Scheduling In and Out Forests in the Presence of Communication Delays

Theodora A. Varvarigou, *Member, IEEE*, Vwani P. Roychowdhury, *Member, IEEE*, Thomas Kailath, *Fellow, IEEE*, and Eugene Lawler

Abstract—We consider the problem of scheduling tasks on multiprocessor architectures in the presence of communication delays. Given a set of dependent tasks, the *scheduling problem* is to allocate the tasks to processors such that the pre-specified precedence constraints among the tasks are obeyed and certain cost-measures (such as the computation time) are minimized. Several cases of the scheduling problem have been proven to be NP-complete [16], [10]. Nevertheless, there are polynomial time algorithms for interesting special cases of the general scheduling problem [12], [14], [10]. Most of these results, however, do not take into consideration the delays due to message passing among processors. In this paper we study the increase in time complexity of scheduling problems due to the introduction of communication delays. In particular, we address the open problem of scheduling Out-forests (In-forests) in a multiprocessor system of m identical processors when communication delays are considered. The corresponding problem of scheduling Out-forests (In-forests) without communication delays admits an elegant polynomial time solution as presented first by Hu in 1961 [12]; however, the problem in the presence of communication delays has remained unsolved. We present here first known polynomial time algorithms for the computation of the optimal schedule when the number of available processors is given and bounded and both computation and communication delays are assumed to take one unit of time. Furthermore, we present a *linear-time* algorithm for computing a *near-optimal* schedule for unit-delay out-forests. The schedule's length exceeds the optimum by no more than $(m-2)$ time units, where m is the number of processors. Hence for two processors the computed schedule is strictly optimum.

Index Terms—Communication delays, out-forest precedence graphs, multiprocessor architectures, out-forest precedence graphs, optimal deterministic schedules, polynomial-time algorithms.

1 Introduction

We consider *deterministic scheduling* problems in the presence of communication delays among the processors. The area of deterministic scheduling is of obvious importance, and has been extensively studied since the early 1950s. In most of the cases, however, the communication delays among processors were ignored. We study here the increase in the complexity of scheduling problems due to the introduction of communication delays.

Scheduling of dependent tasks in a multiprocessor system is such a well studied topic that any reference to the available material has to be selective. The general problem of scheduling a set of n unit delay and arbitrarily dependent tasks on a set of m identical processors where both m and n are variables of the problem and unbounded has been proven to be NP-complete [16]. Moreover, the scheduling problem remains NP-complete even in some cases of restricted kind of dependencies among the tasks (see [10], [13], [19], [11], [9]).

- *T.A. Varvarigou is with the Department of Electronic and Computer Engineering, Technical University of Crete, Chania 73100, Crete. Email: theodora@ced.tuc.gr.*
- *V.P. Roychowdhury is with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095.*
- *T. Kailath is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.*
- *E. Lawler was with the Department of Electrical Engineering and Computer Science, University of California, Berkeley. He is deceased.*

Manuscript received Sept. 1, 1992.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95195.

Nevertheless, there are polynomial time algorithms for several interesting special cases of the general scheduling problem. One of the first polynomial time algorithms was developed by Hu [12], who presented a linear time algorithm for computing the optimum schedule of an In-forest (or Out-forest) precedence graph, for a given number of available processors. The general strategy used in this algorithm is the *highest-level-first* strategy, modified versions of which give optimal results for other scheduling problems such as for the *interval order precedence graphs* [14], [7] or for the special case where only two processors are available for the computation of the tasks [6], [8]. Polynomial time algorithms have been presented for the special cases of opposing forests [10], [5] as well as level order [5] precedence graphs.

The above mentioned results and algorithms assume that the communication delays due to message passing among processors that compute dependent tasks is negligible. A more realistic approach is to take communication delays into consideration for the computation of the optimal schedule. However, introducing communication delays may destroy the optimality of the schedules computed for precedence graphs without communication delays. Hence a new analysis of the scheduling problem is needed when communication delays are considered. Unfortunately, the general scheduling problem with communication delays is NP-complete; this is because it is a general version of the usual scheduling problem (i.e., without communication delays), which is NP-complete [16]. The problem remains NP-complete even when the number of available proces-

sors is *arbitrary* (number of processors is said to be arbitrary if the algorithm can use unrestricted number of processors). In particular, Chretienne showed that for general precedence graphs, and for arbitrary computational and communication delays, the problem of computing the optimal schedule remains NP-complete, even if no restrictions are imposed on the number of available processors (see [3]). Note that the corresponding scheduling problem with arbitrary number of processors but without communication delays admits a simple linear time algorithm [9]. An even stronger NP-complete result was shown by Papadimitriou and Yannakakis in 1990. They proved that even if the computation delays are restricted to be unit, and the communication delays are restricted to be a positive integer t, the problem still remains NP-complete (see [15]).

However, as already mentioned, there are polynomial time algorithms for different special cases of the scheduling problem, such as In/Out forest precedence graphs, interval order precedence graphs, *etc.*, without communication delays. The question we ask is: *How does the complexity of these special cases change when communication delays are introduced?* In particular, we investigate the complexity of the scheduling problem with communication delays for the special case of In/Out forest precedence graphs.

An attempt to incorporate communication delays in scheduling of general graphs was made by Colin and Chretienne in [4]. They consider general precedence graphs, and they allow short communication delays, as well as duplication of tasks. The number of available processors is considered to be arbitrary. Under these constraints, for each task i, they propose polynomial time algorithms to compute lower bounds b_i of the starting time of any of its copies. A first attempt to consider communication delays in In-tree precedence graphs was made by Chretienne in [2]. He developed a greedy but optimal algorithm for computing the optimal schedule of In-tree precedence graphs under the assumptions of *arbitrary number of available processors* and *short communication delays*. The same problem was also considered by Anger et al. in [1], and the proposed algorithm works for the case of In-forests/Out-forests precedence graphs and arbitrary number of processors. None of these algorithms, however, is applicable to the case where the number of processors at each time step are not as many as needed (i.e., arbitrary) but *given as a variable of the problem.*

We address here the open problem of scheduling Out-forest/In-forest precedence graphs with communication delays when the number of available processors is *not arbitrary* but *a variable of the problem*. This case where m is a constant or a variable of the problem (but bounded) is the *most frequently encountered case in practical applications*. We present algorithms of $O(n^{2m-2})$ time complexity for the problem, where m is the number of available processors, and the computation and communication delays are assumed to take unit time. Hence, for constant number of processors the problem of scheduling In/Out forest precedence graphs admits polynomial time solution. *This is the first known result of its type and demonstrates that certain scheduling problems remain of polynomial time complexity even in the presence of communication delays.* We next give a detailed description of our model, assumptions, and our results.

1.1 The Model and Results

The general model that we are going to assume in this paper is similar to the models used by other researchers (see [1], [5], [10]) and can be described as follows: The model consists of m identical processors $P_1, P_2, \ldots, P_m$ where m is a parameter of the problem and bounded by a constant c. There exist n computational tasks $T_1, T_2, \ldots, T_n$ that can be executed in any of the m available processors. There is a partial order among the tasks. This implies that if $T_i \rightarrow T_j$ (i.e., T_j is dependent on T_i) then the computation of T_j in some processor P_j cannot start before task T_i is computed (say in processor P_i) and the results of this computation have been transmitted from processor P_i to processor P_j. The partial order among the tasks introduces a *precedence graph* associated with the scheduling problem. The tasks $\{T_i\}$ are the nodes of the graph and we assume directed edges between the nodes $T_i \rightarrow T_j$ whenever there is a dependency between tasks T_i and T_j.

For our purposes we shall assume that the precedence graph is of the **Out-forest** (or **In-forest**) form which is defined as follows: A precedence graph is an Out-Forest (In-Forest) if and only if: For every task T_i there is only one task T_j for which the dependency $T_j \rightarrow T_i$ ($T_i \rightarrow T_j$) exists. All the tasks are of unit computational time, i.e., it takes one unit of time to compute any of the tasks $T_1, \ldots, T_n$ in any of the processors $P_1, \ldots, P_m$. The processors are fully connected, i.e., any processor can communicate with any other processor. Moreover, the computation of the tasks in the processors is independent of the communication among them. This implies that the processors can execute tasks at the same time that communication is taking place among them. The communication delay among any two processors takes unit time, i.e., it takes one unit of time for the transmission of the results of the computation of any task from any processor P_i to any processor P_j different than P_i. There is no communication delay involved when some task T_i and its dependent task T_j are computed in the same processor. Immediately after the completion of the computation of any task T_i, the results are sent to the processors where the tasks that depend on T_i are going to be computed.

Given the above general model, the scheduling problem can be formulated as follows:

PROBLEM 1. *Given a set of processors $P_1, \ldots, P_m$ and a set of dependent tasks $T_1, \ldots, T_n$ whose corresponding precedence graph is of the Out-forest (In-forest) form, find a valid schedule for the assignment of the given tasks to the processors, i.e., assign each task T_i to a processor P_i and to a time slot t_i in such a way that:*

1) *If $T_i \rightarrow T_j$ then T_j is not scheduled to begin before T_i is completed and the results of T_i have been sent from P_i (where T_i is computed), to P_j where T_j is computed; and*
2) *The total completion time is minimum.*

In this paper we first solve Problem 1 and prove the following result (see Theorem 6):

Given an out-forest G with communication delays, the optimal schedule for G can be computed in $O(n^{2m-2})$ time.

Note that our algorithm is of polynomial time complexity when m is bounded by a constant. The question whether

the problem is NP-complete for unbounded m remains unanswered. The analysis of the In-forest precedence graph scheduling problem reduces to the Out-forest precedence graph scheduling problem by just reversing the direction of the dependencies between the tasks and then inverting the resulting optimal schedule.

In the process of proving this theorem we develop efficient ways of transforming the given graphs that are subject to communication delays into *delay free* graphs that can be scheduled without taking into consideration communication delays between the processors, and whose optimal schedules *obey the precedence and communication delay constraints* of the original graph and have *the same length* as the optimal schedule of the original graph. This allows us to use dynamic programming techniques to obtain polynomial time algorithms for the computation of the optimal schedules.

Our algorithm is a generalization of the algorithm presented in [5] for graphs with no communication delays. Using similar principles as in [5], we prove that we can actually take communication delays among the processors into consideration while keeping the time complexity of the algorithm polynomial. Moreover, it appears that our result regarding the equivalence of a given precedence graph to a corresponding delay free precedence graph (see Section 3.1 and Theorem 3) is the first of its kind. We believe that such ideas can be used to *generalize other algorithms* that have been developed for scheduling precedence graphs without communication delays, into polynomial time algorithms for the case when communication delays are considered.

Furthermore, we present a *linear-time* algorithm for computing a **near-optimal** schedule for unit-delay out-forests (see Theorem 8). The schedule's length exceeds the optimum by no more than $m - 2$ time units. Hence for two processors the computed schedule is strictly optimum.

1.2 Notations and an Outline of the Paper

The schedule that results as a solution of Problem 1 will be called an *optimum schedule*. Any assignment of the tasks to the processors and to the time slots that obeys the precedence constraints and the communication delay constraints is called a *valid schedule* (or simply a *schedule*). The total number of time slots that a certain schedule occupies is called the *length* of the schedule. We often refer to the precedence graph of the tasks simply as a *graph*. In the analysis that follows we assume only Out-forest precedence graphs. The computational tasks are often referred to as *nodes*. Whenever there is a dependency $T_i \rightarrow T_j$ between tasks we refer to T_i as the parent of T_j and we refer to T_j as the child of T_i. If there is a directed path in the precedence graph from a node T_i to a node T_j then we refer to T_i as the predecessor of T_j and we refer to T_j as the successor of T_i. Height $h(G)$ of a precedence graph G is the length of the longest path in G. A subgraph of a given out-forest is called a *tree* if there is a single node in the subgraph that has no predecessors. *Root* is a node that has no predecessors and *leaf* is a node with no successors. A graph F is called a subgraph of a (precedence) graph G if every node in F has the same successors as it has in G.

The rest of the paper is as follows: In Section 2, we present some results on scheduling dependent tasks when communication delays are ignored. In Section 3.1, we discuss the *favored child property* that allows us to derive delay free graphs out of graphs that are subject to communication delays. In Section 3.2.1, we introduce the notion of the *shortest delay free graph*. In Sections 3.2.2 and 3.2.3, we develop polynomial time algorithms for computing first the length of the optimal schedule for a given set of dependent tasks and then for actually computing the optimal schedule. In Section 4, we present a *linear-time* algorithm for computing a *near-optimal* schedule for unit-delay out-forests. Finally, in Section 5, we make some concluding remarks.

2 The MERGE Algorithm

We shall discuss here the so-called MERGE algorithm, which was first developed in [5] for scheduling out-forests *without* communication delays. We shall use the algorithm and the related concepts in Section 3.2, where communication delays will be taken into account as well.

DEFINITION 1. *A* **component** *C of a graph G is any subgraph of G for which the following is true: there are no edges among nodes of C and nodes of G − C. That is, C is the union of one or more disjoint connected components of G. The* **median** *of an out-forest G, $\mu(G)$, is the height of some m*th *highest tree of G plus one, where m is the number of available processors. The* **high subgraph** H_G *of a given out-forest G is the subgraph of G that contains all the trees, with height strictly greater than the median of G (i.e., $\mu(G)$). The* **low subgraph** L_G *of G, is the subgraph of G that contains all the trees of G that are of height less or equal to the median $\mu(G)$.*

Given a graph G we can obviously compute the high and the low subgraphs of G, H_G, and L_G, respectively, in $O(n)$ time. The following theorem suggests a way of computing the length of the optimum schedule of an out-forest G, given the length of an optimal schedule of its high subgraph H_G, and is due to Dolev and Warmuth [5].

THEOREM 1. *For any delay free out-forest G,*

$$\lambda(G) = \max\left\{\lambda(H_G),\ \min_k\left\{k \mid km \geq |G|\right\}\right\}.$$

The **MERGE** algorithm presented in [5] computes the optimum schedule for a precedence graph G, given an optimum schedule for the high subgraph H_G of G (see also [5]), in $O(n)$ time.

The following theorem (see [5] for a proof), gives the time complexity for the MERGE algorithm.

THEOREM 2. *Let G be an out-forest precedence graph. Then given an optimum schedule for the high subgraph of G, there is an O(n)-time algorithm that computes an optimum schedule for the whole graph G.*

3 Polynomial Time Algorithms for Optimal Scheduling in the Presence of Communication Delays

3.1 Delay Free Transformations of Precedence Graphs

As already mentioned in the introduction, if task T_i, scheduled in processor P_i, depends on task T_j scheduled in processor P_j at time slot t, then task T_i cannot be scheduled to begin at time slot $t + 1$ or earlier. This is because the results of task T_j are needed for the computation of task T_i and cannot be transmitted from processor P_j to P_i before time slot $t + 2$. The transmission of results, as already stated in the introduction, is independent of the actual computation in the different processors, so processor P_i or P_j can be used for the computation of other tasks while the communication between P_i and P_j is taking place. The idea that is going to allow us to compute optimum schedules for the case when communication delays have to be taken into account is the *transformation* of the given graph into an *equivalent delay free graph*. The delay free graph can then be scheduled under precedence constraints only.

So the general idea is to transform any given graph into a delay free graph, in which whenever a node n has more than one child, namely $n_1, \ldots n_k$ (see Fig. 1a), then one of them, say n_i, is chosen to be the *favored child* that is scheduled *before* all the other children of the parent n. Then the delay dependencies between node n and its children $n_1, \ldots, n_k$ are removed and delay free dependencies are added between node n and the favored child n_i, and between n_i and the rest of the children $n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k$ of n as shown in Fig. 1b. *Of course one has to prove that the transformation results into a graph that obeys the precedence constraints of the original graph and that there always exists a delay free graph which has an optimum schedule of the same length as the optimum schedule of the original graph.* Before proceeding into Lemmas 1 and 2 that prove the above we define the following:

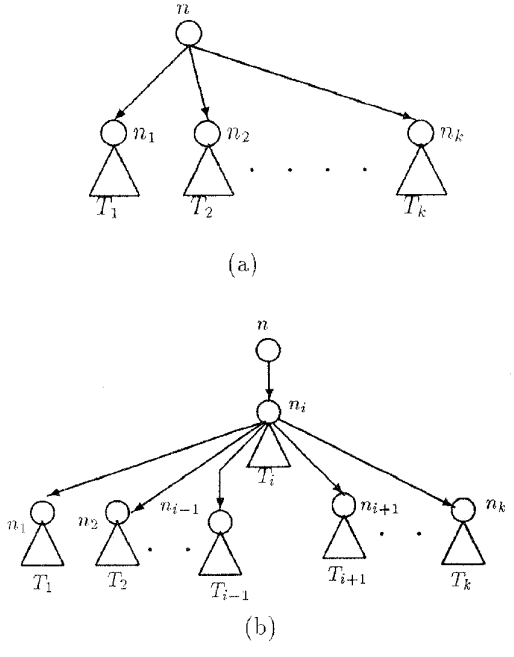


Fig. 1. Delay free transformation of delay graphs: the strategy.

Delay dependencies are the dependencies that are subject to communication delays. So, if a node n has more than one delay dependent children, only one of them can be scheduled in the time slot immediately following the time slot n is scheduled. **Delay free dependencies** are the dependencies that are not subject to communication delays. So, if a node n has more than one delay free dependent children, then there is no restriction on how many out of the n children that can be scheduled in the time slot, immediately following the time slot where n is scheduled.

DEFINITION 2. *Suppose that G is a graph whose dependencies are delay dependencies. Then we define the corresponding* **delay free** *graph G^s as the graph that results if we replace the* **delay dependencies** *among every node n and its children $n_1, n_2, \ldots$, with two stages of delay free dependences:*

1) *between n and some child n_j of n (node n_j will be referred to as the favored child), and*
2) *among n_j and the rest of the children of n.*

The following lemma states that given a precedence graph G, any schedule of a corresponding delay free graph G^s is a valid schedule for G as well (for a proof, please see [17]).

LEMMA 1. *Suppose that G is a delay graph and G^s is a corresponding delay free graph of G. Then if S^s is a valid schedule for G^s (that obeys the precedence constraints of G^s) then S^s is a valid schedule for G as well (that obeys both the precedence and the communication delay constraints).*

Hence, any schedule valid for graph G^s is valid for the original graph G as well. What we are going to show now is that for any graph G there exists a corresponding delay free graph G^s that has an optimum schedule of the same length as the optimum schedule of the original delay graph G. This optimum schedule of G^s is an optimum schedule of G as well. The following two definitions are going to be used in the presentation of Lemma 2.

DEFINITION 3. *We say that a schedule S of graph G has the* **favored child property**, *if and only if for every node $i \in G$ exactly one of i's children, j, is assigned to an earlier time slot than the other children of i, if i has a child at all. Node j is called the favored child of i.*

DEFINITION 4. *Given a schedule S of graph G, of length l we define S_i, $1 \le i \le l$, to be the nodes of G scheduled in time slot i, in schedule S.*

LEMMA 2. *For every graph G there exists an optimum schedule S with the favored child property.*

PROOF. Consider an optimum schedule $S = (S_1, \ldots, S_l)$ of G for which the favored child property does not hold. Let t be the latest time slot containing a node i without a favored child. Let t', where $t' \ge t + 2$, be the earliest time slot containing a child j of i. Each node in $S_{t'-1}$ has at most one child in $S_{t'}$, and j itself is not the child of a node in $S_{t'-1}$. Hence, there is at least one node k in $S_{t'-1}$ without a child in $S_{t'}$. So let us swap nodes j and k, moving j from $S_{t'}$ to $S_{t'-1}$ and k from $S_{t'-1}$ to $S_{t'}$. After this interchange, j is the favored child of i, and every

node that had a favored child before the interchange still has a favored child. It follows that a finite number of such interchanges creates an optimum schedule with the favored child property. For an alternative proof of this lemma you can also see [17], [18]. □

However, the optimal schedule of a delay graph G that has the favored child property is also the optimum schedule of a delay free transformation of G. Hence, the following Corollary follows immediately from Lemma 2.

COROLLARY 1. *Suppose G is a delay graph that has an optimum schedule of length L. Then there is a delay free graph G^s corresponding to G, which has an optimum schedule of the same length L.*

Theorem 3 below follows easily from the above lemmas and corollary.

THEOREM 3. *Given an out-forest G, there exists a corresponding delay free graph G^s that has an optimum schedule of the same length as the optimum schedule of the original graph G.*

The result of Theorem 3 plays a key role in the proof of Theorem 5. In fact, in the proof of Theorem 5, we show how to efficiently determine a delay free graph G^s that has an optimum schedule of the same length as the optimum schedule of any given graph G.

3.2 Scheduling in the Presence of Communication Delays

In this section, we are going to present polynomial time algorithms for the computation of the optimal schedule of out-forest graphs. The following section presents a linear-time algorithm for computing a near-optimal schedule for unit-delay out-forests.

3.2.1 Shortest Delay Free Graph

First we present an algorithm for computing the *shortest delay free graph* that corresponds to a given precedence graph which we define below.

DEFINITION 5. *Shortest delay free graph G^s of a given graph G is a delay free graph such that every subgraph of G^s has height less than or equal to the height of the corresponding (i.e., containing the same nodes) subgraph in any other delay free graph for G.*

LEMMA 3. *Given a tree T, one can construct a shortest delay free tree T^s, as well as the shortest delay free trees that correspond to all the subtrees of T in $O(n)$ time.*

PROOF. A constructive algorithm can be described as follows:

The Algorithm: Consider the out-tree T as shown in Fig. 1a rooted at a node n. Let $n_1, n_2, \ldots, n_k$ be the children of n and T_i are the subtrees of T corresponding to nodes n_i, $i = 1, 2, \ldots, k$ (see Fig. 1b). Construct the shortest delay free tree T^s in the following recursive fashion:

1) Compute the shortest delay free trees T_i^s, $i = 1, \ldots, k$, corresponding to the subtrees T_i of the original tree T.
2) Construct the shortest delay free tree T^s as follows:
   a) Suppose that

$$\text{height}\left(T_i^s\right) = \max_{j=1,\ldots,k} \text{height}\left(T_j^s\right)$$

then set node n_i as the only children of node n. If there are more than one nodes n_i for which

$$\text{height}\left(T_i^s\right) = \max_{j=1,\ldots,k} \text{height}\left(T_j^s\right),$$

then break the tie arbitrarily.

   b) Add subtrees $T_1^s, \ldots, T_{i-1}^s, T_{i+1}^s, \ldots, T_k^s$ as successors of node n_i (keeping the rest of the successors of n_i as they are in T_i^s) as shown in Fig. 1b.

It is easy to see that the above algorithm is correct and requires $O(n)$ time. □

The generalization of Lemma 3 for the case of the *shortest delay free graph* G^s corresponding to an out-forest G is straightforward. One should just take the shortest delay free trees that correspond to the trees of graph G. Note also that in the process of computing G^s out of G, one computes the shortest delay free trees that correspond to all the subtrees of G, as well as their height. So we get the following corollary:

COROLLARY 2. *Given an out-forest G, one can construct a shortest delay free out-forest G^s in $O(n)$ time. Given G^s, one can find the shortest delay free transformation of any subtree of G, and its corresponding height in $O(1)$ time.*

DEFINITION 6. *Given a subgraph F of graph G, we define the corresponding shortest delay free graph F^s to be the subgraph of G^s (where G^s is the shortest delay free graph of G) that contains only the nodes that belong to F.*

3.2.2 High and Low Subgraphs for Unit-Delay Graphs

In this section, we present a basic theorem that relates the length of the optimum schedule of a graph that is subject to communication delays, to the length of the optimal schedule of its high subgraph, the latter being defined for graphs that are subject to communication delays as follows.

DEFINITION 7 (High/Low subgraphs of a graph that is subject to communication delays). *Given a graph F and the corresponding sortest delay free graph $F^s = H_{F^s} \cup L_{F^s}$, where H_{F^s} and L_{F^s} are the high and the low subgraphs of F^s, respectively, we define $H^e_{F^s}$ and $L^e_{F^s}$ to be the subgraphs of F that correspond to H_{F^s} and L_{F^s} subgraphs of F^s (i.e., contain the same nodes as H_{F^s} and L_{F^s}). We call $H^e_{F^s}$ and $L^e_{F^s}$ the high and low subgraphs of F, respectively.*

For example graph F of Fig. 2a has graph F^s of Fig. 2b as its corresponding shortest delay free graph. H_{F^s} and L_{F^s} are illustrated in Fig. 2b and $H^e_{F^s}$ and $L^e_{F^s}$ are illustrated in Fig. 2a.

DEFINITION 8. *We define $\lambda(F)$ as the length of the optimum schedule of a graph F.*

As proven in Theorem 3 there exists a delay free graph H corresponding to $H^e_{F^s}$ that has a schedule of length equal to the length of the optimal schedule for $H^e_{F^s}$, i.e., $\lambda(H^e_{F^s}) = \lambda(H)$. The trees of H are at least as high as the

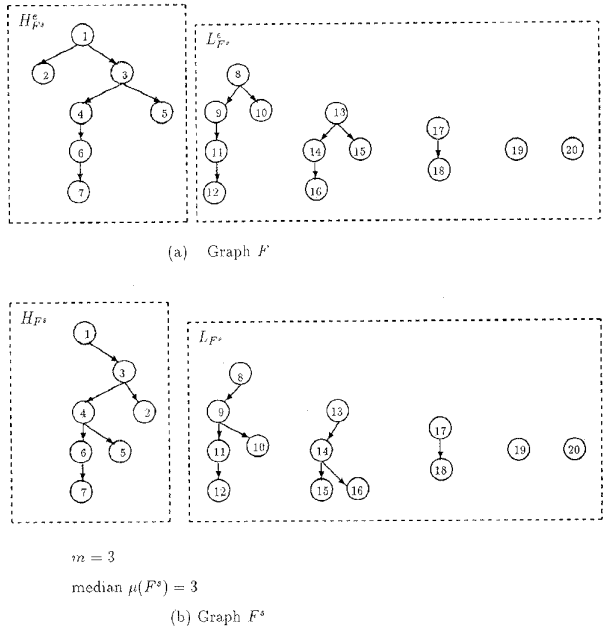


Fig. 2. Examples of $H^e_{F^s}$ and $L^e_{F^s}$, corresponding to an out-forest G.

trees of H_{F^s} (since both H_{F^s} and H are delay free graphs corresponding to $H^e_{F^s}$, and H_{F^s} is the shortest delay free graph corresponding to delay graph $H^e_{F^s}$). So H is the high graph of the delay free graph $H \cup L_{F^s}$ (since H_{F^s} was the high tree of $H_{F^s} \cup L_{F^s}$). We claim that the following is true:

CLAIM 1. $\lambda(H \cup L_{F^s}) = \max\{\lambda(H), \min_k\{k \mid km \geq |H| + |L_{F^s}|\}\}$.

We now make the following claim:

CLAIM 2. $\lambda(F) = \lambda(H \cup L_{F^s})$.

The above two claims lead to the following result.

THEOREM 4. *For any subgraph F of a given out-forest G,*

$$\lambda(F) = \max\left\{\lambda\left(H^e_{F^s}\right), \min_k\left\{k \mid km \geq |F|\right\}\right\}.$$

PROOF. The proof follows immediately from Claim 2 by noting that:

1) $\lambda(H^e_{F^s}) = \lambda(H)$
2) $|L^e_{F^s}| = |L_{F^s}|$ and
3) that the optimal schedules S and S^* of $H^e_{F^s}$ and H respectively have the same number of idle periods $p = p^*$ (since they have the same length and the same number of nodes).

Thus,

$$\lambda(F) = \begin{cases} \lambda\left(H^e_{F^s}\right) & \text{if } p \geq \left|L^e_{F^s}\right|, \\ \min_k\left\{k \mid km \geq |F|\right\} & \text{if } p < \left|L^e_{F^s}\right|. \end{cases}$$

Since $\lambda(F)$ is always greater than or equal to $\min_k\{k \mid km \geq |F|\}$, then

$$\lambda(F) = \max\left\{\lambda\left(H^e_{F^s}\right), \min_k\left\{k \mid km \geq |F|\right\}\right\}. \qquad \square$$

3.2.3 A Dynamic Programming Algorithm for Computing the Optimal Schedule

As already proven in Theorem 4, given a subgraph F of G, and the length of the optimal schedule for $H^e_{F^s}$, we can compute $\lambda(F)$ in $O(n)$ time. So, in order to be able to compute the length of the optimal schedule for any subgraph F of G, we need to compute $\lambda(H^e_{F^s})$ for every subgraph F.

Since we want to compute a schedule that corresponds to a delay free transformation, it is useful to characterize the nodes of any given subgraph, such that the favored child property is maintained. Let us define as **Init(G)** the set of nodes of G that have no predecessors. Let us also define two nodes n_1, n_2 in G to be **brothers** if these nodes have the same parent in G. Then there are two kind of nodes in Init($H^e_{F^s}$):

1) The nodes whose all of their brothers do not belong in $H^e_{F^s}$. These nodes can be scheduled independently of the rest of the nodes in Init($H^e_{F^s}$) and are called **initial independent nodes**; and
2) The nodes whose brothers are all in Init($H^e_{F^s}$).

These nodes cannot be arbitrarily scheduled, but one of them has to be scheduled before all its brothers, because it corresponds to the root of a tree in H_{F^s}. Hence we define a **bunch** to be a set of nodes of a graph G such that every two nodes in the set are brothers and moreover, every brother of any node in the set is also in the set. A bunch that consists of initial nodes is called an **initial bunch**.

Given a subgraph F of a graph G we define $n \in Init(F)$ to be an **initial independent node** if all its brothers are not in F. We define **initial components** of a subgraph F of graph G to be the set of all initial independent nodes and initial bunches of F. For example, in the subgraph F in Fig. 3a, *Initial independent nodes(F)* = {1}. The set of initial components of F in Fig. 3a is {1, {8, 13}, {17, 19, 20}}. Note that the set of initial components of F and the set of initial nodes of F contain the same nodes, with the only difference that in the set of initial components of F the nodes are grouped into initial bunches and initial independent nodes. We now make the following claim:

CLAIM 3. *Consider graph G with shortest delay free transformation G^s computed using the algorithm presented in Lemma 3. Then, given a subgraph F of G where some of F's initial nodes are initial independent nodes and some belong to initial bunches, one can compute $H^e_{F^s}$ and L_{F^s} of F in O(1) time.*

The following claim follows easily from Definition 1 and Definition 6.

CLAIM 4. *Given a subgraph F of G, its subgraph $H^e_{F^s}$ contains at most $m - 1$ initial components.*

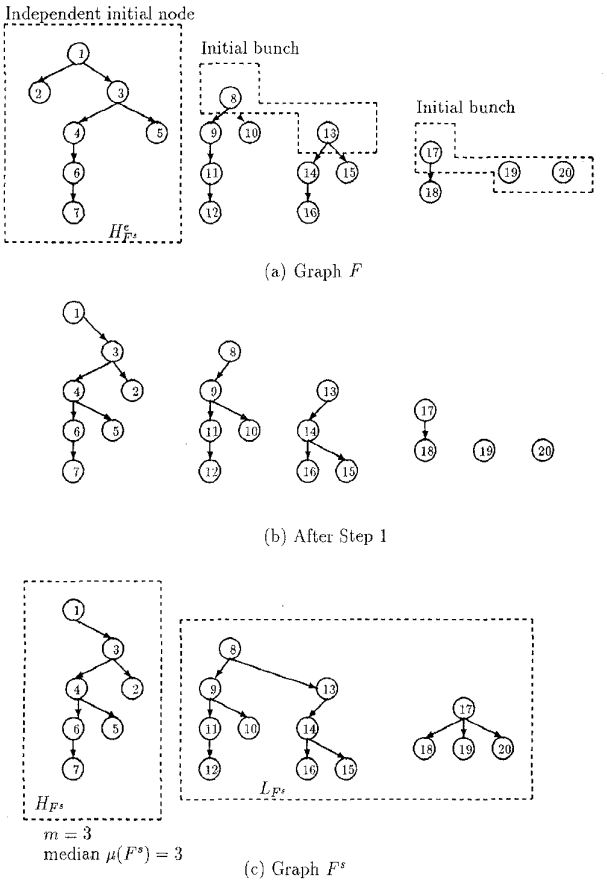


Fig. 3. Independent initial nodes, initial bunches.

DEFINITION 9. *Given a graph F we introduce the following notation:* $F \Rightarrow^R F'$ *where R is a subset of Init(F) with the restriction that no more than one node from the same initial bunch of F are in R. F′ is the graph that results from F if we delete the nodes of R.*

We know that $H^e_{F^s}$ contains at most $m-1$ initial components. Hence, the set of all the subgraphs of G that have no more than $m-1$ initial components, contains the set of all the possible high graphs $H^e_{F^s}$, corresponding to any subgraph F of G. So, if we compute the length of the optimal schedule for all subgraphs of G that contains at most $m-1$ initial components, we can then compute in $O(n)$ time the length of the optimal schedule for G itself or any subgraph of G. In the following theorem we give a recursive procedure for computing the length of the optimal schedule for all subgraphs of G that contain at most $m-1$ initial components. In Theorem 6, we show how to use the results of Theorem 5 to compute the optimal schedule for G.

Our constructive proof for the following theorem is based on the results of Theorem 3, namely, the existence of a delay free graph for any G with the same length of optimal schedule.

THEOREM 5. *Let G be an out-forest. Then for every subgraph F of G with no more than* $m-1$ *initial components,* $\lambda(F)$ *can be computed in* $O(n^{2m-2})$ *time.*

PROOF. The algorithm that computes $\lambda(F)$ for every subgraph F of G with no more than $m-1$ initial components can be described as follows:

1) $\lambda(\varnothing) = 0$
2) For $i = 1$ to n do:

For all subgraphs F of G with up to $m-1$ initial components and $|F| = i$ do:

$$\lambda(F) = 1 + \min_R \left\{ \lambda\left(H^e_{F'^s} \cup L_{F'^s}\right) \mid F \Rightarrow^R F' \right\}$$

where $|R| \le m$.

PROOF OF CORRECTNESS. A proof of correctness of the above algorithm follows from the following comments:

1) As follows from Definition 9, sets R *contain only initial nodes of F and at most one node from each of the initial bunches of F.* It is easy to see why only initial nodes of F are allowed to be scheduled in the first time slot of the schedule of F (nodes are allowed to be scheduled only after all their predecessors have been computed).

We next show why it is enough to consider only sets R that have at most one node from any initial bunch for the computation of the optimal schedule of F is more involved. In Theorem 3, we proved that there is a delay free transformation of F, namely F^s, that has an optimal schedule of the same length as the optimal schedule of F. But from Definition 2 we have that for every bunch $\{n_1, \ldots, n_k\}$ of F, F^s has some n_j as the parent of all the n_is for $i = 1, \ldots, j-1, j+1, \ldots, k$. But in this optimal schedule of F^s (and of F as well) only n_j is allowed to get scheduled when none of the $\{n_1, \ldots, n_k\}$ have been scheduled yet. This translates in the following: *F has an optimal schedule in which at most one of the nodes of each initial bunch of F is scheduled in the first time slot.* Hence, it is enough to consider only sets R that have at most one node from any initial bunch for the computation of the optimal schedule of F.

2) $\lambda(H^e_{F'^s} \cup L_{F'^s})$ *can be computed in* $O(1)$ *time.* Consider a certain set R and the resulting graph F'. Then from Claim 3 we know that given F', $H^e_{F'^s}$ and $L_{F'^s}$ can be computed in $O(1)$ time. But $H^e_{F'^s}$ is a subset of G with up to $m-1$ initial components itself (since it is the high subgraph of F'). Moreover, $\left|H^e_{F'^s}\right| \le |F'| < |F| = i$. So $\lambda(H^e_{F'^s})$ has been computed in an earlier step of the algorithm. But having $\lambda(H^e_{F'^s})$, we can compute $\lambda(F')$ in $O(1)$ time (see Theorem 4). So, for every given R, $\lambda(H^e_{F'^s} \cup L_{F'^s})$ can be computed in $O(1)$ time.
3) Among all the eligible sets R, we chose the one that minimizes $\lambda(F')$. It is then obvious that the total length of the schedule for F is going to be minimum and equal to $1 + \lambda(F')$.

Complexity issues: We can compute the time complexity of the algorithm presented above as follows:

1) There are $O(n)$ possible independent nodes in graph G and $O(n)$ different bunches (note that each bunch corresponds to a common parent; since there are n nodes in G, the number of different bunches has to be less than n). So there are $O(n)$ possible initial components for any graph F (total number of independent nodes and bunches).
2) There are $O(n^{m-1})$ different sets that contain *up to* $m-1$ initial components (total number of ways we can choose up to $m-1$ initial components out of the $O(n)$ available ones). Hence, Step 2a gets executed at most $O(n^{m-1})$ times (one time for each of the different subgraphs F of G with up to $m-1$ initial components; note that to each set of initial components corresponds one and only one subgraph of graph G and visa versa).
3) Every execution of Step 2a of the algorithm is of complexity $O(n^{m-1})$ because:
   a) For every different graph F of Step 2.a there are $O(n^{m-1})$ different sets R (total number of different ways to choose up to $m-1$ nodes out of the $O(n)$ initial nodes of F; note that F contains up to $m-1$ initial components but may contain $O(n)$ initial nodes).
   b) Given a certain R, F' is uniqly determined and it takes $O(1)$ time to compute $H^e_{F'^s}$ and $L_{F'^s}$ as mentioned in Claim 3.

   Having determined $H^e_{F'^s}$, one knows $\lambda(H^e_{F'^s})$ which has been computed in an earlier step (since $H^e_{F'^s}$, from Definition 5, only contains up to $m-1$ initial components and $|H^e_{F'^s}| \le |F'| < |F| = i$). Hence, for a specific R one can compute $\lambda(F')$ in $O(1)$ time using Theorem 4.
4) The whole algorithm is of $O(n^{m-1})O(n^{m-1}) = O(n^{2m-2})$ time complexity. □

REMARK. The algorithm presented in the above theorem, can be used to efficiently determine a delay free graph G^s that has an optimum schedule of the same length as the optimum schedule of the any given graph G.

In the following theorem a polynomial time algorithm is presented for the computation of the optimal schedule of a given precedence graph G.

THEOREM 6. *Given out-forest G, the optimal schedule for G can be computed in $O(n^{2m-2})$ time.*

PROOF. Applying the algorithm of Theorem 5, we can determine $\lambda(F)$ for every subgraph F, of G with no more than $m-1$ initial nodes in $O(n^{2m-2})$ time. Having computed $\lambda(H^e_{G^s})$ we can compute $\lambda = \lambda(H^e_{G^s} \cup L_{G^s})$ in $O(1)$ time (see Theorem 4). The schedule corresponding to the optimal length can be computed by the following recursive procedure:

SCHEDULE (G)

Step 1:

1) Determine $H^e_{G^s}$ and L_{G^s}.
2) Find R such that $H^e_{G^s} \Rightarrow^R G'$ and $\lambda(H^e_{G'^s})$ is minimum.
3) Schedule nodes of R in the first time slot.

Step 2:

1) SCHEDULE($H^e_{G'^s} \cup L_{G'^s}$)
2) SCHEDULE($H^e_{G^s}$) = R *appended with* SCHEDULE($H^e_{G'^s} \cup L_{G'^s}$)

Step 3: MERGE(SCHEDULE($H^e_{G^s}$), L_{G^s})

PROOF OF CORRECTNESS. SCHEDULE(G) algorithm presented above first computes $H^e_{G^s}$ and L_{G^s} subgraphs of G. Theorem 2 guarantees that if we find an optimum schedule for $H^e_{G^s}$, then we can find an optimum schedule for G using MERGE algorithm of Theorem 2. So all we need to prove is that the schedule that is computed for $H^e_{G^s}$ in steps 1 and 2 of the SCHEDULE(G) algorithm is optimum. Then in step 3 MERGE algorithm will be used to compute the optimum schedule for the whole graph G.

Any schedule for $H^e_{G^s}$ will have some nodes R scheduled in the first time slot and will have the rest of the graph $G' = H^e_{G^s} - (R)$ scheduled in later time slots. The total length of the schedule of $H^e_{G^s}$ then will be $1 + \lambda(G')$. So all we really need to guarantee is that the choice of R is such that $\lambda(G')$ is the minimum possible. But schedule of minimum length for graph G', according to Theorem 4, is the schedule that results in minimum $\lambda(H^e_{G'^s})$. So R has to be such that the resulting $H^e_{G'^s}$ has a schedule of minimum possible length. This is exactly how nodes R are chosen in step 1.b. The total length of the schedule for $H^e_{G^s}$ is $1 + \lambda(G') = 1 + \lambda(H^e_{G'^s} \cup L_{G'^s})$; hence the total length is minimum.

In Step 2 of the SCHEDULE algorithm we recursively compute the optimum schedule for G' which we append to the set R to find the optimal schedule for $H^e_{G^s}$. The latter we use in step 3 to compute the optimum schedule for the whole graph G.

The resulting schedule has divided the nodes into sets that correspond to different time slots. Using the algorithm ALLOCATE that is presented below we can decide on which node is going to be scheduled in which processor so that the communication delay constraints are not violated. ALLOCATE procedure takes the output of the SCHEDULE algorithm as its input. Suppose that $S(k)$ is the set of nodes scheduled in time slot k. Then ALLOCATE algorithm can be described as follows:

ALLOCATE (SCHEDULE (G))

1) Assign nodes $S(1)$ to processors at random.
2) For $k = 2$ to $\lambda(G)$ do:

For every node x in $S(k)$, *if* the parent of x has been scheduled in time slot $k-1$ *then* assign node x to the

same processor as its parent *else* assign node x at random.

The correctness of the ALLOCATE algorithm is obvious since at most one of the children of every node x is scheduled in the time slot immediately following the time slot in which x is scheduled. So the communication delay constraints are met (no communication delays are involved for dependent tasks scheduled in the same processor).

Complexity issues: Suppose that SCHEDULE(G) is of $T(n)$ complexity. Then the complexity of Step 1a of the algorithm is $O(n)$ (see Theorem 4), the complexity of Step 1b is $O(n^m)$, since there are $O(n^m)$ different ways to choose R (assuming that $\lambda(F)$ has already been computed for all the subgraphs F of G of no more than $m-1$ initial components). Step 2 is of at most $T(n-1)$ complexity and step 3 is of $O(n)$ complexity (as follows from Theorem 2). So $T(n) = T(n-1) + O(n^m) \Rightarrow T(n) = O(n^{m+1})$. Hence the total time complexity of step 2 is $O(n^{m+1})$, and the SCHEDULE algorithm takes $O(n^{m+1})$ time. ALLOCATE algorithm is of $O(n)$ complexity. Taking into consideration the time complexity of computing $\lambda(F)$ for all subgraphs F of G with no more than $m-1$ initial nodes we conclude that the total complexity of the algorithm is $O(n^{2m-2})$. □

4 A Linear Time Approximation Algorithm for Computing a Near-Optimal Schedule for Unit-Delay Out-Forests

In Section 3.1, we showed that for every out-forest G with communication delays, there exists a delay-free out-forest G^s such that the optimal schedule for G^s is also an optimal schedule for G. The rest of the previous section discussed how to determine such a delay-free transformation. In the process we had also introduced the concept of shortest delay-free transformations; the optimal schedule of the resultant out-forest, however, need not correspond to an optimal schedule of the parent out-forest. In this section we show that the length of an optimal schedule of a shortest delay-free forest does not exceed the optimal schedule of the underlying out-forest by more than $(m-2)$ time units. Since there are linear time algorithms for both determining a shortest delay-free forest, and for determining its optimal schedule, it follows that there exists a linear time algorithm for determining a near-optimal schedule for any given out-forest with unit communication and computation delays.

Following the construction in Section 3.2.1, we can define the height, $h(i)$, of a shortest delay-free tree rooted at node i as follows:

$$h(i) = \begin{cases} 1 & \text{if } i \text{ is a leaf node,} \\ 1 + \max\{h(j) \mid j \text{ is a child of node } i\} & \text{otherwise.} \end{cases}$$

We shall apply critical path scheduling for obtaining the optimal schedule for a shortest delay-free out-forest. When critical path scheduling is applied to a delay-free forest G_{df}, tasks are assigned to time slots, one slot at a time, from first to last. An unassigned task is said to be "available" for slot t if all of its predecessors have been assigned to prior slots. If no more than m tasks are available at t, all the available tasks are put in S_t. If more than m tasks are available, then m highest tasks are put in S_t, where the height of a node i, is the height of the subtree rooted at i, i.e., the length of a longest path from i to a leaf node. The following theorem first presented in [12] proves the optimality of the schedule and also establishes an important property that is used in the proof of Theorem 8.

THEOREM 7. *Critical path scheduling yields an optimal schedule for any delay-free out-forest G_{df}.*

We next establish the main result.

THEOREM 8. *If G^s is a shortest delay-free out-forest corresponding to a given out-forest G, then*

$$\lambda(G_s) \le \lambda(G) + (m-2).$$

PROOF. Let S be an optimal schedule for G^s.

Case 1: All time slots, other than the first and last, are completely filled. In this case the schedule is clearly optimal for both G^s and G.

Case 2: There is a time slot, other than the first or last, that is incompletely filled. Let t be the latest such time slot. From the proof of Theorem 7, we know that each node in this slot has a continuous chain of predecessors executed in slots $t-1, t-2, \ldots$, back to slot 1. Furthermore, at least one node in slot t must have height $h(i) \ge 2$ in G^s, else t would be the last nonempty slot in S. Suppose node i has a predecessor k in G^s that is not the favored child of its parents in G, where k is in slot $t' < t$. Then node j, the predecessor of k in G^s, is the favored child of the parent of both j and k in G and j is in slot $t'-1$. This implies that $h(j) \ge h(k)$ (since G^s is a shortest delay-free out-forest), and that j has a successor in G^s, different from i, in slot t. It follows that node i has at most $m-2$ predecessors that are not favored children. Hence node i is executed at most $m-2$ time units later than it could possibly be executed in any feasible schedule for G. It follows that S itself is no more than $m-2$ time units longer than an optimal schedule for G. □

5 Conclusion

In this paper, we have presented polynomial time algorithms for optimally scheduling n dependent tasks in a system of m processors when the precedence graph is in the form of Out-forest or In-forest, m is a variable of the problem and bounded by a constant, the tasks are of unit computational delay and the communication between the processors is taken into consideration and is assumed to be of unit delay. We also presented a linear-time algorithm for computing a near-optimal schedule for a unit-delay out-forest. Some interesting problems that could follow as possible extensions of the results presented in this paper are:

1) The nonstraight profile scheduling problem with communication delays, meaning the scheduling problem when the number of processors available at each time step varies but is still bounded by a constant.
2) Scheduling problem with communication delays under the assumption of non identical processors; for example an interesting generalization of the scheduling problem would be the case where the available processors are of different computational capacity.
3) Scheduling with communication delays in more general precedence graphs.

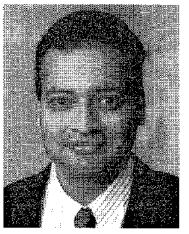
ACKNOWLEDGMENTS

This work was supported in part by the SDIO/IST U. S. Army Research Office through Contract DAAL03-87-K-0033.

REFERENCES

[1] F.D. Anger, J.-J. Hwang, and Y.-C. Chow, "Scheduling with Sufficient Loosely Coupled Processors," *J. Parallel and Distributed Computing*, vol. 9, pp. 87–92, 1990.
[2] P. Chretienne, "A Polynomial Algorithm to Optimally Schedule Tasks on a Virtual Distributed System Under Tree-Like Precedence Graphs," *European J. Operational Research*, vol. 43, pp. 225–230, 1988.
[3] P. Chretienne, "Task Scheduling Over Distributed Memory Machines," *Proc. Int'l Workshop Parallel and Distributed Algorithms*, 1988.
[4] J. Colin and P. Chretienne, "Cpm Scheduling with Small Communication Delays and Task Duplication," Technical Report M.A.S.I 90.1, Université Pierre et Marie Curie, France, Jan. 1990.
[5] D. Dolev and M.K. Warmuth, "Profile Scheduling of Opposing Forests and Level Orders," *SIAM J. Algorithms and Discrete Methods*, vol. 6, pp. 87–92, Oct. 1985.
[6] M. Fujii, T. Kasami, and K. Ninomiya, "Optimal Scheduling of Two Equivalent Processors," *SIAM J. Applied Math.*, vol. 17, pp. 784–789, 1969.
[7] H.N. Gabow, "A Linear Time Recognition Algorithm for Interval DAGs," *Information Processing Letters*, vol. 12, pp. 20–22, 1981.
[8] H.N. Gabow, "An Almost Linear Algorithm for Two Processor Scheduling," *J. ACM*, vol. 29, pp. 766–780, 1982.
[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979.
[10] M.R. Garey, D.S. Johnson, R.E. Tarjan, and M. Yannakakis, "Scheduling Opposing Forests," *SIAM J. Algorithms and Discrete Methods*, vol. 4, pp. 72–93, Mar. 1983.
[11] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals Discrete Math.*, vol. 5, pp. 287–326, 1979.
[12] T.C. Hu, "Parallel Sequencing and Assembly Line Problems," *Operations Research*, vol. 9, pp. 841–848, 1961.
[13] E.W. Mayr, "Well Structured Parallel Programs Are Not Easy to Schedule," Technical Report STAN-CS-81-880, Stanford Univ., Stanford, Calif., 1981.
[14] C.H. Papadimitriou and M. Yannakakis, "Scheduling Interval Order Tasks," *SIAM J. Computing*, vol. 10, pp. 405–409, 1979.
[15] C.H. Papadimitriou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. Computing*, vol. 19, no. 2, pp. 322–328, Apr. 1990.
[16] J.D. Ullman, "NP-Complete Scheduling Problems," *J. Computer and Systems Science*, vol. 10, pp. 384–393, 1975.
[17] T.A. Varvarigou, "Scheduling and Fault Tolerance Issues in Multiprocessor Systems," PhD thesis, Stanford Univ., Stanford, Calif., Dec. 1991.
[18] T. Varvarigou, V.P. Roychowdhury, and T. Kailath, "Scheduling In and Out Forests in the Presence of Communication Delays," *Proc. Int'l Parallel Processing Symp.* (IPPS '93), pp. 222–229, Newport Beach, Calif., Apr. 1993.
[19] M.K. Warmuth, "Scheduling in Profiles of Constant Breadth," PhD thesis, Dept. of Computer Science, Univ. of Colorado, Boulder, Aug. 1981.

Theodora A. Varvarigou received the BTech degree from the National Technical University of Athens, Greece, in 1988, the MS degree from Stanford University, Stanford, California, in 1989, and the PhD degree from Stanford University in 1991. She is currently an assistant professor at the Technical University of Crete, Greece. From December 1991 until December 1994, she was a member of the technical staff at AT&T Bell Labs, Holmdel, New Jersey. Her research interests include parallel algorithms and architectures, fault-tolerant system design, and parallel scheduling on multiprocessor systems.



Vwani P. Roychowdhury received the BTech degree from the Indian Institute of Technology, Kanpur, India, and the PhD degree from Stanford University, Stanford, California, in 1982 and 1989, respectively, both in electrical engineering.

He is currently a professor in the Department of Electrical Engineering at the University of California, Los Angeles. From August 1991 until June 1996, he was a faculty member at the School of Electrical and Computer Engineering at Purdue University. His research interests include parallel algorithms and architectures, computational paradigms for nanoelectronics, design and analysis of neural networks, special purpose computing arrays, VLSI design, and fault-tolerant computation.

Thomas Kailath was educated in Poona, India, and at the Massachusetts Institute of Technology. Until December 1962, he worked at the Jet Propulsion Laboratories, Pasadena, California, and also taught part-time at the California Institute of Technology. He then went to Stanford University as an associate professor of electrical engineering. He served as director of the Information Systems Laboratory during a period of rapid growth from 1971 through 1980, as associate chairman of the department until 1987, and was then appointed the first holder of the Hitachi America Professorship of Engineering. Prof. Kailath has also held shorter-term appointments at several institutions around the world, including AT&T Bell Laboratories, University of California at Berkeley, the Indian Institute of Science, Cambridge University, Delft University, and the Massachusetts Institute of Technology.

His research has spanned a large number of disciplines, emphasizing information theory and communication in the 1960s; linear systems, estimation, and control in the 1970s; and VLSI design and sensor array signal processing in the 1980s. Alongside these have been contributions to several fields of applied mathematics. His current interests emphasize applications of signal processing, computation, and control to problems in manufacturing and telecommunications.

Prof. Kailath serves on the editorial boards of more than a dozen engineering and mathematics journals, and has been editor of the Prentice Hall Series in Information and System Sciences since 1963. He is an author of nearly 300 papers, two textbooks, and several research monographs and patents. He is also active in several professional societies. He has received awards from the IEEE Information Theory Society, which he served as president in 1975, the IEEE Signal Processing Society, and the American Control Council. He is the recipient of the 1995 IEEE Education Medal. He has held Guggenheim and Churchill fellowships, among others, and has honorary doctorates from Linkoping University in Sweden and Strathclyde University in Scotland. He is a fellow of the IEEE and of the Institute of Mathematical Statistics, and a member of the National Academy of Engineering, the American Academy of Arts and Sciences, and the Third World Academy of Sciences.

Eugene Lawler obtained an AM at Harvard University in 1957, and was a senior electrical engineer at Sylvania Electric Products in Needham, Massachusetts, from 1959 until 1961 when he returned to Harvard to obtain a PhD in 1962. He taught at the University of Michigan, Ann Arbor, from 1962-1970. He was a member of the faculty at the University of California at Berkeley from 1971 until his death in 1995.

Dr. Lawler studied algorithmic issues in combinatorial optimization for more than 30 years. His contributions were fundamental in giving the discipline the breadth and depth it has obtained. He was the author of the textbook *Combinatorial Optimization: Networks and Matroids* (1976). He was a coeditor of *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (1985) which became benchmark reference. He was also the coauthor of a classic paper on branch-and-bound (with D.E. Wood) and of one on dynamic programming (with J.M. Moore).