SCHEDULING PARTIALLY ORDERED TASKS WITH PROBABILISTIC EXECUTION TIMES

K. M. Chandy and P. F. Reynolds
The University of Texas at Austin

The objective of this paper is to relate models of multi-tasking in which task times are known or known to be equal to models in which task times are unknown. We study bounds on completion times and the applicability of optimal deterministic schedules to probabilistic models. Level algorithms are shown to be optimal for forest precedence graphs in which task times are independent and identically distributed exponential or Erlang random variables. A time sharing system simulation shows that multi-tasking could reduce response times and that response time is insensitive to multi-tasking scheduling disciplines.

Key Words and Phrases: deterministic models, probabilistic models, multi-tasking, multi-processing

CR Categories: 4.3, 4.32, 4.34, 4.35, 5.3, 5.32, 5.4, 5.42, 8.1

A substantial amount of literature exists on the scheduling of partially ordered tasks with fixed execution times given two or more processors [1]. Results for models in which task times are fixed, henceforth called deterministic models, include (1) the definition of optimal polynomial algorithms for models satisfying certain assumptions regarding both the number and type of processors and the nature of task precedence constraints [2, 3, 4, 7, 10]; (2) the establishment of bounds for comparing optimal and worst case scheduling algorithms [6, 10]; and (3) the identification of models for which optimal algorithms are necessarily NP-complete [11].

A major drawback of deterministic models rests in the assumption that task execution times are fixed. Programs written to recognize potential parallelism in other programs are generally unable to determine exact execution times for the tasks they generate. Thus, models in which task times are not known are clearly needed.

We define multi-tasking to be both the partitioning of single processes into smaller tasks and the subsequent parallel execution of those tasks. As multi-processor systems become common, multi-tasking will find an increasing domain of applicability. For example, multi-tasking in a real-time monitoring system could allow an unacceptably slow process to be executed in a reasonable amount of time. Also, systems characterized by frequently run, lengthy, high priority jobs would benefit from multi-tasking. In this case the ratio of the costs of partitioning a process and the net reduction in total processing time becomes more favorable. If a process has a high priority and is lengthy then a reduction in its total execution time is favorable not only for that process but also for processes that may have been blocked due to their lower priority.

There is a need to identify cases in which multi-tasking will be useful. Browne, et al., [8] have concluded that multi-tasking in a multi-programmed system would not produce significant improvement in thoughput. A simulation described at the end of this paper supports that conclusion. Systems designers need to know these and other results so that they might determine whether the incorporation of multi-tasking into an operating system would be beneficial.

In this paper we consider a partially ordered set of tasks $T=\{T_1,\ldots,T_n\}$ and its related precedence graph G. We also use T to refer to the set of vertices in G, and we define the value of T_i to be the mean execution time for T_i. The level of a vertex in G is defined the same way it is defined in [1]. At any given time, an initial task is one all of whose predecessors have been completed. A Highest Levels First (HLF) Schedule is defined as: Whenever a processor becomes free, assign that initial task (if any) which is at the highest level of those initial tasks not yet assigned. A B-schedule [1] is an HLF schedule in which ties among initial highest level tasks are broken arbitrarily. An A-schedule is an HLF schedule in which ties between highest level tasks are broken by a labeling scheme [1]. Hu [2] has shown that B-schedules are optimal for forest precedence graphs (in which every task has at most one successor) if all task execution times are known to be equal. Coffman and Graham [3] have shown that the A-schedule is an optimal nonpreemptive schedule for arbitrary task graphs given two processors and provided all task execution times are known to be equal. Muntz and Coffman

have shown that processor-sharing all highest level tasks is an optimal preemptive schedule if the precedence graph is a forest or if there are only two processors [4, 7]. Adam, Chandy and Dickson [5] have shown empirically that the B-schedule is near-optimal for arbitrary precedence graphs and stochastic task times.

In the sections that follow we describe first known results for models in which task times are unknown. Our objective is to extend the results for deterministic models to models in which task times are not known. We begin in section 1 by pointing out that a bound derived in [6] for worst case total execution time can be applied as well to processes with unknown task times. In section 2 we summarize the simulations described in [5], in which it was found that HLF algorithms could be expected to perform near-optimally for all precedence graphs with unknown task times. Section 3 contains our results for systems having two or more processors. For systems with two identical processors we show that forest precedence graphs with independent identical exponentially distributed task times can be scheduled optimally only by HLF algorithms. We next extend this result to tasks having identical n-order Erlang distributions. Results for both preemptive and non-preemptive cases are derived. We then give cases in which we know that HLF is not necessarily optimal, and for which we know no optimal polynomial scheduling algorithm. We complete the section by giving our results for models with two or more non-identical processors. We finish with the results of a simulation of multi-tasking in a time-sharing system which indicate that multi-tasking in a multi-programming environment does not improve response time substantially. The parameters for the time-sharing simulation were obtained from a software monitor on a CDC-6400 at the University of Texas at Austin.

1. Bounds for Scheduling When Task Times Are Unknown. Graham [6] has shown that given a set of partially ordered tasks and assuming (1) m identical processors, (2) known task times, and (3) non-preemptive scheduling algorithms that always schedule a ready task if possible, the upper bound for the ratio of finishing times for an arbitrary schedule ω_a, and an optimal schedule ω_o, is:

$$\omega_a/\omega_o \leq (2 - 1/m) \qquad (1)$$

This ratio can be applied as well to the case in which task times are unknown. Consider a process consisting of component tasks with execution times that vary across separate executions of the process. Let ω_F be the random variable which is the optimum finishing time for a particular execution of the process given task execution times in advance. Let ω_h, also a random variable, be the finishing time of an arbitrary schedule when applied to the same set of given task times. Then, from (1)

$$\omega_a \leq (2 - 1/m) * \omega_F \qquad (2)$$

Taking expectations, it follows that

$$\bar{\omega}_h \leq (2 - 1/m) * \bar{\omega}_F \qquad (3)$$

Hence, given any two scheduling algorithms, "a" and "b" with finishing times ω_a and ω_b, the average finishing times, $\bar{\omega}_a$ and $\bar{\omega}_b$ for algorithms "a" and "b" are related by

$$\bar{\omega}_a \leq (2 - 1/m) * \bar{\omega}_b \qquad (4)$$

In particular, equation (4) holds if ω_b is the optimum finishing time given distributions for task execution times.

2. Simulation Results for Precedence Graphs with Unknown Task Times. In general, we have found that HLF schedules for task graphs with unknown task execution times produce results significantly better than the bound derived in the preceding section might suggest. Simulations performed by Adam, et al., [5] on arbitrary graphs with unknown task times clearly indicate that B-schedules are near-optimal.

Algorithms simulated by Adam et al. include (1) HLFET (Highest Levels First with Estimated Times) and (2) SCFET (Smallest Co-levels First with Estimated Times). Table I compares the preemptive list schedules derived from these algorithms to an optimal preemptive schedule derived using dynamic programming. Table II compares the performance of the non-preemptive list schedules derived from SCFET and HLFET to each other. Optimal bounds were not available in this case.

The conclusions that we draw from these simulations are, that on the average: (1) preemptive B-schedules can be expected to perform near-optimally (within 5% of optimal) for task graphs in which task times are unknown, and (2) non-preemptive B-schedules can be expected to be at least superior to other scheduling algorithms derivable in polynomial time.

# Processors	# Nodes	# Edges	Percentage deviation from Optimal HLFET	SCFET
5	20	66	0.03	1.01
5	23	236	0.00	0.01
5	20*	66	0.02	0.59
5	19	56	0.00	0.00
5	15*	55	0.00	0.00
5	18	68	0.00	0.28
5	20	43	0.19	1.47
5	15	55	0.00	0.00
5	18	69	0.00	0.05
5	19	107	0.01	0.01
5	12	43	0.00	0.00
5	8	13	0.00	0.00
5	20	48	0.02	0.52

*These graphs were processed assuming all tasks have equal mean duration time.

TABLE I

# Graphs run	Range # nodes	Average # nodes	Average # edges	% deviation from HLFET SCFET
22	1-50	36	76	13.37
38	51-100	63	119	13.63
4	101-150	130	214	10.94
3	200+	234	398	8.79

TABLE II

3. Scheduling Two or More Processors When Task Times Are Unknown. Consider a forest precedence graph (in which every vertex has at most one successor) where all task times are independent and are derived from identical exponential distributions (iid exponential). Assume without loss of generality that the mean execution time for all such tasks is unity. We refer to such precedence graphs as Exponential Forests (ExFs). Similarly, consider a forest precedence graph where all task times are independent and are derived from identical nth order Erland distributions (iid Erlang). We assume without loss of generality that the mean execution time for all such tasks is n, and we refer to these precedence graphs as Erlang Forests (ErFs).

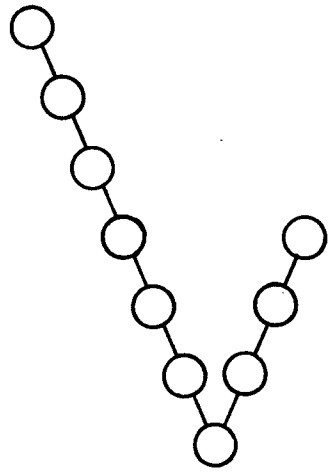
Our results in this section are the following: (1) For ExFs and two identical processors, HLF algorithms are optimal. (2) For ErFs and two identical processors, HLF is optimal in the non-preemptive case and HLF with processor sharing is optimal in the preemptive case. (3) For models with more than two identical processors HLF algorithms are not necessarily optimal for ExFs. (4) For arbitrary precedence graphs with task times which are iid exponential random variables HLF scheduling algorithms are not necessarily optimal for two or more identical processors. (5) For models with two or more identical processors and forests in which task times are derived from non-identical independent exponential distributions, HLF algorithms are not necessarily optimal. (6) For ExFs and two non-identical processors HLF with preemption is optimal. (7) For both ExFs and ErFs and with two or more non-identical processors, keeping a processor idle even when there is a ready task can produce smaller finishing times if non-preemptive scheduling is assumed.

3.1. Results for Two or More Identical Processors

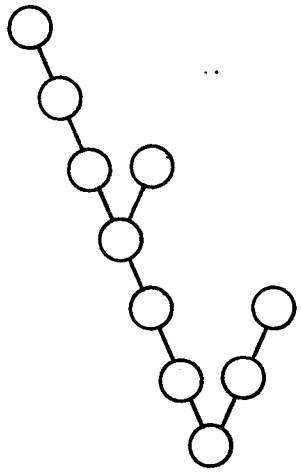
3.1.1. Optimality of HLF Algorithms for ExFs.

Due to the detailed nature of our theorems and proofs for the optimality of HLF algorithms, we have chosen to present a more intuitive description of our approach in this section. Theorems and proof outlines for this section appear in the appendix.

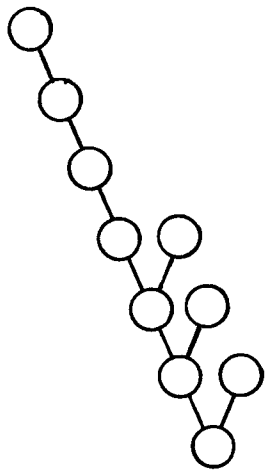
In order to prove the optimality of HLF algorithms for ExFs we develop the concept of flatness of an ExF by defining a set of partial relations for relating the flatness of one ExF to another. Given two ExFs, G and H, we define G to be as flat as H ($G \sim H$) if and only if the number of tasks at every level in G is equal to the number of tasks at every level in H. G is as flat



(a)



(b)



(c)

Fig. 1. Flatness examples.

as or flatter than H ($G \propto H$) if and only if the number of tasks above (away from the root) a given level in G is less than or equal to the number of tasks above the same level in H. G is flatter than H ($G \propto H$) if and only if the number of tasks above a particular level in G is strictly less than the number of tasks above the same level in H, and $G \propto H$. Note that the removal of a task from a given ExF always produces an ExF that is flatter than the original. The ExF in Fig. 1a is as flat as the ExF in Fig. 1c and flatter than the one in Fig. 1b.

If G and H each have at least two initial tasks, then we define x_1 and x_2 to be the tasks that are selected by an arbitrary algorithm applied to G, and we assume without loss of generality that the level of x_1 is greater or equal to the level of x_2. Similar assumptions are made for y_1 and y_2, the tasks selected by an arbitrary algorithm for H. We denote G with x_i removed as $(G - x_i)$, and H with y_i removed as $(H - y_i)$.

Consider two ExFs, G and H, that satisfy a flatness relationship, (1) $G \propto H$ or (2) $G \not\propto H$. Assuming that G and H each have at least two initial tasks, then when $(G - x_i)$ is compared with $(H - y_i)$ we say that the original flatness relation between G and H is preserved if condition (2) is met, that it is maintained if $(G - x_i)$ and $(H - y_i)$ have the same relation as G and H, and that it is enhanced if relation (2) originally applied to G and H and relation (1) applies to $(G - x_i)$ and $(H - y_i)$.

We now establish (theorems 1 - 3) that if 1) G and H each have at least two initial tasks, 2) either $G \propto H$ or $G \not\propto H$, and 3) an HLF algorithm is applied to G and an arbitrary algorithm is applied to H, then when we compare $(G - x_1)$ to $(H - y_1)$ and $(G - x_2)$ to $(H - y_2)$, flatness will be preserved, at least, in one of these comparisons, and it will be maintained or enhanced in the other. Also, if $G \sim H$ and HLF algorithms are applied to both G and H then $(G - x_1) \sim (H - y_1)$ and $(G - x_2) \sim (H - y_2)$.

We leave the preceding result for a moment in order to discuss an important characteristic of our original assumptions. Since we have assumed task execution times to be iid exponential it follows from the "memoryless" property of exponential distributions that the expected execution time of a partially processed task is independent of the amount of processing the task has received and is equal to the original expected value. Furthermore, when two tasks having exponential distributions with unit means are considered in parallel, the expected time until one of the tasks completes is 1/2 (units of time). With these results we can express a recursive cost function for the expected finishing time of G using scheduling algorithm X (denoted T(G, X)) as

$$T(G,X) = 1/2 + 1/2 * T(G-x_1,X) + 1/2 * T(G--x_2,X)$$

assuming there are at least two initial tasks in G. If G is a chain, then the function becomes

$$T(G,X) = 1 + T(G - x_1,X).$$

In theorems 4 - 7 we derive the exclusive optimality of HLF algorithms primarily by inductively applying the cost functions and flatness results given above. We conclude that preemption is of no benefit for the following reasons: Due to the memoryless property of exponential random variables, the state of a partially processed ExF is itself an ExF. It follows that any schedule that is optimal for a given ExF will continue to be optimal until a task completes. If preemption is allowed, then it follows that a processor should never be idle when an executable task is available. Since a task only needs to be preempted when another task completes it also follows that there would never be more than N - 1 preemptions for an N task ExF. If a task is selected by an HLF algorithm, it could continue to be selected by an HLF algorithm independent of the number of other tasks that are completed. From this it follows that if we have identical processors, preemption is not necessary in an HLF algorithm for ExFs.

Results of this section are unique in the following respects: (1) Task times are assumed to be unknown. (2) All optimal schedules must be of a certain type (i.e., HLF). Thus, we have a necessary and sufficient condition for optimality whereas other problems [2, 3, 4, 7] have yielded only sufficient conditions. (3) Given two identical processors, there is no benefit in preemption.

3.1.2. Optimality of HLF Algorithms for ErFs. We present a brief outline of our results. A discussion of the theorems used to prove our results appears in the appendix.

HLF is an optimal scheduling algorithm if non-preemption is assumed. Theorems used to prove this result incorporate the flatness concept defined in the preceding section. Given task time distributions that are n-order Erlang, a task is assumed to consist of n exponential stages. We show that an ExF can be constructed such that tasks in the ExF represent stages of tasks in an ErF. It is assumed that the scheduling policy cannot know how many stages a partially processed task has left. (To allow this would reduce ErFs to ExFs.) Using the constructed ExF we can then demonstrate the optimality of HLF algorithms using the techniques of the previous section.

In the non-preemptive case HLF with processor sharing is an optimal scheduling algorithm. If two or more tasks exist at the highest level, then they should be processor shared. If only one task exists at the highest level, then it should be assigned to one processor and initial tasks at the next highest level should be processor shared on the remaining processor. We derive our results by constructing a corresponding ExF as described above and by using the techniques of the previous section.

Details of our results appear in a forthcoming paper.

3.1.3. Schedules for ExFs and More Than Two Identical Processors.

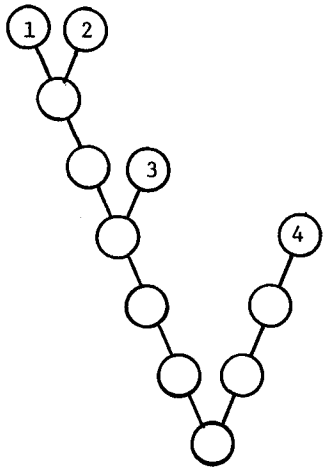


Fig. 2. ExF for which 3 processor HLF algorithm is not optimal.

The optimal three processor schedule for the ExF shown in Fig. 2 is not an HLF schedule; tasks 1, 2 and 4 are processed initially in the optimal schedule whereas tasks 1, 2 and 3 are processed initially in the HLF schedule.

The reason that the results of Section 3.1.1 break down with three or more processors is interesting. If H has 3 executable tasks and G∝H, and G and H have the same number of tasks, it is possible that G has only two executable tasks (Figs. 1a, 1b). Hence, given three processors, H offers more opportunity for exploiting parallelism and hence has a smaller mean completion time. It is obvious that if G∝H, and G and H have the same number of tasks, then G must have at least two executable tasks; thus, G can always exploit two processors.

3.1.4. Precedence Graphs Which Are Not Forests.

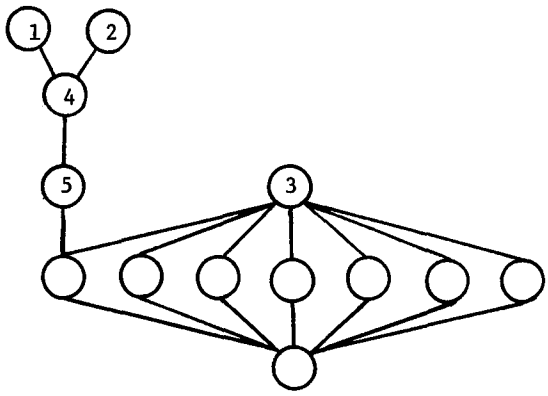


Fig. 3. iid exponential precedence graph for which HLF is not optimal.

The optimal two-processor schedule for the graph in Fig. 3 in which task times are iid exponential is not HLF; the optimal solution is to process tasks 1 (or 2) and 3 initially whereas the HLF schedule processes tasks 1 and 2 initially. Therefore, the A-schedule is not optimal in this case. A polynomial algorithm for this problem has not been obtained.

The reason that the HLF algorithm is not optimal in this case is that task 3 has more successors than 2. If we start with 1 and 2, there is a greater probability that we will finish tasks 1, 2, 4 and 5 before 3 than if we start with 1 and 3 (or 2 and 3).

3.1.5. Forests with Unequal Exponentially Distributed Task Times.

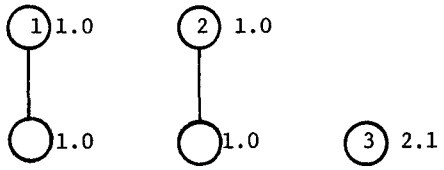


Fig. 4. Forest with unequal exponentially distributed task times (mean times appear beside tasks).

The optimal two processor schedule for the forest precedence graph shown in Fig. 4 is not an HLF schedule. An optimal solution in this case is to first schedule tasks 1 and 2 and then to schedule the remaining task with task 3. An HLF schedule would have selected task 3 and either task 1 or task 2 initially. A polynomial algorithm for this problem has not been obtained.

HLFNET is an algorithm that has been proposed [5] for unknown mean task times. This algorithm is essentially an HLF algorithm assuming that all tasks have equal mean times. Note that HLFNET would be optimal in this example and that HLF is not. This supports findings in [5] that HLFNET is a reasonable algorithm when means are unknown.

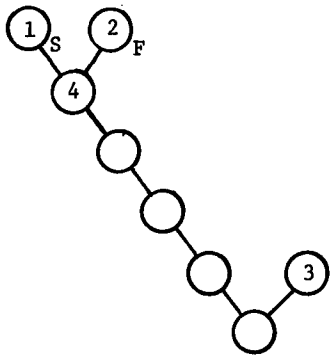
3.2. Results for Unequal Processors.

3.2.1. ExFs with Two Non-identical Processors. The results of section 3.1.1 can be extended to the case where the two processors have different rates. Let the mean time for all tasks on the fast processor be unity and on the slow processor be 1/a. If the fast processor is assigned a task x_1 of ExF G and the slow processor is assigned x_2 in a schedule X, we have

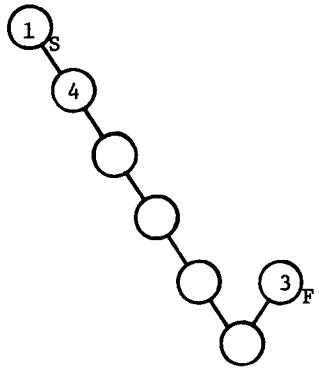
$$T(G,X) = \frac{1}{1+a} + \frac{1}{1+a} T(G - x_1, X) + \frac{a}{1+a} T(G-x_2, X)$$

A schedule is defined to be HLF if and only if, at all times, the fast processor is assigned an executable task at a higher level than all other executable tasks and the slow processor is then assigned an executable task at the highest level among all remaining executable tasks. Thus a task may be initially assigned to the slow processor and later switched to the fast processor. As discussed before, the maximum number of preemptions of tasks (on the slow processor) is N-1. The theorems for this case are identical to those in section 3.1.1.

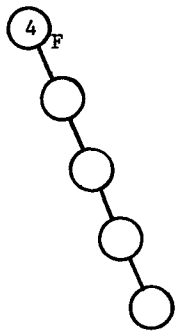
3.2.2. ExFs and ErFs with Non-Preemptive Scheduling.



(a) Task 2 completes.



(b) Task 1 completes and S kept idle.



S idle

(c) F assigned to task 4.

F: fast processor
S: slow processor

Fig. 5 Keeping a slow processor idle.

It has been shown for non-preemptive scheduling of some deterministic models that keeping a processor idle even when there is a ready task can produce smaller finishing times [6]. The same result applies to ExFs and ErFs when there are two or more non-identical processors. A two processor example is given in Figure 5 where, initially, the slow processor is assigned to task 1 and the fast processor is assigned to task 2 (Fig. 5a). If the fast processor completes its task first then it is assigned next to task 3 (Fig. 5b). Assuming that the slow processor completes task 1 next, it now becomes possible, given widely differing processor speeds, that it would be advantageous to not assign the slow processor to task 4 but rather to wait for the fast processor to complete task 3 so that it could then be assigned to task 4, and subsequently the remainder of the chain (Fig. 5c).

4. A Simulation of Multi-tasking in a Time-sharing System. Browne et. al [8] predicted that multi-tasking in a multi-programming environment would not result in a significant reduction in user response time. In Table III we give the results of our simulation of multi-tasking in a time-sharing system that concur with that prediction. The parameters for this simulation were obtained from a software monitor on a CDC-6400 at the University of Texas at Austin. A description of the system can be found in [9].

A simulation of the University of Texas single processor time-sharing system was written and validated and then modified so that two processing units were simulated in the place of the one original processor. The service rates for the two processors were equal to the service rate of the original processor. The degree of multi-tasking was varied from none to extremely optimistic.

We now discuss five cases that were simulated.

Case (a). The maximum response time is obtained when we assume that

(1) All tasks can always be processed in parallel by both processors.

(2) All task execution times are known.

(3) Both CPUs always work in parallel on that job in the CPU queue with the Shortest Remaining Time (SRT).

Case (b). Consider another model where the two processors always process different jobs if there are 2 or more jobs in the CPU queue. The two processors cooperate on a single job only if there is one job in the CPU queue. In this case too we assume that all jobs can be processed in parallel by both CPUs at all times. We assume that the CPU discipline is processing-sharing.

Case (d). In this case we assume that an optimal multi-tasking scheduling algorithm is one that results in a single job processing rate of 2μ where μ is the processing rate of one CPU. We then construct a worst case multi-tasking scheduling algorithm (in comparison to the assumed optimal algorithm) in accordance with the bound discussed in section 1. In this case the single job processing rate by both CPUs would be 1.33μ.

Case (c). In general it is unrealistic to assume that the multi-tasking of a single process could utilize two processors throughout the duration of the execution of the process. The assumption of a single job processing rate of 1.5μ with an optimal multi-tasking scheduling algorithm

is not pessimistic.

Case (e). Here we assume that no multi-tasking is done. It should be noted that this is the worst case for all cases in which the effective single process multi-tasking rate is less than or equal to 1.5μ.

Optimal vs. Non-Optimal Multi-tasking Scheduling Algorithms.

If case (b) is considered the optimum multi-tasking algorithm, then case (d) is the corresponding worst-multi-tasking algorithm. Similarly case (e) is the worst case if case (c) corresponds to the optimum multi-tasking schedule. The difference in response time between case (b) and (d) is 10% and between (c) and (e) is 8%. Since HLF is a near-optimal algorithm, the difference in response times between the HLF case and the optimum multi-tasking case would be very small.

Multi-tasking vs. No Multi-tasking.

Excluding all multi-tasking overhead costs, there is approximately a 12% reduction in user response time between cases (e) and (a). From this we conclude that response time is relatively insensitive to the scheduling algorithm chosen for multi-tasking. We note that the 12% difference found between cases (e) and (a) would probably be reduced significantly if multi-tasking overhead costs were included in the case (a) simulation.

Case	CPU Scheduling Discipline	Single Job Service Rate	Service Rate for More Than One Job	Degree of Multi-tasking	User Response Time (sec.)
e	processor sharing	μ*	2μ	none	1.016
d	processor sharing	1.33μ	2μ	moderate	0.998
c	processor sharing	1.50μ	2μ	optimistic	0.934
b	processor sharing	2μ	2μ	very optimistic	0.900
a	processor sharing	2μ	2μ	extremely optimistic	0.894

*μ is the single processor service rate.

Table III. Results from the Simulation of Multi-tasking in a Time-sharing System.

SUMMARY

We conclude that

(1) Multi-tasking in a typical time sharing system results in small reduction in response time.

(2) Response time is not very sensitive to the multi-tasking scheduling algorithm used. Thus, without regard to scheduling costs, the response time resulting from using known near-optimal polynomial algorithms such as HLF (and HLFNET) can be expected to differ insignificantly from the response time resulting from an optimal algorithm. Since optimal algorithms are NP-complete HLF (and HLFNET) are preferred algorithms.

(3) Though multi-tasking may not reduce response significantly in multi-programming systems, it could be useful in speeding up high priority jobs.

(4) Parallelism recognizers should attempt to divide programs into tasks with approximately equal mean times.

APPENDIX

Definitions

An ExF is a forest where all task times are independent identical exponential random variables. We define G and H as ExFs and N_i and M_i as the number of vertices at level i for G and H respectively, for i = 1,2,3,...,. Clearly N_i and M_i are non-negative integers. Let S(G,m) be the number of tasks at level m or higher in G. Then

$$S(G,m) = \sum_{i \geq m} N_i \qquad (1)$$

G is defined to be as flat as H, denoted by $G \backsim H$,

IFF $S(G,m) = S(H,m)$ for m = 1,2,3,... (2)

G is defined to be as flat or flatter than H, denoted by $G \propto H$, if and only if:

$S(G,m) \leq S(H,m)$ for m = 1,2,3,... (3)

G is defined to be flatter than H, denoted by $G \propto H$, if and only if (3) is true, and there exists some m, such that

$S(G,m) < S(H,m)$ (4)

We define X to be an HLF schedule on G which initially processes tasks x_1 and x_2 if there are at least two initial tasks in G and processes task x_1 if there is only one. Similarly, we define Y and Z to be schedules on H; where Y is HLF and Z is arbitrary. If H has at least two initial tasks, then Y will begin with tasks y_1 and y_2 and Z will begin with z_1 and z_2. If H has only one initial task, then Y will begin with y_1 and Z with z_1. Let $G - x_i$ be the subgraph of G obtained by deleting x_i. We define $H - y_i$ and $H - z_i$ similarly.

Theorems for Optimality of HLF for ExFs and Two Processors

We present the following theorems without detailed proof in order to establish the optimality of the HLF scheduling algorithm for ExFs and two processors.

Theorem 1

Let H have two or more executable tasks. Then

$$H - y_i \preceq H - z_i \qquad i=1,2 \qquad (5)$$

Furthermore, if the level of Y_i is greater than the level of z_i then

$$H - y_i \prec H - z_i \qquad i=1,2 \qquad (6)$$

Proof: A Proof for (5) is shown by establishing that

$S(H - y_1,m) \leq S(H - z_1,m)$ for all m, i=1,2.

(6) can be proven by showing

$S(H - y_i,\text{level of } y_i) < S(H - z_i, \text{ level of } z_i)$

and using (5).

Theorem 2

Let G and H have two or more initial tasks. If $G \preceq H$, then

$$G - x_i \preceq H - y_i \qquad (7)$$

Furthermore, if $G \prec H$, then either

$$G - x_1 \prec H - y_1 \qquad (8)$$

or $$G - x_2 \prec H - y_2$$

or both (8) and (9) are true.

Proof: (7) can be proved by showing that

$S(G - x_i,m) \leq S(H - y_i,m)$ for all m, i=1,2

(8) and (9) are proven by showing that for some m

$S(G - x_i,m) < S(H - y_i,m)$

and by using (7).

Corollary 1

If $G \sim H$, then $G - x_i \sim H - y_i$, i-1,2

Proof: Follows from Theorem 2.

Theorem 3

Let G and H have two or more executable tasks. If $G \preceq H$, then

$$G - x_i \preceq H - z_i \qquad i=1,2 \qquad (10)$$

Furthermore, if $G \prec H$, then either

$$G - x_1 \prec H - z_1 \qquad (11)$$

or $$G - x_2 \prec H - z_2 \qquad (12)$$

or both (11) and (12) are true.

Proof: Proof follows from theorems 1 and 2.

Theorem 4

If $G \sim H$, then $T(G,X) = T(H,Y)$ where X and Y are arbitrary HLF schedules.

Proof: The proof for when G and H are both chains follows from (2). The proof for when G and H have at least two initial tasks is an induction on the number of tasks in G and H.

It is important to realize here that due to the memoryless property of a random exponential distribution the costs of executing G and H can be expressed as

$$T(G,X) = 1/2 + 1/2 * T(G - x_1,X) + 1/2 * T(G - x_2X)$$

and

$$T(H,Y) = 1/2 + 1/2 * T(H - y_1,Y) + 1/2 * T(H - y_2,Y)$$

respectively. Corollary 1 and the induction hypothesis can be applied directly to these cost functions.

Theorem 4 implies that the mean finishing times of all HLF schedules for a given ExF are equal. Therefore, we henceforth define T(G,HLF) as the mean finishing time of all HLF schedules for the ExF G.

Theorem 5

Let G have at least two initial tasks. If G contains N tasks, then

$$\text{Level of } x_i \leq T(G,HLF) \leq N$$

Proof: By induction on N.

Theorem 6

If $G \prec H$, then $T(G,HLF) < T(H,HLF)$

Proof: By induction on M, the number of nodes in H. Theorems 2, 4 and 5 are used to prove four cases. These cases are 1) Both G and H have at least two initial nodes, 2) Both G and H have less than two initial nodes, 3) H has less than two initial tasks while G has at least two, and 4) G has less than two executable tasks while H has at least two.

Theorem 7

A two processor schedule on an ExF is optimal if and only if it is HLF.

Proof: By induction on the number of tasks. Note that since all HLF schedules have the same completion time it follows that all HLF schedules are optimal.

Theorems for Optimality of HLF for ErFs and Two Processors

An ErF is a forest where all task times are iid Erlang random variables. If preemption is not allowed with two processors and ErF precedence graph, then HLF is an optimal scheduling policy. The proof is inductive and the 2 induction assumptions are outlined below; proofs are found in a forthcoming paper.

The definitions of flatness have to be somewhat modified when non-preemptive tasks are considered. Let G be an ErF. Let a task at level ℓ in G have received t units of processing; this task will still have a processor assigned to it. We shall refer to this state of the system as (G,ℓ,t). The state (G,ℓ,t) is said to be flatter than the state (G',ℓ',t') if for $t \geq t'$,

$$S(G,m) \leq S(G',m) \text{ all } m$$

$$S(G-\ell,m) \leq S(G'-\ell',m) \text{ all } m$$

The induction assumptions are obviously true for ErFs with 1, 2, and 3 nodes. We show that if the assumption is true for ErFs with n-1 or fewer nodes, then it is true for ErFs with n nodes.

Induction Assumptions

For all ErFs with n or fewer nodes, in which a processor may have been assigned to a task.

(1) HLF is an optimal policy for processing all unprocessed tasks. (If a task is partially processed, the processor must complete that task and all succeeding tasks will be assigned in an HLF manner.)

(2) If G and G′ are ErFs, the optimal expected time for G cannot exceed the optimal expected time for G′.

If preemption is allowed, the optimal policy is to processor-share all tasks at the highest level if there are two or more tasks at the highest level; if there is only one task at the highest level, then it should be processed and all initial tasks at the next highest level should be processor-shared. We shall refer to this policy as the preemptive HLF policy. The theorem that the preemptive HLF policy is optimal for two processors and ErFs is proved inductively in a manner similar to that of the non-preemptive case.

The Erlang random variable may be represented by a sequence of iid exponential random variables. Since preemption is allowed we may have several partially processed tasks. The "state" of a partially processed task consists of the number of exponential "stages" left in that task. The state of the ErF is the forest of unfinished exponential stages. Note that the scheduling policy cannot know the state of the system because it cannot know the number of exponential stages left in a partially processed task, but it may attempt to surmise the state from the amount of processing that each task has had.

Given a state G of an ErF we define an ExF G′ with a one-to-one correspondence between stages in G and tasks in G′; if stage i must precede stage j in G then task i must precede task j in G′. If G and H are states of ErFs and G′ and H′ are corresponding ExFs, we define G∝H if and only if G′ ∝ H′.

The inductive proof that the preemptive HLF policy is optimal for ErFs is similar to that for ExFs. The induction assumptions are: (1) Preemptive HLF policy is optimal for ErFs with n or fewer unfinished tasks (note: not stages).

(2) If G and H are ErFs and if G∝H, then the optimal expected completion time for G cannot exceed that for H.

REFERENCES

1. Coffman, E.G. and Denning, P.J., Operating Systems Theory, Prentice-Hall, Englewood Cliffs, N.J. (1974).

2. Hu, T.C., "Parallel sequencing and assembly line problems", Operations Research, 9, (Nov - Dec 1961), 841-848.

3. Coffman, E.G. and Graham, R.L., "Optimal scheduling for two-processor systems", Acta Informatica, 1.3. (1972), 200-213.

4. Muntz, R.R. and Coffman, E.G., "Optimal preemptive scheduling on two-processor systems", IEEE Trans. C-18, 11 (Nov 1969), 1014-1020.

5. Adam, T.T., Chandy, K.M., and Dickson, J.R., "A comparison of list schedules for parallel processing systems", Comm. ACM, 17, 12, (1974, 685-691.

6. Graham, R.L., "Bounds on Multiprocessing Timing Anomalies", SIAM J. Appl. Math., 17, 2 (1969), 416-440.

7. Muntz, R.R. and Coffman, E.G., "Preemptive Scheduling of Real-Time Tasks on Multi-processor Systems", J. ACM 17, 2 (Apr. 1972), 324-338.

8. Browne, J.C., Chandy, K.M., Hogarth, J. and Lee, C.C., "The Effect on Throughput of Multi-Processing in a Multi-Programming Environment", IEEE Trans. on Comp., C-22, 8 (Aug 1973), 728-735.

9. Brown, R.M., "An Analytic Model of a Large Scale Interactive System Including the Effects of Finite Main Memory", Univ. of Texas, Computer Science Report TR-31.

10. Lam, S. and Sethi, R., "Analysis of a Level Algorithm for Preemptive Scheduling". To appear in these proceedings.

11. Ullman, J. D., "Polynomial Completeness of the Equal Execution Time Scheduling Problem", Princeton Univ., Dept. of Elec. Engrg., Computer Science Report TR-115 (Dec. 1972).

Acknowledgement

This research was supported by NSF Grant DCR74-13302.