SIAM J. COMPUT.
Vol. 18, No. 2, pp. 244-257, April 1989

SCHEDULING PRECEDENCE GRAPHS IN SYSTEMS WITH INTERPROCESSOR COMMUNICATION TIMES*

JING-JANG HWANG†, YUAN-CHIEH CHOW‡, FRANK D. ANGER§, AND CHUNG-YEE LEE‡

Abstract. The problem of nonpreemptively scheduling a set of m partially ordered tasks on n identical processors subject to interprocessor communication delays is studied in an effort to minimize the makespan. A new heuristic, called Earliest Task First (ETF), is designed and analyzed. It is shown that the makespan ω_{ETF} generated by ETF always satisfies $\omega_{\mathrm{ETF}} \leqq (2-1/n)\omega_{\mathrm{opt}}^{(i)} + C$, where $\omega_{\mathrm{opt}}^{(i)}$ is the optimal makespan without considering communication delays and C is the communication requirements over some immediate predecessor-immediate successor pairs along one chain. An algorithm is also provided to calculate C. The time complexity of Algorithm ETF is $O(nm^2)$.

Key words. multiprocessor scheduling, worst-case analysis, communication delays

AMS(MOS) subject classifications. 68Q20, 68Q25

1. Introduction. An extensively studied problem in deterministic scheduling theory is that of scheduling a set of partially ordered tasks on a nonpreemptive multiprocessor system of identical processors in an effort to minimize the overall finishing time, or "makespan." So much literature has been produced in the related area that a number of review articles have been published, including excellent summaries by Coffman [1], Graham et al. [3], and Lawler, Lenstra, and Rinnooy Kan [6]. The underlying computing system of this classical problem is thought to have no interprocessor overhead such as processor communication or memory contention. Such an assumption is a reasonable approximation to some real multiprocessor systems; therefore, applications can be found for the derived theory [5]. The assumption, however, is no longer valid for message-passing multiprocessors or computer networks, since interprocessor communication overhead is clearly an important aspect in such systems and is not negligible.

In this paper, interprocessor communication overhead is made part of the problem formulation and corresponding solutions are derived. The augmented multiprocessing model starts with a given set $\Gamma = \{T_1, T_2, \cdots, T_m\}$ of m tasks, each with processing time $\mu(T_i)$, and a system of n identical processors. The tasks form a directed acyclic graph (DAG) in which each edge represents the temporal relationship between two tasks and is associated with a positive integer $\eta(T, T')$, the number of messages sent from an immediate predecessor T to an immediate successor T' upon the termination of the immediate predecessor. The task model is called an "enhanced directed acyclic graph (EDAG)" and is denoted as a quadruple $G = G(\Gamma, \rightarrow, \mu, \eta)$.

To characterize the underlying system, a parameter $\tau(P, P')$ is introduced to represent the time to transfer a message unit from processor P to P'. The system is then denoted as $S = S(n, \tau)$, where n is the number of identical processors. By varying the values of $\tau(P, P')$, the system model can be used to model several types of systems such as a fully connected system, a local area network, or a hypercube. To accommodate the deterministic scheduling approach, we further assume that the communication subsystem is contention free. Mathematically speaking, the time to take $\eta(T, T')$ units of messages from P to P' is $\tau(P, P') \times \eta(T, T')$, a deterministic value. Figure 1 shows an example of the computing model and a feasible schedule.

* Received by the editors August 31, 1987; accepted for publication (in revised form) June 13, 1988.

† National Chiao Tung University, Hsinchu, Taiwan 30049, Republic of China.

‡ Department of Computer Science, University of Florida, Gainesville, Florida 32611.

§ Computer Science Department, Florida Institute of Technology, Melbourne, Florida 32901.

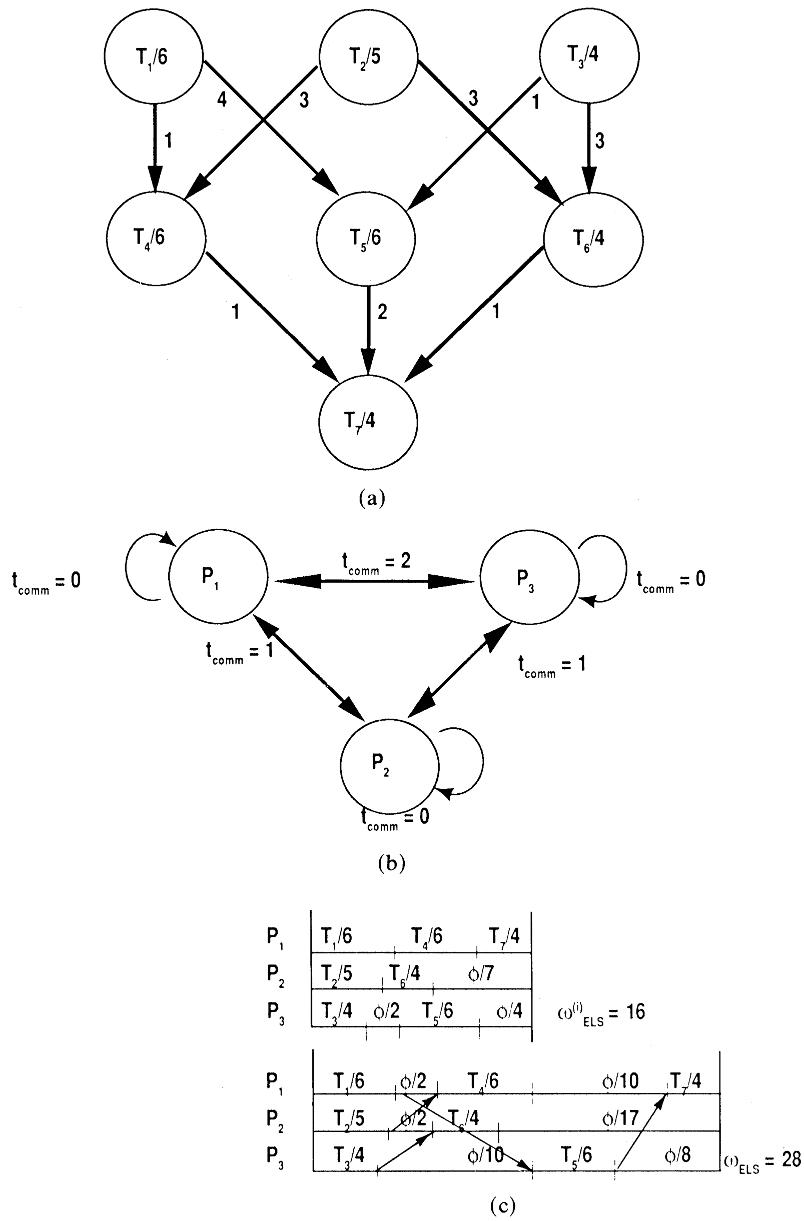


FIG. 1. *An example.* (a) *The* EDAG *model*; (b) *the system model*; (c) *schedules with and without communication delays.*

The computing model described above is an extension of Rayward-Smith's earlier model [7], which was confined to unit communication times (UCT) and unit execution times (UET). He shows that the problem of finding the minimum makespan is NP-complete and also presents a heuristic. The heuristic, called "generalized list scheduling," adopts the same greedy strategy as Graham's list scheduling [2]: No processor remains idle if there is some task available that it could process. For UET and UCT models, a task T can be processed on the processor P_i at time t if T has no immediate predecessors, or each immediate predecessor of T has been scheduled (to start) on processor P_i at time $\leqq t-1$ or on processor $P_j \neq P_i$ at time $\leqq t-2$. A schedule using Rayward-Smith's heuristic always satisfies

$$\omega_g^c \leqq \left(3-\frac{2}{n}\right) \times \omega_{\mathrm{opt}}^c - \left(1-\frac{1}{n}\right), \tag{1.1}$$

where ω_g^c is the length of the greedy schedule and ω_{opt}^c is the length of the optimal schedule.

This work generalizes Rayward and Smith's assumptions on communication times and execution times. The main result offers a new heuristic and its performance bound. The method is named "Earliest Task First (ETF)" and was first presented in Hwang's dissertation [4]. This paper simplifies the presentation, reduces its time complexity, and improves the performance bound. The main result indicates that the length of an ETF schedule is bounded by the sum of Graham's bound for list scheduling and the "communication requirement" over some immediate predecessor-immediate successor pairs along one chain that can be calculated using Algorithm C of § 3. In notation this is expressed as follows:

$$\omega_{\mathrm{ETF}} \leqq \left(2-\frac{1}{n}\right)\omega_{\mathrm{opt}}^{(i)} + C. \tag{1.2}$$

In (1.2), $\omega_{\mathrm{opt}}^{(i)}$ denotes the optimal schedule length obtained by ignoring the inter-processor communication and is different from ω_{opt}^c in (1.1). The term $(2-1/n)\omega_{\mathrm{opt}}^{(i)}$ is the Graham bound for list scheduling [2].

When ETF is applied to a problem with UET and UCT assumptions, its schedule length satisfies

$$\omega_{\mathrm{ETF}} \leqq \left(3-\frac{1}{n}\right) \times \omega_{\mathrm{opt}}^{(i)} - 1 \tag{1.3}$$

since $C \leqq \omega_{\mathrm{opt}}^{(i)} - 1$. Both bounds ((1.1) and (1.3)) are close since 3 is the dominant factor in both expressions; but we should note that the bound on ETF uses $\omega_{\mathrm{opt}}^{(i)}$, which is smaller than ω_{opt}^c in (1.1), as the base multiplier.

2. ETF—A new heuristic.

Section 2.1 presents a simple approach, named ELS, for solving the scheduling problem formulated in § 1. ELS (extended list scheduling) is a straightforward extension of Graham's LS (list scheduling) method. The unsatisfactory performance of ELS motivates the development of ETF.

2.1. ELS—a simple solution.

The ELS method adopts a two-phase strategy. First, it allocates tasks to processors by applying LS as if the underlying system were free of communication overhead. Second, it adds necessary communication delays to the schedule obtained in the first phase.

Graham's worst-case bound of LS as stated in (2.1) [2] provides a basis for deriving a similar bound for ELS:

$$\omega_{\mathrm{LS}}^{(i)} \leqq \left(2-\frac{1}{n}\right)\omega_{\mathrm{opt}}^{(i)}. \tag{2.1}$$

It is obvious that the difference between an ELS and an LS schedule, using the same task list L, is bounded by the total maximum communication requirement:

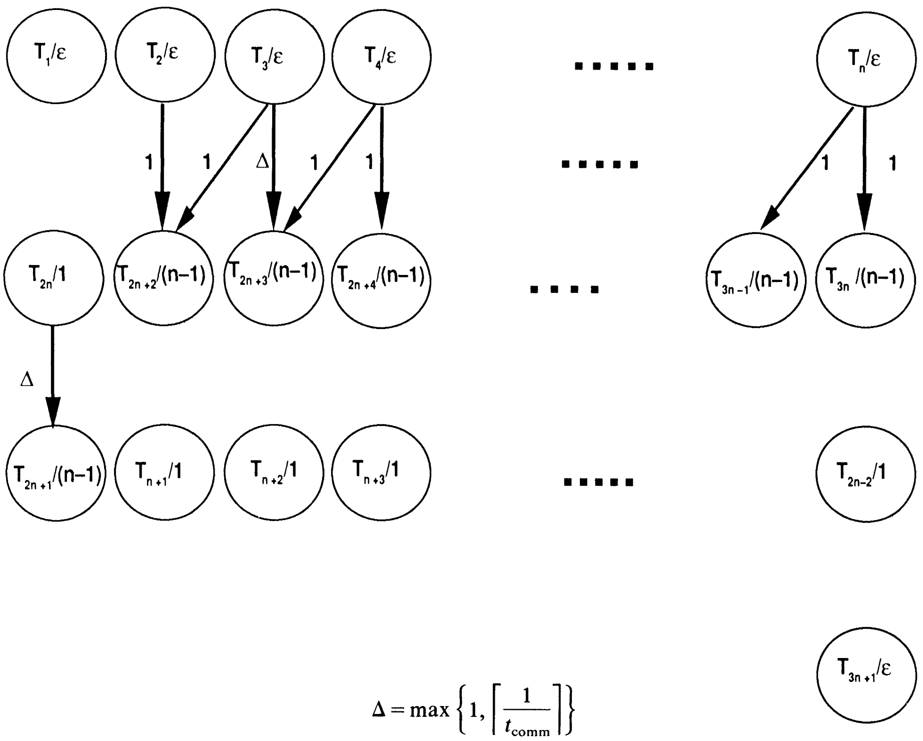
$$\omega_{\mathrm{ELS}}(L) - \omega_{\mathrm{LS}}^{(i)}(L) \leqq \tau_{\max} \times \sum_{T\in\Gamma, T'\in S_T} \eta(T, T'), \tag{2.2}$$

where $\tau_{\max} = \max\{\tau(P, P')\}$ is used to calculate the maximum communication requirements, and S_T denotes the set of immediate successors of T. We shall also use D_T to denote the set of immediate predecessors of T in later sections. Combining (2.1) and (2.2), we obtain (2.3):

$$\omega_{\mathrm{ELS}} \leqq \left(2 - \frac{1}{n}\right)\omega_{\mathrm{opt}}^{(i)} + \tau_{\max} \times \sum_{T\in\Gamma, T'\in S_T} \eta(T, T'). \tag{2.3}$$

The question raised here is whether the bound shown in (2.3) is the best possible. It is not hard to show that all maximum communication requirements may actually produce communication delays. However, to show that the bound stated in (2.3) is tight, it is necessary to produce an ELS schedule in which the schedule of computational tasks is the worst and, at the same time, almost no communications can be overlapped with the computations in the same schedule. This is not obvious since, in general, a longer schedule will allow more overlapping between computation and communication. Nevertheless, the example in Fig. 2 shows that the bound stated in (2.3) is asymptotically achievable as ε and δ approach zero. We summarize the result in Theorem 2.1.

THEOREM 2.1. *Any* ELS *schedule is bounded by the sum of Graham's bound and the total communication requirement* ((2.3)). *Furthermore, this bound is the best possible.*



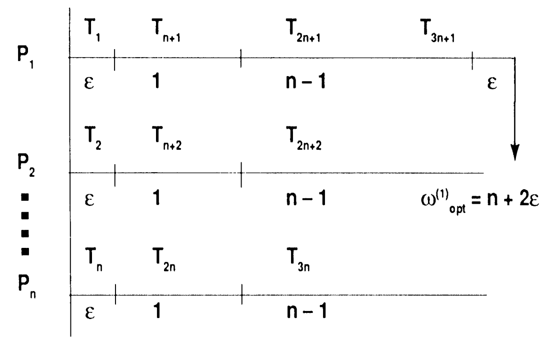
$$\Delta = \max\left\{1, \left\lceil \frac{1}{t_{\mathrm{comm}}} \right\rceil\right\}$$

(a)

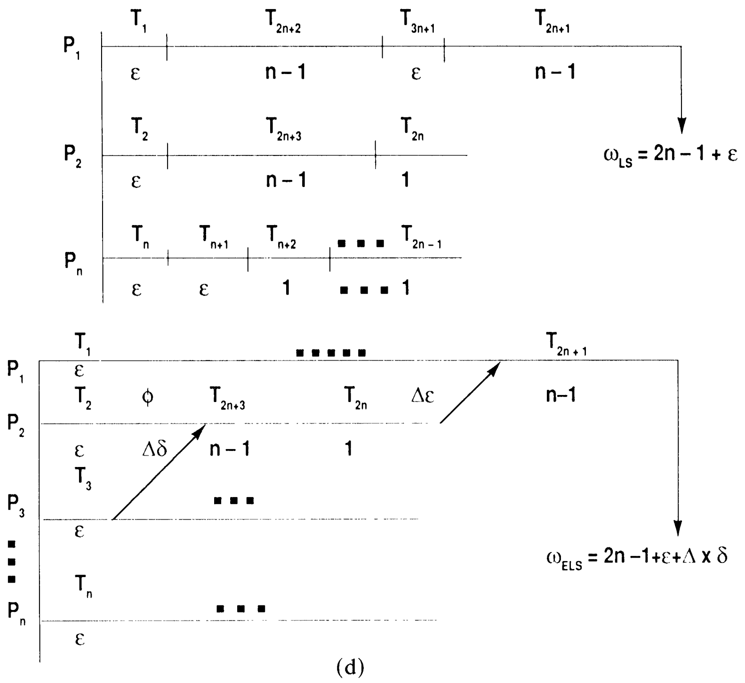
$S(n, t_{\mathrm{comm}})$
where $t_{\mathrm{comm}} = \delta$ for all $P_i \neq P_j$

(b)

FIG. 2. *A worst-case example for Theorem* 2.1. (a) *The* EDAG *model*; (b) *the system model*; (c) *an optimal schedule on an ideal system*; (d) *an* LS *schedule and its corresponding* ELS *schedule.*



(c)



(d)

FIG. 2—continued

2.2. ETF—a new heuristic. The performance of ELS is unsatisfactory when dealing with communication delays, as indicated by Theorem 2.1. ETF, the core of this paper, has a much better performance bound. ETF adopts a simple greedy strategy: the earliest schedulable task is scheduled first. The algorithm is event-driven and is to be described in detail in the following paragraphs.

A task is called *available* when all its predecessors have been scheduled. Let A and I be the sets of available tasks and free processors, respectively, with $A = \{T: D_T = \varnothing\}$ and $I = \{P_1, \cdots, P_n\}$ initially. The starting time of T is denoted as $s(T)$; the finishing time is $f(T)$; the processor to which T is assigned is $p(T)$. These three values for all tasks are the output of the scheduling algorithm.

The starting time of an available task is determined by several factors: when its preceding tasks are finished, how long the communication delays take, and where the task and its predecessors are allocated. Let $r(T, P)$ denote the time the last message for T arrives at processor P; mathematically,

$$r(T, P)=\begin{cases} 0 & \text{if } T \text{ has no predecessors,} \\ \max\limits_{T' \in D_T} \{f(T')+\eta(T', T)\times \tau(p(T'), P)\}. \end{cases} \tag{2.4}$$

Then the earliest starting time of an available task T can be calculated by

$$e_s(T)=\max\{CM, \min_{P\in I}\{r(T, P)\}\} \tag{2.5}$$

where CM, called "current moment," denotes the curent time of the event clock. By definition, I is the set of free processors at time = CM. Now, the earliest starting time among all available tasks, $e_s^\wedge$, is calculated as follows:

$$e_s^\wedge=\min_{T\in A}\{e_s(T)\}. \tag{2.6}$$

The greedy strategy of ETF wants to find $\hat{T}\in A$ and $\hat{P}\in I$ such that $e_s^\wedge = e_s(\hat{T}) = \max\{\text{CM}, r(\hat{T}, \hat{P})\}$.

Since we assume arbitrary communication delays in our model, a newly available task after the event clock is advanced may have an earlier starting time than that of the select task $\hat{T}$. To overcome such a difficulty, a second time variable, called "next moment" and abbreviated as NM, is introduced to keep track of the scheduling process. NM denotes the earliest time after CM at which one or more currently busy processors become free. NM is set to ∞ if all processors are free after CM. It is clear that the scheduler will not generate any available task that can be started earlier than $e_s^\wedge$ if $\text{NM} \geqq e_s^\wedge$. The scheduling decision will be made in this case; otherwise the event clock is advanced and the decision is postponed.

Conflicts may occur during the scheduling process when two available tasks have the same starting time. To resolve the conflicts, we adopt a priority task list L as we did in LS and ELS methods. Such a strategy is called ETF/LS. If the priority list L is obtained through the critical-path analysis, then the method is called ETF/CP. Algorithm ETF below is a simple ETF implementation in which conflicts are resolved arbitrarily.

Example. Consider the same EDAG task graph and the same three-processor system as given in Fig. 1. The ETF schedule is presented in Fig. 3. Table 1 shows the detail of the scheduling process. In Table 1, i denotes ith execution of the inner loop of executing Algorithm ETF when it is applied to the given problem. The values of CM and NM at the beginning of ith execution of the inner loop are denoted as cm(i) and nm(i), respectively. $\hat{T}$ is the available task selected when CM = cm(i) and NM = nm(i); $\hat{T}$ = NONE means no task available at the current moment. The scheduling

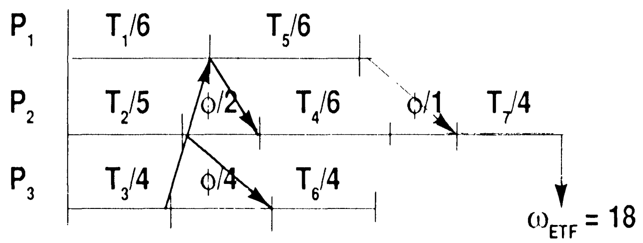


FIG. 3. *An* ETF *schedule.*

TABLE 1

I	CM (I)	NM (I)	$\hat{T}$	Decision
1	0	∞	T_1	made
2	0	6	T_2	made
3	0	5	T_3	made
4	0	4	None	
5	4	5	None	
6	5	6	T_6	postponed
7	6	∞	T_5	made
8	6	12	T_4	made
9	6	12	T_6	made
10	6	12	None	
11	12	13	None	
12	13	∞	T_7	made
13	All tasks have been scheduled			

decision of $\hat{T}$ is made if $e_s^\wedge \leqq \mathrm{nm}(i)$ and postponed otherwise.

ALGORITHM ETF.

```
(0) Initialize: $I \leftarrow \{P_1, \cdots, P_n\}$, $A \leftarrow \{T: D_T = \emptyset\}$, $Q \leftarrow \emptyset$, CM $\leftarrow 0$, and NM $\leftarrow \infty$,
    $r(T, P) \leftarrow 0$ for each $T \in A$ and each $P \in I$.
(1) While $|Q| < m$ Do
    begin
    (1.1) While $I \neq \emptyset$ and A $\neq \emptyset$ Do
          begin
          (1.1.1) Find $\hat{T} \in A$ and $\hat{P} \in I$ such that
                  $\min_{T \in A} \min_{P \in I} r(T, P) = r(\hat{T}, \hat{P}) =$ temp, let $e_s^\wedge = \max\{\mathrm{CM}, \mathrm{temp}\}$.
          (1.1.2) If $e_s^\wedge \leqq$ NM
                  then
                  Assign $\hat{T}$ to run on $\hat{P}$: $p(\hat{T}) \leftarrow \hat{P}$; $s(\hat{T}) \leftarrow e_s^\wedge$; $f(\hat{T}) \leftarrow s(\hat{T}) + \mu(\hat{T})$,
                  $A \leftarrow A - \{\hat{T}\}$, $I \leftarrow I - \{\hat{P}\}$, append $\hat{T}$ to $Q$,
                  if $f(\hat{T}) \leqq$ NM then NM $\leftarrow f(\hat{T})$
                  else exit the inner loop.
          end
    (1.2) Proceed: CM $\leftarrow$ NM, find new NM.
    (1.3) Repeat for each $T$, $P$ such that $T$ just finished on $P$:
          $I \leftarrow I + \{P\}$, repeat for each $T' \in S_T$:
            $d_{T'} = d_{T'} - 1$; if $d_{T'} = 0$ then $T'$ is newly available
    (1.4) For each newly available task $T'$ do
          $A \leftarrow A + \{A'\}$,
          for each processor $P$, $r(T', P) = \max_{T \in D_{T'}} \{f(T) + \eta(T, T') \times \tau(p(T), P)\}$
      end.
```

3. Analysis of ETF. Some properties of ETF are to be established in this section. The time complexity is analyzed first, followed by the three lemmas that help to explain the way ETF works. The establishment of a performance bound concludes this section.

THEOREM 3.1. *The time complexity of* ETF *is* $O(\mathrm{nm}^2)$, *where n and m are the number of processors and the number of tasks, respectively.*

Proof. The ETF algorithm consists of a main loop containing an inner loop. It is obvious that the main loop repeats at most m times since some task "finishes" in step 1.2 each time. The running time depends on the number of repetitions of the inner loop. We observe that there are two mutually exclusive possibilities after executing an iteration of the inner loop: (1) When $e_s^{\wedge} \leqq \mathrm{NM}$, the selected available task $\hat{T}$ was scheduled. (2) When $e_s^{\wedge} > \mathrm{NM}$ or $I = \emptyset$ or $A = \emptyset$, the inner loop was exited. For the latter case, the CM was moved forward to NM (by executing step 1.2, which follows the inner loop) and at least one task was finished at the new CM. Consequently, the inner loop repeats at most $2m$ times (since the algorithm either schedules a task or finishes at least one task for each iteration of the inner loop).

Each task becomes newly available once during the scheduling process; therefore, the total running time of step 1.4 is $O(\mathrm{nm}^2)$. The most time-consuming step is generally step 1.1.1, which takes $O(\mathrm{nm})$ for each execution and $O(\mathrm{nm}^2)$ for at most $2m$ executions. □

We introduce some conventions here. Let z be the number of iterations of statement 1.1, and let i be an index of a particular iteration, $i \leqq z$. By an execution of statement 1.1, we mean the execution of the whole inner loop. When $I = \emptyset$ or $A = \emptyset$, the statement 1.1 is also considered being executed once, though the substatements 1.1.1 and 1.1.2 are skipped. Let the sequences $\mathrm{cm}(1), \mathrm{cm}(2), \cdots, \mathrm{cm}(z)$ and $\mathrm{nm}(1), \mathrm{nm}(2), \cdots, \mathrm{nm}(z)$ denote values carried by variables CM and NM, respectively, at the beginning of each execution of statement 1.1. If the scheduling decision of a task T is made during the ith execution of statement 1.1, then i is called the *decision order* of T and denoted as $\mathrm{do}(T)$, and $\mathrm{cm}(i)$ is called the *decision time* of T and denoted as $\mathrm{dt}(\hat{T})$. Note that not every integer i, $1 \leqq i \leqq z$, is a decision order of some task. We define $e_s(i)$ as the earliest starting time among all available tasks while CM carries the value $\mathrm{cm}(i)$; $e_s(T, i)$ is similarly defined for each available task T and is calculated by (2.5).

LEMMA 1. *For any task T, $\mathrm{dt}(T) \leqq s(T)$ holds and there exists no task T' such that $\mathrm{dt}(T) < f(T') < s(T)$ or $\mathrm{dt}(T) < \mathrm{dt}(T') < s(T)$.*

Proof. The first part is obvious since each task was scheduled to start at a time no less than the value of the current moment at which the decision is made (see statements 1.1.1 and 1.1.2 of Algorithm ETF).

Suppose that T' is a task such that $\mathrm{dt}(T) < f(T') < s(T)$. Let $\mathrm{do}(T) = i$ and $\mathrm{do}(T') = j$. We have four cases:

(1) T was scheduled after $T'(i > j)$. We get $\mathrm{nm}(i) \leqq f(T')$ by definition and thus, $\mathrm{nm}(i) \leqq f(T') < s(T)$ follows. According to the decision principle (statement (1.1.2)), T should not be scheduled at the ith execution of statement 1.1. This contradicts $\mathrm{do}(T) = i$.

(2) T was scheduled before $T'(i < j)$ and T' has no predecessors. T' became available from the first execution of statement 1.1. Thus, $e_s(T, i) = s(T) > f(T') > s(T') = e_s(T', j) \geqq e_s(T', i)$. (The last inequality is due to the fact that the earliest starting time of a task is nondecreasing after it becomes available, since CM is nondecreasing (see (2.25)). The result $e_s(T, i) > e_s(T', i)$ implies that T' instead of T should have been selected in the ith execution of statement 1.1. This leads to a contradiction.

(3) T was scheduled before T' and $f(T^*) \leqq \mathrm{cm}(i)$, where T^* is a last finished predecessor of T'. The assumption $f(T^*) \leqq \mathrm{cm}(i)$ implies T' became available before the ith execution of statement 1.1. Thus, the same argument for case (2) leads to a contradiction.

(4) T was scheduled before T' and $f(T^*) > \mathrm{cm}(i)$, where T^* is a last finished predecessor of T'. For this case, we have $\mathrm{dt}(T) < f(T^*) < s(T)$. We replace T' by T^*, find the last predecessor of the new T', called T^* again, and repeat the same argument. One of the four cases may happen. If one of cases (1), (2), and (3) occurs, then we are done; otherwise the process is continued. Since a new task T^* with earlier finish

time is obtained in each cycle, we will reach the case $f(T^*) \leqq \mathrm{cm}\,(i)$ in a finite number of cycles if case (4) occurs repeatedly. This concludes the proof of the second part.

Suppose now that $\mathrm{dt}\,(T) < \mathrm{dt}\,(T') < s(T)$. We can find a task T^* such that $f(T^*) = \mathrm{dt}\,(T')$ since a decision is made only at some cm at which a task is finished. This contradicts the second part of this lemma. □

LEMMA 2. *If* $s(T) < s(T')$, *then* $\mathrm{do}\,(T) < \mathrm{do}\,(T')$ *and* $\mathrm{dt}\,(T) \leqq \mathrm{dt}\,(T')$.

Proof. Suppose $s(T) < s(T')$, $\mathrm{do}\,(T) = i$, $\mathrm{do}\,(T') = j$, $i > j$. Since the current-moment sequence is nondecreasing, two possibilities can be discussed: (1) $\mathrm{dt}\,(T) > \mathrm{dt}\,(T')$, or (2) $\mathrm{dt}\,(T) = \mathrm{dt}\,(T')$. For (1), we apply the first part of Lemma 1 and get $\mathrm{dt}\,(T') < \mathrm{dt}\,(T) \leqq s(T) < s(T')$. This contradicts the second part of Lemma 1. For (2) T and T' are available at $\mathrm{cm}\,(i) = \mathrm{cm}\,(j)$, and $e_s(T', j) = s(T') > s(T) = e_s(T, i) \geqq e_s(T, j)$. During the jth execution of statement 1.1, T should have been selected instead of T. This is a contradiction. □

LEMMA 3. *Let T and T' be two tasks satisfying* $f(T) < \mathrm{dt}\,(T')$. *Then there exists an integer i such that* $\mathrm{cm}\,(i) = f(T)$. *In other words, the current-moment sequence cannot skip any time point at which a task is finished until all tasks have been scheduled.*

Proof. Let $\mathrm{do}\,(T') = j$. Then $0 = \mathrm{cm}\,(1) \leqq f(T) < \mathrm{cm}\,(j)$. Since the current-moment sequence is nondecreasing, an index k can be found such that $\mathrm{cm}\,(k) \leqq f(T) < \mathrm{cm}\,(k+1)$. If $\mathrm{cm}\,(k) = f(T)$, then we are done. If $\mathrm{cm}\,(k) < f(T) < \mathrm{cm}\,(k+1)$ and T is not a task scheduled when $\mathrm{CM} = \mathrm{cm}\,(k)$, then T is a task whose scheduling decision was made earlier. Then $f(T) < \mathrm{cm}\,(k+1) = \mathrm{nm}\,(k)$ contradicts $\mathrm{nm}\,(k) \leqq f(T)$. If $\mathrm{cm}\,(k) < f(T) < \mathrm{cm}\,(k+1)$ and T is a task scheduled at *cm* (k), then NM should have been moved backward to $f(T)$, and thus $\mathrm{cm}\,(k+1) = \mathrm{nm}\,(k) = f(T)$ after the scheduling of T. This is also a contradiction. □

THEOREM 3.2. *For any* EDAG *task model* $G = G(\Gamma, \rightarrow, \mu, \eta)$ *to be scheduled on a system* $S = S(n, \tau)$, *the schedule length,* ω_{ETF}, *obtained by* ETF *always satisfies*

$$\omega_{\mathrm{ETF}} \leqq \left(2 - \frac{1}{n}\right) \times \omega_{\mathrm{opt}}^{(i)} + C_{\max} \tag{3.1}$$

where $C_{\max}$ is the maximum communication requirement along all chains in Γ. That is,

$$C_{\max} = \max \left\{ \tau_{\max} \times \sum_{i=1}^{l-1} \eta(T_{c_i}, T_{c_{i+1}}) \colon (T_{c_1}, \cdots, T_{c_l}) \text{ is a chain in } \Gamma \right\}. \tag{3.2}$$

Proof. The set of all points of time in $(0, \omega_{\mathrm{ETF}})$ can be partitioned into two subsets A and B. A is defined to be the set of all points for which all processors are executing some tasks. B is defined to be the set of all points of time for which at least one processor is idle (maybe all processors are idle due to simultaneous communication delays). If B is empty, then all processors complete their last assignment at ω_{ETF} and no idle interval can be found with $(0, \omega_{\mathrm{ETF}})$. The ETF schedule is indeed optimal and thus the theorem holds obviously. We thus assume B is nonempty. In the interest of mathematical rigor, we suppose B is the disjoint union of q open intervals (b_{l_i}, b_{r_i}) as below:

$$B = (b_{l_1}, b_{r_1}) \cup \cdots \cup (b_{l_q}, b_{r_q})$$

where $b_{l_1} < b_{r_1} < b_{l_2} < b_{r_2} < \cdots < b_{l_q} < b_{r_q}$.

We claim that we can find a chain of tasks,

$$X : T_{j_l} \rightarrow T_{j_{l-1}} \rightarrow \cdots \rightarrow T_{j_1},$$

such that

$$\sum_{i=1}^{q} (b_{r_i} - b_{l_i}) \leqq \sum_{k=1}^{l} \mu(T_{j_k}) + \sum_{k=1}^{l-1} \tau_{\max} \times \eta(T_{j_{k+1}}, T_{j_k}). \tag{3.3}$$

In other words, the total length of B is no longer than the sum of all computational times and all pessimistic communication requirements along the chain X. We say that the chain X covers the set B.

Let T_{j_1} denote a task that finishes in the ETF schedule at time ω_{ETF}. Let $s(T_{j_1})$ denote the time at which T_{j_1} is started as scheduled by ETF. There are three possibilities regarding the starting time of T_{j_1}:

(1) $s(T_{j_1}) \leqq b_{l_1}$.

(2) $s(T_{j_1}) \in B$, i.e., there exists an integer h, $h \leqq q$, such that

$$b_{l_h} < s(T_{j_1}) < b_{r_h}. \tag{3.4}$$

(3) $s(T_{j_1}) \in A$ but $s(T_{j_1}) > b_{l_1}$, i.e., there exists an integer h, $h \leqq q-1$, such that

$$b_{r_h} \leqq s(T_{j_1}) \leqq b_{l_{h+1}} \quad \text{or} \quad b_{r_q} \leqq s(T_{j_1}). \tag{3.5}$$

If the first possibility occurs, then the task T_{j_1} by itself constitutes a chain that satisfies our claim. We shall show for the second and the third possibilities, respectively, that we can always add one or more tasks to the chain to extend the covered part of B to the left. To say a set of points is covered by a chain means that the length of the set is less than or equal to the sum of computation times and communication requirements along the chain, where communication requirements are estimated pessimistically (using $\tau_{\max}$).

Let us first consider the second possibility. Let h be the index satsifying (3.4). Then T_{j_1} alone has covered part of B from its right end to somewhere in between b_{l_h} and b_{r_h}. The mission here is to add the second task, T_{j_2}, to the chain. By the definition of B there is some processor P_α that was idle during $(s(T_{j_1})-\varepsilon, s(T_{j_1}))$ for some $\varepsilon > 0$. We consider three cases separately:

Case 1. T_{j_1} did not become available until $s(T_{j1})$.

Case 2. T_{j_1} became available at a time earlier than $s(T_{j_1})$ but later than time zero.

Case 3. T_{j_1} became available at time zero.

For Case 1, we simply take the last finished predecessor of T_{j_1} as T_{j_2}.

For Case 2, let T^* be the last finished immediate predecessor of T_{j_1}. Then $f(T^*) < s(T_{j_1})$. We further consider three subcases: (i) There exists no task T assigned to run P_α with $s(T) > s(T_{j_1}) - \varepsilon$. (ii) There exists at least one task T assigned to run on P_α with $s(T) > s(T_{j_1})$, but there exists no task T assigned on P_α with $s(T) = s(T_{j_1})$. (iii) There exists a task T assigned to run on P_α with $s(T) = s(T_{j_1})$.

For the first subcase (of Case 2), since no task is blocking T_{j_1} from starting earlier on P_α, it must be that $r(T_{j_1}, P_\alpha) \geqq s(T_{j_1})$. Let T_{j_2} be an immediate predecessor of T_{j_1} whose message to T_{j_1} arrives at P_α at time $r(T_{j_1}, P_\alpha)$. Let $p(T_{j_2}) = P_\beta$. Then $\tau(P_\beta, P_\alpha) \times \eta(T_{j_2}, T_{j_1}) = r(T_{j_1}, P_\alpha) - f(T_{j_2}) \geqq s(T_{j_1}) - f(T_{j_2})$. This result indicates that the communication requirement from T_{j_2} to T_{j_1} covers the interval $(f(T_{j_2}), s(T_{j_1}))$, and hence the covered part of B is shifted further left from $s(T_{j_1})$ after T_{j_2} is added to the chain. Furthermore, the parameter $\tau(P_\beta, P_\alpha)$ can be replaced by $\tau_{\max}$. This completes the first subcase.

For the second subcase (of Case 2), we let T_α be the first task allocated on P_α after $s(T_{j_1})$. By Lemma 2, do $(T_{j_1}) <$ do (T_α). The allocation of T_α on P_α cannot be a reason to block the possibility of T_{j_1} utilizing P_α at an earlier time. The reason T_{j_1} did not take such an advantage is again due to $r(T_{j_1}, P_\alpha) \geqq s(T_{j_1})$. T_{j_2} can therefore be found in the same way as in the first subcase.

We assume a task T_α satisfying $p(T_\alpha) = P_\alpha$ and $s(T_\alpha) = s(T_{j_1})$ in the last subcase. Now both do $(T_\alpha) >$ do (T_{j_1}) and do $(T_\alpha) <$ do (T_{j_1}) are possible. If do $(T_\alpha) >$ do (T_{j_1}), then T_α was scheduled later than T_{j_1}; hence, the same argument used in the previous subcase prevails. We need to argue why $r(T_{j_1}, P_\alpha) \geqq s(T_{j_1})$ when do $(T_\alpha) <$ do (T_{j_1}). Let us assume do $(T_\alpha)=k$ and consider two situations separately: (1) dt $(T_\alpha)<s(T_\alpha)$, and (2) dt $(T_\alpha)=s(t_\alpha)$. For (1), at the beginning of the kth execution of the inner loop, both T_α and T_{j_1} were available. (T_{j_1} was available since $f(T^*)\leqq$ dt (T_α) guaranteed by Lemma 1.) If $r(T_{j_1}, P_\alpha)<s(T_\alpha)$, then T_{j_1}, and not T_α, should have been selected in the kth execution of the inner loop. For (2), we let cm be the largest value in the current-moment sequence satisfying $f(T^*)\leqq \text{cm}>s(T_\alpha)$. (The existence of such cm is guaranteed by Lemma 3.) At CM = cm, T_α was unscheduled and T_{j_1} was available. T_α could not block T_{j_1} from occupying P_α at a time earlier than $s(T_\alpha)$. If $r(T_{j_1}, P_\alpha)<s(T_{j_1})$, then T_{j_1} should have been allocated on P_α and started at an earlier time when CM carried the value cm. This contradiction implies that $r(T_{j_1}, P_\alpha)\geqq s(T_{j_1})$. Based on this result, the same technique as before can be applied to add a second task T_{j_2} to the chain finishing all subcases of Case 2.

Case 3 is indeed impossible. We can obtain $r(T_{j_1}, P_\alpha)\geqq s(T_{j_1})$ as we did above for Case 2; on the other hand, we have a contradictory fact that $r(T_{j_1}, P_\alpha)=0$ due to $D_T=\emptyset$.

We have completed our discussion of the second possibility. Let us summarize the results obtained so far. Suppose $T_{j_1}<b_{r_h}$ for some $h\leqq q$. We constructed a second task T_{j_2} such that T_{j_2} precedes T_{j_1} and

$$\begin{aligned}&\mu(T_{j_1})+\mu(T_{j_2})+\tau_{\max}\times\eta(T_{j_2}, T_{j_1})\\&\quad\geqq(b_{r_q}-b_{l_q})+\cdots+(b_{r_{h+1}}-b_{l_{h+1}})+(b_{r_h}-s(T_{j_1}))+s((T_{j_1})-s(T_{j_2})).\end{aligned}$$

Since $s(T_{j_1})$ is an interior point of (b_{l_h}, b_{t_h}), at least some new portion (maybe all) of (b_{l_h}, b_{r_h}) has been covered now.

The location of $s(T_{j_2})$, once again, has three possibilities:

(1) $s(T_{j_2})\leqq b_{l_1}$.

(2) $s(T_{j_2})\in B$.

(3) $s(T_{j_2})\in A$ but $s(T_{j_2})>b_{l_1}$.

If the first possibility happens, then we are done. If not, we repeat our arguments to construct a third task or more tasks. (The third possibility may lead to adding more than one task in one cycle.) The cycle can be repeated until the starting time of the last added task satisfies the first possibility.

Now, it remains to show how to add tasks to the chain in the case of the third possibility (3.5). Suppose that h is the integer such that $b_{r_h}\leqq s(T_{j_1})\leqq b_{l_{h+1}}$ or $h=q$ when $b_{r_q}\leqq s(T_{j_1})$. Let $\hat{\Gamma}=\{T\colon T$ is a predecessor of T_{j_1} satisfying $s(T)\geqq b_{r_h}$ or $T=T_{j_1}\}$. Clearly, $\hat{\Gamma}\neq\phi$. Let $T^{\#}\in\hat{\Gamma}$ such that $s(T)<b_{r_h}$ for any immediate predecessor T of $T^{\#}$. Then $T^{\#}$ is either a predecessor of T_{j_1} or T_{j_1} itself. In either case, we can always construct a sequence of tasks:

$$T^{\#}=T_{j_{d-1}}\rightarrow T_{j_{d-2}}\rightarrow\cdots\rightarrow T_{j_2}\rightarrow T_{j_1}.$$

Although these tasks are not scheduled during B, the last added task, $T^{\#}$, is closer to uncovered points of B. The task $T^{\#}$ will play the same role as T_{j_1} did in the discussion about the second possibility. The mission again is to add a task T_d to the chain. There are three cases:

Case 1. $T^{\#}$ did not become available until b_{r_h}.

Case 2. $T^{\#}$ became available at a time earlier than b_{r_h} but later than time zero.

Case 3. $T^{\#}$ became available at time zero.

We finish Case 1 by setting T_{j_d} to be the last finished predecessor of $T^{\#}$. Case 3 is impossible by the same argument for Case 3 of the second possibility.

For Case 2, let T^* be the last finished predecessor of $T^{\#}$. Then $f(T^*)<b_{r_h}\leqq s(T^{\#})$. By the definition of B there is some processor P_α that was idle during the time $(b_{r_h}-\varepsilon, b_{r_h})$ for some $\varepsilon>0$. We ask the same question as before. Why had $T^{\#}$ not been allocated on P_α while it was available and P_α was idle? Let T_α be the task assigned to run on P_α at b_{r_h}. Then $f(T^*) \leqq \mathrm{dt}\,(T_\alpha) \leqq s(T_\alpha)$ by Lemma 1. As in the last subcase of Case 2 for the second possibility, (1) $\mathrm{do}\,(T_\alpha) > \mathrm{do}\,(T^\#)$ and (2) $\mathrm{do}\,(T_\alpha) < \mathrm{do}\,(T^\#)$ are distinguished. Two situations for (2) are further discussed separately: (1) $\mathrm{dt}\,(T_\alpha) < s(T_\alpha)$, and (2) $\mathrm{dt}\,(T_\alpha) = s(T_\alpha)$. The same techniques can be applied to conclude that $r(T^\#, P_\alpha) \geqq b_{r_h}$. Let T_{j_d} be the immediate predecessor whose message reaches P_α at $r(T^\#, P_\alpha)$ assuming $\hat{T}$ be allocated on P_α. This task is exactly what we want.

After T_{j_d} together with $T_{j_{d-1}}, \cdots, T_{j_2}$ is added to the chain, the enlarged chain clearly covers all of B from some point on, and this is a greater part of B than the initial one. We repeat the whole process by considering three possibilities regarding the position of $s(T_{j_d})$. The process is repeated until $s(T_{j_l}) \leqq b_{l_1}$ is satisfied by the added task in the chain. Finally, a chain satisfying our claim (see (3.3)) is constructed.

The important thing to note about this claim is

$$\sum_{\phi_i \in \Phi} \mu(\phi_i) \leqq (n-1) \times \sum_{k=1}^{l} \mu(T_{j_k}) + n \times \tau_{\max} \times \sum_{k=1}^{l-1} \eta(T_{j_{k+1}}, T_{j_k})$$

where the left-hand sum is over all empty tasks. (A processor is said to be executing an empty task when it is idle.) But the fact that $T_{j_l}, \cdots, T_{j_1}$ is a chain implies that it takes at least $\sum_{k=1}^{l} \mu(T_{j_k})$ to finish all tasks in Γ in any schedule, even on an ideal system, i.e.,

$$\omega_{\mathrm{opt}}^{(i)} \geqq \sum_{k=1}^{l} \mu(T_{j_k}).$$

The following inequality is obvious:

$$\sum_{T \in \Gamma} \mu(T) \leqq n \times \omega_{\mathrm{opt}}^{(i)}.$$

Consequently,

$$\begin{aligned}
\omega_{\mathrm{ETF}} &= \frac{1}{n}\left(\sum_{T_k \in \Gamma} \mu(T_k) + \sum_{\phi_i \in \Phi} \mu(\phi_i)\right) \\
&\leqq \frac{1}{n}\left(n\omega_{\mathrm{opt}}^{(i)} + (n-1)\omega_{\mathrm{opt}}^{(i)} + n \times \tau_{\max} \times \sum_{k=1}^{l-1} \eta(T_{j_{k+1}}, T_{j_k})\right) \\
&= \left(2 - \frac{1}{n}\right)\omega_{\mathrm{opt}}^{(i)} + \tau_{\max} \times \sum_{k=1}^{l-1} \eta(T_{j_{k+1}}, T_{j_k}).
\end{aligned}$$

The theorem follows immediately by replacing the communication requirements along a particular chain by $C_{\max}$. □

The bound stated in Theorem 3.2 can be reduced by giving an algorithm to construct the task chain and the corresponding communication requirements used to cover idle times. The proof of Theorem 3.2 actually finds a chain X: $T_{j_l}, \cdots, T_{j_1}$ such that $|B| \leqq \sum_{k=1}^{l} \mu(T_{j_k}) +$ *certain communication times.* The last term, called C in the following discussion, corresponds to the sum of just enough terms of the form $\tau(p(T_{j_{k+1}}), P_\alpha) \times \eta(T_{j_{k+1}}, T_{j_k})$ to "cover" the times at which (1) no task in the constructed chain is running, and (2) at least one processor is idle. Since it can frequently happen that all processors are busy while communication is taking place, much of the time $C_{\max}$ is "hidden" by being overlapped with computation of other tasks. We are therefore led to a better bound for ω_{ETF} that even coincides with Graham's bound if all communication is hidden. The algorithm assumes that all processors were idle for

$t<0$, and each task was scheduled to run during a closed time interval $[t_1, t_2]$. It calculates an upper bound C for the "unhidden" communication of the constructed chain.

```
ALGORITHM C
Input: An ETF Schedule
Output: C
(0) $C \leftarrow 0$, find a task $T$ that was scheduled to finish last.
(1) Let $u = \text{l.u.b.}^1 \{t\colon t \leqq s(T)$ and at least one processor is idle at $t\}$.
(2) If $u = 0$ then output $C$ and stop.
(3) If $u < s(T)$ then find a task $T'$ such that $T'$ is either a predecessor of $T$ with
    $s(T') \geqq u$ or $T$ itself but $s(T^*) < u$ for any immediate predecessor $T^*$ of $T'$. Replace
    $T$ by $T'$.
(4) Let $T'$ be the last finished predecessor of $T$. If $f(T') = u$ then replace $T$ by $T'$ and
    go to (1).
(5) Find a processor $P_\alpha$ that was idle during $(u-\varepsilon, u)$ for some $\varepsilon > 0$, find a task
    $T' \in D_T$ such that $f(T') + \tau(p(T'), P_\alpha) \times \eta(T', T) = r(T, P_\alpha)$, $C \leftarrow C +$
    $\tau(p(T'), P_\alpha) \times \eta(T', T)$, replace $T$ by $T'$, go to step 1.
```

In general, C is much smaller than $C_{\max}$ depending on the case. It serves, however, to replace $C_{\max}$ in the performance bound of Theorem 3.2, and this is stated in the corollary below.

COROLLARY. *An* ETF *schedule is bounded from above by the sum of the Graham's bound and the output of Algorithm* C. *Mathematically,*

$$\omega_{\text{ETF}} \leqq \left(2 - \frac{1}{n}\right) \omega_{\text{opt}}^{(i)} + C. \tag{3.6}$$

4. Conclusion and discussion. An essential result of multiprocessor scheduling theory is the introduction of the list-scheduling method and the establishment of its performance guarantee. In this paper, we first extend the LS method to address the issue of communication delays. Theorem 2.1 indicates that the extended method, named ELS, fails to shorten communication delays, at least in worst cases. The new heuristic presented in this paper effectively reduces communication delays and also maintains the strength of the classical list scheduling.

It should be noted that the performance bound of ETF is made possible by the assumption of no communication contention; however, it is clear that the contention-free attribute alone does not guarantee the same bound.

Assume no communication contention is indispensable for maintaining the deterministic approach; but, strictly speaking, this assumption is only valid in fully connected systems or systems with contention-free protocols. For the later case, the protocol overhead should be included in the estimation of the communication parameters.

To expand the application domain of this theoretic work, we would rather consider the deterministic scheduling as a planning tool for task allocation suitable for most highly connected systems. Though each task is scheduled to start at some specific time, it will be started whenever enabled in the actual execution. In the case the actual start time of a task is later than its scheduled start time due to the queueing delay caused by communication contention, the succeeding tasks may become incapable of starting their execution at their scheduled times. This "postponement" may propagate. In the extreme case the postponement may propagate down to the critical execution path

[1] Least upper bound.

and then lengthen the total scheduled time of the graph. However, to some extent, queueing delay can be absorbed in noncritical paths. Only the portion of queueing delay that cannot be absorbed will lengthen the total graph's execution time. For this reason, the sensitivity of the no-communication contention assumption can be relaxed in real applications.

Acknowledgments. The authors are grateful to Drs. R. L. Graham and S. R. Kosaraju for their careful review and suggestions to the ETF algorithm. They are also indebted to the University of Florida and National Chiao Tung University. Without their support of the e-mail facility, the revision of the paper that required much communication across half the globe would have been very difficult.

REFERENCES

[1] E. G. Coffman, Jr., ed., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, 1976.

[2] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[3] 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*, Ann. Discrete Math., 5 (1979), pp. 287–326.

[4] J.-J. Hwang, *Deterministic scheduling in systems with interprocessor communication times*, Ph.D. dissertation, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 1987.

[5] H. Kasahara and S. Narita, *Parallel processing on robot-arm control computation on multiprocessing systems*, IEEE J. Robotics and Automation, (1985), pp. 104–113.

[6] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Recent development in deterministic sequencing and scheduling: a survey*, in Deterministic and Stochastic Scheduling, M. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, eds., D. Reidel, Dordrecht, the Netherlands, 1982, pp. 367–374.

[7] V. J. Rayward-Smith, UET *scheduling with interprocessor communication delays*, Internal Report SYS-C86-06, School of Information Systems, University of East Anglia, Norwich, United Kingdom, 1986.