SIAM J. APPL. MATH.
Vol. 17, No. 2, March 1969

BOUNDS ON MULTIPROCESSING TIMING ANOMALIES*

R. L. GRAHAM†

1. Introduction. It is well known (cf. [5], [6], [8]) to workers in the field of parallel computation that a multiprocessing system consisting of many identical processors acting in parallel may exhibit certain somewhat unexpected "anomalies," even though the system operates under a rather natural set of rules; e.g., it can happen that increasing the number of processors can *increase* the length of time required to execute a given set of tasks. In this paper we study a typical model of such a multiprocessing system, and we determine the precise extent by which the execution time for a set of tasks can be influenced because of these timing anomalies. A special case of this model will be shown to generate an interesting number-theoretic question, partial answers to which are given in the latter half of the paper.

2. Description of the system; examples of anomalies. Let us suppose we are given n (abstract) identical processing units P_i, $i = 1, \cdots, n$, and a set of tasks $T = \{T_1, \cdots, T_r\}$ which is to be processed by the P_i. We are also given a partial order[1] $\prec$ on T and a function $\mu: T \to (0, \infty)$. Once a processor P_i begins to execute a task T_j, it works without interruption until the completion of that task, requiring altogether $\mu(T_j)$ units of time. It is also required that the partial order be respected in the following sense: If $T_i \prec T_j$ then T_j cannot be started until T_i has been completed. Finally, we are given a sequence $L = (T_{i_1}, \cdots, T_{i_r})$ consisting of all the tasks of T and called a *priority list*. The P_i execute the T_j as follows: Initially, at time 0, all the processors (instantaneously) scan the list L from the beginning, searching for tasks T_i which are "ready" to be executed, i.e., which have no predecessors under $\prec$. The first ready task T_j in L which P_i comes to is started by P_i; P_i continues to execute T_j for the $\mu(T_j)$ units of time required to complete T_j. In general, at any time a processor P_i completes a task, it immediately scans L for the first available ready task to execute. If there are currently no such tasks, then P_i becomes idle. (We shall also say that P_i is executing an *empty task* denoted by φ_k.) P_i remains idle until some other P_j completes a task, at which time P_i (and, of course, P_j) immediately scans L for ready tasks (which may now exist because of the completion of P_j). If two (or more) processors both attempt to start executing a task, it will be our convention to assign the task to the processor with the smaller index. The least time at which all tasks of T have been completed will be denoted by ω. In the interests of mathematical rigor, it will be convenient to consider the $\mu(T_j)$ units of time required for the execution of T_j to be a *half-open* interval $[t, t + \mu(T_j))$ on the time axis.

---

* Received by the editors January 7, 1968. Presented by invitation at the Symposium on Combinatorial Mathematics, sponsored by the Office of Naval Research, at the 1967 Fall Meeting of Society for Industrial and Applied Mathematics held at the University of California, Santa Barbara, November 29–December 2, 1967.

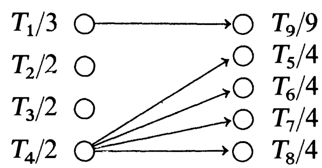
† Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey 07974.

[1] Compare with [4].

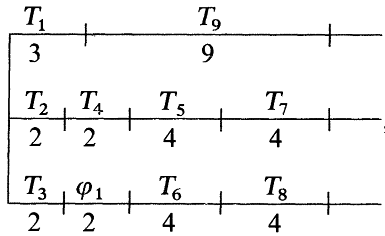
We now consider an example which illustrates the working of the preceding multiprocessing system and various anomalies associated with it. We indicate the partial order $\prec$ on T and the function μ by a *directed graph* $G(\prec, \mu)$. In $G(\prec, \mu)$ the vertices correspond to the T_i and a directed edge from T_i to T_j denotes $T_i \prec T_j$. The vertex T_j of $G(\prec, \mu)$ will actually be labeled with the symbols $T_j/\mu(T_j)$. The activity of each P_i is conveniently represented by a *timing diagram D. D* consists of n horizontal half-lines (labeled by the P_i) in which each line is a time axis starting from time 0 and is subdivided into labeled half-open segments according to the corresponding activity of P_i.

Example. $n = 3; L = (T_1, T_2, \cdots, T_9)$.

$G(\prec, \mu)$:



D:



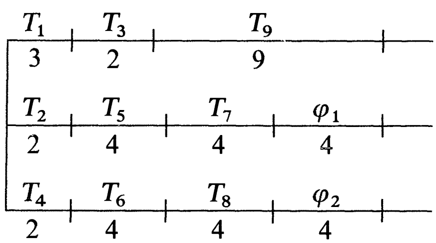
, $\omega = 12$.

Note that in D we have labeled the intervals above by the task and below by its length.

It is evident from the definition of ω that it is a function of L, μ, $\prec$ and n. Let us vary each of these four parameters in the example and see the effect this variation has on ω.

(i) Replace L by $L' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$, leaving μ, $\prec$ and n unchanged. In this case we obtain

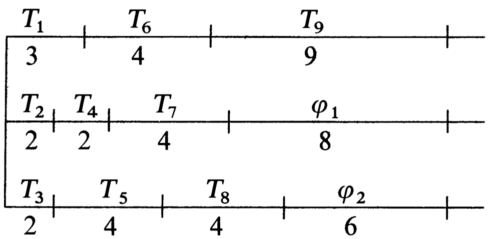
D':



and $\omega' = \omega'(L', \mu, \prec, n) = 14$.

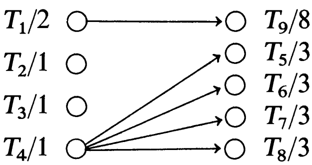
(ii) Change $\prec$ to $\prec'$ by removing $T_4 \to T_5$ and $T_4 \to T_6$. Then

D':



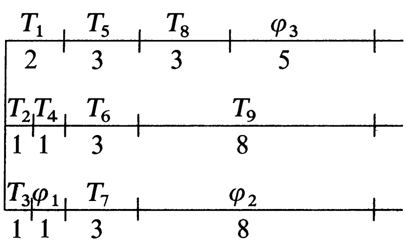
and $\omega' = \omega'\,(L, \mu, \prec', n) = 16$.

(iii) Decrease μ to μ' by defining $\mu'(T_i) = \mu(T_i) - 1$ for all i. In this case $G(\prec, \mu)$ becomes



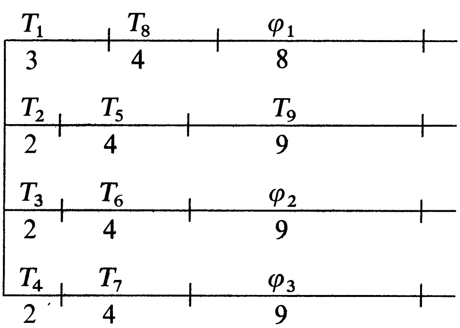
and

D':



with $\omega' = \omega'(L, \mu', \prec, n) = 13$.

(iv) Increase n from 3 to 4. Then



and $\omega' = 15$.

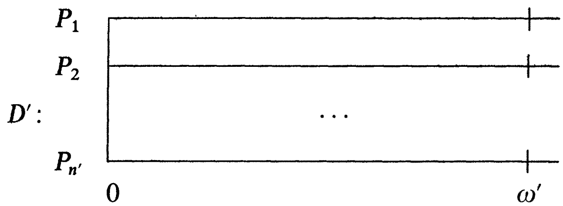
The examples in (ii), (iii) and (iv) show that contrary to what might be generally expected, *relaxing* $\prec$, *decreasing* μ, or *increasing* n can all cause ω to *increase.* In the next section we obtain an upper bound on the factor by which ω can increase by simultaneously changing L, relaxing $\prec$, decreasing μ and changing n. This bound is *optimal* in the sense that it cannot be replaced by any smaller function of the same variables.

3. The general bound. Suppose we are given a set T of tasks which we wish to execute two separate times. The first time we are given a time function μ, a partial order $\prec$, a priority list L and a multiprocessing system composed of n identical processors $P_i, i = 1, \cdots, n$. The second time we are given a time function $\mu' \leqq \mu$, a partial order[2] $\prec' \subseteq \prec$, a priority list L', and n' identical processors $P_i, i = 1, \cdots, n'$. Let ω and ω' denote the respective finishing times. Our next goal is to establish the following theorem.

THEOREM 1.

$$\frac{\omega'}{\omega} \leqq 1 + \frac{n-1}{n'}.$$

Proof. Consider the timing diagram D' obtained by executing the tasks T_i of T using the primed parameters.



The set of all points of time in $[0, \omega')$ can be partitioned into two subsets A and B. A is defined to be the set of all points of time for which *all* processors are executing some task of T. Similarly, B is defined to be the set of all points of time for which *at least one* processor is idle (but not all processors are idle). We note that both A and B are the disjoint union of half-open intervals. Let T_{j_1} denote a task which finishes in D' at time ω'. Let $S(T_{j_1})$ denote the time at which T_{j_1} is started. There are two possibilities. If $S(T_{j_1}) \in B$ and $S(T_{j_1})$ is not a boundary point of B, then by the definition of B there is some processor P_i which for some $\varepsilon > 0$ is idle during the time $[S(T_{j_1}) - \varepsilon, S(T_{j_1}))$. The question which naturally occurs is why T_{j_1} was not started until time $S(T_{j_1})$ when P_i was idle before and during this time. The only possible answer is that there must be some task T_{j_2} in D' such that $T_{j_2} \prec' T_{j_1}$ and T_{j_2} is completed at time $S(T_{j_1})$ (for this would certainly cause T_{j_1} to wait until time $S(T_{j_1})$ to be started).

On the other hand, suppose $S(T_{j_1}) \in A$ or $S(T_{j_1}) \neq 0$ is a boundary point of B and there exists $x < S(T_{j_1})$ such that $x \in B$. Let $x_1 = \text{l.u.b.}\{x : x < S(T_{j_1})$ and

[2] Since a partial order on T is a subset of $T \times T$, $\prec' \subseteq \prec$ has the obvious meaning.

$x \in B\}$. By the construction of A and B, we see that $x \in A$, and for some processor P_i and some $\varepsilon > 0$, P_i is idle during the time $[x_1 - \varepsilon, x_1)$. We again ask why T_{j_1} was not started during this time period. The only possible answer is that some task T_{j_2} must have been executed during this period upon which T_{j_1} depended in order to be started. For certainly if this were not the case, then either T_{j_1} or some predecessor of T_{j_1} should have been started during this time.

In either case ($S(T_{j_1}) \in A$ or B) we have seen that *either* there exists a task T_{j_2} which is completed at time $F(T_{j_2})$ such that $T_{j_2} \prec' T_{j_1}$ and $y \in [F(T_{j_2}), S(T_{j_1}))$ implies $y \in A$ *or* $x < S(T_{j_1})$ implies $x \in A$ or $x < 0$. We can repeat this construction inductively, forming $T_{j_3}, T_{j_4}, \cdots$, etc., until we reach a task T_{j_m} for which $x < S(T_{j_m})$ implies $x \in A$ or $x < 0$. Consequently, we have shown the existence of a *chain* of tasks

(1) $$T_{j_m} \prec' T_{j_{m-1}} \prec' \cdots \prec' T_{j_2} \prec' T_{j_1}$$

in D' such that *at every time $t \in B$, some T_{j_k} is being executed.* We say that this chain *covers B*. The important thing to notice about this chain is

(2) $$\sum_{\varphi_i' \in D'} \mu'(\varphi_i') \leqq (n' - 1) \sum_{k=1}^{m} \mu'(T_{j_k}),$$

where the left-hand sum is over all empty tasks φ_i' in D'. But (1) and the hypothesis $\prec' \subseteq \prec$ imply

(3) $$T_{j_m} \prec T_{j_{m-1}} \prec \cdots \prec T_{j_2} \prec T_{j_1}.$$

Thus

(4) $$\omega \geqq \sum_{k=1}^{m} \mu(T_{j_k}) \geqq \sum_{k=1}^{m} \mu'(T_{j_k}).$$

Consequently, by (2) and (4),

(5) $$\omega' = \frac{1}{n'}\left\{\sum_{T_k \in T} \mu'(T_k) + \sum_{\varphi_i' \in D'} \mu'(\varphi_i')\right\} \leqq \frac{1}{n'}(n\omega + (n' - 1)\omega).$$

From this we obtain

(6) $$\frac{\omega'}{\omega} \leqq 1 + \frac{n-1}{n'},$$

and the theorem is proved.

Examples are given in [2] which show that the bound in (6) is best possible. In fact, for $n = n'$, it is shown that the ratio of $2 - 1/n$ for ω'/ω can be achieved (to within an arbitrary $\varepsilon > 0$) by the variation of any one of L, μ or $\prec$. It can be noted that for $n = 1$, ω' is never greater than ω, while for $n > 1$, ω' can be greater than ω even though n' is quite large.

4. A modified system. It may be pointed out that it is quite reasonable to consider a multiprocessor system in which the priority list L is "dynamically formed" as opposed to the fixed list we have used thus far. For example one quite reasonable way of doing this is as follows: At any time a processor is free, it immediately begins to execute the ready task which currently heads the *longest chain* of unexecuted tasks (in the sense that the sum of the task times in the chain is maximal). Suppose by following this algorithm of choosing tasks we have a finishing time of ω_L. If we denote by ω_0 the minimum possible finishing time (for all possible lists), then we would like to assert something about the ratio ω_L/ω_0. It follows from Theorem 1 that $\omega_L/\omega_0 \leqq 2 - 1/n$; we could hope in fact that ω_L/ω_0 would always be considerably closer to 1 than this. Unfortunately, however, this is not the case since it can be shown that the best possible bound on this ratio is given by

$$\frac{\omega_L}{\omega_0} \leqq 2 - \frac{2}{n+1},$$

(which *is* a slight improvement). Similarly, we might use the algorithm that a processor always tries to execute the ready task T_i which has the *largest* sum $\mu(T_i) + \sum_{T_i \prec T_j} \mu(T_j)$ (i.e., the sum of the descendant lengths is maximal). If we denote the finishing time using this algorithm by ω_M, then it is again possible to produce examples for which ω_M/ω_0 is as large as $2 - 2/(n + 1)$.

However, there is a special case for which it is possible to lower the preceding bounds significantly and still use algorithms which require relatively little effort. This is the case in which $\prec$ is empty, and it is this case to which we shall restrict our attention for the remainder of the paper.

5. The special case in which $\prec$ is empty. As before we are given tasks $T = \{T_1, \cdots, T_r\}$, a time function $\mu: T \to (0, \infty)$, and n processing units. We could ask for an algorithm for choosing the T_i for which the finishing time is *optimal*, i.e., as small as possible. However, in general, it seems quite likely[3] that this could only be achieved by an exponential (in r) number of steps, and even for moderate r, this would be prohibitive. It is more reasonable to ask for a method of obtaining a finishing time ω such that ω/ω_0 is known to be relatively close to 1, and only a modest amount of energy is expended in obtaining ω. The algorithm which generates ω_L described in the preceding section is an example of such a method. In this algorithm, since $\prec$ is now empty, a free processor always starts to execute the longest remaining unexecuted task. With ω_L as the finishing time for this algorithm we have the following theorem.

THEOREM 2.

$$\frac{\omega_L}{\omega_0} \leqq \frac{4}{3} - \frac{1}{3n}, \tag{7}$$

and this bound is best possible.

[3] This has never been proved.

Proof. Assume there exist a set of tasks $T = \{T_1, \cdots, T_r\}$ and $\mu: T \to (0, \infty)$ which contradict (7). Let α_i denote $\mu(T_i)$ and let us renumber the T_i so that

$$\alpha_1 \geqq \alpha_2 \geqq \cdots \geqq \alpha_r. \tag{8}$$

The theorem clearly holds for $n = 1$; hence, we can assume that $n \geqq 2$ and r is minimal.

First note that, by the definition of ω_L, the order in which the tasks are executed corresponds precisely to using the priority list $L = (T_1, T_2, \cdots, T_r)$. Suppose in the corresponding timing diagram D_L, T_m is a task with the latest finishing time (which must be ω_L), where $m < r$. If we consider the truncated set $T' = \{T_1, \cdots, T_m\}$ with the list $L' = (T_1, \cdots, T_m)$, we see that the execution time ω' for T' using L' is exactly ω_L. On the other hand, for the optimal value ω'_0 for T', it is true that $\omega'_0 \leqq \omega_0$, where ω_0 denotes the optimal time for the set T. Hence

$$\frac{\omega'}{\omega'_0} \geqq \frac{\omega_L}{\omega_0} > \frac{4}{3} - \frac{1}{3n},$$

and the set T' forms a smaller counterexample to the theorem. This contradicts the minimality assumption on r. Thus, we can assume that T_r is the only task which finishes at time ω_L.

It is immediate that

$$\omega_0 \geqq \frac{1}{n} \sum_{i=1}^{r} \alpha_i. \tag{9}$$

Also, it follows that if τ denotes the starting time of T_r then

$$\sum_{i=1}^{r-1} \alpha_i \geqq n\tau, \qquad \omega_L = \tau + \alpha_r \tag{10}$$

since no processor is idle before T_r starts being executed. Therefore

$$\begin{aligned}\frac{\omega_L}{\omega_0} = \frac{\tau + \alpha_r}{\omega_0} &\leqq \frac{\alpha_r}{\omega_0} + \frac{1}{n\omega_0} \sum_{i=1}^{r-1} \alpha_i \\ &= \frac{(n-1)\alpha_r}{n\omega_0} + \frac{1}{n\omega_0} \sum_{i=1}^{r} \alpha_i \\ &\leqq \frac{(n-1)\alpha_r}{n\omega_0} + 1.\end{aligned}$$

Since T contradicts (7),

$$1 + \frac{(n-1)\alpha_r}{n\omega_0} \geqq \frac{\omega_L}{\omega_0} > \frac{4}{3} - \frac{1}{3n},$$

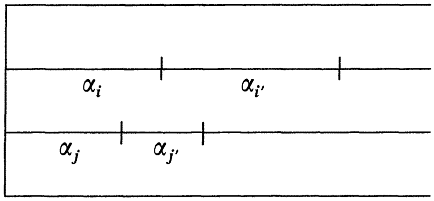
$$\frac{(n-1)\alpha_r}{n\omega_0} > \frac{1}{3} - \frac{1}{3n} = \frac{n-1}{3n},$$

and finally

$$\alpha_r > \frac{\omega_0}{3}. \tag{11}$$

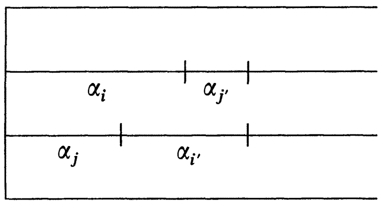
Hence, if (7) is false, in an optimal solution (which has timing diagram D_0), *no processor can execute more than two tasks.*

Suppose the following configuration occurs in D_0:

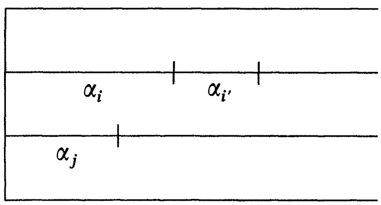
D_0:


, $\alpha_i > \alpha_j$, $\alpha_{i'} > \alpha_{j'}$.

If we interchange $\alpha_{i'}$ and $\alpha_{j'}$ to form

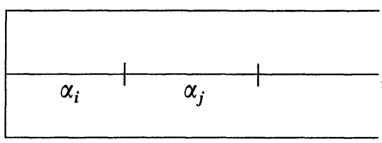
D_0':


then in D_0' the (possibly) new finishing time ω' certainly satisfies $\omega' \leqq \omega_0$, i.e., this single interchange could not have caused ω_0 to increase. Also, if the configuration

D_0:


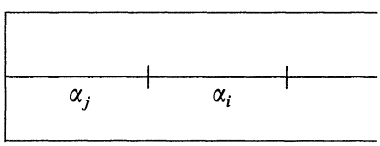
, $\alpha_i > \alpha_j$,

occurs in D_0, then moving α_i from the line with α_i to the line with α_j cannot cause ω_0 to increase. Let us call either of these two preceding operations a Type 1 operation. By a Type 2 operation on D_0 we mean changing any occurrence of

D_0:


, $\alpha_i < \alpha_j$,

to the form

D_0':


.

Clearly, this operation does not affect ω_0. For any timing diagram D we define a function $\mathscr{S}(D)$ as follows: Let F_i denote the *least* time t such that for every time $t' \geqq t$, the processor P_i is idle in D. Then

$$\mathscr{S}(D) = \sum_{1 \leqq i < j \leqq n} |F_i - F_j|.$$

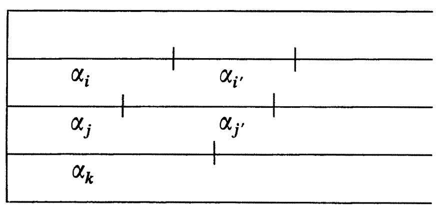
It is not difficult to check:

(a) If D' is obtained from D by a Type 1 operation, then $\mathscr{S}(D') < \mathscr{S}(D)$;

(b) if D' is obtained from D by a Type 2 operation, then $\mathscr{S}(D') = \mathscr{S}(D)$.

Now we start from D_0 and apply all possible Type 1 and Type 2 operations until the resulting timing diagram D^* has no internal configurations to which either type of operation may be applied. That such a D^* exists follows from the facts that there are only a finite number of possible arrangements of the r tasks on n lines, between two Type 1 operations only a finite number of Type 2 operations may be performed, and only a finite number of Type 1 operations may be performed because of (a). Hence, in D^* it follows that for any configuration of the form

D^*:

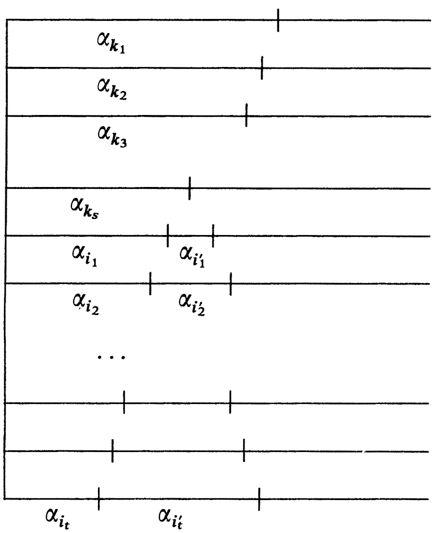


we have:

(12) $\qquad \alpha_i > \alpha_{j'}$ implies $\alpha_{j'} \geqq \alpha_{i'}, \quad \alpha_i \leqq \alpha_k$ and $\alpha_i > \alpha_{i'}$.

Thus, by a suitable rearrangement of the lines of D^* we can bring D^* into the form

$\bar{D}$:



where

$$\alpha_{k_1} \geqq \alpha_{k_2} \geqq \cdots \geqq \alpha_{k_s},$$

$$\alpha_{i_1} \geqq \alpha_{i_2} \geqq \cdots \geqq \alpha_{i_t},$$

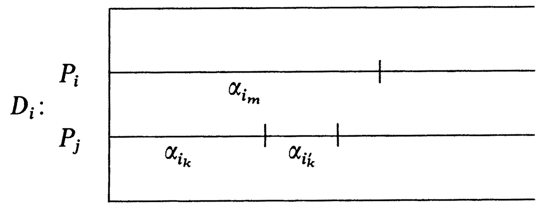
and, by (12),

$$\alpha_{i'_1} \leqq \alpha_{i'_2} \leqq \cdots \leqq \alpha_{i'_t}.$$

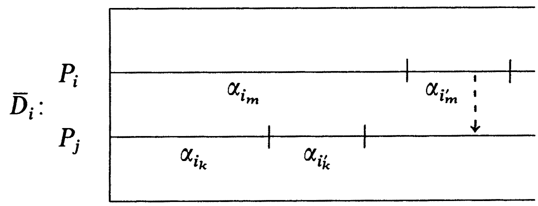
But (12) also implies $\alpha_{k_s} \geqq \alpha_{i_1}$ and $\alpha_{i_t} \geqq \alpha_{i'_t}$. Hence we combine these to obtain

$$(13) \qquad \alpha_{k_1} \geqq \cdots \geqq \alpha_{k_s} \geqq \alpha_{i_1} \geqq \cdots \geqq \alpha_{i_t} \geqq \alpha_{i'_t} \geqq \cdots \geqq \alpha_{i'_1}.$$

Since none of the operations applied to D_0 caused ω_0 to increase, by the optimality of ω_0 the finishing time of $\bar{D}$ must also be ω_0. But note that $\bar{D}$ looks very much like[4] the timing diagram D_L obtained by using the decreasing length list $(T_1, \cdots, T_r)$. In fact the only way in which D_L could differ from $\bar{D}$ is in the assignment of the second-layer tasks $T_{i'_c}$. Specifically, a difference could occur only if for some pair T_{i_k}, $T_{i'_k}$ we have $\alpha_{i_k} + \alpha_{i'_k} \leqq \alpha_{i_m}$ for some $m < k$,



In this case, in D_L, $T_{i'_m}$ with length $\alpha_{i'_m}$ might be assigned to P_j instead of P_i. However, if this situation were possible, then, in $\bar{D}$,



and it would be possible to move $\alpha_{i'_m}$ from P_i to P_j; and since the finishing time is not increased, it is still ω_0. This is a contradiction since this is now an optimal solution which has *three* tasks assigned to one processor. Hence, we conclude that D_L and $\bar{D}$ are isomorphic (in the obvious sense) and $\omega_0 = \omega_L$. But this *contradicts* the hypothesis that $\omega_L/\omega_0 > 4/3 - 1/(3n)$, and the validity of the bound given by the theorem is established.

To show that this bound is best possible, we consider the following set of task lengths:

$$(\alpha_1, \alpha_2, \cdots, \alpha_r) = (2n - 1, 2n - 1, 2n - 2, 2n - 2, \cdots, n + 1, n + 1, n, n, n),$$

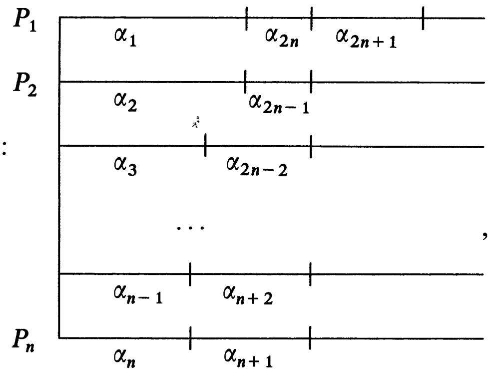
[4] Up to renaming tasks of equal length.

where $r = 2n + 1$. Specifically we have

$$\alpha_k = 2n - \left[\frac{k+1}{2}\right], \qquad k = 1, \cdots, 2n, \quad \text{and} \quad \alpha_{2n+1} = n.$$

In this case

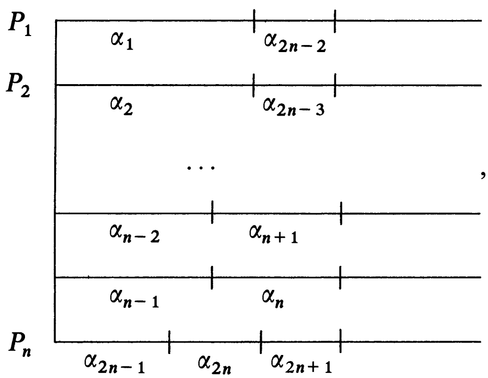
D_L:



, $\omega_L = 4n - 1,$

and

D_0:



, $\omega_0 = 3n,$

and therefore

$$\frac{\omega_L}{\omega_0} = \frac{4}{3} - \frac{1}{3n}.$$

It will be admitted that the bound $4/3 - 1/(3n)$ is not the first expression which comes to mind if one were to make an a priori guess at the answer. In the following section, however, it will be seen that this as well as the earlier bound of $2 - 1/n$ are both simple special cases of a more general result.

6. A general algorithm. Following a suggestion of D. Kleitman and D. Knuth, we were led to consider the following algorithm (still for the case in which $\prec$ is empty): For an integer $k \geqq 0$, choose the k longest tasks of the set of tasks $T = \{T_1, \cdots, T_r\}$ and arrange them in a list L which gives the *optimal* solution

ω_k for these k tasks. Now, extend L to a sequence containing all the tasks of T by adjoining the remaining $r - k$ tasks arbitrarily to L, forming the list $L(k)$. Let $\omega(k)$ denote the finishing time in the timing diagram $D(k)$ for T using $L(k)$. Again, let ω_0 denote the minimum possible finishing time for T. Our final result is the following theorem (where $[\cdot]$ denotes the greatest integer function).

THEOREM 3.

$$\frac{\omega(k)}{\omega_0} \leqq 1 + \frac{1 - 1/n}{1 + [k/n]}.$$

This bound is best possible for $k \equiv 0 \pmod{n}$.

Proof. If $\omega(k) = \omega_k$, then $\omega(k) = \omega_0$ and the theorem holds. We can therefore assume $\omega(k) > \omega_k$. We can also assume $r > k$. Let α^* denote $\max_{k+1 \leqq j \leqq r} \{\alpha_j\}$ where, as before, $\alpha_j = \mu(T_j)$. By the definition of α^* and the rules of operation of the multiprocessor system it follows that no processor can be idle before time $\omega(k) - \alpha^*$. Hence,

$$\sum_{j=1}^{r} \alpha_j \geqq n(\omega(k) - \alpha^*) + \alpha^* \tag{14}$$

and consequently,

$$\omega_0 \geqq \frac{1}{n} \sum_{j=1}^{r} \alpha_j \geqq \omega(k) - \left(\frac{n-1}{n}\right)\alpha^*. \tag{15}$$

There are at least $k + 1$ tasks T_j which have length $\geqq \alpha^*$. Hence, some processor must execute at least $1 + [k/n]$ of these "long" tasks. This implies

$$\omega_0 \geqq \left(1 + \left[\frac{k}{n}\right]\right)\alpha^*. \tag{16}$$

By (15) and (16) we finally obtain

$$\omega(k) \leqq \omega_0 + \left(\frac{n-1}{n}\right)\alpha^* \leqq \omega_0\left(1 + \frac{n-1}{n}\,\frac{1}{1 + [k/n]}\right), \tag{17}$$

which is the bound stated in the theorem.

To show that this bound is best possible when $k \equiv 0 \pmod{n}$ we present the following example: Define α_i for $1 \leqq i \leqq k + 1 + n(n-1)$ by

$$\alpha_i = \begin{cases} n & \text{for } 1 \leqq i \leqq k+1, \\ 1 & \text{for } k+2 \leqq i \leqq k+1+n(n-1). \end{cases}$$

For this set of tasks and the list $L(k) = (T_1, \cdots, T_k, T_{k+2}, \cdots, T_{k+1+n(n-1)}, T_{k+1})$ we have $\omega(k) = k + 2n - 1$. Since $\omega_0 = k + n$,

$$\frac{\omega(k)}{\omega_0} = \frac{k + 2n - 1}{k + n} = 1 + \frac{n-1}{k+n} = 1 + \frac{1 - 1/n}{1 + k/n} = 1 + \frac{1 - 1/n}{1 + [k/n]}.$$

This completes the proof of the theorem.

We have already seen several special cases of Theorem 3. For $k = 0$, we have

$$\frac{\omega(0)}{\omega_0} \leqq 2 - \frac{1}{n},$$

which is also implied by Theorem 1 for $n = n'$. Theorem 3 implies

$$\frac{\omega_L}{\omega_0} \leqq \frac{\omega(2n)}{\omega_0} \leqq 1 + \frac{1 - 1/n}{1 + [2n/n]} = \frac{4}{3} - \frac{1}{3n},$$

which is also the bound of Theorem 2.

There is an obvious algorithm for achieving the optimal solution for the n largest tasks; namely, just assign one task to each processor. If the remaining $r - n$ tasks are chosen arbitrarily, then by Theorem 3 we conclude

$$\frac{\omega(n)}{\omega_0} \leqq \frac{3}{2} - \frac{1}{2n}.$$

It would be interesting to know other simple algorithms which are optimal for the cases $r = 3n, 4n$, etc.

The problem we have been considering is equivalent to the following: We are given a set (with possible repetition) of positive real numbers $A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$. For each partition π of A into n subsets $A_1, \cdots, A_n$, let $m(\pi)$ denote $\max_i \sum_{\alpha \in A_i} \alpha$ and let m_0 denote $\min_\pi m(\pi)$. For a given $\varepsilon \geqq 0$, we wish to "efficiently" determine a partition $\pi = \pi(\varepsilon)$ such that

$$\frac{m(\pi)}{m_0} \leqq 1 + \varepsilon.$$

We must keep in mind, of course, that we can always find a partition π such that $m(\pi) = m_0$ just by a finite enumeration of *all* partitions of A. This algorithm, however, requires an "exponential" amount of work (in terms of the number of tasks r). On the other hand, by simply ordering the α_i into a nonincreasing sequence, which can be done in roughly $r \log_2 r$ comparisons, we can form a partition π for which $m(\pi)/m_0 < 1 + \frac{1}{3}$ (by Theorem 2).

Clearly a basic problem in this area is to make the preceding concept of "amount of work" precise and to develop strong upper and lower bounds on the work needed to achieve near optimal solutions. For example, suppose we restrict ourselves to the two operations of *addition* and *comparison* and assume $n = 2$. It is probably true that there exists a constant $C > 1$ such that any algorithm which determines an optimal partition π (i.e., such that $m(\pi) = m_0$) for any finite set of tasks T must require at least $C^{|T|}$ operations. However, this is not known at present.

REFERENCES

[1] E. L. CODD, *Multiprogram scheduling*, Comm. ACM, 3 (1960), pp. 347–350.

[2] R. L. GRAHAM, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[3] J. HELLER, *Sequencing aspects of multiprogramming*, J. Assoc. Comput. Mach., 8 (1961), pp. 426–439.

[4] J. L. KELLEY, *General Topology*, Van Nostrand, Princeton, 1955.
[5] B. LIEBESMAN, *The use of a special algebra in schedule analysis*, to appear.
[6] G. K. MANACHER, *Production and stabilization of real-time task schedules*, J. Assoc. Comput. Mach., 14 (1967), pp. 439–465.
[7] B. P. OCHSNER, *Controlling a multiprocessor system*, Record 44, Bell Laboratories, 1966, pp. 59–62.
[8] P. RICHARDS, *Parallel programming*, Rep. TD-B60-27, Technical Operations Inc., 1960.
[9] M. ROTHKOPF, *Scheduling independent tasks on one or more processors*, Interim Tech. Rep. 2, Operations Research Center, M.I.T., Cambridge, 1964.