Optimal Clustering of Tree-Sweep Computations for High-Latency Parallel Environments

Lixin Gao, *Member*, *IEEE*, Arnold L. Rosenberg, *Fellow*, *IEEE*, and Ramesh K. Sitaraman, *Member*, *IEEE*

Abstract—Modern hardware and software systems promote a view of parallel systems in which interprocessor communications are uniform and rather expensive in cost. Such systems demand efficient clustering algorithms that aggregate atomic tasks in a way that diminishes the impact of the high communication costs. We develop here a linear-time algorithm that *optimally* clusters computations that comprise a sequence of disjoint complete up- and/or down-sweeps on a complete binary tree for such parallel environments. Such computations include, for instance, those that implement broadcast, accumulation, and the parallel-prefix operator; such environments include, for instance, networks of workstations or BSP-based programming systems. The schedules produced by our clustering are optimal in the sense of having the *exact minimum* makespan—not just an approximation thereof—accounting for both computation and communication time. We show by simulation that the makespans of the schedules produced by our algorithm are close to half of those produced by the algorithm that yielded the best schedules previously known.

Index Terms—Scheduling, tree computations, clustering.

1 Introduction

1.1 Background

Modern hardware and software systems promote a view of parallel systems in which interprocessor communications are uniform and rather expensive in cost. Networks of workstations foster such a view because of the nature of the hardware implementing such systems [25], programming systems based on abstractions such as BSP [3], [20], [31], or LogP [2], [3], [8] foster such a view because of their underlying programming models. The theoretical "architecture-independent" model in [24] is well-suited for developing algorithms with such a view. Within such uniform, high-communication-cost environments, the *clustering* component of parallel scheduling—which aggregates atomic tasks for assignment to the same processor in a way that diminishes the impact of the high communication costs [12], [13], [28]—becomes increasingly important for the development of good schedules.

We focus throughout on computations that are "fine-grain," in the sense that each interprocessor communication costs more than the computation time for an atomic task. The complementary "coarse-grain" situation is far less challenging algorithmically.

Unfortunately, as we indicate in our survey of related work (in Section 1.3), even in the presence of simplifying assumptions, there is likely no efficient way to achieve optimal clusterings of tasks, as most such problems are NP-hard. The main result of the current paper is an efficient, *exactly optimal* clustering algorithm for a class of computations whose scheduling problem is close to the border between efficient and NP-hard problems. Specifically, we develop a linear-time algorithm for optimally clustering computations that comprise a sequence of disjoint complete up- and/or down-sweeps[1] on a complete binary tree.

As we discuss further in Section 1.2, the claimed optimality is within the realm of fine-grain computations, in multiport parallel environments having uniform interprocessor communication costs.

Tree-sweep computations can be used to implement operations and operators such as broadcast and reduction (or accumulation) (via single sweeps) and scan (or parallel-prefix) (via double sweeps).

The "tree-sweep" schedules produced by our clustering algorithm are *exactly* optimal in the sense of having *exactly* minimum makespans, not just constant-factor approximations thereof. Indeed, one can achieve *approximately* optimal clusterings—whose schedules have makespans within a factor of 2 of optimal—via algorithms that are significantly simpler than ours. To wit, let $\mathcal{T}_n$ denote the height-n complete binary tree (which has $2^n - 1$ nodes) and let τ denote the uniform cost of interprocessor communication. The reader can verify easily that the following algorithm achieves 2-approximate optimality. This algorithm is a straightforward generalization of a scheduling algorithm due to Papadimitriou and Yannakakis [24] in that this algorithm works for all values of τ, not just τ of the form $2^k - 1$.

- *L. Gao is with the Department of Computer Science, Smith College, Northampton, MA 01060. E-mail: gao@cs.smith.edu.*
- *A.L. Rosenberg and R.K. Sitaraman are with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003. E-mail: {rsnbrg; ramesh}@cs.umass.edu.*

Manuscript received 27 June 1997; revised 3 Aug. 1998.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 105296.

1. An "up-sweep" proceeds from the leaves to the root of the tree; a "down-sweep" proceeds from the root to the leaves.

Algorithm P.Y($\mathcal{T}_n$)
{Generate the **P.Y** schedule $\Sigma_{P.Y}$ for a complete binary tree $\mathcal{T}_n$}
begin
Let S denote a set of $\lfloor \tau + 1 \rfloor$ nodes that are closest to $\mathrm{root}(\mathcal{T}_n)$. (If $\mathcal{T}_n$ has fewer than $\lfloor \tau + 1 \rfloor$ nodes, S contains all nodes in the tree.)
Schedule all nodes of S in processor 0.
Schedule each tree in the forest induced by the nodes of $\mathcal{T}_n - S$ recursively, using a new set of processors for each.
end

Our research was motivated both by a desire to demarcate better the boundary between efficient scheduling problems and NP-hard ones, and by the recognition that even small factors (e.g., 2) can be critical in actual applications. The latter recognition led us to verify, via simulations we describe in Section 3.3, that the schedules produced by our algorithm are, in fact as well as theory, almost twice as fast as those produced by Algorithm **P.Y**, which yielded the best schedules known prior to our work.

In common with scheduling approaches, such as [33], we make no attempt to minimize the number of *virtual processors* our algorithm uses to achieve an optimal makespan. In consonance with the cluster-then-map scheduling strategy advocated in, e.g., [12], [13], [21], [28], we envision following our clustering algorithm with a virtual-to-physical processor mapping in order to complete the implementation of our algorithm on an actual parallel machine. We do not presently know how such mapping will affect the "virtual" optimality of our schedules, Importantly, though, one can employ an algorithmic device from [24] to convert our algorithm, at the cost of a factor of 2 in makespan, into one that employs precisely

$$\frac{\text{tree-size}}{\text{optimal makespan}}$$

processors. Also in common with the just-cited studies of the cluster-then-map strategy, we assume that our parallel computing platforms enjoy a *multiport* communication capability for both sending and receiving messages.

1.2 The Formal Framework

1.2.1 The Computational Load

Our work resides in the arena of scheduling algorithms for *directed acyclic graphs* (*dags*, for short) whose nodes represent uniform-size atomic tasks and whose arcs represent precedence constraints due to intertask dependencies. We focus in particular on complete binary[2] tree-dags, with arcs uniformly oriented either from the root to the leaves (an *out-tree* or *down-tree*) or from the leaves to the root (an *in-tree* or *up-tree*). The computations of interest comprise sequences of time-disjoint total sweeps up and/or down complete tree-dags; the orientations of a tree's arcs reflecting the direction of the current sweep. (We allow consecutive sweeps to be in opposing directions.) We focus on scheduling a single up-sweep, simply running the schedule "backwards" to obtain a schedule for a down-sweep. We lose some generality by using this approach since, while recomputation of nodes/tasks can never help on an up-sweep—hence will never appear in our optimal up-sweep schedules—they can help on down-sweeps. Therefore, whereas the up-sweep schedules produced by our algorithm are optimal (within the assumed computing environment) whether recomputation is allowed or not, the down-sweep schedules produced by our algorithm are optimal only among schedules that do not allow recomputation.

2. We concentrate on *binary* trees only for definiteness. Our algorithms adapt easily to complete trees of arbitrary fixed arities.

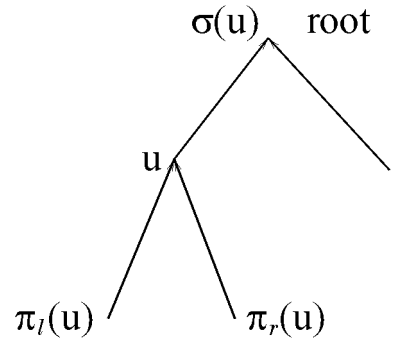


Fig. 1. The up-tree $\mathcal{T}_3$; $\mathrm{HGT}(u) = 2$.

Notation and terminology. For any up-tree $\mathcal{T}$: $\mathrm{SIZE}(\mathcal{T})$ is the number of nodes in $\mathcal{T}$, $\mathtt{root}(\mathcal{T})$ is its root, and $\mathcal{T}(u)$ is the (complete) subtree rooted at node u. Every nonroot node u of $\mathcal{T}$ has an edge to its unique *successor*, $\sigma(u)$; every nonleaf node u of $\mathcal{T}$ has two edges entering it, one from its *left predecessor*, $\pi_\ell(u)$, and one from its *right predecessor*, $\pi_r(u)$. See Fig. 1. The *height* of a node u of $\mathcal{T}$, denoted $\mathrm{HGT(u)}$, is defined recursively as follows: A leaf has height 1 and, inductively, $\mathrm{HGT}(\sigma(u)) = 1 + \mathrm{HGT}(u)$. All nodes of $\mathcal{T}$ that share the same height form a *level* of $\mathcal{T}$. The *height* of $\mathcal{T}$ is $\mathrm{HGT}(\mathtt{root}(\mathcal{T}))$.

1.2.2 The Computational Model

Our time unit throughout is the (common) execution time of each atomic task (= tree-node). In "wall-clock" terms, the (common) interprocessor communication delay τ is, thus, the ratio of communication-delay to computation-delay, hence, implicitly specifies the granularity of the computation. We focus only on *fine-grain* computations, for which $\tau > 1$.

Since we do not allow recomputation of tasks, every task is executed by precisely one processor at precisely one time; hence, specifying these processors and times completely characterizes a *schedule* Σ for a tree $\mathcal{T}$. Formally, then, a schedule Σ is an assignment of a unique processor $\mathrm{PROC}(u; \Sigma)$ and a unique *actual execution time* $\mathrm{AET}(u; \Sigma)$ to each node u of $\mathcal{T}$.

Clearly, a *valid* schedule must observe the constraints that are implicit in the dependencies of the tree and in the semantics of the architectural model. These two constraints take the following forms:

THE PROCESSOR LOAD CONSTRAINT. *A processor can execute at most one task at a time:* for distinct nodes u and v of $\mathcal{T}$, if $\text{PROC}(u;\Sigma) = \text{PROC}(v;\Sigma)$, then

$$|\text{AET}(u;\Sigma) - \text{AET}(v;\Sigma)| \geq 1.$$

THE PRECEDENCE CONSTRAINT. *A node of $\mathcal{T}$ cannot be executed before both of its predecessors have been executed and have had time to arrive at* $\text{PROC}(u;\Sigma)$. *Implicit in this constraint is that a leaf-node can be executed at any time.*

A more complicated approach to the Precedence Constraint is useful as we develop and verify our algorithm. Central to this approach is the *eligible-for-execution time*—often called the "ready time"—of task u under schedule Σ, denoted $\text{EET}(u;\Sigma)$.

- If u is a leaf, then $\text{EET}(u;\Sigma) = 0$.
- If u is not a leaf, then $\text{EET}(u;\Sigma)$ is computed as follows. First, add the three quantities:
  1. Quantity: $\text{AET}(\pi_\ell(u);\Sigma)$, which is the time when $\pi_\ell(u)$ is computed
  2. Quantity: 1, which is the time to compute $\pi_\ell(u)$
  3. Quantity: If $\text{PROC}(\pi_\ell(u);\Sigma) \neq \text{PROC}(u;\Sigma)$, then τ else 0, which is the time to transmit the result from $\pi_\ell(u)$

  Then, compute the corresponding sum for $\pi_\ell(u)$. $\text{EET}(u;\Sigma)$ is the larger of these two sums.[3]

The PRECEDENCE CONSTRAINT now takes the following form: *For every node u of $\mathcal{T}$,* $\text{AET}(u;\Sigma) \geq \text{EET}(u;\Sigma)$.

We assume henceforth, without explicit mention, that all given schedules are valid. (Of course, we prove explicitly that any schedules we produce are valid.)

The goal of our scheduling algorithms is to minimize the *makespan* of a computation, namely, the time for computing the entire tree. Since $\texttt{root}(\mathcal{T})$ is always the last-executed node in a valid schedule, the makespan of $\mathcal{T}$ under schedule Σ, denoted $\text{MKSPN}(\mathcal{T};\Sigma)$, is precisely $\text{AET}(\texttt{root}(\mathcal{T});\Sigma) + 1$. Our target is to achieve makespan $T^\star(\mathcal{T})$, which is the minimum makespan of any computation of tree $\mathcal{T}$:

$$T^\star(\mathcal{T}) \stackrel{\text{def}}{=} \min_{\text{valid } \Sigma} \text{MKSPN}(\mathcal{T};\Sigma).$$

In Sections 2 and 3 we develop, for any given up-tree $\mathcal{T}$, a schedule $\Sigma^\star$ that is *exactly optimal*, in the sense that $\text{MKSPN}(\mathcal{T};\Sigma^\star) = T^\star(\mathcal{T})$.

1.3 Relevant Prior Work

We briefly survey a number of sources in the literature that are directly relevant to our study. We refer the reader to [7], [22] for more complete surveys.

1.3.1 NP-Hard Scheduling Problems

It has been known for decades that the problem of constructing optimal schedules for dags is extremely hard, with even specialized cases falling within the class of NP-hard problems. The most general problem, optimally scheduling unrelated tasks that take arbitrary computation times on a given number of processors, is shown in [11, p. 238] to be NP-hard. Even when interprocessor communication time is negligible ($\tau = 0$) or commensurate with per-task computation time ($\tau = 1$), the problem of optimally scheduling a dag on a given number of processors is shown in [30] and [23], [27], respectively, to be NP-hard. Even when the number of processors is restricted to 2 and the dags are restricted to be binary trees, the problem of optimally scheduling for any given τ is NP-hard [1]. The difficulty of scheduling persists in many virtual-processor situations. If the dag to be scheduled is a tree of height ≤ 2, but each interprocessor communication is allowed to involve an arbitrary amount of data, then the problem of optimally scheduling the dag is NP-hard [6]. Even in the single-parameter "architecture-independent" model of [24], when one may use unboundedly many processors, the problem of optimally scheduling arbitrary dags is NP-hard [24]; in fact, even the problem of optimally scheduling binary trees is still NP-hard in this model [15].

1.3.2 Efficient Optimal Scheduling Algorithms

Prior to the algorithm we develop in the current paper, there have existed only the following few specialized scheduling algorithms that efficiently (i.e., in polynomial time) construct exactly optimal schedules. One finds in [16] an algorithm that produces optimal architecture-independent schedules for an arbitrary N-node dag within time $O(N^\tau)$; whenever τ is an absolute constant, this algorithm operates in polynomial time. In environments having only two processors, one can schedule any dag in linear time when $\tau = 0$ [9] and any forest in linear time when $\tau = 1$ [23]. Turning to tree-dags, linear-time algorithms suffice to produce optimal architecture-independent schedules for coarse-grain computations [5] and for complete trees within two-processor environments [1]. Finally, moving from the realm of scheduling algorithms to that of scheduling strategies, it is proven in [12] that coarse-grain dags can always be scheduled optimally via *linear clustering*—a technique that assigns all nodes on a dag's critical path to the same processor. (Linear clustering algorithms diverge on how to proceed after this first processor assignment [28].)

1.3.3 Approximately Optimal Scheduling Algorithms

The literature contains several polynomial-time algorithms that, at least in special cases, produce schedules that are provably within a constant factor of optimal. Among these, the most comprehensive is the architecture-independent algorithm of [24], which produces, for arbitrary dags and arbitrary τ, a schedule whose makespan is within a factor of 2 of optimal; this algorithm recomputes tasks when convenient. Also presented in [24] is the simple version of Algorithm **P.Y** that suffices when τ has the form $2^k - 1$ for some k that divides the height of the up-tree; recomputation cannot help when scheduling up-trees. There is a "greedy" algorithm for environments in which $\tau = 1$, which produces p-processor schedules for arbitrary dags, that are within a factor of $3 - 2/p$ of optimal [27]. (A scheduling algorithm is "greedy" if it never allows a processor to remain idle while some task is eligible for execution.) When interprocessor communication time is negligible ($\tau = 0$), there are a

3. Implicit in our evaluation of EET is the assumption that our parallel computing platforms observe a *multiport* communication protocol.

TABLE 1
Classes for Node v

u	$u \in C(\mathcal{T}_n;\Sigma)$?	u is a nonleaf, and $\{\pi_\ell(u), \pi_r(u)\} \subseteq C(\mathcal{T}_n;\Sigma)$?
Σ-*external*	NO	IMPOSSIBLE
Σ-*internal*	YES	YES
Σ-*boundary*	YES	NO

number of near-optimal scheduling algorithms. For general dags whose atomic tasks differ arbitrarily in complexity, there is a greedy algorithm whose schedules have makespan within a factor of 2 of optimal [14]. Moving from the realm of scheduling algorithms to that of scheduling strategies, one finds in [12] the sweeping result that *any* schedule for a coarse-grain dag that is produced by an algorithm that uses linear clustering has makespan within a factor of 2 of optimal. Next, one finds in [4] a nonconstructive strategy for scheduling N-node dags on p-processor machines with makespan $N/p + h$, where h is the length of the dag's critical path. This strategy, which has been implemented for numerous specific computational problems, easily produces schedules that are within a factor of 2 of optimal: To wit, N/p is the best possible speedup on a p-processor machine and h is the inherent sequential computation time of the dag. The literature on scheduling dags on multiprocessors with fixed topologies is rather sparse. One notable study is [29], which presents a polynomial-time algorithm for scheduling tree-dags on rings of processors. The resulting schedules are proven to be within a factor of 11 of optimal in makespan, but simulations show the deviation from optimality to be closer to 2 or 3.

1.3.4 Other Related Problems

Of course, there exist voluminous literatures on two classes of scheduling problems that are quite different from ours. First is the quest for (nearly or asymptotically) optimal schedules for specific functions, both in an architecture-independent setting (see, e.g., [18], [19]) and in a fixed-topology setting (see, e.g., [10], [17], [26]). Second is the simulation study of algorithms for scheduling classes of dags (see, e.g., [21], [28], [32]). None of these studies is really relevant to the current paper because of either focus or methodology.

We turn now to the scheduling algorithm that is our main contribution, laying the mathematical foundations in Section 2 and presenting and validating the algorithm in Section 3.

2 Clustering Fine-Grain Tree-Sweeps

This section is devoted to developing the theory that underlies the main contribution of the paper, the linear-time algorithm of Section 3 that produces exactly optimal schedules for fine-grain ($\tau > 1$) up-sweeps of a complete binary tree. In Sections 2.1-2.4, we develop the underpinnings of our main algorithm. We compile a list of five constraints on a scheduling algorithm and prove that we lose no generality by restricting our search for an optimal schedule to those that satisfy these constraints. In fact, we prove that there is a *unique* optimal schedule having the five properties. For definiteness, we assume throughout that we are scheduling the height-n complete binary tree $\mathcal{T}_n$; for succinctness, we henceforth abbreviate $T^\star(\mathcal{T}_n)$ by $T^\star(n)$.

2.1 The Five Underlying Properties

This section is devoted to specifying the five properties of tree-sweep schedules that help us narrow our search for an optimal schedule and that are enjoyed by precisely one optimal schedule.

Notation and terminology. Let Σ be a schedule for the complete binary tree $\mathcal{T}_n$. The *top-cluster* of $\mathcal{T}_n$ under Σ, denoted $C(\mathcal{T}_n;\Sigma)$, is the subtree of $\mathcal{T}_n$ comprising all nodes whose paths to $\texttt{root}(\mathcal{T}_n)$ contain no communication-edge. Note that all nodes in $C(\mathcal{T}_n;\Sigma)$ are executed in the same processor—with no loss of generality, processor 0. We use the top-cluster of schedule Σ to assign each node v of $\mathcal{T}_n$ to one of three classes. 1) v is Σ-*external* if it does not belong to $C(\mathcal{T}_n;\Sigma)$; 2) $v \in C(\mathcal{T}_n;\Sigma)$ is Σ-*internal* if both of its predecessors also belong to $C(\mathcal{T}_n;\Sigma)$; 3) v is a Σ-*boundary* node in all other cases. Table 1 codifies this classification.

When the specific schedule Σ in question is either irrelevant or clear from context, we omit the prefix "Σ-," thereby shortening the three node-classes to just "external node," "internal node," and "boundary node." By extension, we designate the subtree rooted at node u of $\mathcal{T}_n$ an *external subtree* of Σ if u is a Σ-external node and $\sigma(u)$ is a Σ-boundary node.

Henceforth, let us say that a schedule Σ' is *as fast as* a schedule Σ if, for all complete binary trees $\mathcal{T}$, $\textsc{Mkspn}(\mathcal{T};\Sigma') \leq \textsc{Mkspn}(\mathcal{T};\Sigma)$. By extension, schedule Σ' is *optimal* if it is as fast as any other schedule.

We now present the five properties that collectively prune our search space for optimal schedules.

External-subtree optimality (the ESO property). Our first property requires each external subtree of $\mathcal{T}_n$ to be scheduled exactly optimally. More formally,

Schedule Σ has the *external-subtree-optimal property* if it schedules all Σ-external subtrees optimally, employing a set of processors that are used nowhere else in the computation of $\mathcal{T}_n$.

Maximum-height boundaries (the MHB property). Our second property has a schedule "move" its boundary nodes as high as possible subject to the precedence constraints.

Schedule Σ has the *maximum-height-boundary property* if each Σ-boundary node u has

$$\text{AET}(u;\Sigma) \leq T^{\star}(\text{HGT}(u)) + \tau - 1.$$

Our informal description of the MHB property follows from this formal specification of the property for the following reason. For *any* valid schedule Σ, the precedence constraints imply that $T^{\star}(\text{HGT}(u) - 1) + \tau \leq \text{AET}(u;\Sigma)$. Thus, the MHB property requires that each boundary node u has the maximum allowable height allowed by its AET value and the precedence constraints.

Importantly, for our later analysis, any schedule Σ with the MHB property executes its boundary nodes in nondecreasing order of height: If the boundary nodes u and v satisfy $\text{HGT}(u) < \text{HGT}(v)$, then $\text{AET}(u;\Sigma) < \text{AET}(v;\Sigma)$.

Postorder execution (the PO property). The PO property requires a schedule to execute the nodes of its top-cluster in order of their positions in the postorder linearization of the input tree. In other words:

For any $u, v \in C(\mathcal{T}_n;\Sigma)$, if u precedes v in the postorder linearization of $\mathcal{T}_n$, then $\text{AET}(u;\Sigma) < \text{AET}(v;\Sigma)$.

Eagerness. A schedule Σ is *eager* if it executes each top-cluster node as soon as possible, while retaining the same set of top-cluster nodes and their order of execution. This means formally that, for each $v \in C(\mathcal{T}_n;\Sigma)$:

$$\text{AET}(v;\Sigma) = \begin{cases} \text{EET}(v;\Sigma), \\ \quad \text{if } v \text{ is the first node executed by processor 0} \\ \max\{\text{AET}(u;\Sigma) + 1,\ \text{EET}(v;\Sigma)\}, \\ \quad \text{otherwise, where } u \text{ is the top-cluster node} \\ \quad \text{that is executed immediately before } v. \end{cases}$$

Persistence. Our final property requires a schedule to execute top-cluster nodes "almost" consecutively, with no large gaps. For each node $u \in C(\mathcal{T}_n;\Sigma)$, we define

$$gap(v;\Sigma) \stackrel{def}{=} \begin{cases} \text{AET}(v;\Sigma), \\ \quad \text{if } v \text{ is the first node executed by processor 0} \\ \text{AET}(v;\Sigma) - \text{AET}(u;\Sigma) - 1, \\ \quad \text{otherwise, where } u \text{ is the top-cluster node} \\ \quad \text{executed immediately before } v. \end{cases}$$

Schedule Σ is *persistent* if for any top-cluster node $v \in C(\mathcal{T}_n;\Sigma)$, $gap(v;\Sigma) < 1$. This means that processor 0 is always busy during the execution of $\mathcal{T}_n$, from time 0 to time $\text{MKSPN}(\mathcal{T}_n;\Sigma)$, except possibly for a fraction of one unit idle time.

We call a gap of size ≥ 1 *large*.

2.2 A Simplifying Lemma

We now present a lemma that materially simplifies the proofs of the lemmas that allow us to focus on schedules that enjoy the five properties of Section 2.1.

Proposition 2.1. *One can transform any schedule Σ to a schedule Σ':*

- *which is as fast as Σ,*
- *for which both predecessors of each Σ'-boundary node are Σ'-external nodes,*
- *which enjoys the ESO, MHB, and PO properties whenever Σ does.*

Proof. We obtain schedule Σ' from Σ as follows: We convert each nonexternal predecessor of each Σ-boundary node to an external node (by changing its incident edge to a communication-edge). We then schedule all newly created external subtrees optimally, employing a set of processors that are used nowhere else in the computation of $\mathcal{T}_n$. Tree-nodes that are not rescheduled in this way retain both their processor assignments and AET's from Σ.

The validity of schedule Σ' is argued as follows:

- Each Σ'-internal node has the same and AET as it had under Σ.
- Each Σ'-external node is rescheduled via a valid schedule.
- Each Σ'-boundary node u still observes the precedence constraint since

$$\begin{aligned} \text{AET}(u;\Sigma') &= \text{AET}(u;\Sigma) \stackrel{\star}{\geq} (\text{HGT}(u) - 1) + \tau \\ &= \text{EET}(u;\Sigma'). \end{aligned}$$

The preservation of properties is argued as follows:

- Each new external subtree is scheduled optimally by Σ', using new processors.
- All top-cluster nodes of Σ' are executed in the same order as they are under Σ.
- Each Σ'-boundary node was also a Σ-boundary node.

Finally, since $\text{AET}(\texttt{root}(\mathcal{T}_n);\Sigma') = \text{AET}(\texttt{root}(\mathcal{T}_n);\Sigma)$, we conclude that schedule Σ' is as fast as schedule Σ. □

We turn now to our series of enabling lemmas.

2.3 Endowing Schedules with the Five Properties

2.3.1 External-Subtree-Optimal Schedules

Lemma 2.1. *One can transform any schedule Σ into a schedule $eso(\Sigma)$ that is as fast as Σ and that enjoys the ESO property.*

Proof. We obtain schedule $eso(\Sigma)$ from schedule Σ by rescheduling (if necessary) each external subtree optimally, employing a set of processors that are used nowhere else in $\mathcal{T}_n$. Any node of $\mathcal{T}_n$ that is not so rescheduled retains both its and AET from Σ.

The validity of schedule $eso(\Sigma)$ is argued as follows:

- Each $eso(\Sigma)$-internal node has the same and AET as it had under Σ.
- Each $eso(\Sigma)$-external node is rescheduled via a valid schedule.
- Finally, focus on an $eso(\Sigma)$-boundary node u. Letting v denote an external predecessor of u, the fact that $eso(\Sigma)$ schedules external subtrees optimally implies that $\text{AET}(v;eso(\Sigma)) \leq \text{AET}(v;\Sigma)$. It follows that

$$\begin{aligned} \text{AET}(u;eso(\Sigma)) &= \text{AET}(u;\Sigma) \geq \text{EET}(u;\Sigma) \\ &\geq \text{EET}(u;eso(\Sigma)). \end{aligned}$$

By construction, $eso(\Sigma)$ enjoys the ESO property. Finally, since $\texttt{root}(\mathcal{T}_n)$ is in the top-cluster of every schedule, we have $\text{AET}(\texttt{root}(\mathcal{T}_n); eso(\Sigma)) = \text{AET}(\texttt{root}(\mathcal{T}_n); \Sigma)$ so that $eso(\Sigma)$ is as fast as Σ. □

Lemma 2.1 allows us to concentrate henceforth on the problem of identifying and assigning AET's to nodes in the top-cluster of $\mathcal{T}_n$.

2.3.2 Maximum-Height-Boundary Schedules

Lemma 2.2 *One can transform any schedule Σ that enjoys the ESO property into a schedule $\text{MHB}(\Sigma)$ that is as fast as Σ and that enjoys both the ESO and MHB properties.*

Proof. By Proposition 2.1, we may assume that both predecessors of each Σ-boundary node are external. We construct schedule $\text{MHB}(\Sigma)$ from schedule Σ via a finite sequence of applications of a transformation Π_{MHB} that eliminates violations of the MHB property. We thereby produce a sequence of schedules,

$$\Sigma = \Sigma_0, \Sigma_1, \ldots, \Sigma_k = \text{MHB}(\Sigma),$$

each as fast as its predecessor and each retaining the ESO property; moreover, each successive Σ_{i+1} represents "progress" toward $\text{MHB}(\Sigma)$ in the following sense. Either Σ_{i+1} enjoys the MHB property, in which case $\Sigma_{i+1} = \text{MHB}(\Sigma)$, or Σ_{i+1} has a strictly smaller top-cluster than Σ_i. Of course, the former contingency must hold eventually. We now describe a single application of transformation Π_{MHB}, which produces schedule Σ_{i+1} from the non-MHB schedule Σ_i.

Let u be a Σ_i-boundary node that violates the MHB property so that $\text{AET}(u; \Sigma_i) > T^\star(\text{HGT}(u)) + \tau - 1$. We distinguish two cases.

Assume first that $u = \texttt{root}(\mathcal{T}_n)$. Since both predecessors of u are external under Σ_i, we have $C(\mathcal{T}_n; \Sigma_i) = \{\texttt{root}(\mathcal{T}_n)\}$. Construct Σ_{i+1} by assigning $\text{AET}(u; \Sigma_{i+1}) := T^\star(\text{HGT}(u) - 1) + \tau$ and by having Σ_{i+1} inherit all other PROCs and AETs from Σ_i. Now, Σ_{i+1} is valid by construction since $\text{AET}(u; \Sigma_{i+1}) = \text{EET}(u; \Sigma_{i+1})$. Clearly, Σ_{i+1} enjoys the ESO property by inheritance from Σ_i. Additionally, we have

$$\text{AET}(u; \Sigma_{i+1}) = T^\star(\text{HGT}(u) - 1) + \tau \ \leq^\star (\text{HGT}(u)) + \tau - 1.$$

In the last inequality, we use the elementary fact that for all $n > 1$, $T^\star(n) \geq T^\star(n-1) + 1$. This is true because, for any valid schedule Σ, $\text{EET}(\texttt{root}(\mathcal{T}_n); \Sigma) \geq T^\star(n-1)$, so that $\text{MKSPN}(\mathcal{T}_n; \Sigma) \geq T^\star(n-1) + 1$. Thus, Σ_{i+1} enjoys the MHB property. Finally, since we have assigned u a smaller AET than it had under Σ_i (in fact, we have assigned it the smallest valid AET), we have $\text{MKSPN}(\mathcal{T}_n; \Sigma_{i+1}) < \text{MKSPN}(\mathcal{T}_n; \Sigma_i)$.

Assume next that $u \neq \texttt{root}(\mathcal{T}_n)$. We transform Σ_i to Σ_{i+1} by making node $\sigma(u)$ a Σ_{i+1}-boundary node and by making both predecessors of $\sigma(u)$ Σ_{i+1}-external. We then schedule the new external subtrees rooted at $(\sigma(u))$ and $(\sigma(u))$ (one of which is node u) optimally, employing a new set of processors for each. We have Σ_{i+1} inherit all other assignments from Σ_i. Note first that Σ_{i+1} is a valid schedule. To wit, all nodes other than $\sigma(u)$ satisfy the precedence constraint either by construction or by inheritance from Σ_i; and node $\sigma(u)$ satisfies the precedence constraint from the fact that:

$$\begin{aligned}\text{AET}(\sigma(u); \Sigma_{i+1}) &= \text{AET}(\sigma(u); \Sigma_i) \\ &\geq \text{AET}(u; \Sigma_i) + 1 \\ &> T^\star(\text{HGT}(u)) + \tau - 1 + 1 \\ &= \text{EET}(\sigma(u); \Sigma_{i+1}).\end{aligned}$$

Additionally, Σ_{i+1} enjoys the ESO property by construction and inheritance from Σ_i. Moreover, Σ_{i+1} has fewer nodes in its top-cluster than does Σ_i, because $u \in C(\mathcal{T}_n; \Sigma_i) - C(\mathcal{T}_n; \Sigma_{i+1})$. Finally, since we have not modified the AET of $\texttt{root}(\mathcal{T}_n)$, we have

$$\text{MKSPN}(\mathcal{T}_n; \Sigma_{i+1}) = \text{MKSPN}(\mathcal{T}_n; \Sigma_i).$$

In both of the enumerated cases, transformation Π_{MHB} preserves the validity and the external-subtree optimality of its initial schedules; moreover, the transformation can speed up a schedule but can never slow it down. Finally, the transformation either results in a MHB schedule, or it decreases the size of the top-cluster. Clearly, then, a finite sequence of applications of Π_{MHB} will ultimately produce the desired schedule $\text{MHB}(\Sigma)$. □

2.3.3 Postorder Schedules

Lemma 2.3. *One can transform any schedule Σ that enjoys the ESO and MHB properties into a schedule $\text{PO}(\Sigma)$ that is as fast as Σ and that enjoys the ESO, MHB, and PO properties.*

Proof. First, note that we can simplify our task by ignoring the MHB property. To wit, if we can transform Σ into a schedule $\text{PO}(\Sigma)$ that is as fast as Σ and that enjoys the ESO and PO properties, then we can invoke Lemma 2.2 to endow $\text{PO}(\Sigma)$ with the MHB property without jeopardizing the PO property—because transformation Π_{MHB} does not modify the AET's of top-cluster nodes. So, we concentrate on constructing schedule $\text{PO}(\Sigma)$ from schedule Σ.

Creating $C(\mathcal{T}_n; \text{PO}(\Sigma))$. As before, Proposition 2.1 allows us to assume that for each Σ-boundary node u, all nodes in subtree $\mathcal{T}(u)$ are external except u. We specify $C(\mathcal{T}_n; \text{PO}(\Sigma))$ to be a subtree of $\mathcal{T}_n$ such that:

- $C(\mathcal{T}_n; \text{PO}(\Sigma))$ and $C(\mathcal{T}_n; \Sigma)$ have equally many nodes at each level;
- The nodes of $C(\mathcal{T}_n; \text{PO}(\Sigma))$ at each level are the leftmost nodes of $\mathcal{T}_n$ at that level.

Let $\widetilde{N}_\ell$ represent the number of nodes at height ℓ of $C(\mathcal{T}_n; \Sigma)$. We construct $C(\mathcal{T}_n; \text{PO}(\Sigma))$ by choosing the $\widetilde{N}_\ell$ leftmost nodes at each height ℓ of $\mathcal{T}_n$. This is possible because, for all $\ell < n$, $\widetilde{N}_\ell \leq 2\widetilde{N}_{\ell+1}$. Not only do $C(\mathcal{T}_n; \Sigma)$ and $C(\mathcal{T}_n; \text{PO}(\Sigma))$ have equally many nodes of each height, they also have equally many *boundary* nodes of each height. Since Σ-boundary nodes have no predecessors in the top-cluster, the same applies to Σ_{PO}-boundary

nodes. Therefore, the number of boundary nodes at height ℓ in either top-cluster equals $\widetilde{N}_\ell - \widetilde{N}_{\ell-1}/2$.

Scheduling $C(\mathcal{T}_n; \text{PO}(\Sigma))$. Since we design $\text{PO}(\Sigma)$ to enjoy the ESO property, we need explicitly schedule only the nodes of $C(\mathcal{T}_n; \text{PO}(\Sigma))$. First, we establish a one-to-one correspondence that associates a unique Σ-boundary node $\kappa(u)$ with each $\text{PO}(\Sigma)$-boundary node u. We begin by ordering the $\text{PO}(\Sigma)$-boundary nodes by height, from smallest to largest, breaking ties by left-to-right order. Next, we order the Σ-boundary nodes by AET, also from smallest to largest. We create the correspondence κ by pairing the elements of these two lists seriatim. Note that u and $\kappa(u)$ have the same height since $\text{PO}(\Sigma)$ and Σ have equally many boundary nodes of each height and since the MHB property mandates that boundary nodes be executed in nondecreasing order of height.

We visit the nodes of $C(\mathcal{T}_n; \text{PO}(\Sigma))$ in postorder fashion to assign AETs. Let $v \in C(\mathcal{T}_n; \text{PO}(\Sigma))$ be the node we are visiting currently and let u be the node visited immediately before v. Then we assign

$$\text{AET}(v; \text{PO}(\Sigma)) := \begin{cases} \max\{\text{AET}(u; \text{PO}(\Sigma)) + 1,\ \text{AET}(\kappa(v); \Sigma)\}, \\ \qquad \text{if } v \text{ is a } \text{PO}(\Sigma)\text{-boundary node} \\ \text{AET}(u; \text{PO}(\Sigma)) + 1, \\ \qquad \text{otherwise.} \end{cases}$$

We note first that schedule $\text{PO}(\Sigma)$ is valid because each internal node observes the precedence constraint from the postorder construction and each boundary node v inherits validity from the corresponding boundary node $\kappa(v)$ in Σ. Next, we note that, by construction, $\text{PO}(\Sigma)$ enjoys both the ESO and PO properties. Finally, we prove via two cases that $\text{PO}(\Sigma)$ is as fast as Σ.

Assume first that $\text{PO}(\Sigma)$ keeps processor 0 busy at every time-step, starting at time 0. It is immediate, then, that $\text{MKSPN}(\mathcal{T}_n; \text{PO}(\Sigma)) = |C(\mathcal{T}_n; \text{PO}(\Sigma))|$. In contrast, $\text{MKSPN}(\mathcal{T}_n; \Sigma)$ is *no smaller than* $|C(\mathcal{T}_n; \Sigma)|$. Since our construction of $\text{PO}(\Sigma)$ guarantees that

$$|C(\mathcal{T}_n; \text{PO}(\Sigma))| = |C(\mathcal{T}_n; \Sigma)|,$$

it follows that $\text{MKSPN}(\mathcal{T}_n; \text{PO}(\Sigma)) \leq \text{MKSPN}(\mathcal{T}_n; \Sigma)$.

Alternatively, say that $\text{PO}(\Sigma)$ does not keep processor 0 busy at every time-step. Let v be the last node that has $gap(v; \text{PO}(\Sigma)) > 0$ (so that processor 0 is busy at every time-step from time $\text{AET}(v; \text{PO}(\Sigma))$). From the construction of $\text{PO}(\Sigma)$, v is a $\text{PO}(\Sigma)$-boundary node and $\text{AET}(v; \text{PO}(\Sigma)) = \text{AET}(\kappa(v); \Sigma)$. Since there is no gap between consecutively executed nodes' AETs after v, $\text{MKSPN}(\mathcal{T}_n; \text{PO}(\Sigma))$ is the sum of $\text{AET}(v; \text{PO}(\Sigma))$ and the number of nodes executed from v to $\texttt{root}(\mathcal{T}_n)$ under schedule $\text{PO}(\Sigma)$. From similar considerations, $\text{MKSPN}(\mathcal{T}_n; \Sigma)$ is *no smaller than* the sum of $\text{AET}(\kappa(v); \Sigma)$ and the number of nodes executed from $\kappa(v)$ to $\mathcal{T}\mathit{troot}(\mathcal{T}_n)$ under schedule Σ. Since

$$|C(\mathcal{T}_n; \Sigma_{\text{PO}})| = |C(\mathcal{T}_n; \Sigma)|,$$

if we can prove that $\text{PO}(\Sigma)$ executes at least as many top-cluster nodes before v as Σ does before $\kappa(v)$, then we shall have proven that $\text{PO}(\Sigma)$ is as fast as Σ. We turn now to this task. We are guided by the intuition that $\text{PO}(\Sigma)$ is "greedy," in the sense that it schedules as many internal nodes as possible before scheduling the next boundary node. To obtain a formal argument, we first establish an upper bound on the number of top-cluster nodes that are executed before $\kappa(v)$ under Σ. For each i, the number of height-i top-cluster nodes that are executed before $\kappa(v)$ under Σ is no greater than the number of top-cluster nodes made eligible for execution by the boundary nodes executed before $\kappa(v)$. Say that N_ℓ height-ℓ boundary nodes are executed before $\kappa(v)$ under Σ; note that the same number of height-ℓ boundary nodes are executed before v under $\text{PO}(\Sigma)$. The nodes executed before $\kappa(v)$ under Σ can render eligible for execution no more than

$$n_i \stackrel{def}{=} \left\lfloor \sum_{l=1}^{i} N_l 2^{l-1}/2^{i-1} \right\rfloor$$

height-i top-cluster nodes: To wit, each height-i node has 2^{i-1} leaves below it and a node is eligible for execution only if all of its leaves belong to subtrees rooted at boundary nodes executed before $\kappa(v)$. It is easy to check that exactly n_i height-i top-cluster nodes are executed before v under $\text{PO}(\Sigma)$. Summing up top-cluster nodes executed at heights $1, 2, \ldots, n$, we conclude that at least as many nodes are executed before v under $\text{PO}(\Sigma)$ as are executed before $\kappa(v)$ under Σ. The lemma follows. □

2.3.4 Eager Schedules

Lemma 2.4. *One can transform any schedule Σ that enjoys the ESO, MHB, and PO properties into an eager schedule $\text{EAGER}(\Sigma)$ that is as fast as Σ and that enjoys the same properties.*

Proof. We construct schedule $\text{EAGER}(\Sigma)$ from schedule Σ via a sequence of applications of a transformation Π_{EAGER} that produces a sequence of schedules,

$$\Sigma = \Sigma_0, \Sigma_1, \ldots, \Sigma_k = \text{EAGER}(\Sigma),$$

each retaining all of the salient properties and each as fast as Σ; moreover, each Σ_{i+1} will enjoy the following additional property. Either Σ_{i+1} is eager, in which case, $\Sigma_{i+1} = \text{EAGER}(\Sigma)$, or the first node that violates eagerness in Σ_{i+1} is farther along the postorder linearization of $\mathcal{T}_n$ than is the first violating node in Σ_i.

Let node v be the first node in the PO linearization of $\mathcal{T}_n$ that violates eagerness. Let u be the node in $C(\mathcal{T}_n; \Sigma_i)$ that schedule Σ_i executes immediately before v. Transformation Π_{EAGER} eliminates this violation of eagerness by decreasing the AET of v and otherwise leaving schedule Σ_i unchanged. This results in having schedule Σ_{i+1} agree with Σ_i in all respects, except for the AET of node v, which is set to

$$\text{AET}(v, \Sigma_{i+1}) := \max\{\text{EET}(v; \Sigma_i),\ \text{AET}(u; \Sigma_i) + 1\}.$$

Easily, schedule Σ_{i+1} is as fast as schedule Σ_i. Moreover, Σ_{i+1} inherits from Σ_i both validity and the ESO, PO, and MHB properties. Finally, the earliest violation of eagerness in schedule Σ_{i+1} appears—if at all—further along the postorder linearization of $\mathcal{T}_n$ than node v. Transfor-

mation Π_{EAGER} thus steadily pushes violations of eagerness farther along the postorder linearization of the finite tree $\mathcal{T}_n$; it must, therefore, ultimately produce the schedule $\text{EAGER}(\Sigma)$. □

2.3.5 Persistent Schedules

Lemma 2.5. *One can transform any schedule Σ that enjoys the ESO, MHB, PO, and eagerness properties into a persistent schedule $\text{PERS}(\Sigma)$ that is as fast as Σ and that enjoys the same properties.*

Proof. In the presence of a large gap, we transform Σ to the desired persistent schedule $\text{PERS}(\Sigma)$ via a finite sequence of applications of a transformation Π_{PERS} that produces a sequence of schedules,

$$\Sigma = \Sigma_0, \Sigma_1, \ldots, \Sigma_k = \text{PERS}(\Sigma),$$

each retaining the ESO, MHB, PO, and eagerness properties, and each having the following additional property. Either Σ_{i+1} is persistent, in which case $\Sigma_{i+1} = \text{PERS}(\Sigma)$, or $|C(\mathcal{T}_n; \Sigma_{i+1})| = |C(\mathcal{T}_n; \Sigma_i)| + 1$. Since the top-cluster cannot grow in size indefinitely, we are guaranteed to terminate with a persistent schedule having no large gaps, which enjoys the other four salient properties.

We now describe and validate a single application of transformation Π_{PERS}, which produces schedule Σ_{i+1} from schedule Σ_i. Find the first node v in the postorder linearization of $\mathcal{T}_n$ that is in $C(\mathcal{T}_n; \Sigma_i)$ and that has a large gap, i.e., $gap(v; \Sigma_i) \geq 1$. First, we note that Σ_i's eagerness ensures that v is a Σ_i-boundary node. Second, we claim that v is not a leaf node. If it were, then we would have $\text{EET}(v; \Sigma_i) = 0$. The eagerness of Σ_i would then ensure that $\text{AET}(v; \Sigma_i)$ is either 0 or $\text{AET}(u; \Sigma_i) + 1$, where u is the the top-cluster node that is executed immediately before v. In either case, we would have $gap(v; \Sigma_i) = 0$, which contradicts the assumed large gap at v. With the preceding two facts, we are ready to transform Σ_i to Σ_{i+1}, by moving one of v's predecessors to the top-cluster as follows: Let v' be the leftmost external node that is a predecessor of v. Move v' into the top-cluster so that $C(\mathcal{T}_n; \Sigma_{i+1}) = C(\mathcal{T}_n; \Sigma_i) \cup \{v'\}$. Assign $\text{AET}(v'; \Sigma_{i+1}) := \text{AET}(v; \Sigma_i) - 1$ and have Σ_{i+1} schedule both $\mathcal{T}((v'))$ and $\mathcal{T}((v'))$—if they exist—optimally. Finally, have Σ_{i+1} agree with Σ_i in all other respects.

It is transparent that Σ_{i+1} enjoys the ESO and PO properties. Let us, therefore, verify its validity and the MHB property. Since $gap(v; \Sigma_i) \geq 1$ and Σ_i is an eager schedule, we have

$$\text{AET}(v; \Sigma_i) \;=\; \text{EET}(v; \Sigma_i) \;=\; T^\star(\text{HGT}(v) - 1) + \tau.$$

We can, therefore, see that Σ_{i+1} enjoys the MHB property from the fact that

$$\begin{aligned}\text{AET}(v'; \Sigma_{i+1}) &= \text{AET}(v; \Sigma_i) - 1 \\ &= T^\star(\text{HGT}(v) - 1) + \tau - 1 \\ &= T^\star(\text{HGT}(v')) + \tau - 1.\end{aligned}$$

The validity of Σ_{i+1} follows from the fact that

$$\text{AET}(v'; \Sigma_{i+1}) \;=\; T^\star(\text{HGT}(v')) + \tau - 1 \;\geq^\star\; (\text{HGT}(v') - 1) + \tau.$$

The last inequality uses the elementary fact that, for all $n > 1$, $T^\star(n) \geq T^\star(n-1) + 1$. Finally, one sees easily that we can make Σ_{i+1} an eager schedule by applying transformation Π_{EAGER} repeatedly, all the while maintaining the ESO, MHB, and PO properties. □

2.4 Pruning the Search Space for Optimal Schedules

Lemmas 2.1–2.5 assure us that we can focus our quest for optimal fine-grain tree-sweep schedules on schedules that enjoy the five underlying properties of this section. Summing up the lemmas formally, we have:

Theorem 2.1. *One can transform any schedule Σ to an eager, persistent schedule $\Sigma^\star$ that is as fast as Σ and that enjoys the ESO, MHB, and PO properties.*

It follows from Theorem 2.1 that there is an optimal schedule that enjoys the five "underlying" properties. Importantly for the development of our efficient scheduling algorithm in the next section, these five properties actually narrow down our search for an optimal schedule to a unique one!

Theroem 2.2. *There is precisely one eager, persistent schedule for a fine-grain complete-binary-tree up-sweep which enjoys the ESO, MHB, and PO properties.*

Proof. We prove that any two schedules, $\Sigma^{(1)}$ and $\Sigma^{(2)}$, that enjoy all five properties must be identical. To this end, for some arbitrary i, let $v_i^{(1)}$ (resp., $v_i^{(2)}$) be the ith node to be executed in the top-cluster of $\Sigma^{(1)}$ (resp., of $\Sigma^{(2)}$). We show by induction on i that $v_i^{(1)} = v_i^{(2)}$, and $\text{AET}(v_i^{(1)}; \Sigma^{(1)}) = \text{AET}(v_i^{(2)}; \Sigma^{(2)})$.

The base case, $i = 1$, is immediate by persistence and the PO property. Being persistent, both $\Sigma^{(1)}$ and $\Sigma^{(2)}$ have processor 0 execute the first top-cluster node at time 0. By the PO property, this first top-cluster node is the leftmost leaf of $\mathcal{T}_n$.

Assume for induction that, for every $k \leq i$, the kth nodes executed under $\Sigma^{(1)}$ and $\Sigma^{(2)}$ are identical and have the same AET. We show via two cases that $v_{i+1}^{(1)} = v_{i+1}^{(2)}$ and that $\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)}) = \text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)})$.

Assume first that some internal node is eligible for execution after the first i nodes in the top-cluster have been executed. (The coincidence of the two schedules to this point means that this case holds for both schedules if it holds for either one.) Because of the PO property, there is exactly one node that is eligible. Thus, both $\Sigma^{(1)}$ and $\Sigma^{(2)}$ must execute this node next; in other words, $v_{i+1}^{(1)} = v_{i+1}^{(2)}$. Since both schedules are eager, $\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)}) = \text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)}) = \text{AET}(v_i^{(1)}; \Sigma^{(1)}) + 1$.

Alternatively, assume that no internal node is eligible for execution after the first i nodes in the top-cluster have been executed. As long as $v_i^{(1)} \neq \texttt{root}(\mathcal{T}_n)$, both $\Sigma^{(1)}$ and

$\Sigma^{(2)}$ must choose a boundary node to execute. Since both schedules enjoy the MHB property, we must have the following bounds on execution times. On the one hand, for node $v_{i+1}^{(1)}$, we have

$$\begin{aligned} T^\star(\text{HGT}(v_{i+1}^{(1)}) - 1) + \tau \ &\leq aet(v_{i+1}^{(1)}; \Sigma^{(1)}) \\ &\leq^\star (\text{HGT}(v_{i+1}^{(1)})) + \tau - 1. \end{aligned} \quad (1)$$

On the other hand, for node $v^{(2)}$, we have

$$\begin{aligned} T^\star(\text{HGT}(v_{i+1}^{(2)}) - 1) + \tau \ &\leq aet(v_{i+1}^{(2)}; \Sigma^{(2)}) \\ &\leq^\star (\text{HGT}(v_{i+1}^{(2)})) + \tau - 1. \end{aligned} \quad (2)$$

The bounds in (1) and (2) imply that

$$\text{HGT}(v_{i+1}^{(1)}) = \text{HGT}(v_{i+1}^{(2)}).$$

Were this not so, the bounds would imply that

$$|\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)}) - \text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)})| \geq 1. \quad (3)$$

However, since both schedules are persistent, if we let A ambiguously denote $\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)})$ or $\text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)})$, we have

$$\text{AET}(v_i^{(1)}; \Sigma^{(1)}) + 1 \ \leq \ A \ < \ \text{AET}(v_i^{(1)}; \Sigma^{(1)}) + 2,$$

so that

$$|\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)}) - \text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)})| < 1.$$

Since this last bound contradicts (3), we conclude that $\text{HGT}(v_{i+1}^{(1)}) = \text{HGT}(v_{i+1}^{(2)})$. In the presence of the PO property, this equality of heights implies that $v_{i+1}^{(1)} = v_{i+1}^{(2)}$. By the eagerness of both schedules, then,

$$\text{AET}(v_{i+1}^{(1)}; \Sigma^{(1)}) = \text{AET}(v_{i+1}^{(2)}; \Sigma^{(2)}).$$

Having thus extended the induction, we conclude that schedules $\Sigma^{(1)}$ and $\Sigma^{(2)}$ are identical. □

The upshot of Theorems 2.1 and 2.2 is:

Corollary 2.1. *Any eager, persistent schedule for a fine-grain sweep up a complete binary tree, which enjoys the ESO, MHB, and PO properties, is optimal in makespan.*

3 An Optimal Scheduling Algorithm for Fine-Grain Tree-Sweeps

In this section, we specify and analyze the promised linear-time algorithm that optimally schedules sweeps up fine-grain complete binary trees. It is quite simple to verify that the schedules produced by our algorithm are optimal since the schedules possess the five "underlying" properties of Section 2.1, hence, are "covered" by Corollary 2.1. Our algorithm builds on the following elementary fact.

Proposition 3.1. *Let Σ be a schedule for $\mathcal{T}_n$ that enjoys all five underlying properties. If v is the leftmost node at some level of $\mathcal{T}_n$, then $\text{AET}(v; \Sigma) = T^\star(\text{HGT}(v)) - 1$. In other words, Σ schedules subtree $\mathcal{T}(v)$ optimally.*

Proof. Let v be the leftmost node in $\mathcal{T}_n$ at some level and let Σ' be schedule Σ restricted to the nodes in subtree $\mathcal{T}(v)$ i.e., for all $u \in \mathcal{T}(v)$,

$$\text{AET}(u; \Sigma') = \text{AET}(u; \Sigma),$$

and $(u; \Sigma') = (u; \Sigma)$. Clearly, Σ' is persistent; moreover, it inherits the four other properties from Σ. Therefore, Corollary 2.1 assures us that Σ' schedules $\mathcal{T}(v)$ optimally. □

3.1 Description of Algorithm~Fine-Grain

Algorithm **Fine-Grain** produces an optimal schedule, Σ_{opt}, for $\mathcal{T}_n$ as follows:

1. Scheduling the top-cluster. The algorithm chooses a sequence of nodes $v_1, v_2, \ldots, v_k = \texttt{root}(\mathcal{T}_n)$ to constitute the top-cluster. It then assigns each v_i to PE 0 ($(v_i; \Sigma_{opt}) := 0$) and computes $\text{AET}(v_i; \Sigma_{opt})$ as follows:

1.1. Base step: Choose v_1 to be the leftmost leaf of $\mathcal{T}_n$ and set $\text{AET}(v_1; \Sigma_{opt}) := 0$.

1.2. Inductive step: Say that we have chosen nodes $v_1, v_2, \ldots, v_{i-1}$ and their respective AETs. Choose node v_i and its AET as follows:

(a) If some node v is eligible for execution[4]—i.e., both of v's predecessors have been assigned AETs—then choose v as v_i and set $\text{AET}(v_1; \Sigma_{opt}) := \text{AET}(v_{i-1}; \Sigma_{opt}) + 1$. Note that, in this case, v_i must equal $\sigma(v_{i-1})$.

(b) If no node is eligible for execution, then choose v_i to be a new boundary node that satisfies the MHB property, in the following way:

(i) Choose the height h_i (at which we shall select v_i) to be the smallest value of ℓ that satisfies the following constraint:

$$\text{AET}(v_{i-1}; \Sigma_{opt}) \ \leq \ T^\star(\ell) + \tau - 2. \quad (4)$$

Then choose v_i to by the leftmost node of $\mathcal{T}_n$ at height h_i that has not yet been assigned an AET.

In greater detail, we proceed as follows. Let v_j be the last boundary node scheduled before v_i, and let $h_j = \text{HGT}(v_j)$. We consecutively set ℓ to $h_j, h_j + 1, \ldots$, until we find a height ℓ that satisfies (4). Now, in order to perform this search, we need to compute $T^\star(\ell)$ for each $\ell \in \{h_j, h_j + 1, \ldots, h_i\}$. Since v_i is not the first boundary node to be scheduled, it is never the leftmost node of $\mathcal{T}_n$ at its level. Thus, when we choose node v_i, we already know the AET's of the leftmost nodes at all heights h_i and below. Proposition 3.1 allows us to use these AET's to compute the relevant $T^\star(\ell)$. After we determine h_i, we can find the node v_i in constant time, as long as we maintain an array that points to the leftmost unscheduled

4. Recall that a leaf-node is always eligible for execution.

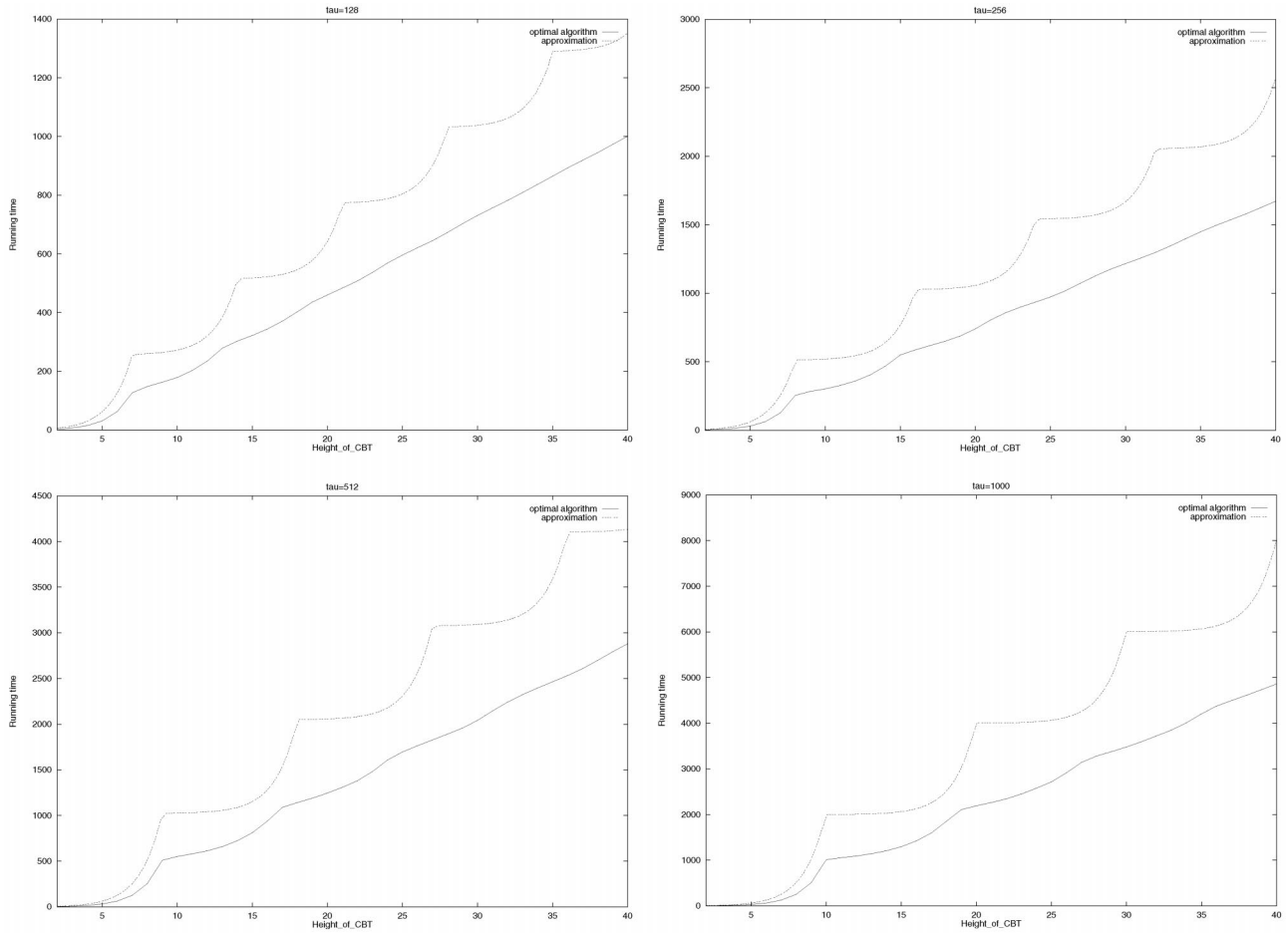


Fig. 2. The makespan when (clockwise from the upper left) $\tau = 128, 256, 1000, 512$. The solid curves describe our schedules; the dotted curves describe the **P.Y** schedules.

node at each level of $\mathcal{T}_n$. We update this array every time a new node is assigned an AET.

(ii) Set[5] $\text{AET}(v_i; \Sigma_{opt})$:
$= \max\{\text{AET}(v_{i-1}; \Sigma_{opt}) + 1, {}^{\star}(h_i - 1) + \tau\}$.

2. Scheduling the external trees. We turn now to the nodes of $\mathcal{T}_n$ that do not reside in the top-cluster. We schedule the external subtrees of $\mathcal{T}_n$ in nondecreasing order of height and in left-to-right order of their roots, as follows: Let v be the root of an external subtree $\mathcal{T}(v)$, and let v' be the leftmost node of $\mathcal{T}_n$ at height $\text{HGT}(v)$. We schedule $\mathcal{T}(v)$ on a set of processors that are used nowhere else by Σ_{opt}. We set the AET of each node in $\mathcal{T}(v)$ to coincide with the AET of the corresponding node of $\mathcal{T}(v')$. This scheduling is possible because of the order in which we schedule external trees: all nodes of $\mathcal{T}(v')$ have been scheduled before we begin to scheduling subtree $\mathcal{T}(v)$.

The schedules produced by Algorithm **Fine-Grain** may leave some boundary nodes in the top-cluster with only one external predecessor. If desired, one can invoke Proposition 2.1 to remedy this situation.

5. Here again, we compute the value of $T^{\star}(h_i - 1)$ from the AET of the leftmost node at height $h_i - 1$.

3.2 Verification and Analysis of Algorithm Fine-Grain

We show first that Algorithm **Fine-Grain**'s schedules possess the five underlying properties, hence are optimal.

Theorem 3.1. *Schedule Σ_{opt} is optimal.*

Proof. We comment on each of the five underlying properties in turn.

First, Σ_{opt} enjoys the ESO property because of Proposition 3.1 and the fact that Algorithm **Fine-Grain** schedules each external subtree in the same manner as the leftmost subtree of $\mathcal{T}_n$ of the same height. The MHB property is ensured by the constraint imposed by inequality (4). The PO property results from the facts that each internal node is scheduled as soon as it is eligible (Step 1.2.a) and that boundary nodes are chosen to be the leftmost unscheduled nodes at their respective heights (Step 1.2.b.i). Eagerness and persistence follow from the fact that processor 0 starts executing at time 0 (Step 1.1) and from the manner in which AETs are assigned to successive nodes (Steps 1.2.a and 1.2.b.ii).

We now invoke Corollary 2.1 to conclude the optimality of Σ_{opt}. □

Next, we analyze the computational complexity of Algorithm **Fine-Grain**, proving that it operates in time linear in the size of the input tree.

Theorem 3.2. *For all tree-heights n, Algorithm **Fine-Grain** generates an optimal schedule for a fine-grain up-sweep of $\mathcal{T}_n$ in time $O(2^n)$, i.e., in time linear in $(\mathcal{T}_n)$.*

Proof. We maintain an array to store information (such as AET's) about each node of $\mathcal{T}_n$. The nodes of $\mathcal{T}_n$ are numbered in breadth-first-search order and the ith numbered node is mapped to the ith slot of the array. Note that the following queries can be executed in constant time: 1) given node u, find the location of its predecessors and successor in the array; 2) given a height ℓ, find the location of the leftmost node at height ℓ in the array. Now, we bound the time spent on each step of the algorithm.

- The initialization in Step 1.1 takes constant time.
- Each execution of Step 1.2.a to schedule a new internal node takes constant time. Since Step 1.2.a is executed at most $|C(\mathcal{T}_n; \Sigma_{\mathrm{PO}})|$ times, the total time spent on this step is proportional to $|C(\mathcal{T}_n; \Sigma_{\mathrm{PO}})|$, hence is $O(2^n)$.
- We bound the total time spent on Step 1.2.b by bounding the total number of times (4) is tested. Now, whenever we find a value of ℓ that satisfies the inequality, we create a new boundary node. Moreover, if a value of ℓ fails to satisfy the inequality, then we never try that value of ℓ again. Thus, the number of times that (4) is tested is no greater than n (the height of $\mathcal{T}_n$) plus the number of boundary nodes. This total is thus proportional to $|C(\mathcal{T}_n; \Sigma_{\mathrm{PO}})|$, hence is $O(2^n)$.
- Step 2 consists entirely of copying information from one node to another, which clearly takes constant time per node. Therefore, the total time spent in Step 2 is $O(2^n)$.

The running time Algorithm **Fine-Grain** is the sum of all the above bounds, hence is clearly $O(2^n)$. □

3.3 An Empirical Evaluation of Algorithm Fine-Grain

The analysis of Algorithm **Fine-Grain** in Section 3.2 shows that it produces optimal schedules, but gives no information about how much the algorithm actually improves previously known fine-grain tree-sweep schedules. As we note in Section 1.1, the best we can hope for is a factor-of-2 speedup since the schedules produced by Algorithm **P.Y** are within a factor of 2 of optimal.

In order to gauge the actual quality of Algorithm **Fine-Grain**, we compare the schedules it produces with those produced by Algorithm **P.Y**, for a variety of tree-sizes and values of τ. Our results indicate that, at least for large trees and large values of τ, the schedules produced by Algorithm **Fine-Grain** actually do approach being twice as fast as those produced by Algorithm **P.Y**.

Our simulations compare the actual performance of a C-program implementation of Algorithm **Fine-Grain** with the easily derived explicit expression for

$$T_{P.Y}(n) \stackrel{def}{=} M_{\mathrm{KSPN}}(\mathcal{T}_n; \Sigma_{P.Y}).$$

Fig. 2 illustrates the results of the simulations, exhibiting the functions $T^{\star}(\mathcal{T}_n)$ and $T_{P.Y}(\mathcal{T}_n)$ for the cases $\tau = 128, 256, 512, 1,000$ and $n \in \{1, 2, \ldots, 40\}$. The plots indicate that, even for moderate values of n and τ, the makespan of Σ_{opt} is a fraction of that of $\Sigma_{P.Y}$ and, moreover, that the fraction seems to approach $1/2$ as n and τ increase. Further analysis shows that $T^{\star}(\mathcal{T}_n)$, considered as a function of n, is "almost linear," exhibiting a "slope" that is periodic with lengthening periods. Comparing the "slope" with the analytical behavior of $T_{P.Y}(n)$, as a function of n, strengthens our belief that $T^{\star}(\mathcal{T}_n)$ tends to $\frac{1}{2}T_{P.Y}(\mathcal{T}_n)$ for large n and τ.

ACKNOWLEDGMENTS

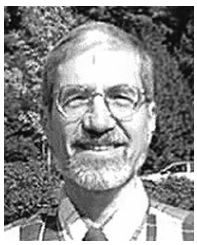
The research of L. Gao was supported in part by U.S. National Science Foundation Grants CCR-92-21785, CCR-94-10077, DUE-97-50789, and NCR-97-29084. The research of A. Rosenberg was supported in part by U.S. National Science Foundation Grants CCR-92-21785 and CCR-97-10367. The research of R. Sitaraman was supported in part by a U.S. National Science Foundation Grant CCR-94-10077 and a U.S. National Science Foundation CAREER Award CCR-97-03017. We would like to thank the anonymous referees for their valuable comments and suggestions. An early version of this paper was presented at the *Seventh IEEE Symposium on Parallel and Distributed Processing*, San Antonio, Texas, 1995, under the title, "Optimal Architecture-Independent Scheduling of Fine-Grain Tree-Sweep Computations."

REFERENCES

[1] F. Afrati personal communication, 1996.
[2] A. Alexandrov, M.I. Ionescu, K.E. Schauser, and C. Scheiman, "LoGP: Incorporating Long Messages into the LogP Model—One Step Closer Towards a Realistic Model for Parallel Computation," *Seventh ACM Symp. Parallel Algorithms and Architectures,* pp. 95-105, 1995.
[3] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis, "BSP vs. LogP," *Eighth ACM Symp. Parallel Algorithms and Architectures,* pp. 25-32, 1996.
[4] R.P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," *J. ACM,* vol. 21, pp. 201-206, 1974.
[5] P. Chretienne, "A Polynomial Algorithm to Optimally Schedule Tasks on a Virtual Distributed System under Tree-Like Precedence Constraints," *European J. Operations Research,* vol. 43, pp. 225-230, 1989.
[6] P. Chretienne and C. Picouleau, "A Basic Scheduling Problem with Interprocessor Communication Delays," *Proc. Summer School on Scheduling Theory and Its Applications,* pp. 81-100, 1992.
[7] P. Chrétienne and C. Picouleau, "Scheduling with Communication Delays: A Survey," *Scheduling Theory and Its Applications,* P. Chrétienne, E.G. Coffman, J.K. Lenstra, and Z. Liu, eds. pp. 65-90. New York: John Wiley & Sons, 1995.
[8] D. Culler, R.M. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Comm. ACM,* 1996.
[9] M. Fujii, T. Kasami, and K. Ninomiya, "Optimal Sequencing of Two Equivalent Processors," *SIAM J. Applied Math.,* vol. 17, pp. 784-789, 1969.

[10] L.-X. Gao and A.L. Rosenberg, "Toward Efficient Scheduling of Evolving Computations on Rings of Processors," *J. Parallel and Distributed Computing,* vol. 38, pp. 92-100, 1996.
[11] M.R. Garey and D.S. Johnson, *Computers and Intractability.* San Francisco: W.H. Freeman, 1979.
[12] A. Gerasoulis, S. Venugopal, and T. Yang, "Clustering Task Graphs for Message Passing Architectures," *Proc. ACM Int'l Conf. Supercomputing,* pp. 447-456, 1990.
[13] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Dags on Multiprocessors," *J. Parallel and Distributed Computing,* vol. 16, pp. 276-291, 1992.
[14] R.L. Graham, "Bounds for Certain Multiprocessor Timing Anomalies," *Bell Systems Technology J.,* vol. 45, pp. 1,563-1,581, 1966.
[15] A. Jakoby and R. Reischuk, "The Complexity of Scheduling Problems with Communication Delays for Trees," *Proc. Scandinavian Workkshop Algorithmic Theory,* pp. 163-177, 1992.
[16] H. Jung, L. Kirousis, and P. Spirakis, "Lower Bounds and Efficient Algorithms for Multiprocessor Scheduling of Dags with Communication Delays," *Information Computing,* vol. 105, pp. 94-104, 1993.
[17] C. Kaklamanis and G. Persiano, "Branch-and-Bound and Backtrack Search on Mesh-Connected Arrays of Processors," *Math. Systems Theory,* vol. 27, pp. 471-489, 1994.
[18] R.M. Karp, A. Sahay, E. Santos, and K.E. Schauser, "Optimal Broadcast and Summation in the LogP Model," *Proc. Fifth ACM Symp. Parallel Algorithms and Architectures,* pp. 142-153, 1993.
[19] R.M. Karp and Y. Zhang, "Randomized Parallel Algorithms for Backtrack Search and Branch-and-Bound Computation," *J. ACM,* vol. 40, pp. 765-789, 1993.
[20] J.-S. Kim, S. Ha, and C.S. Jhon, "Efficient Barrier Synchronization Mechanism for the BSP Model on Message-Passing Architectures," *Proc. 12th IEEE Int'l Parallel Processing Symp.,* pp. 255-259, 1998.
[21] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computations upon Multiprocessor Architectures," *Proc. Int'l Conf. Parallel Processing,* vol. III, pp. 1-8, 1988.
[22] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity," *Handbooks of Operations Research and Management Science,* S.C. Graves and A.H.G. Rinnooy Kan, eds. Amsterdam: Elseviers, vol. 4, pp. 445-522 1993.
[23] J.K. Lenstra, M. Veldhorst, and B. Veltman, "The Complexity of Scheduling Trees with Communication Delays," *J. Algorithms,* vol. 20, pp. 157-173, 1996.
[24] C.H. Papadimitriou and M. Yannakakis, "Towards an Architecture-independent Analysis of Parallel Algorithms," *SIAM J. Computing,* vol. 19, pp. 322-328, 1990.
[25] G.F. Pfister, *In Search of Clusters.* Upper Saddle River, N.J.: Prentice Hall, 1995.
[26] A.G. Ranade, "Optimal Speedup for Backtrack Search on a Butterfly Network," *Math. Systems Theory,* vol. 27, pp. 85-101, 1994.
[27] V.J. Rayward-Smith, "UET Scheduling with Unit Interprocessor Communication Delays," *Discrete Applied Math.,* vol. 18, pp. 55-71, 1987.
[28] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors.* Cambridge, Mass.: MIT Press, 1989.
[29] H.K. Tadepalli and E.L. Lloyd, "An Improved Approximation Algorithm for Scheduling Task Trees on Linear Arrays," *10th Int'l Parallel Processing Symp.,* pp. 584-590, 1996.
[30] J.D. Ullman, "NP-Complete Scheduling Problems," *J. Comput. Syst. Scis., //please spell out journal name//* vol. 10, pp. 384-393, 1975.
[31] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM,* vol. 33, pp. 103-111, 1990.
[32] M.-Y. Wu and D. Gajski, "A Programming Aid for Hypercube Architectures," *J. Supercomputing,* vol. 2, pp. 349-372, 1988.
[33] T. Yang and A. Gerasoulis, "A Fast Static Scheduling Algorithm for Dags on an Unbounded Number of Processors," *Proc. Supercomputing '91,* pp. 633-642, 1991.

Lixin Gao received the BS degree in computer science from the University of Science and Technology of China in 1986, and the MS degree in computer engineering from Florida Altantic University in 1987. Between 1987 and 1990, she was an engineer at the Avionics Division of Allied Signal. She received her PhD degree in computer science from the University of Massachusettes at Amherst in 1996. Her research interests include multimedia networking and parallel and distributed systems. Since 1996, she has been an assistant professor of computer science at Smith College. Dr. Gao is a recipient of the U.S. National Science Foundation CAREER Award, and she is a member of the IEEE, the ACM, and Sigma Xi.



Arnold L. Rosenberg received an AB degree in mathematics from Harvard College in 1962, and AM and PhD degrees in applied mathematics from Harvard University in 1963 and 1966, respectively. Dr. Rosenberg is Distinguished University Professor of Computer Science at the University of Massachusetts (UMass) at Amherst, where he co-directs the Theoretical Aspects of Parallel and Distributed Systems (TAPADS) Laboratory. Prior to joining UMass, he was a professor of computer science at Duke University from 1981 to 1986, and a research staff member at the IBM Watson Research Center from 1965 to 1981. He has held visiting positions at the Technion (Israel Institute of Technology), Yale University, and the University of Toronto. Dr. Rosenberg's research focuses on theoretical aspects of parallel architectures and communication networks, with emphasis on developing algorithmic techniques for designing better networks and architectures and using them more efficiently. He is the author of more than 130 technical papers on these and other topics in theoretical computer science and discrete mathematics. Dr. Rosenberg is a fellow of the ACM, a fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a member of SIAM. He is the editor-in-chief of *Theory of Computing Systems* (formerly, *Mathematical Systems Theory*) and serves on other editorial boards.

Ramesh K. Sitaraman received his BTech in electrical engineering from the Indian Institute of Technology, Madras. He obtained his PhD degree in computer science from Princeton University in 1993. Dr. Sitaraman is currently an assistant professor of computer science at the University of Massachusetts at Amherst, where he is a member of the Theoretical Computer Science group, and co-directs the Theoretical Aspects of Parallel and Distributed Systems (TAPADS) Laboratory. Dr. Sitaraman's research focuses on fundamental theoretical issues in the design and use of parallel and distributed systems, and communication networks. His specific interests include communication in parallel and distributed networks, fault tolerance, and algorithms for scheduling and load-balancing. He is a recipient of an U.S. National Science Foundation CAREER Award and a Lilly Fellowship. He is a member of the IEEE and the ACM.