors

Sandeep N. Bhatt, Geppino Pucci, Abhiram Ranade, and Arnold L. Rosenberg, *Senior Member, IEEE*

***Abstract*— The operations of scattering and gathering in a network of processors involve one processor of the network—call it $\mathcal{P}_0$—communicating with all other processors. In scattering, $\mathcal{P}_0$ sends (possibly) distinct messages to all other processors; in gathering, the other processors send (possibly) distinct messages to $\mathcal{P}_0$. We consider networks that are trees of processors. We present algorithms for scattering messages from and gathering messages to the processor that resides at the root of the tree. The algorithms are: 1) quite general, in that the messages transmitted can differ arbitrarily in length; 2) quite strong, in that they send messages along noncolliding paths, hence do not require any buffering or queuing mechanisms in the processors; and 3) quite efficient: The algorithms for scattering in general trees are optimal, the algorithm for gathering in a path is optimal, and the algorithms for gathering in general trees are nearly optimal. Our algorithms can easily be converted, via the use of spanning trees, to efficient algorithms for scattering and gathering in networks of arbitrary topologies.**

***Index Terms*—Bufferless communication, communication primitives, communication schedules, distributed scheduling, interconnection networks, message routing, one-to-all personalized communication.**

I. Introduction

A. Communication in Parallel Computation

COMMUNICATION is an essential component of parallel computation. A variety of modes of communication have been studied within the framework of *networks of processors* —identical processing elements (PE's) that communicate by means of an interconnection network. The most commonly studied modes are the following.

- (Partial) permutation routing [1], [3], [10], [13], [17] is a form of communication in which each PE is both the sender and recipient of (at most) one message.
- Broadcasting [8], [12] is a form of communication in which one PE sends one specific message to all other PE's.
- Gossiping (or all-to-all broadcasting) [7], [16] is a form of communication in which each PE sends one specific message to all other PE's.

Baumslag and Annexstein [1], Johnsson and Ho [8], and Saad and Schultz [14] (among others) point out that these popular forms of communication do not exhaust the algorithmically useful possibilities. Specifically, they add to the menu of communication modes the operations of *scattering and gathering*[1]

- Scattering (or one-to-all personalized communication) is a form of communication in which one PE sends (possibly) distinct messages to all other PE's.
- Gathering is a form of communication in which all PE's send (possibly) distinct messages to one specific PE.

Efficient algorithms for a general version of the operations of scattering and gathering form the subject matter of the current paper. Specifically, we present efficient algorithms for scattering from and gathering to the root PE of a general tree-structured network.[2] We present an optimal algorithm for scattering from the root of a general tree, an optimal algorithm for gathering to the root of a unary tree (i.e., the end-PE of a path), and a nearly optimal algorithm for gathering to the root of a general tree. Via the use of spanning trees, our (nearly) optimal tree-oriented algorithms become efficient algorithms for scattering and gathering in networks of arbitrary topology. The generality of our study manifests itself in three ways.

1) We allow messages to differ in length by arbitrary amounts; indeed, some messages may be null. This contrasts with the studies in [1], [4], [8], [14], wherein all messages have the same length.
2) We scatter and gather messages in trees of arbitrary shape and, hence, via the use of spanning trees, in networks of arbitrary topologies. This contrasts with the studies in [4], [8], [14], [15], which focus on a small repertoire of networks, such as rings, meshes, and hypercubes.
3) We transmit messages along noncolliding paths in our networks, hence do not require any buffering or queueing mechanisms in the PE's. This constrasts with virtually all other studies of message transmission in networks.

Manuscript received December 9, 1991; revised June 26, 1992. This work was supported in part by the National Science Foundation under Grants CCR-89-07426 and CCR-90-13184, in part by the NSF/DARPA under Grant CCR-89-08285, in part by Air force Grant AFOSR-89-0382, in part by the CNR project "Sistemi Informatici e Calcolo Parallelo, and in part by AFOSR under Grant F49620-87-C-0041.

S. N. Bhatt is with Bell Communications Research, Morristown, NJ 07960.

G. Pucci is with the Dipartimento di Elettronica ed Informatica, Universita' di Padova, 35131 Padova, Italy.

A. Ranade is with the Division of Computer Science, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

A. L. Rosenberg is with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 9208770.

[1] Other important modes have also been studied, including *multiscattering* [11] and *exchange* [2], but less frequently.

[2] Henceforth, for brevity: we use the term "tree" for "tree-structured network;" also, we use the term "network" to denote both a network of processors and its underlying interconnection network; context should always disambiguate each occurrence of the word.

One might be able to rationalize our demand for unbuffered communication in terms of resource conservation. Buffering requires both additional memory (each PE must be prepared to store the longest message in the system) and time (e.g., for the processing of addresses). However, our overriding motivation in this study was to understand communication in networks better, by determining the cost of this strict assumption in terms of the complexity of the problems of scattering and gathering general messages in general networks.

B. The Computing Model

1) Networks of Processors We study the problems of scattering from and gathering to the root-PE of a *synchronous* tree of arbitrary shape. Each network $\mathcal{A}$ comprises $n+1$ identical PE's, $\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_n$. By convention, we always let $\mathcal{P}_0$ denote the root of the tree, i.e., the PE which is the source of messages in a scattering operation and the target of messages in a gathering operation.

The PE's of the networks we study have neither message buffers nor queues. Messages within networks must, therefore, be scheduled so as never to "collide" with one another. For the operation of scattering, the fact that we scatter within a tree guarantees such avoidance; for the operation of gathering, this scheduling is a major challenge.

The networks we study use the *single-port* communication regimen: during each communication step, a PE can send information to at most one of its immediate neighbors and, simultaneously, receive information from at most one of its immediate neighbors; the sending and receiving neighbors may be distinct. We do, however, allow a PE to perform (say, arithmetic) computations while communicating, as well as to access its local memory. This regimen is to be contrasted with the *multiport* communication regimen, in which a PE can send and receive information from *each* of its immediate neighbors in one step. In Section IV, we indicate briefly how our results extend to a multiport model.

The networks we study communicate *in rounds;* i.e., while a scattering (resp., a gathering) operation is in progress, there is no other communication going on in the network. This means that the only resource contention we must worry about arises from the many messages that are being scattered (resp., gathered) in the current operation. This regimen is to be contrasted with the one studied in [2], wherein the present study of bufferless communication is generalized to allow each PE to be both the source of and the destination for abitrarily many messages at once. As an aside, the study in [2] compensates for the generality of its communication setting—bufferless PE's passing messages in arbitrary ways—by restricting attention to simple network topologies, specifically, one- and two-dimensional meshes (i.e., rings and toroidal meshes).

Porting to General Networks: Our efficient collision-free algorithms can be transported easily to networks of arbitrary topology via the use of an "efficient" spanning tree of (the undirected graph underlying) the network in question, rooted at the singular PE for the scattering or gathering operation. For the operation of *scattering* and for the operation of *gathering under a multiport regimen,* one would sensibly choose a *breadth-first* spanning tree, in order to ensure that every message travels the shortest possible distance to its destination. The possibility of large node-degrees in breadth-first trees causes no concern, because in a scattering operation, a PE is receiving or transmitting at most one message at each step, and in a multiport gathering operation, a PE can service as many ports as it has at each step. For the operation of *gathering under a single-port regimen,* the time required to accommodate large node-degrees in the tree can dominate the time for single-port gathering. Broadcasting is typically part of the synchronization protocol needed for gathering in multisuccessor networks, and high-degree nodes can slow down single-port broadcasts. (As an extreme example, compare the times for single-port broadcasting in an n-PE network $\mathcal{A}$ in which every pair of nodes is connected by an edge: a) using a complete binary spanning tree of $\mathcal{A}$, versus b) using a single-level degree-$(n-1)$ spanning tree.) Consequently, in this case, one might seek a spanning tree whose structure approximates that of a *minimum broadcast tree* [9].

Remark 1: The framework just outlined may represent only the communication subsystem of a heterogeneous parallel architecture; for instance, the architecture viewed as a whole may have PE's of differing powers and sizes, which operate asynchronously except during global communication operations (such as scattering and gathering).

2) Messages and Message Sequences Each message $\mathcal{M}_i$ involved in a scattering or gathering operation is a sequence of some number L_i (perhaps zero) of atomic *flits.* A flit is the largest unit of information that the network can transmit between adjacent nodes in one communication step (i.e., in one so-called transit time).

A message is treated as an *indivisible unit* during a scattering or gathering operation, in the sense that the L flits of a message are never interrupted by flits from other messages. Initially, the L flits of the message are all in the originating PE; after the message has begun to travel through the network, its flits are always in contiguous PE's; the lack of buffering ensures that each flit is in a separate PE once it leaves the originating PE. A consequence of the indivisibility of messages is that addressing information needs appear only in the first flit of the message, thereby lessening both the setup time for messages and the aggregate length devoted to addressing information.

Let $\mathcal{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n \rangle$ be a sequence of messages (to be scattered or gathered). Let

$$\boldsymbol{N}(\mathcal{M}) = \langle i_1, i_2, \cdots, i_k \rangle$$

denote, in increasing order, the subsequence of message indices whose messages are nonnull, i.e., for which $L_{i_j} > 0$.

3) The Scattering and Gathering Problems In a scattering operation, the root-PE $\mathcal{P}_0$ has a message $\mathcal{M}_i$ of length L_i to send to each PE $\mathcal{P}_i$, for $i > 0$. In a gathering operation, each PE $\mathcal{P}_i$, where $i > 0$, has a message $\mathcal{M}_i$ of length L_i destined for PE $\mathcal{P}_0$. (For both operations, some messages $\mathcal{M}_i$ may be null, so that $L_i = 0$.) We perform these operations in trees of arbitrary shapes, subject to the following:

- Once a message has been dispatched by its originating PE, it encounters no interruption until it is received by its destination PE. In particular, each intermediate PE must relay the message with no queueing or buffering, and messages are treated as indivisible units (in the sense described earlier).
- For each $i > 0$, message M_i will be routed along the unique path ρ_i that connects PE $\mathcal{P}_0$ and PE $\mathcal{P}_i$ in the tree. We let $\delta(i)$ denote the length of path ρ_i, i.e., the distance that message $\mathcal{M}_i$ must travel.

4) Problem Complexity We measure the complexity of a scattering or gathering operation in terms of the time for delivering all relevant messages. Focussing on a fixed but arbitrary message sequence $\mathcal{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n \rangle$, this time is formalized as follows:

The Time for Scattering: A *schedule for scattering message sequence* $\mathcal{M}$ is a permutation σ (for "scattering-schedule") of the index-sequence $\boldsymbol{N}(\mathcal{M}) = \langle i_1, i_2, \cdots, i_k \rangle$, i.e., a one-to-one function

$$\sigma: \{1, 2, \cdots, k\} \rightarrow \boldsymbol{N}(\mathcal{M}).$$

The intended interpretation is that PE $\mathcal{P}_0$ sends out message $\mathcal{M}_{\sigma(1)}$, then message $\mathcal{M}_{\sigma(2)}$, then $\mathcal{M}_{\sigma(3)}$, and so on, in that order, in a steady stream, with no intervening gaps. Thus, under schedule σ, given index i with $L_i \neq 0$, PE $\mathcal{P}_0$ begins transmitting message $\mathcal{M}_i$ at *dispatch time*

$$\tau_\sigma(i) = \sum_{\{j \mid \sigma^{-1}(j) < \sigma^{-1}(i)\}} L_j. \tag{1}$$

(Note the effect of the single-port regimen.) Message $\mathcal{M}_i$ arrives at its destination, PE $\mathcal{P}_i$, at *arrival time*

$$\alpha_\sigma(i) = \tau_\sigma(i) + L_i + \delta(i). \tag{2}$$

The *time for scattering message sequence* $\mathcal{M}$ under scattering-schedule σ is the time it takes for every flit of $\mathcal{M}$ to reach its final destination; symbolically,

$$T^{\text{scat}}(\sigma; \mathcal{M}) = \max_{i \in \boldsymbol{N}(\mathcal{M})} \{\alpha_\sigma(i)\}. \tag{3}$$

Equation (3) implies the following simple result, which delimits the difference between the best and worst scattering-schedules. The proof is left to the reader.

Proposition 1: Let σ be a scattering-schedule for message sequence $\mathcal{M}$. Assuming that message $\mathcal{M}_i$ of $\mathcal{M}$, for $1 \leq i \leq n$, has length L_i, the time for σ satisfies the following bounds:

$$\begin{aligned} &\max \left(\sum_{i \in \boldsymbol{N}(\mathcal{M})} L_i, \max_{i \in \boldsymbol{N}(\mathcal{M})} \{\delta(i)\} \right) \\ &\quad \leq T^{\text{scat}}(\sigma; \mathcal{M}) \\ &\quad \leq \sum_{i \in \boldsymbol{N}(\mathcal{M})} L_i + \max_{i \in \boldsymbol{N}(\mathcal{M})} \{\delta(i)\}. \end{aligned}$$

The Time for Gathering: A *schedule for gathering message sequence* $\mathcal{M}$ is a sequence of integers

$$\gamma = \tau_\gamma(i_1), \tau_\gamma(i_2), \cdots, \tau_\gamma(i_k),$$

(for "gathering-schedule"), where $\boldsymbol{N}(\mathcal{M}) = \{i_1, i_2, \cdots, i_k\}$. The intended interpretation is that each $\tau_\gamma(i)$ (where $i \in \boldsymbol{N}(\mathcal{M})$) is the *dispatch time* for message $\mathcal{M}_i$, i.e., the time when PE $\mathcal{P}_i$ begins transmitting $\mathcal{M}_i$. The last flit of message $\mathcal{M}_i$ is received by PE $\mathcal{P}_0$ at *arrival time*

$$\alpha_\gamma(i) = \tau_\gamma(i) + L_i + \delta(i).$$

The *time for gathering message sequence* $\mathcal{M}$ under gathering-schedule γ is the time it takes for every flit of $\mathcal{M}$ to reach PE $\mathcal{P}_0$; symbolically,

$$T^{\text{gath}}(\gamma; \mathcal{M}) = \max_{i \in \boldsymbol{N}(\mathcal{M})} \{\alpha_\gamma(i)\}.$$

The Challenges: Note that neither the time for scattering, T^{scat}, nor the time for gathering, T^{gath}, allows for any delay of messages at nodes other than the originating node. This means that our message-scheduling algorithms cannot rely on—so the network need not provide—any mechanism for buffering or queueing messages in PE's. This lack of buffering provides an additional challenge in scheduling the gathering operation, which is lacking in the scattering operation. Namely, the scheduling algorithm must provide, in a distributed manner, for the dispatching of messages in the network so that messages never collide on their paths to PE $\mathcal{P}_0$.

Remark 2: Our timing model is somewhat simpler than that of some of the earlier cited sources. Specifically, we charge L time units to transmit a message containing L flits; some sources (such as [4]) would charge a message setup time of β time units, plus a per-flit transmission time of τ time units for this message, for a total cost of $\beta + L\tau$ time units. This change of model would not affect our analyses in a material way.

Remark 3: As suggested earlier, our algorithms for scattering and gathering in arbitrary networks employ spanning trees that are *fixed,* independent of the message sequence $\mathcal{M}$. For many networks, there exists no single spanning tree that is simultaneously optimal for the single-port regimen and for all message sequences, especially because messages can be null. This means that our algorithms for general networks will often be suboptimal.

C. Related Work

Saad and Schultz [14] define the operations of scattering and gathering in full generality but present algorithms only for a specific repertoire of network topologies and for the case of equal-length messages. Fraigniaud *et al.* [4] prove the optimality of the Saad–Schultz algorithm for scattering on a unidirectional ring of processors. Stout and Wagar [15] and Johnsson and Ho [8] present optimal algorithms for scattering equal-length messages on a hypercube, using both the single-port and multiport communication regimes. Li [11] considers performing several scattering operations at once on a reconfigurable network of processors. Bhatt *et al.* [2] study the most general type of communication, wherein each PE has

a distinct message for each other PE, in bufferless rings and toroidal networks. All of these references, save the last, assess time $\beta + L\tau$ for transmitting an L-flit message.

II. Scattering on Networks of Processors

Say that scattering-schedule σ is *optimal* for message sequence $\mathcal{M}$ on a given tree if *on that tree,*

$$T^{\text{scat}}(\sigma; \mathcal{M}) \leq T^{\text{scat}}(\sigma'; \mathcal{M})$$

for any other scattering-schedule σ' for $\mathcal{M}$.

It is shown in [4] that the *unique* optimal scattering-schedule for equal-length messages on a unidirectional ring is given by the permutation $\sigma(i) = n - i + 1$, i.e., by sending out messages according to a *farthest-destination-first* (FDF) regimen, one in which nonnull messages are dispatched in *decreasing* order of the distances to their destinations. We now prove that the optimality of FDF schedules persists when the lengths of the scattered messages are general and when the scattering is done from the root-PE of an arbitrary tree. Specifically, we show that, within this setting, for every message sequence $\mathcal{M}$, every FDF scattering-schedule is optimal for $\mathcal{M}$ (although there may be optimal non-FDF schedules also). It is consistent with intuition that FDF scattering-schedules need no longer be the *unique* optimal ones when one considers messages of arbitrary lengths, because a single enormous message could so dominate the message transmission time as to mask the order of a collection of small messages sent out right after it. Since the optimality of all FDF schedules ensures the optimality of a large family of scattering algorithms, we present the following theorem in lieu of a specific optimal algorithm.

Theorem 1: Every FDF scattering-schedule for scattering from the root-PE of an arbitrary tree is optimal.

Proof: Let the tree $\mathcal{T}$ with root-PE $\mathcal{P}_0$ be fixed.

The Theorem makes two claims, which we treat in turn. First, we prove that every optimal scattering-schedule for a given message sequence can be replaced by an FDF scattering-schedule for the sequence with no increase in scattering time (so the FDF schedule is also optimal). Second, we prove that every FDF schedule for a message sequence is optimal, i.e., that messages destined for equidistant PE's can be dispatched in any order. The reader should note the crucial role of our communicating on a tree in what follows.

Claim 1: For every message sequence $\mathcal{M}$ and every scattering-schedule σ for $\mathcal{M}$, there is an FDF scattering-schedule σ' for $\mathcal{M}$ with

$$T^{\text{scat}}(\sigma'; \mathcal{M}) \leq T^{\text{scat}}(\sigma; \mathcal{M}).$$

Moral: Every message sequence has an optimal FDF scattering-schedule.

Claim 1 asserts that one can never decrease the scattering time of a schedule by dispatching a nonnull message that is destined for a nearby PE before a nonnull message that is destined for a more distant PE. This is not surprising, as one hopes to use pipelining to make progress in sending the nearby message while the distant message is in transit.

Proof of Claim: Assume, for contradiction, that there is a message sequence $\mathcal{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n \rangle$ such that no optimal scattering-schedule for $\mathcal{M}$ observes the FDF regimen. Let σ_1 be any optimal scattering-schedule for $\mathcal{M}$. Because σ_1 does not observe the FDF regimen, there must exist PE indexes i and j, both in $\boldsymbol{N}(\mathcal{M})$, such that

$$\sigma_1^{-1}(i) = \sigma_1^{-1}(j) + 1; \qquad \delta(i) > \delta(j); \qquad L_j > 0.$$

Let σ_2 be the scattering-schedule for $\mathcal{M}$ obtained from σ_1 by interchanging $\sigma_1^{-1}(i)$ and $\sigma_1^{-1}(j)$; i.e.,

$$\sigma_2(k) = \begin{cases} i & \text{if } \sigma_1(k) = j \\ j & \text{if } \sigma_1(k) = i \\ \sigma_1(k) & \text{otherwise.} \end{cases}$$

We claim that

$$T^{\text{scat}}(\sigma_2; \mathcal{M}) \leq T^{\text{scat}}(\sigma_1; \mathcal{M}). \tag{4}$$

By (3), inequality (4) will follow from the inequality

$$\max\{\alpha_{\sigma_1}(i), \alpha_{\sigma_1}(j)\} \geq \max\{\alpha_{\sigma_2}(i), \alpha_{\sigma_2}(j)\};$$

we establish this inequality by analyzing the dispatch and arrival times of messages under schedules σ_1 and σ_2. We begin by noting that (1) implies the following relations among the *dispatch* times under schedules σ_1 and σ_2. (All indexes referred to are associated with nonnull messages.)

$$\tau_{\sigma_1}(i) = \tau_{\sigma_1}(j) + L_j$$

$$\tau_{\sigma_2}(k) = \begin{cases} \tau_{\sigma_1}(j) & \text{if } k = i \\ \tau_{\sigma_1}(i) + L_i - L_j & \text{if } k = j \\ \tau_{\sigma_1}(k) & \text{otherwise.} \end{cases}$$

(Note that $\tau_{\sigma_1}(i) > \tau_{\sigma_1}(j)$ because $L_j > 0$.) By (2), therefore, we infer the following relations among the *arrival* times under schedules σ_1 and σ_2

$$\begin{aligned} \alpha_{\sigma_1}(j) &= \tau_{\sigma_1}(j) + L_j + \delta(j) \\ \alpha_{\sigma_1}(i) &= \tau_{\sigma_1}(i) + L_i + \delta(i) \\ \alpha_{\sigma_2}(i) &= \tau_{\sigma_2}(i) + L_i + \delta(i) \\ &= \tau_{\sigma_1}(j) + L_i + \delta(i) \\ \alpha_{\sigma_2}(j) &= \tau_{\sigma_2}(j) + L_j + \delta(j) \\ &= \tau_{\sigma_1}(i) + L_i + \delta(j) \end{aligned}$$

while $\alpha_{\sigma_2}(k) = \alpha_{\sigma_1}(k)$ for all $k \notin \{i, j\}$. (These last equations on α_{σ_1} and α_{σ_2} hold because we route messages within a tree.) We can now deduce that

$$\alpha_{\sigma_1}(i) = \max\{\alpha_{\sigma_1}(i), \alpha_{\sigma_1}(j), \alpha_{\sigma_2}(i), \alpha_{\sigma_2}(j)\}.$$

Specifically,

$$\begin{aligned} &\alpha_{\sigma_1}(i) > \alpha_{\sigma_1}(j) && \text{because } \delta(i) > \delta(j) \\ &&& \text{and } \tau_{\sigma_1}(i) = \tau_{\sigma_1}(j) + L_j \\ &\alpha_{\sigma_1}(i) > \alpha_{\sigma_2}(i) && \text{because } \tau_{\sigma_1}(i) > \tau_{\sigma_1}(j) \\ &\alpha_{\sigma_1}(i) > \alpha_{\sigma_2}(j) && \text{because } \delta(i) > \delta(j). \end{aligned}$$

It follows from this chain of reasoning that $T^{\text{scat}}(\sigma_2; \mathcal{M}) \leq T^{\text{scat}}(\sigma_1; \mathcal{M})$, with strict inequality whenever message $\mathcal{M}_i$ is the last message to arrive at its destination under schedule

σ_1. Now, if scattering-schedule σ_2 observes the FDF regimen, then this inequality already contradicts the assumption that no FDF scattering-schedule is optimal for $\mathcal{M}$. If scattering-schedule σ_2 does not observe the FDF regimen, then it is "one transposition closer" to observing the regimen than is schedule σ_1. In particular, we can iterate the operation of transposing transmission times that violate the FDF regimen a finite number of times (in fact, no more than $n(n-1)/2$ times) to arrive at a scattering-schedule σ that *does* observe the FDF regimen and that has scattering time no greater than that of schedule σ_1, thus contradicting the assumption that no FDF scattering-schedule is optimal for $\mathcal{M}$. □

Claim 2: All FDF scattering-schedules take the same time.

Moral: Every scattering-schedule that observes the FDF regimen is optimal.

Proof of Claim: Say that the scattering-schedule σ observes the FDF regimen. The only way to alter σ without violating the regimen is to rearrange the transmission order of messages destined for equidistant PE's. We claim that such rearrangement does not alter the time for the schedule and, hence, must preserve optimality. To wit, (1) and (2) imply the following. If messages $\mathcal{M}_{j_1}, \mathcal{M}_{j_2}, \cdots, \mathcal{M}_{j_l}$ are all destined for PE's at distance Δ from PE $\mathcal{P}_0$, and if the earliest dispatch time of any of these messages is τ, then the latest arrival time of any of these messages is

$$\tau + \sum_{h=1}^{l} L_{j_h} + \Delta,$$

independent of the specific order of dispatching the messages. □

Note that Claim 2 verifies that optimal scattering-schedules for a message sequence $\mathcal{M}$ do not depend on the lengths of the messages in $\mathcal{M}$.

The Theorem follows. □

Let us focus momentarily on the simplest possible tree, namely, a path having PE $\mathcal{P}_0$ as its root. For notational convenience, say that in this tree: $\mathcal{P}_{i+1}$ is the child of $\mathcal{P}_i$, for $0 \leq i < n$; $\mathcal{P}_{i-1}$ is the parent of $\mathcal{P}_i$, for $0 < i \leq n$; and $\mathcal{P}_n$ is the (sole) leaf. When one is scattering messages from $\mathcal{P}_0$ in such a tree, the proof of Theorem 1 can be visualized easily. As one can see in Fig. 1, for instance, in this case, each message dispatched by PE $\mathcal{P}_0$ sweeps out a parallelogram in the space-time domain. (The parallelogram associated with the length-L_i message $\mathcal{M}_i$ destined for PE $\mathcal{P}_i$ has length-L_i sides parallel to the time axis, corresponding to the path traversed by the L_i flits of message $\mathcal{M}_i$, and length-i sides at a 45-degree angle to the time axis, corresponding to the progress of the flits along the line of PE's.) Constructing examples of scattering operations on paths, visualized via space-time parallelograms, will convince the reader that often a portion of the upper slanted sides of the space-time parallelogram of one message can be "hidden in the shadow" of the space-time parallelogram of an earlier dispatched message; this corresponds to pipelining the use of the intermediate PEs to decrease the overall time of the scatter operation. Constructing analogs of the competing dispatch orders of Fig. 1(a) and (b) will illustrate what Theorem 1 verifies, namely, that more hiding occurs when the parallelogram of a message destined for a more distant PE "provides shadow for" the parallelogram of a message destined for a nearby PE than when the dispatch times of the two messages are reversed. In Fig. 1, we make message $\mathcal{M}_4$ longer than message $\mathcal{M}_5$ to emphasize the independence of the "hiding" phenomenon from the lengths of messages.

Legend:

Processors: $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5$
Message Lengths: $L_1 = 0$, $L_2 = 0$, $L_3 = 0$, $L_4 = 4$, $L_5 = 3$
Message Notation: $\mathcal{M}_i = m_{i,1} m_{i,2} \cdots m_{i,L_i}$

(*a*) **Network traffic when $\mathcal{M}_5$ precedes $\mathcal{M}_4$:**

Step	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
0	$m_{5,1}$					
1	$m_{5,2}$	$m_{5,1}$				
2	$m_{5,3}$	$m_{5,2}$	$m_{5,1}$			
3	$m_{4,1}$	$m_{5,3}$	$m_{5,2}$	$m_{5,1}$		
4	$m_{4,2}$	$m_{4,1}$	$m_{5,3}$	$m_{5,2}$	$m_{5,1}$	
5	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$	$m_{5,3}$	$m_{5,2}$	$m_{5,1}$
6	$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$	$m_{5,3}$	$m_{5,2}$
7		$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$	$m_{5,3}$
8			$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	
9				$m_{4,4}$	$m_{4,3}$	
10					$m_{4,4}$	

(*b*) **Network traffic when $\mathcal{M}_4$ precedes $\mathcal{M}_5$:**

Step	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
0	$m_{4,1}$					
1	$m_{4,2}$	$m_{4,1}$				
2	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$			
3	$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$		
4	$m_{5,1}$	$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	$m_{4,1}$	
5	$m_{5,2}$	$m_{5,1}$	$m_{4,4}$	$m_{4,3}$	$m_{4,2}$	
6	$m_{5,3}$	$m_{5,2}$	$m_{5,1}$	$m_{4,4}$	$m_{4,3}$	
7		$m_{5,3}$	$m_{5,2}$	$m_{5,1}$	$m_{4,4}$	
8			$m_{5,3}$	$m_{5,2}$	$m_{5,1}$	
9				$m_{5,3}$	$m_{5,2}$	$m_{5,1}$
10					$m_{5,3}$	$m_{5,2}$
11						$m_{5,3}$

Fig. 1. Illustrating the effect of message order on the time for scattering on a path.

III. Gathering on Networks of Processors

Say that gathering-schedule γ is *optimal* for message sequence $\mathcal{M}$ on a given tree if, *on that tree,*

$$T^{\text{gath}}(\gamma; \mathcal{M}) \leq T^{\text{gath}}(\gamma'; \mathcal{M})$$

for any other gathering-schedule γ' for $\mathcal{M}$.

In an ideal world, we would implement the gathering operation by running an FDF scattering algorithm "backwards;" by reasoning analogous to that in the proof of Theorem 1, an algorithm that accomplished this would be optimal. Of course, one can not literally run an FDF scattering algorithm "backwards," because in the scattering operation, PE's other

than $\mathcal{P}_0$ are passive, while in the gathering operation, they are active: they must initiate their message transmissions. To compensate for this fact, any algorithm for a bufferless gathering operation must precede the transmission of messages by a *distributed protocol* that schedules the dispatch times of the messages so that no two collide in transit. A straightforward synchronization-like protocol suffices to accomplish this scheduling. We begin this section with a simple version of this protocol, called *shoulder tapping* (Section III-A), that implements the operation of gathering messages to one end of a path by interlacing the synchronization and scheduling activities. Although shoulder tapping yields an optimal algorithm for gathering on a path, it is too simple to work on general tree structures. Since altering shoulder tapping to operate on general trees leads to a cumbersome algorithm, we opt instead for a version of the protocol which decouples the synchronization and scheduling activities. The resulting protocol, called *transmission certification* (Section III-B), is readily adapted to general tree structures, but only at the cost of added time for separate synchronization and scheduling activities.

It is worth stressing here that gathering must in general be more time consuming than scattering, because of the need for a scheduling protocol that precedes message transmission. In particular, in a gathering operation, a PE cannot safely begin transmitting its message until "told to," for fear of interfering with the transit of another PE's message.

A. Shoulder Tapping: A Solution for Paths of Processors

The shoulder-tapping protocol we present now exploits the single-child structure of a path in an essential way; it is this feature that precludes its graceful extension to trees of more complicated structure. The algorithm that implements shoulder tapping seeks, for a message sequence $\mathcal{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n \rangle$, a gathering-schedule

$$\gamma = \tau_\gamma(i_1), \tau_\gamma(i_2), \cdots, \tau_\gamma(i_k)$$

where $i_j \in \boldsymbol{N}(\mathcal{M})$, which minimizes each dispatch time $\tau_\gamma(i_j)$ subject to the requirement that messages never collide, and subject to the inequalities

$$\tau_\gamma(i) \geq \begin{cases} i+2 & \text{if } i < n \\ i+1 & \text{if } i = n. \end{cases} \tag{5}$$

Inequalities (5) must hold for any distributed gathering algorithm on a path; they reflect the following facts, which hold for *all* PE indexes, not just those in $\boldsymbol{N}(\mathcal{M})$.

- Each PE $\mathcal{P}_i$ (save, of course, $\mathcal{P}_0$) must receive a *wakeup call* telling it when to begin transmitting its message $\mathcal{M}_i$ (assuming that the message is nonnull).
- The sequence of wakeup calls must be initiated by $\mathcal{P}_0$ (since, in general, it is the best arbiter of when it is ready to receive the message sequence), hence must take at least i steps to reach $\mathcal{P}_i$.
- The single-port communication regimen does not allow $\mathcal{P}_i$ to overlap dispatching its message (toward $\mathcal{P}_0$) and transmitting a wakeup call to $\mathcal{P}_{i+1}$.

The algorithm operates as follows. Each PE $\mathcal{P}_i$, where $i > 0$, remains dormant until its shoulder is tapped by PE $\mathcal{P}_{i-1}$ with a wakeup call: the call is a (one-flit) message consisting of the order

$$\text{TRANSMIT AFTER} \geq s_i \text{ STEPS}$$

where s_i is a positive integer. Assume that $\mathcal{P}_i$ receives its wakeup call at time t_i. It responds by serially entering the following operational phases, which embody Algorithm *Shoulder-Tap*.

Algorithm Shoulder-Tap

Phase 0: $\mathcal{P}_0$ transmits $\mathcal{P}_1$ the wakeup call

$$\text{TRANSMIT AFTER} \geq (s_1 = 1) \text{ STEPS}$$

Phase 1: If $i = n$, then this phase is ignored; else, if $i < n$, then: At time $t_i + 1$ (one step after receiving its wakeup call), $\mathcal{P}_i$ transmits to $\mathcal{P}_{i+1}$ a wakeup call of the form

$$\text{TRANSMIT AFTER} \geq s_{i+1} \text{ STEPS}$$

where the positive integer s_{i+1} is computed using the following time-line. (Note the effect of the single-port communication regimen.)

time t_i: $\mathcal{P}_i$ receives its wakeup call (from $\mathcal{P}_{i-1}$).
time$t_i + 1$: $\mathcal{P}_{i+1}$ receives its wakeup call from $\mathcal{P}_i$.
time $t_i + \max(2, s_i)$: $\mathcal{P}_{i-1}$ receives the first flit of $\mathcal{M}_i$ from $\mathcal{P}_i$ (when $L_i > 0$).
time $t_i + L_i + \max(1, s_i - 1)$: $\mathcal{P}_{i-1}$ receives the last (i.e., the L_ith) flit of $\mathcal{M}_i$ from $\mathcal{P}_i$; hence, at this time, $\mathcal{P}_i$ is ready to relay (passively) any messages it receives from PE's $\mathcal{P}_j$ for $j > i$.

Since $\mathcal{P}_{i+1}$ receives its wakeup call at time $t_i + 1$, and since s_{i+1} must be positive, $\mathcal{P}_i$ sets the value of s_{i+1} as follows:

$$s_{i+1} \leftarrow \max(1, L_i + \max(0, s_i - 2)).$$

Phase 2: If $L_i = 0$, then this phase is ignored; else, from time $t_i + \max(2, s_i)$ to time $t_i + L_i + \max(1, s_i - 1)$, $\mathcal{P}_i$ transmits message $\mathcal{M}_i$ to $\mathcal{P}_0$, via $\mathcal{P}_{i-1}$, one flit at a time.

Phase 3: From time $t_i + L_i + \max(1, s_i - 1)$ on, $\mathcal{P}_i$ begins to relay (passively) any messages it receives from PE's $\mathcal{P}_j$ for $j > i$. □

Two small instances of Algorithm *Shoulder-Tap* appear in Figs. 2 and 3. Fig. 2 attempts to depict a "typical" message sequence: Fig. 3 depicts a somewhat pathological sequence which illustrates that the dispatch times of messages under the algorithm may not be monotonic in the indexes of the dispatching PE's.

We show now that Algorithm *Shoulder-Tap* produces an optimal gathering-schedule for paths.

Theorem 2: Algorithm *Shoulder-Tap* is an optimal algorithm for gathering on a path.

Proof: Let us consider the behavior of Algorithm *Shoulder-Tap* on an arbitrary message sequence $\mathcal{M} = \langle \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n \rangle$.

Note first that when all of the messages in sequence $\mathcal{M}$ are nonnull, Algorithm *Shoulder-Tap* delivers the messages to $\mathcal{P}_0$ in a *gap-free* fashion. When $n = 1$, this transmission takes place from time–step 2 to time–step $1 + L_1$; when $n > 1$, this transmission takes place from time–step 3 to time–step 2 +

Legend:

Processors: $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5$
Message Lengths: $L_1 = 2$, $L_2 = 3$, $L_3 = 0$, $L_4 = 2$, $L_5 = 1$
Message Notation: $\mathcal{M}_i = m_{i,1}m_{i,2}\cdots m_{i,L_i}$
TRANSMIT AFTER $\geq s_i$ STEPS order: (s_i)

Network Traffic:

Step	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
1		(1)				
2		$m_{1,1}$	(2)			
3	$m_{1,1}$	$m_{1,2}$	$m_{2,1}$	(3)		
4	$m_{1,2}$	$m_{2,1}$	$m_{2,2}$		(1)	
5	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$		$m_{4,1}$	(2)
6	$m_{2,2}$	$m_{2,3}$		$m_{4,1}$	$m_{4,2}$	$m_{5,1}$
7	$m_{2,3}$		$m_{4,1}$	$m_{4,2}$	$m_{5,1}$	
8		$m_{4,1}$	$m_{4,2}$	$m_{5,1}$		
9	$m_{4,1}$	$m_{4,2}$	$m_{5,1}$			
10	$m_{4,2}$	$m_{5,1}$				
11	$m_{5,1}$					

Fig. 2. Gathering via shoulder-tapping on a six-node path: A "typical" example.

Legend:

Processors: $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5$
Message Lengths: $L_1 = 9$, $L_2 = 0$, $L_3 = 1$, $L_4 = 0$, $L_5 = 1$
Message Notation: $\mathcal{M}_i = m_{i,1}m_{i,2}\cdots m_{i,L_i}$
TRANSMIT AFTER $\geq s_i$ STEPS order: (s_i)

Network Traffic:

Step	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
1		(1)				
2		$m_{1,1}$	(9)			
3	$m_{1,1}$	$m_{1,2}$		(7)		
4	$m_{1,2}$	$m_{1,3}$			(6)	
5	$m_{1,3}$	$m_{1,4}$				(4)
6	$m_{1,4}$	$m_{1,5}$				
7	$m_{1,5}$	$m_{1,6}$				
8	$m_{1,6}$	$m_{1,7}$				$m_{5,1}$
9	$m_{1,7}$	$m_{1,8}$		$m_{3,1}$	$m_{5,1}$	
10	$m_{1,8}$	$m_{1,9}$	$m_{3,1}$	$m_{5,1}$		
11	$m_{1,9}$	$m_{3,1}$	$m_{5,1}$			
12	$m_{3,1}$	$m_{5,1}$				
13	$m_{5,1}$					

Fig. 3. Gathering via shoulder-tapping on a 6-node path: A "pathological" example.

$\Sigma_i L_i$. In this case, the Algorithm can clearly not be improved, since the small additive constant in excess of the message-stream length is needed for synchronization, as in inequality (5).

In order to establish the optimality of Algorithm *Shoulder-Tap* when some of the messages in sequence $\mathcal{M}$ are null, we introduce the following analogue of FDF scattering-schedules.

We have already remarked that the ideal gathering-schedule would be one that ran an FDF scattering-schedule "backwards." From the perspective of PE $\mathcal{P}_0$, as recipient of the messages, such a schedule would have messages that originate at nearby PE's arrive before messages that originate at more distant PE's, i.e., would observe a *nearest-received-first* (NRF) regimen. The formal verification that there is an optimal NRF gathering-schedule satisfying inequality (5) for every message sequence follows the lines of the analogous result for FDF scattering-schedules (Theorem 1), hence is left to the reader. In common with Theorem 1, this verification can be visualized geometrically when the underlying tree is a path. Messages in gathering operations sweep out the same type of parallelograms in the space-time domain as they do in scattering operations; the main difference is that gathering-parallelograms slant from the northeast to the southwest, whereas scattering-parallelograms slant from the northwest to the southeast (Fig. 2).

With no loss of generality, we henceforth compare Algorithm *Shoulder-Tap* only with gathering-schedules that honor the NRF regimen.

Consider, therefore, an arbitrary NRF gathering-schedule for $\mathcal{M}$,

$$\gamma = \tau_\gamma(i_1), \tau_\gamma(i_2), \cdots, \tau_\gamma(i_k)$$

where $i_j \in \boldsymbol{N}(\mathcal{M})$. For any $2 \leq j \leq k$, we must have

$$\tau_\gamma(i_j) \geq \tau_\gamma(i_{j-1}) + L_{i_{j-1}} + i_{j-1} - i_j \tag{6}$$

or else messages $\mathcal{M}_{i_j}$ and $\mathcal{M}_{i_{j-1}}$ would either collide or violate the NRF regimen. By combining inequalities (5) and (6), we obtain

$$\tau_\gamma(i_j) \geq \begin{cases} i_1 + 2 \\ \quad \text{if } j = 1 \text{ and } i_j \neq n \\ \max\left(i_j + 2, \tau_\gamma(i_{j-1}) + L_{i_{j-1}} + i_{j-1} - i_j\right) \\ \quad \text{if } j > 1 \text{ and } i_j < n \\ \max\left(n+1, \tau_\gamma(i_{j-1}) + L_{i_{j-1}} + i_{j-1} - n\right) \\ \quad \text{if } i_j = n. \end{cases} \tag{7}$$

A straightforward induction establishes that the gathering-schedule produced by Algorithm *Shoulder-Tap* satisfies inequality (7) *as an equality*. It follows that the gathering time for Algorithm *Shoulder-Tap* is minimal among algorithms for gathering on a path, that schedule message deliveries in a distributed fashion, hence obey inequality (5). □

Generalizing the interlaced synchronization-plus-message passing strategy of Algorithm *Shoulder-Tap* to trees whose PE's have multiple children seems to require a rather complicated protocol. Messages must have end-of-message delimiters so that each PE $\mathcal{P}_i$ can coordinate the message streams of its children and their descendants. We turn now to an alternative strategy which accomplishes this coordination in a simpler way, hence extends gracefully to trees of arbitrary structure.

B. Transmission Certification: A Solution for General Trees

We now modify the protocol of Algorithm *Shoulder-Tap* by decoupling the synchronization and message passing activities. The resulting Algorithm *Transmission-Certification* operates in four phases.

Algorithm Transmission-Certification

{The first two phases represent the decoupled synchronization part of the protocol.}

Phase 1: PE $\mathcal{P}_0$ "awakens" all other PE's in the tree by broadcasting a *synchronization token.* (This wakeup call lets the PE's know that $\mathcal{P}_0$ is ready to "gather" their messages.)

Phase 2: Each PE $\mathcal{P}_i$ responds to the synchronization token by sending a (one-flit) *transmission certificate* to its parent PE. The certificate indicates how soon $\mathcal{P}_i$ can initiate a *gap-free* transmission of all the messages in the subtree whose root it occupies. The PE's at the leaves of the tree are the first to send certificates; a nonleaf PE's certificate is computed using the length of its message, together with the certificates of its children.

{The second two phases are reminiscent of Algorithm *Shoulder-Tap.*}

Phase 3: When $\mathcal{P}_0$ receives its children's certificates, it initiates a wave of TRANSMIT–MESSAGE orders. Inductively, the orders transmitted by a PE $\mathcal{P}_i$ to its children schedule the children's gap-free transmissions: the scheduled dispatch time for each child is calculated from $\mathcal{P}_i$'s own dispatch time, its own message length L_i, and the certificates it received (during Phase 2) from its children.

Phase 4: Finally, the PE's follow the schedule of phase 3, transmitting messages in a gap-free stream toward $\mathcal{P}_0$, via their parents. □

Since $\mathcal{P}_0$ eventually receives the entire set of messages in a gap-free stream (of length $\Sigma_i L_i$), Algorithm *Transmission-Certification* is optimal, up to the time required for the synchronization-and-scheduling protocol. This protocol comprises three phases: two of the phases (Phases 1 and 3) are essentially broadcasts in the tree; the other (Phase 2) is essentially a leaf-to-root reverse broadcast, with children's messages being combined into a single message by each parent. We now describe these phases in detail.

Assume henceforth that each PE $\mathcal{P}_i$ which is not a leaf in the tree has d_i children, denoted $\mathcal{P}_{i,1}, \mathcal{P}_{i,2}, \cdots, \mathcal{P}_{i,d_i}$ in some arbitrary but fixed order.

Broadcasting and Receiving Messages: Because the single-port communication regimen allows a PE to communicate with at most two neighbors in a single step (one by sending a message and one by receiving a message), communications in the various phases of Algorithm *Transmission Certification* must be orchestrated as illustrated in the following scenario. When PE $\mathcal{P}_i$ receives a synchronization token "SEND–CERTIFICATE" from its parent, it relays the token in turn to its children, $\mathcal{P}_{i,1}, \mathcal{P}_{i,2}, \cdots, \mathcal{P}_{i,d_i}$. After sending the token to a child, $\mathcal{P}_i$ waits to receive the child's transmission certificate before sending the token to the next child. $\mathcal{P}_i$ continues in this fashion, until it has collected transmission certificates from all d_i children. The reader should note that the Algorithm requires $\mathcal{P}_i$ to "remember" which certificate came from which child.

An Overview of Transmission Certificates: During Phase 2 of the Algorithm, each PE $\mathcal{P}_i$ $(i > 0)$ sends its parent a *transmission certificate;* this message consists of a pair of integers (c_i, n_i), where $c_i > 0$ is the *certified lag time,* and $n_i \geq 0$ is the *certified stream length.* The intended interpretation of $\mathcal{P}_i$'s transmission certificate is: c_i steps after receiving a TRANSMIT–MESSAGE order, PE $\mathcal{P}_i$ can start transmitting toward $\mathcal{P}_0$ a *gap-free stream* of n_i flits, comprising all the messages originating at PE's in the subtree rooted at $\mathcal{P}_i$.

Each PE that is a leaf of the tree can compute its certificate directly from the length of its message; each nonleaf PE $\mathcal{P}_i$ computes its certificate from the length of its message, together with the certificates of its children. ($\mathcal{P}_i$ needs both the certified lag times and the certified stream lengths from its children for scheduling purposes, in order to coalesce the children's d_i message streams into a single stream.) When $\mathcal{P}_0$ receives the certificates from its children, it can proceed to schedule all the transmissions, using TRANSMIT–MESSAGE orders that are essentially identical to the shoulder taps that characterize Algorithm *Shoulder-Tap.* The transmission schedule produced by Algorithm *Transmission-Certification* differs from that produced by Algorithm *Shoulder-Tap* mainly in its avoidance of gaps in message transmission (such as that observed at Step 8 in Fig. 2). We now describe how the transmission certificates are computed.

Computing Transmission Certificates: Say that PE $\mathcal{P}_i$ has received the certificates

$$(c_{i,1}, n_{i,1}), (c_{i,2}, n_{i,2}), \cdots, (c_{i,d_i}, n_{i,d_i})$$

from its d_i children. It uses these certificates, plus the length L_i of its message, to compute its certificate (c_i, n_i), as follows:

Stream Length: The computation of $\mathcal{P}_i$'s certified stream length n_i is straightforward, since the message stream that $\mathcal{P}_i$ will transmit is just the concatenation of its message, $\mathcal{M}_i$, with the message streams of its children; hence,

$$n_i = L_i + \sum_{j=1}^{d_i} n_{i,j}.$$

Lag Time: A PE $\mathcal{P}_i$ that resides at a *leaf* of the tree does not have to wait for any other PE before starting to transmit its message stream, which is just its message $\mathcal{M}_i$; therefore, it can start transmitting its message stream with no gaps one step after receiving a TRANSMIT–MESSAGE order, so its certified lag time is just $c_i = 1$. In contrast, a PE $\mathcal{P}_i$ that is not at a leaf of the tree must consider how its message interacts with the message streams that will come from its children PE's. Specifically, PE $\mathcal{P}_i$ computes its certified lag time c_i from the certificates $c_{i,1}, c_{i,2}, \cdots, c_{i,d_i}$ of its d_i children, via the following reasoning, which is presented most easily by means of a time-line similar to that used to compute the wakeup calls in Algorithm *Shoulder-Tap.* Say that (at some time in the future) $\mathcal{P}_i$ will receive the order

TRANSMIT IN s_i STEPS

at time t. The following actions will ensue:

time $t + j$ (for $1 \leq j \leq d_i$): $\mathcal{P}_i$ will relay the order to its child $\mathcal{P}_{i,j}$, with an appropriately modified value $s_{i,j}$ of s_i.
time $t + s_i, \cdots, t + s_i + L_i - 1$: $\mathcal{P}_i$ will transmit message $\mathcal{M}_i$, as the first stage of transmitting the message stream from the PE's in the subtree rooted at $\mathcal{P}_i$. Note that the

integer s_i can be no smaller than $d_i + 1$, because of the single-port communication regimen.[3]

time $t + s_i + L_i$ on: $\mathcal{P}_i$ will begin to relay, without gaps, the message streams sent to it by its d_i children. Note that the integer s_i can be no smaller than $\min\{c_{i,j} | 1 \leq j \leq d_i\}$, because some child of $\mathcal{P}_i$ must begin its gap-free transmission one step before $\mathcal{P}_i$ begins its gap-free relaying. s_i may be larger than this lower bound because of the requirement that message transmission be gap free.

With this time–line in mind, $\mathcal{P}_i$ computes its certified lag time in four steps, as follows:

1) $\mathcal{P}_i$ adopts the *preliminary* certified lag time $c'_{i,0} =_{\text{def}} d_i + 1$; this acknowledges the fact that $\mathcal{P}_i$ cannot begin transmitting its message, $\mathcal{M}_i$, until it has dispatched a TRANSMIT–MESSAGE order to each of its children.

2) $\mathcal{P}_i$ "adjusts" each of its children's certified lag times, amending the lag time of $\mathcal{P}_{i,j}$, where $1 \leq j \leq d_i$, to $c'_{i,j} =_{\text{def}} d_i + c_{i,j}$; this acknowledges the fact that $\mathcal{P}_i$ cannot begin relaying its children's message streams until it has dispatched a TRANSMIT–MESSAGE order to each of its children.

3) $\mathcal{P}_i$ sorts the certified lag times $\{c_{i,j} | 1 \leq j \leq d_i\}$ of its children, thereby obtaining a permutation π of the set $\{1, 2, \cdots, d_i\}$ which orders the children of $\mathcal{P}_i$ in *increasing* order of their certified lag times. ($\mathcal{P}_i$ will use the permutation π now, in computing its certified lag time, and later, in computing the TRANSMIT–MESSAGE times for its children.)

4) Finally, $\mathcal{P}_i$ computes its certified lag time, using a geometrical model. Visualize the nonnegative x-axis, with the following $d_i + 1$ *movable* line segments $X_{i,j}$, where $0 \leq j \leq d_i$.

- $X_{i,0}$ is a length-L_i segment whose left endpoint can be placed anywhere at or to the right of point $c'_{i,0} = d_i + 1$.
- For $1 \leq j \leq d_i$, $X_{i,j}$ is a length- $n_{i,j}$ segment whose left endpoint can be placed anywhere at or to the right of point $c'_{i,j} = d_i + c_{i,j}$.

The intended interpretation is that the x-axis is the time axis, and each line segment represents the time interval during which the corresponding message stream is being transmitted by PE $\mathcal{P}_i$. Specifically, line segment $X_{i,0}$ represents the length-L_i time interval during which $\mathcal{P}_i$ transmits message $\mathcal{M}_i$; each other line segment $X_{i,j}$, for $1 \leq j \leq d_i$, represents the length-$n_{i,j}$ time interval during which $\mathcal{P}_i$ relays the message stream it receives from its jth child $\mathcal{P}_{i,j}$. The restrictions on the placements of the line segments are compatible with this interpretation: any line segment can be moved to the right, representing a delay in the transmission time of the corresponding message stream; no line segment can be moved to the left of its indicated limit (the points $c'_{i,k}$), for such a move would represent $\mathcal{P}_i$'s transmitting the corresponding message stream before the stream is available to it.

$\mathcal{P}_i$ now computes its certified lag time by shifting the line segments $X_{i,k}$ along the x-axis, moving segments rightward at will, but never moving any segment $X_{i,k}$ so that its left endpoint goes to the left of point $c'_{i,k}$, with the goal of combining all d_i segments (by concatenation) into a single line segment of length n_i, whose left endpoint is as small, i.e., as far to the left, as possible; call this combined line segment $X_i^\star$. The left endpoint of $X_i^\star$ is $\mathcal{P}_i$'s sought certified lag time c_i. Straightforward reasoning allows us to compute c_i explicitly

$$c_i = d_i + 1 + \max(0, c_{i,\pi(1)} - L_i) + \sum_{j=2}^{d_i} \max(0, c_{i,\pi(j)} - c_{i,\pi(j-1)} - n_{i,\pi(j-1)}).$$

Remark 4:

1) Combining the d_i+1 line segments into *a single* line segment $X_i^\star$ represents scheduling a *gap-free* transmission by $\mathcal{P}_i$ of all messages originating in its subtree.
2) Placing the line segment $X_i^\star$ as far to the left as possible (subject to the constraints of the points $c'_{i,k}$) represents an attempt to schedule $\mathcal{P}_i$'s transmission as early as possible.
3) If we denote by $c^\star_{i,k}$ the left endpoint of line segment $X_{i,k}$ within line segment $X_i^\star$, then the increasing sequence of values of the endpoints $c^\star_{i,k}$ represents a schedule for the gap-free transmission of the (combined) message streams of $\mathcal{P}_i$'s children.

To clarify the connection between moving line segments and scheduling messages, let us focus on just two segments: for $i = 1, 2$, say that line segment X_i has length n_i and left constraint c'_i. Say, moveover, that $c'_1 \leq c'_2$. Three cases arise.

1) If $c'_2 = n_1$, then both segments X_1 and X_2 can be positioned in their leftmost legal positions (namely, c'_1 and c'_2, respectively) and just juxtaposed to form segment $X^\star$. In this situation, the PE's associated with X_1 and X_2 can both honor their certified lag times.
2) If $c'_2 < n_1$, then segment X_1 can be positioned in its leftmost possible position (namely, c'_1), but segment X_2 must be shifted right $n_1 - c'_2$ positions before being juxtaposed with segment X_1 in order to form segment $X^\star$. This corresponds to having the PE associated with time interval X_2 delay its message transmission for $n_1 - c'_2$ time units so as not to interfere with the transmission by the PE associated with interval X_1.
3) If $c'_2 > n_1$, then segment X_2 can be positioned in its leftmost possible position (namely, c'_2), but segment X_1 must be shifted right $c'_2 - n_1$ positions before being juxtaposed with segment X_2 in order to form segment $X^\star$. This corresponds to having the PE associated with time interval X_1 delay its message transmission for $c'_2 - n_1$ time units so that the final transmission of messages will be free of gaps.

Remark 5: Because line segments start out in their leftmost feasible positions, one can combine them by moving line segments to the right but never to the left, i.e., by delaying message streams but never advancing one. This ensures that a single pass over the line segments, in decreasing order of their indices under the permutation π, suffices to produce line segment $X_i^\star$, hence to compute c_i.

The Message Scheduling Protocol: After PE $\mathcal{P}_0$ receives a transmission certificate from its last (i.e., d_0th) child, it spends the next d_0 steps sending TRANSMIT–MESSAGE orders to

[3] There is an implicit inductive assumption here that s_i has been assigned a feasible value by $\mathcal{P}_i$'s parent.

its children. Each order is a one-flit message of the form

TRANSMIT AFTER s STEPS

where the *transmission time* s is a positive integer; the intended interpretation is that, if a PE $\mathcal{P}$ receives the indicated order at time t, then it begins transmitting its (gap-free) message stream at time $t+s$; if $\mathcal{P}$ is a nonleaf PE, then it will begin this message transmission only after it has relayed to its children versions of the order with appropriately modified transmission times.[4] The issue we must focus on is how a PE ($\mathcal{P}_0$ or any other nonleaf PE) computes its children's transmission times. This computation can be described more uniformly if we imagine that $\mathcal{P}_0$ has received the (imaginary) order TRANSMIT AFTER 0 STEPS. Now we can say, uniformly, that nonleaf PE $\mathcal{P}_i$ receives the order TRANSMIT AFTER s_i STEPS at time t_i, and we can ask, uniformly, how $\mathcal{P}_i$ computes the transmission times $\{s_{i,j} | 1 \leq j \leq d_i\}$ for its children $\{\mathcal{P}_{i,j} | 1 \leq j \leq d_i\}$.

Computing Transmission Times: Say that $\mathcal{P}_i$ receives its transmission time s_i from its parent at time t_i. Earlier, when $\mathcal{P}_i$ computed its certified lag time c_i, it created a tentative transmission schedule for its children (and itself), which is embodied in the $d_i + 1$ start times $\{c^\star_{i,j} | 0 \leq j \leq d_i\}$. (Recall that these were computed while constructing the line segment $X^\star_i$.) Indeed, c_i is just the minimum of these values. The transmission time s_i can be viewed as just an adjustment to this tentative schedule, i.e., as a mandate to adjust the schedule by delaying it uniformly by $s_i - c_i$ time units (equivalently, by shifting $X^\star_i$ to the right $s_i - c_i$ units). Therefore, $\mathcal{P}_i$ assigns to each of its children $\mathcal{P}_{i,j}$, where $1 \leq j \leq d_i$, the transmission time $s_{i,j} = c^\star_{i,j} + s_i - c_i$ and sends it the order

TRANSMIT AFTER $s_{i,j}$ STEPS.

After dispatching all these orders, $\mathcal{P}_i$ proceeds to transmit, according to the schedule implicit in the set $\{s_i\} \cup \{s_{i,j} | 1 \leq j \leq d_i\}$, a gap-free stream containing all the messages in its subtree.

Timing Analysis: The time required by Algorithm *Transmission-Certification* is divided into four packets.

1) Broadcasting the synchronization token (Phase 1) and distributing the TRANSMIT–MESSAGE orders (Phase 3) each takes time essentially equal to the time B for a root-to-leaf broadcast in the tree.
2) The time C for collecting transmission certificates (Phase 2) is dominated by the accumulated time for sorting certified lag times at each PE along the left-to-root paths of the tree. This time is estimated as follows. Assign each leaf-PE the weight 0 and each nonleaf-PE having d children the weight $d \log_2 d$. Assign each root-to-leaf path a weight that is the sum of the weights of its nodes. Then C is a small multiple of the maximum weight of a root-of-leaf path.
3) Since message transmission (Phase 4) is gap-free, it requires time $M =_{\text{def}} \Sigma_i L_i$.

[4] When $\mathcal{P}$ computed its certified lag time, it included time for relaying orders to its children; hence, we can safely assume that s has been chosen large enough to allow time for this relaying.

Legend:

Processors: $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5$
Message Lengths: $L_1 = 2$, $L_2 = 3$, $L_3 = 0$, $L_4 = 2$, $L_5 = 1$
Message Notation: $\mathcal{M}_i = m_{i,1} m_{i,2} \cdots m_{i,L_i}$
Synchronization Token: [$\star$]
Certificate Notation: $< c_i >$
TRANSMIT AFTER s_i STEPS order: (s_i)

Network Traffic:

Step	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$
1		[$\star$]				
2			[$\star$]			
3				[$\star$]		
4					[$\star$]	
5						[$\star$]
6					< 1 >	
7				< 2 >		
8			< 4 >			
9		< 3 >				
10	< 3 >					
11		(3)				
12			(3)			
13		$m_{1,1}$		(4)		
14	$m_{1,1}$	$m_{1,2}$	$m_{2,1}$		(2)	
15	$m_{1,2}$	$m_{2,1}$	$m_{2,2}$		$m_{4,1}$	(2)
16	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{4,1}$	$m_{4,2}$	$m_{5,1}$
17	$m_{2,2}$	$m_{2,3}$	$m_{4,1}$	$m_{4,2}$	$m_{5,1}$	
18	$m_{2,3}$	$m_{4,1}$	$m_{4,2}$	$m_{5,1}$		
19	$m_{4,1}$	$m_{4,2}$	$m_{5,1}$			
20	$m_{4,2}$	$m_{5,1}$				
21	$m_{5,1}$					

Fig. 4. Gathering via transmission certificates on a six-node path.

Easily, *any* gathering algorithm must take at least $\max(B, M)$; in the worst case, this bound increases to $B+M$. To wit, synchronization must take at least B steps, and message transmission must take at least M steps, yielding the universal lower bound; if there is only one message in the sequence, and that message resides at a PE at maximum distance from $\mathcal{P}_0$, then these activities do not overlap. Summarizing this cost assessment, we arrive at the following theorem:

Theorem 3: The time for gathering on a tree using Algorithm *Transmission-Certification* is at most $2B + C + M$. The time for gathering on a tree using any algorithm is at least $\max(B, M)$; in the worst case, this lower bound increases to time $B + M$.

Fig. 4 illustrates the gathering operation of Fig. 2 performed using transmission certificates, rather than shoulder tapping.

As Figs. 2 and 4 indicate, gathering on an n-node ring via transmission certificates is materially slower (by roughly $2n$ steps) than gathering on the path via shoulder-tapping, the

extra time being accounted for by the explicit synchronization protocol. Although a portion of the synchronization time is recovered by the elimination of gaps in the transmission of the message stream, one would normally choose to use shoulder-taps rather than transmission certificates when gathering on a path.

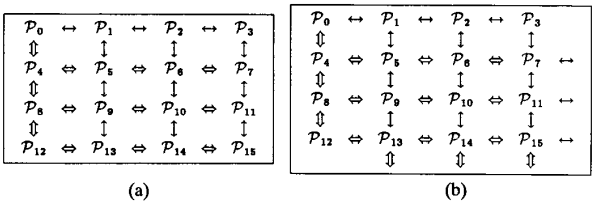


Fig. 5. (a) 4×4 mesh, covered by two jointly spanning edge–disjoint trees; (b) 4×4 toroidal mesh covered by two edge–disjoint spanning trees. The trees' edges are represented, respectively, by $\leftrightarrow$ and by $\Leftrightarrow$.

IV. Algorithms for a Multiport Model

We discuss only briefly how one can extend the gathering algorithms of Section III-B to a multiport communication regimen. Roughly speaking, one can proceed at two levels.

Parallelizing Synchronization: Most simply, in a network with a multiport communication capability, one can parallelize the three tasks in our algorithm that are dedicated to synchronization.

Parallelizing the broadcast of the synchronization token requires no modification of the algorithm.

In contrast, parallelizing the distribution of the TRANSMIT–MESSAGE orders may be tricky. Specifically, each such order has a transmission time associated with it, and each child of a given PE must receive a unique such time in order to insure collision-free message transmission in the absence of message buffers. It is not clear that one can save much time by parallelizing the transmission of the TRANSMIT–MESSAGE orders if the computation of the associated transmission times must be sequential.

Finally, parallelizing the computation of certificates is straightforward and, in fact, simplifies the algorithm by obviating the protocol whereby a PE orchestrates the receipt of certificates from its children.

Parallelizing Message Transmission: We discuss this topic in the context of scattering and gathering in arbitrary networks, via the use of spanning trees. There are two compelling techniques for parallelizing the transmission of messages in a network with a multiport communication capability. Both techniques involve "covering" the network with trees which then cooperate in transmitting the messages, using versions of the algorithms presented in previous sections.

The first technique advocates "covering" the network with mutually edge-disjoint trees rooted at PE $\mathcal{P}_0$, which collectively, though not necessarily individually, span the host network. Fig. 5 depicts two such "coverings." In Fig. 5(a), two trees jointly span the 4×4 mesh; Fig. 5(b) two trees each span the 4×4 toroidal mesh (i.e., the mesh with "wraparound" edges). These disjoint trees are then used just as described in previous sections. The only substantive change in the framework we have been discussing is the role that PE's play relative to each tree, if they belong to more than one. Most simply, each PE will be preallocated to one tree in which it will participate actively; the PE will act solely as a conduit in all other trees. Details can readily be filled in. One attractive feature of this technique is the availability of research in "covering" certain networks with edge–disjoint trees (though the requirement that $\mathcal{P}_0$ be the root of all the trees seems to complicate the problem materially); for instance, one readily shows that the mesh and de Bruijn networks can be so "covered," as can the hypercube [5], [6].

The second technique modifies the first by dropping the requirement that the "covering" trees be mutually edge disjoint. Adapting our algorithms to such a setting any be quite challenging, as one must schedule the traffic on the shared edges.

References

[1] M. Baumslag and F. S. Annexstein, "A unified approach to global permutation routing on parallel networks," *Math. Syst. Theory,* vol. 24, pp. 233–251, 1991.

[2] S. N. Bhatt, G. Bilardi, G. Pucci, A. Ranade, A. L. Rosenberg, and E. J. Schwabe, "On bufferless routing of variable length messages in leveled networks," in *1st European Symp. Algorithms,* 1993, to be published.

[3] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers,* vol. C-36, pp. 547–553, 1987.

[4] P. Fraigniaud, S. Miguet, and Y. Robert, "Complexity of scattering on a ring of processors," *Parallel Computing,* vol. 13, pp. 377–383, 1990.

[5] D. S. Greenberg, Full utilization of communication resources, Ph.D. dissertation, Yale Univ., New Haven, CT, 1991.

[6] D. S. Greenberg and S. N. Bhatt, "Routing multiple paths in hypercubes," *Math. Syst. Theory,* vol. 24, pp. 295–321.

[7] J. Hromkovic, C. D. Jeschke, and B. Monien, "Optimal algorithms for dissemination of information in some interconnection networks," in *Mathematical Foundations of Computer Science '90.* Also in *Lecture Notes in Computer Science 452.* New York: Springer-Verlag, 1990.

[8] S. L. Johnsson and C.-T. Ho, "Optimal broadcasting and personalized communication in hypercubes," *IEEE Trans. Computers,* vol. 38, pp. 1249–1268.

[9] G. Kortsarz and D. Peleg, "Approximation algorithms for minimum time broadcast," *Theory of Computing and Systems (ISTCS '92).* Also in *Lecture Notes in Computer Science 601,* New York: Springer-Verlag, 1992, pp. 67–78.

[10] M. Kunde and T. Tensi, "Multipacket routing on mesh-connected arrays," in *1st ACM Symp. Parallel Algorithms and Architectures,* 1989, pp. 336–343.

[11] J.-J. Li, "Multiscattering on a reconfigurable network of processors," L.I.P., Ecole Normale Supérieure de Lyon, Tech. Rep. 91-03, 1991.

[12] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers," *IEEE Trans. Computers,* vol. C-30, pp. 101–107, 1981.

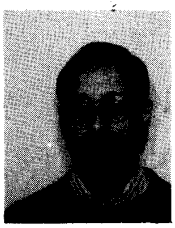
[13] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz, "Fully-adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks," in *3rd ACM Symp. Parallel Algorithms and Architectures,* 1991, pp. 278–290.

[14] Y. Saad and M. H. Schultz, "Data communication in parallel architectures," *Parallel Computing,* vol. 11, pp. 131–150, 1989.

[15] Q. F. Stout and B. Wagar, "Intensive hypercube communication, I: Prearranged communication in link-bound machines," *J. Parallel Distr. Comput.,* vol. 10, pp. 167–181, 1990.

[16] D. M. Topkis, "All-to-all broadcast by flooding in communication networks," *IEEE Trans. Computers,* vol. 38, pp. 1330–1333, 1989.

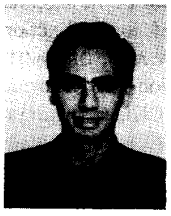
[17] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel computation," in *13th ACM Symp. Theory of Computing,* 1981, pp. 263–277.



Sandeep N. Bhatt received the S.B., S.M., and Ph.D., all in computer science, at the Massachusetts Institute of Technology, Cambridge, MA.

He was an Associate Professor of Computer Science at Yale University, New Haven, CT before joining Bellcore in 1992. During 1990, he was a Visiting Associate Professor of Computer Science at Caltech. His research focuses on algorithmic principles that underlie the design and use of parallel architectures. His contributions include graph embedding techniques to study problems in VLSI layout, to map computations onto parallel machines, and to understand the computational power and limitations of networks. His current work emphasizes algorthmic techniques to support high-level programming abstractions for irregular and adaptive scientific computations on parallel architectures.

Dr. Bhatt is a member of ACM and SIAM.



Geppino Pucci received the "Laurea" degree in Computer Science *summa cum laude* and the Ph.D. degree in Computer Science from the University of Pisa, Italy, in 1987 and 1992, respectively.

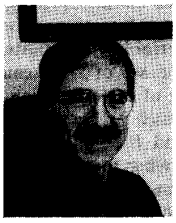
From 1988 to 1990 he was with the Computing Laboratory of the University of Newcastle-upon-Tyne, United Kingdom, where he conducted research in software reliability modeling. In 1991, he spent a six-month research period at the International Computer Science Institute, Berkeley, CA. In 1992, he joined the Dipartimento di Elettronica e Informatica of the University of Padova, Padova, Italy, as an Assistant Professor. His research interests include probabilistic modeling, analysis of parallel algorithms and theory of computation.

Dr. Pucci is a member of ACM.

Abhiram Ranade received the B.Tech in electrical engineering from the Indian Institute of Technology, Bombay, India in 1981 and the doctorate in Computer Science from Yale University, New Haven, CT in 1989.

Currently, he is an Assistant Professor of Electrical Engineering and Computer Science at the University of California, Berkeley. His research interests include parallel architectures and algorithms, parallel programming techniques and data structures.

Dr. Ranade is a member of ACM.



Arnold L. Rosenberg (SM'89) received the A.B., A.M., and Ph.D. degrees from Harvard University, Cambridge, MA.

He spent five years as a Professor of Computer Science at Duke University, Raleigh, NC and 16 years as a Research Staff Member at the IBM Watson Research Center. Additionally, he has held visiting or adjunct positions at New York University, the Polytechnic Institute of New York, the University of Toronto, and Yale University, and he has had short-term visiting positions at the Technion (Israel Institute of Technology) and at several European institutions. He currently is a Distinguished University Professor of Computer Science at the University of Massachusetts at Amherst. His current research focusses on theoretical aspects of parallel algorithms and architectures, with emphasis on: the use of algorithmic techniques to enhance the power of parallel architectures, the logical and physical mapping problems for parallel architectures, and the design of fault-tolerant architectures. He is the author of more than 100 technical papers on these and other topics in theoretical computer science and discrete mathematics.

Dr. Rosenberg is a member of ACM, EATCS, and SIAM.