Reprinted from THE BELL SYSTEM TECHNICAL JOURNAL
Vol. XLV, No. 9, November, 1966
Printed in U.S.A.

Bounds for Certain Multiprocessing Anomalies

By R. L. GRAHAM

(Manuscript received July 11, 1966)

It is known that in multiprocessing systems composed of many identical processing units operating in parallel, certain timing anomalies may occur; e.g., an increase in the number of processing units can cause an increase in the total length of time needed to process a fixed set of tasks. In this paper, precise bounds are derived for several anomalies of this type.

I. INTRODUCTION

In recent years there has been increased interest in the study of the potential advantages afforded by the use of a computer with many processors in parallel. While it is generally true that a set of tasks may be processed in less time by this type of multiprocessing, it has been pointed out that certain anomalies[1,2] may occur, even though the processors are used in a very "natural" way (e.g., it can happen that increasing the number of processors can *increase* the time required to complete a given set of tasks).

It is the purpose of this paper to derive precise bounds on the extent to which these anomalies can affect the time required to process a set of tasks, given certain rather natural rules for the operation of the multiprocessing system.

1.1 *Description of the System*

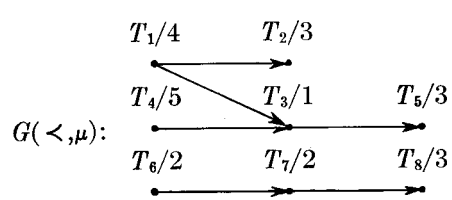
Let us suppose that we are given n identical processing units P_i, $1 \leqq i \leqq n$, and a set of tasks $T = \{T_1, \cdots, T_m\}$ to be processed by the P_i. We are also given a partial-order* $\prec$ on T and a function μ: $T \rightarrow [0, \infty)$. Once a processor P_i begins a task T_j, it works without interruption on T_j until completion of that task, taking altogether $\mu(T_j)$ units of time. It is also required that if $T_i \prec T_j$ then T_j cannot

* See Ref. 2.

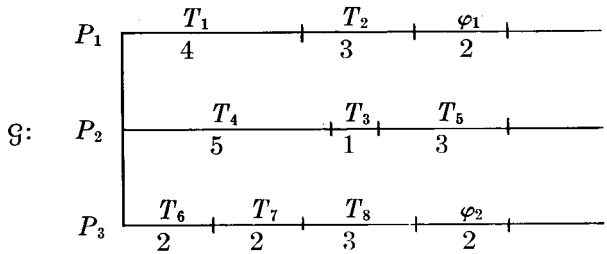
be started until T_i is completed. The P_i execute the T_j in the following way: We are given a linear ordering L: $(T_{k_1}, \cdots, T_{k_m})$ of T called a *task list* (or priority list). In general, at any time t a P_i completes a task, it immediately (and instantaneously) scans the list L (starting from the beginning) until it comes to the *first* task T_j which has not yet begun to be executed. If all the *predecessors* of T_j (i.e., those $T_i \prec T_j$) have been completed by time t then P_i begins working on T_j. Otherwise P_i proceeds to the *next* task $T_{j'}$, in L which has not yet begun to be executed, etc. If P_i proceeds through the entire list L without finding a task to execute then P_i becomes *idle* (we shall also say that P_i is working on an empty task). P_i remains idle until some other P_j *completes* a task at which time P_i (and of course P_j) immediately scans the list L as before for possible tasks to execute. If two processors P_i and P_j, $i < j$, simultaneously attempt to begin the same task T_k, it will be our convention to assign T_k to P_i, the processor with the smaller index. The processors all start scanning L at time $t = 0$ and proceed in the above-mentioned fashion until some time ω, the least time for which all the tasks have been completed.

It will be helpful here to consider several examples. We shall indicate the partial-order $\prec$ on T and the function μ by a *directed graph* $G(\prec,\mu)$. In $G(\prec,\mu)$, the vertices will correspond to the T_j and a directed edge from T_i to T_j will indicate that $T_i \prec T_j$. Each vertex of $G(\prec,\mu)$ will actually be labelled with the symbol $T_j/\mu(T_j)$, the $\mu(T_j)$ indicating the time necessary to execute T_j. The activity of each P_i is conveniently represented by a *timing diagram* $\mathcal{G}$ (also known as a Gantt diagram; see Ref. 1). $\mathcal{G}$ will consist of n horizontal half-lines (labelled by the P_i) in which each line is subdivided into segments* and labelled according to the state of the corresponding processor.

Example 1: $n = 3$, L: $(T_3, T_1, T_2, T_4, T_6, T_5, T_7, T_8)$

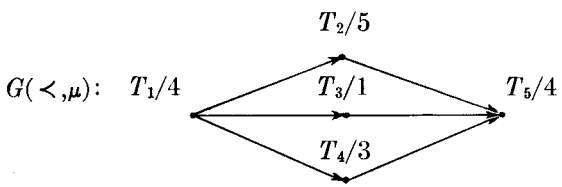


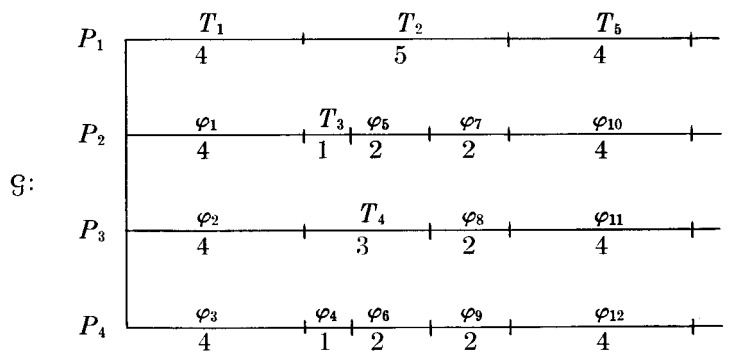
* We always consider the segments as being closed on the left and open on the right.



The symbol φ_i indicates a processor is idle (i.e., working on the empty task φ_i) but not all the T_j have been completed. The indexing of the φ_i is arbitrary. Thus, for $\mathcal{S}$ we have $\omega = 9$.

Example 2: $n = 4$, L: $(T_1, T_2, T_3, T_4, T_5)$





Here, $\omega = 13$. Note that in this example, ω is independent of L. We should also point out here that we are using the convention that whenever any T_j is completed, then *all* current empty tasks φ_i are also terminated. Processors still idle are then given "new" empty tasks to complete (e.g., P_4 in Example 2).

Example 3: $n = 3$, L: $(T_1, T_2, T_3, T_4, T_5, T_6, T_7)$

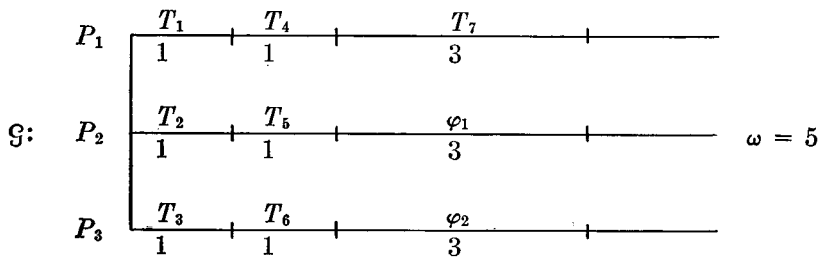
$G(\prec,\mu)$:

$T_1/1$.

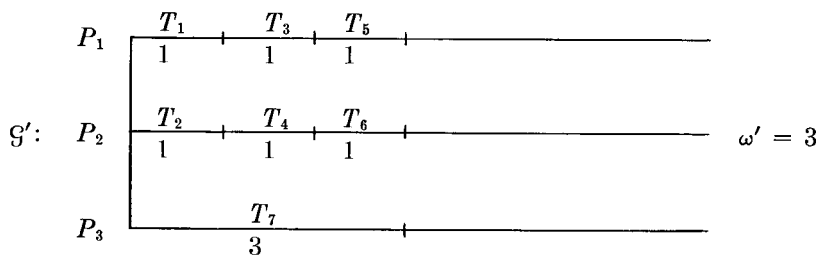
$T_2/1$ . . $T_5/1$

$T_3/1$ . . $T_6/1$

$T_4/1$ . . $T_7/3$



Suppose we use a different list L' given by L': $(T_1, T_2, T_7, T_3, T_4, T_5, T_6)$. We then have



Hence, by simply using a different list L', we have shortened ω by nearly a factor of two. The significance of this and similar examples will be brought out in the next section.

We see that, in general, ω is a function of the task list L, the "time" function μ, the partial-order $\prec$, and the number of processors n (in addition to the rules under which the P_i operate). In this note, we investigate the factor by which ω can increase if we simultaneously:

(*i*) Change* the task list L;

(*ii*) Decrease the function μ;

(*iii*) Relax the partial-order $\prec$;

(*iv*) Change the number of processors from n to n'.

While it might first be expected that (*ii*), (*iii*), or (*iv*) (with $n' > n$) would cause a decrease in ω, easy counterexamples† show that is not always the case. In the next section we obtain an upper bound on the factor by which ω can increase because of (*i*), (*ii*), (*iii*), and (*iv*) (cf. Theorem, p. 1571). This bound is just the expression $1 + n - 1/n'$. We also show that this bound is the *best possible* in the sense that it cannot be replaced by any smaller function of n and n'.

II. THE MAIN RESULTS

We begin this section by considering a special case of the general problem. We include this here in order to acquaint the reader with the basic ideas which will be used later. Suppose we are given a set of tasks $T = \{T_1, \cdots, T_m\}$ and a directed graph $G(\prec,\mu)$ giving a partial-order $\prec$ and a time function μ on T. We execute these tasks twice, each time using two identical processors P_1 and P_2. The first time the tasks are executed we use a task list L while the second time the tasks are executed we use another task list L'. Suppose the corresponding finishing times are ω and ω'. The question we consider now is this: How much can the ratio ω'/ω vary? This is answered by the following

Proposition: $\frac{2}{3} \leqq \frac{\omega'}{\omega} \leqq \frac{3}{2}$.

Proof: By the symmetry of ω and ω' it suffices to show that $\omega'/\omega \leqq \frac{3}{2}$. The basic idea we shall use is a simple one. Consider the timing diagram $\mathcal{G}$ obtained when the tasks are executed using the list L. We want to show that there is a *chain*‡ of tasks $T_{c_1} \prec T_{c_2} \prec \cdots \prec T_{c_r}$ which has the property that *whenever a processor is idle* (i.e., executing an empty task φ_i) *then the other processor is executing one of the* T_{c_k}.

* By "change" we mean "possibly change", etc.

† As far as the author is aware, these facts were first pointed out by Richards.[3]

‡ i.e., a linearly-ordered subset using the partial-order $\prec$.

To define the T_{c_k} we proceed as follows. First, let T_{j_1} be defined to be the task which has the latest finishing time in $\mathcal{G}$ (if there is more than one such task then we choose the task which is executed by the higher-indexed processor). Let φ_{t_1} be the empty task which has the *latest finishing time of all those empty tasks which finish at a time not later than the starting time of* T_{j_1}. By the construction of $\mathcal{G}$, there must be a task T_u which has the *same finishing time as* φ_{t_1}. Define T_{j_2} to be T_u. In general, suppose we have defined T_{j_k} for some $k \geqq 2$. To define $T_{j_{k+1}}$, let φ_{t_k} be the empty task which has the latest finishing time of all those empty tasks φ_i which finish at a time not later than the starting time of T_{j_k}. (If there are no such φ_i then we are done, i.e., $T_{j_{k+1}}$ is not defined.) By hypothesis, there must be a task T_v which has the same finishing time as $\varphi_{t_{k+1}}$ and which has a starting time not later than the starting time of $\varphi_{t_{k+1}}$. Define $T_{j_{k+1}}$ to be T_v. We continue this algorithm for as long as possible, say, until we have defined $T_{j_1}, \cdots, T_{j_r}$.

We first note that since no processor works on *one* empty task φ_i while the other processor works on *more than one task*, then *at any time a processor is executing an empty task,* φ_i, *the other processor is executing one of the* T_{j_k}. We next claim that $T_{j_{k+1}} \prec T_{j_k}$ for $1 \leqq k < r$. Suppose this is not the case. If t_0 denotes the time at which a processor P_i started executing $\varphi_{t_{k+1}}$ then by the hypothesis concerning the operation of the processors, P_i should *not* have been idle (i.e., working on $\varphi_{t_{k+1}}$) since at least one task, namely T_{j_k}, was eligible to be executed at that time. Thus, the timing diagram $\mathcal{G}$ is not valid and we have a contradiction. Hence, we must have $T_{j_{k+1}} \prec T_{j_k}$ for $1 \leqq k < r$. By defining $T_{c_k} \equiv T_{j_{r+1-k}}$ for $1 \leqq k \leqq r$, the first assertion is proved. It follows at once that if we let $\mu(\varphi_i)$ denote the length of time a processor spends executing φ_i, then

$$\sum_{\varphi_i \in \mathcal{G}} \mu(\varphi_i) \leqq \sum_{k=1}^{r} \mu(T_{j_k}). \tag{1}$$

The proof of the proposition now follows directly. Let $T_{i_1} \prec T_{i_2} \prec \cdots \prec T_{i_s}$ be chosen (by the assertion just established) so that

$$\sum_{\varphi_i' \in \mathcal{G}'} \mu(\varphi_i') \leqq \sum_{k=1}^{s} \mu(T_{i_k}), \tag{2}$$

where the φ_i' are taken from $\mathcal{G}'$ (the timing diagram obtained when the list L' is used). Note that ω' can be written as:

$$\omega' = \tfrac{1}{2} \sum_{T_k \in T} \mu(T_k) + \sum_{\phi_i' \in \mathcal{G}'} \mu(\phi_i'). \tag{3}$$

From (2) and (3) we have

$$\omega' \leqq \tfrac{1}{2}\left(\sum_{T_k \in T} \mu(T_k) + \sum_{k=1}^{s} \mu(\phi_i')\right). \tag{4}$$

Since the following inequalities hold:

$$\omega \geqq \tfrac{1}{2} \sum_{T_k \in T} \mu(T_k) \tag{5}$$

$$\omega \geqq \sum_{k=1}^{s} \mu(T_{i_k}) \tag{6}$$

(where (6) follows from the fact that $T_{i_1} \prec T_{i_2} \prec \cdots \prec T_{i_s}$), then we have from (4), (5), and (6)

$$\omega' \leqq \tfrac{1}{2}(2\omega + \omega) = \frac{3\omega}{2}$$

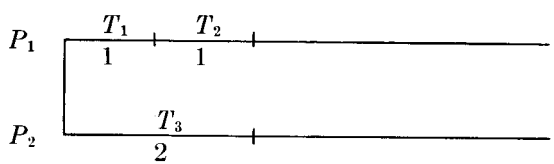
and the proposition follows.

The following example shows that the upper bound of $\frac{3}{2}$ cannot be replaced by any smaller value.

Example 4: $n = 2$, L: (T_1, T_3, T_2), L': (T_1, T_2, T_3)

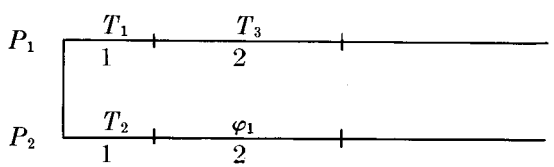
$G(\prec,\mu)$: $T_1/1$, $T_2/1$, $T_3/2$

$\mathcal{S}$:



$\omega = 2$

$\mathcal{S}'$:



$\omega' = 3$

Therefore, $\omega/\omega = \frac{3}{2}$ and the upper bound of the proposition is achieved.

Before stating the main theorem we introduce some notation. Let $T = \{T_1, \cdots, T_m\}$ be a set of tasks. Let $G = G(\prec,\mu)$ and $G' = G'(\prec',\mu')$ be two directed graphs for T with the partial-orders $\prec, \prec'$ and the time functions μ,μ'. We say that $G \leqq G'$ if:

(*i*) $\mu' \leqq \mu$, i.e., $\mu'(T_j) \leqq \mu(T_j)$ for all $T_j \in T$.

(*ii*) $\prec' \subseteq \prec$, i.e., $T_i \prec' T_j$ implies $T_i \prec T_j$ for all $T_i, T_j \in T$.

Finally, suppose we execute the tasks *twice*, one time using the graph G, a task list L and n processors, the other time using the graph G', a task list L' and n' processors. Let ω and ω' denote the respective finishing times. We then have the

Theorem: If $G' \leqq G$ *then*

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

Proof: By a slight modification of the argument used in the proposition, it follows that if φ_i', $1 \leqq i \leqq v$, denote the empty tasks of $\mathcal{G}'$ then there exists a chain of tasks $T_{i_1} \prec' T_{i_2} \prec' \cdots \prec' T_{i_s}$ of tasks in T with the property that *whenever a processor is idle then some other processor is executing one of the* T_{i_k}. From this we conclude

$$\sum_{\varphi_i' \in \mathcal{G}'} \mu'(\varphi_i') \leqq (n' - 1) \sum_{k=1}^{s} \mu'(T_{i_k}). \tag{7}$$

As before we note that

$$\begin{aligned} \omega' &= \frac{1}{n'}\left(\sum_{T_j \in T} \mu'(T_j) + \sum_{\phi_i' \in \mathcal{G}'} \mu'(\phi_i')\right) \\ &\leqq \frac{1}{n'}\left(\sum_{T_j \in T} \mu'(T_j) + (n' - 1) \sum_{k=1}^{s} \mu'(T_{i_k})\right) \end{aligned} \tag{8}$$

where the inequality follows by (7). Since

$$\omega \geqq \frac{1}{n} \sum_{T_j \in T} \mu(T_j) \geqq \frac{1}{n} \sum_{T_j \in T} \mu'(T_j) \tag{9}$$

and

$$\omega \geqq \sum_{k=1}^{s} \mu(T_{i_k}) \geqq \sum_{k=1}^{s} \mu'(T_{i_k}) \tag{10}$$

then by (8), (9), and (10) we conclude

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

Hence,

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

and the theorem is proved.

To show that this bound is best possible, we give several examples, which show that the bound can be attained (to within ε) by varying any *one* of the four parameters L, μ, $\prec$, or n.

Example 5: L is varied.

$$n = n', \qquad \mu = \mu', \qquad \prec = \prec'.$$

$$L = (T_1, T_2, \cdots, T_{n-1}, T_{2n-1}, T_n, T_{n+1}, \cdots, T_{2n-2})$$

$$L' = (T_1, T_n, T_{n+1}, \cdots, T_{2n-2}, T_2, T_3, \cdots, T_{n-1}, T_{2n-1})$$

$G(\prec,\mu)$:

$. T_1/1$

$. T_2/1$

$\vdots$

$. T_{n-1}/1$

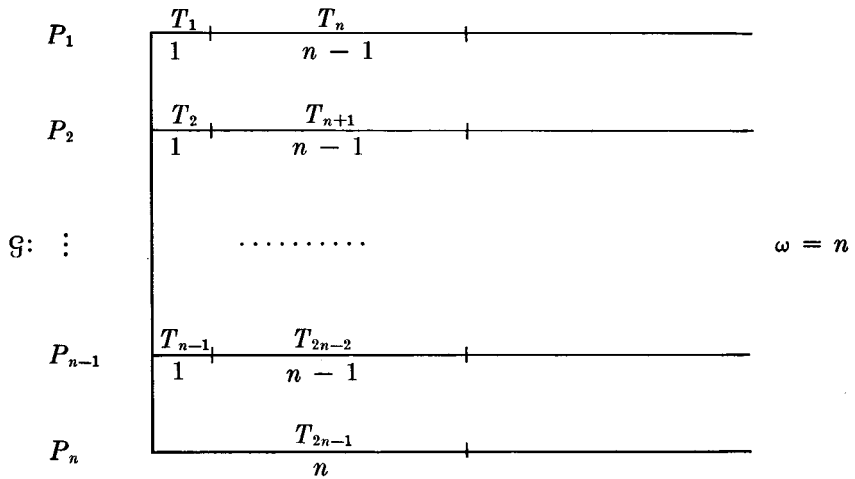
$. T_n/n - 1$

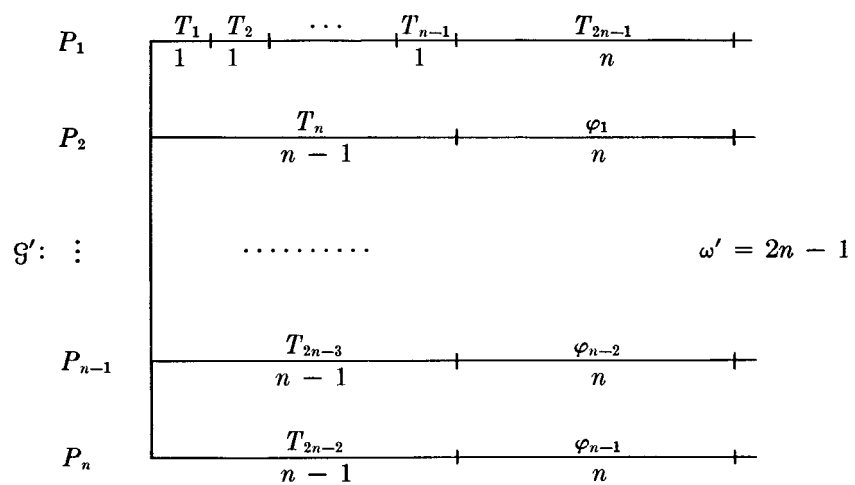
$. T_{n+1}/n - 1$

$\vdots$

$. T_{2n-2}/n - 1$

$. T_{2n-1}/n$


P_1
T_1 T_n
1 $n-1$
P_2
T_2 T_{n+1}
1 $n-1$
$\mathcal{G}$: ⋮ ..... $\omega = n$
P_{n-1}
T_{n-1} T_{2n-2}
1 $n-1$
P_n
T_{2n-1}
n


P_1
T_1 T_2 ⋯ T_{n-1} T_{2n-1}
1 1 1 n
P_2
T_n φ_1
$n-1$ n
$\mathcal{G}'$: ⋮ ..... $\omega' = 2n-1$
P_{n-1}
T_{2n-3} φ_{n-2}
$n-1$ n
P_n
T_{2n-2} φ_{n-1}
$n-1$ n

Thus,

$$\frac{\omega'}{\omega} = 2 - \frac{1}{n},$$

which is the value of $1 + (n - 1)/n'$ when $n = n'$.

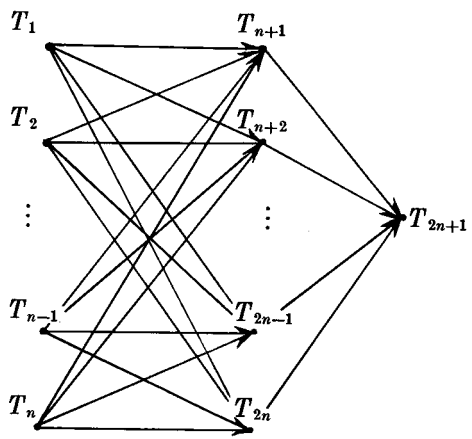
Example 6: μ is decreased.

$$n = n', \quad \mu \geqq \mu', \quad \prec = \prec'$$

$$L = L': \ (T_1, T_2, \cdots, T_{3n})$$

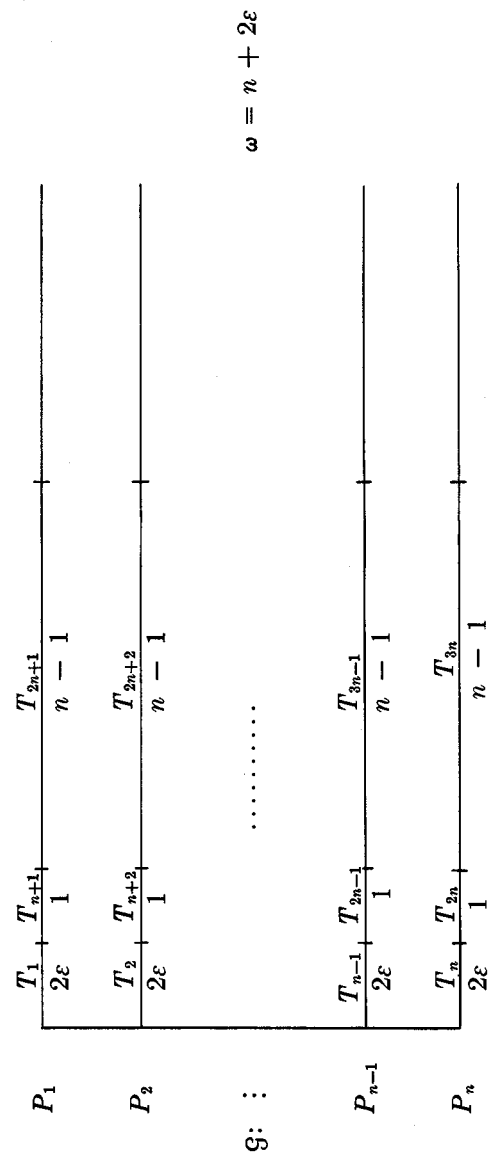
	$\mu(T_i)$	$\mu'(T_i)$
T_1	2ε	ε
T_2	2ε	ε
$\vdots$	$\vdots$	$\vdots$
T_{n-1}	2ε	ε
T_n	2ε	2ε
T_{n+1}	1	1
T_{n+2}	1	1
$\vdots$	$\vdots$	$\vdots$
T_{2n}	1	1
T_{2n+1}	$n - 1$	$n - 1$
T_{2n+2}	$n - 1$	$n - 1$
$\vdots$	$\vdots$	$\vdots$
T_{3n}	$n - 1$	$n - 1$

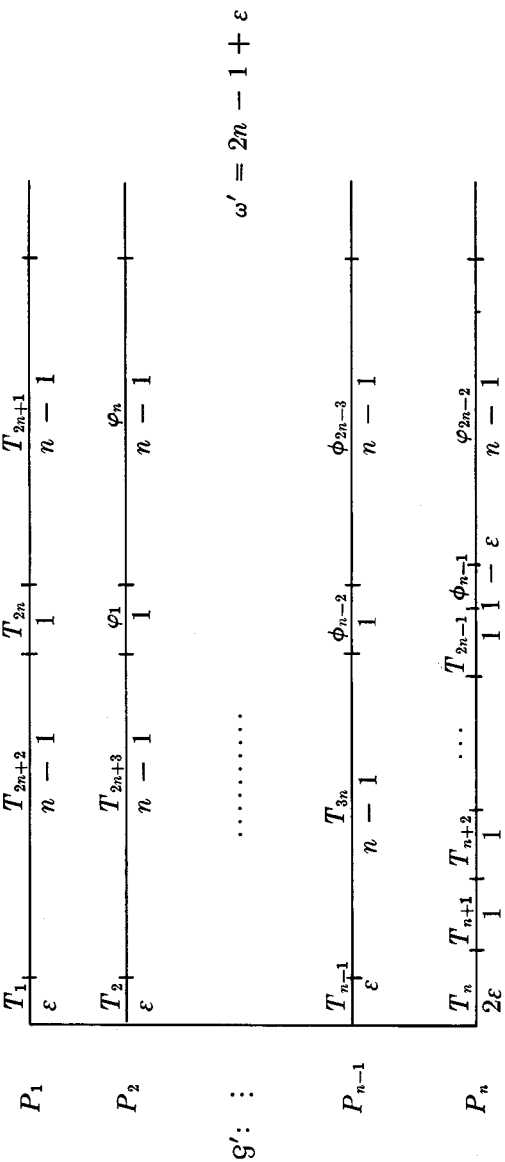
G:



T_{2n+2}, T_{2n+3}, $\vdots$, T_{3n}

(In G, $T_i \prec T_j \prec T_{2n+1}$ for $1 \leqq i \leqq n < j \leqq 2n$.)


ω = n + 2ε
P1
P2
P n−1
P n
T1
T n+1
T 2n+1
T2
T n+2
T 2n+2
T n−1
T 2n−1
T 3n−1
T n
T 2n
T 3n
2ε
1
n − 1


$\omega' = 2n - 1 + \varepsilon$
$\mathcal{G}'$: ⋮⋮
P_1
P_2
P_{n-1}
P_n
T_1 T_{2n+2} T_{2n} T_{2n+1}
T_2 T_{2n+3} φ_1 φ_n
T_{n-1} T_{3n} ϕ_{n-2} ϕ_{2n-3}
T_n T_{n+1} T_{n+2} T_{2n-1} ϕ_{n-1} φ_{2n-2}
ε $n-1$ 1 $n-1$
ε $n-1$ 1 $n-1$
ε $n-1$ 1 $n-1$
2ε 1 1 1 $1-\varepsilon$ $n-1$

Thus,

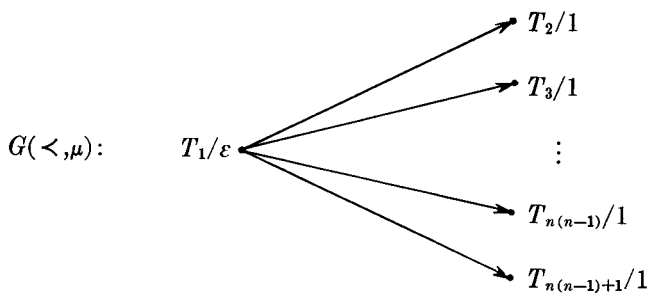
$$\frac{\omega'}{\omega} = \frac{2n - 1 + \varepsilon}{n + 2\varepsilon}$$

which is arbitrarily close to $2 - (1/n)$ for ε sufficiently small. We should note the interesting fact that $\omega' \geqq 2n - 1 + \varepsilon$ for *any* list L' which may be used.

Example 7: $\prec$ is relaxed.

$$n = n', \qquad \mu = \mu', \qquad \prec \supset \prec',$$

$$L = L': \quad (T_1, T_2, \cdots, T_{n(n-1)+2})$$



$$. \; T_{n(n-1)+2}/n$$

$$. \; T_1/\varepsilon$$

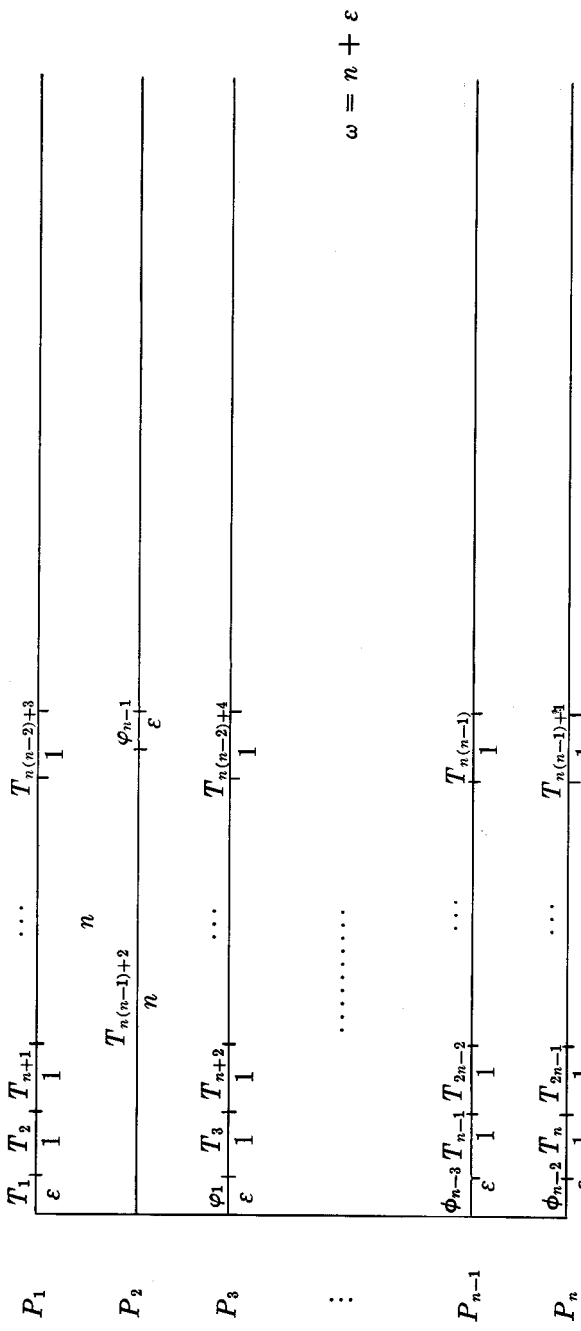
$$. \; T_2/1$$

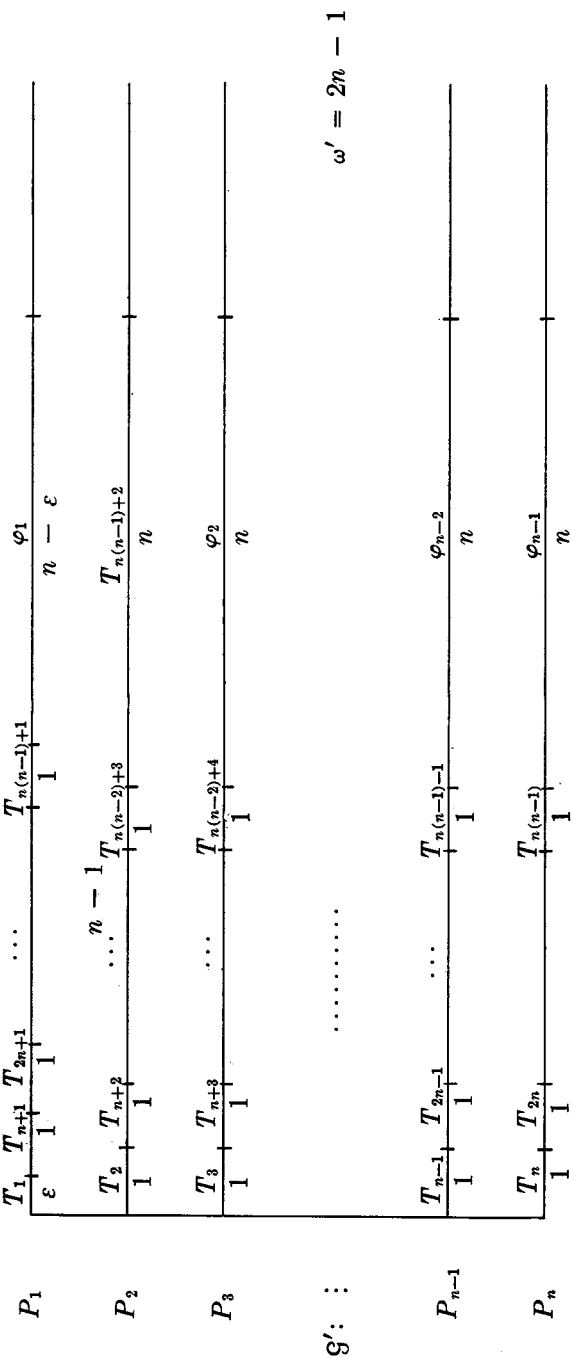
$G(\prec',\mu)$: $\vdots$

$$. \; T_{n(n-1)}/1$$

$$. \; T_{n(n-1)+1}/1$$

$$. \; T_{n(n-1)+2}/n$$


$\omega = n + \varepsilon$
P_1
P_2
P_3
P_{n-1}
P_n
T_1 T_2 T_{n+1} ... $T_{n(n-2)+3}$
ε 1 1 1
n
$T_{n(n-1)+2}$ φ_{n-1}
n ε
φ_1 T_3 T_{n+2} ... $T_{n(n-2)+4}$
ε 1 1 1
ϕ_{n-3} T_{n-1} T_{2n-2} ... $T_{n(n-1)}$
ε 1 1 1
ϕ_{n-2} T_n T_{2n-1} ... $T_{n(n-1)+1}$
ε 1 1 1


P_1
P_2
P_3
$\mathcal{G}'$: : :
P_{n-1}
P_n
T_1 T_{n+1} T_{2n+1} ... $T_{n(n-1)+1}$
T_2 T_{n+2} ... $T_{n(n-2)+3}$
T_3 T_{n+3} ... $T_{n(n-2)+4}$
T_{n-1} T_{2n-1} ... $T_{n(n-1)-1}$
T_n T_{2n} $T_{n(n-1)}$
ε 1 1 1 1
1 1 $n-1$ 1
1 1 1
1 1 1
1 1 1
φ_1 $n-\varepsilon$
$T_{n(n-1)+2}$ n
φ_2 n
φ_{n-2} n
φ_{n-1} n
$\omega' = 2n-1$

Thus,

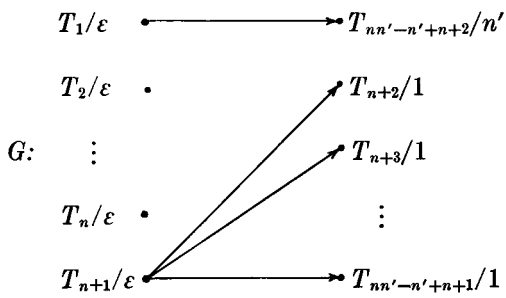
$$\frac{\omega'}{\omega} = \frac{2n - 1}{n + \varepsilon}$$

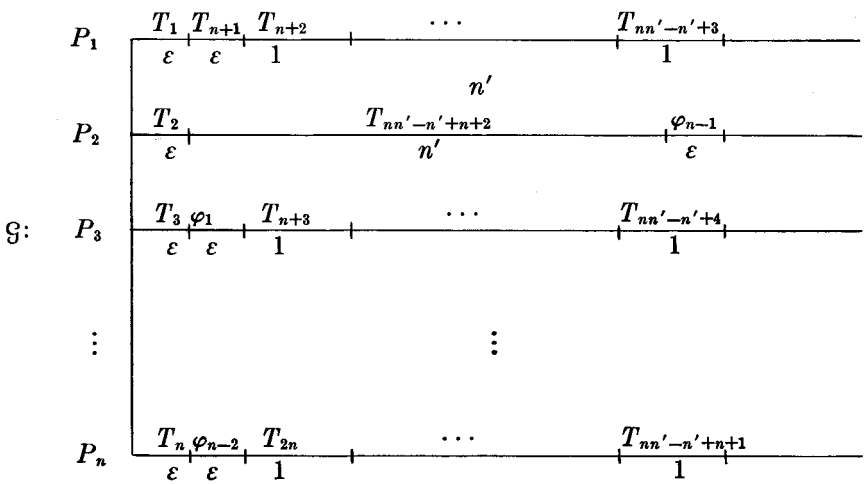
which is arbitrarily close to $2 - (1/n)$ for ε sufficiently small.

Example 8: n is varied.

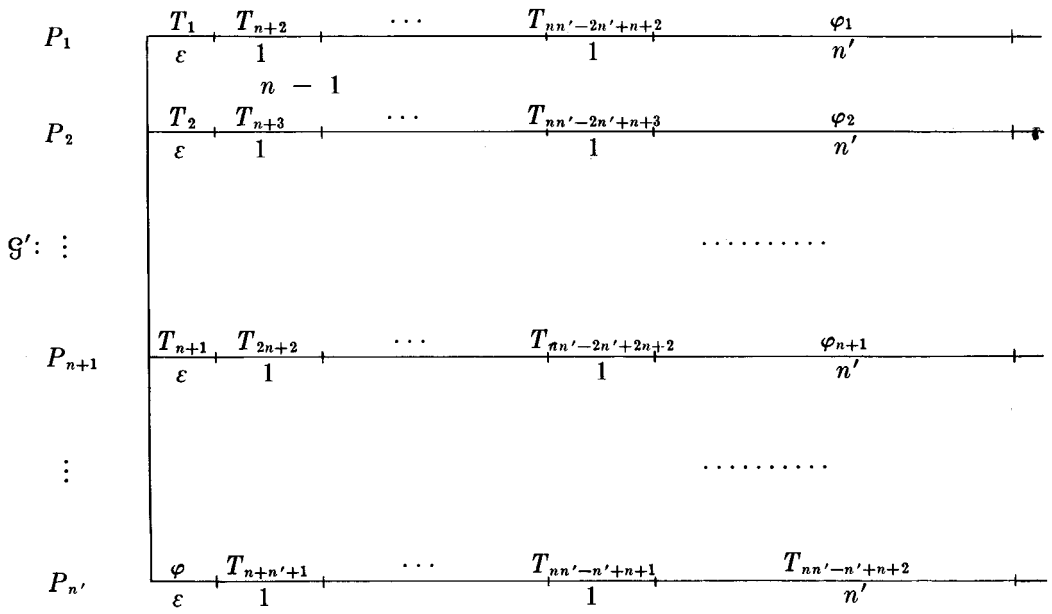
Case 1: $n < n'$, $\mu = \mu'$, $\prec = \prec'$

$L = L'$: $(T_1, T_2, \cdots, T_{nn'-n'+n+2})$





$$\omega = n' + 2\varepsilon$$



$$\omega' = n' + n - 1 + \varepsilon$$

Thus,

$$\frac{\omega'}{\omega} = \frac{n' + n - 1 + \varepsilon}{n' + 2\varepsilon}$$

which is arbitrarily close to $1 + (n - 1/n')$ for ε sufficiently small.

Case 2: $n > n'$. The construction in this case is similar to that of Case 1 and will not be presented.

We should note that in Example 8 we took $L = L'$. If it is of some consolation to a possibly battered intuition, it should be noted that if $n \leqq n'$, $\mu = \mu'$, and $\prec = \prec'$ then for any L which is chosen, it is possible to choose a suitable L' for which $\omega' \leqq \omega$.

III. CONCLUDING REMARKS

It should be pointed out here that we have not considered models of the multiprocessor system in which the priority list L is "dynamically formed" (as opposed to the *fixed* lists we have used thus far). For example, one seemingly quite reasonable way of doing this is as follows: At any time a processor is free, it immediately begins to execute the

"ready" task (i.e., one which has all its predecessors completed) which currently heads the *longest chain* of unexecuted tasks (including itself). Suppose by following this algorithm in choosing tasks, we have a finishing time of ω^*. If we denote by ω_o the least possible finishing time (minimized over *all* lists), then we would like to assert something about the ratio ω^*/ω_o. It follows from what has been proved in this paper that $\omega^*/\omega_o \leqq 2 - (1/n)$, (where n is the number of processors) and we would hope that, in fact, we could show ω^*/ω_o is considerably closer to 1 than this. Unfortunately, this is not possible since it can be shown that the best possible bound on this ratio is given by

$$\frac{\omega^*}{\omega_o} \leqq 2 - \frac{2}{n+1}.$$

It is interesting to note, however, that in the case in which the partial-order $\prec$ on the tasks is *empty*, then this bound can be improved* to

$$\frac{\omega^*}{\omega_o} \leqq \frac{4}{3} - \frac{1}{3n},$$

which, again, is best possible.

In conclusion, one might ask just how "typical" the examples are for which ω'/ω_o is close to the upper bound $2 - (1/n)$. While very little work has been done on this aspect, empirical results (using computer simulation (see Ref. 1)) indicate that examples in which $\omega'/\omega_o \geqq 1.1$ are quite common.

IV. ACKNOWLEDGMENTS

I wish to acknowledge here the stimulating discussions I have had on this subject with S. Lin and with G. K. Manacher, who originally brought these questions to my attention.

REFERENCES

1. Manacher, G. K., The Production and Stabilization of Real-Time Task Schedules, Institute for Computer Research, Quarterly report, Univ. of Chicago, May, 1966.
2. Ochsner, B. P., Controlling a Multiprocessor System, Bell Laboratories Record, February, 1966
3. Richards, P., Parallel Programming, Report No. TD-B60-27, Tech. Operations Inc., August, 1960.
4. Heller, J., Sequencing Aspects of Multiprogramming, JACM, *8*, 1961, pp. 426–439.
5. Kelley, J. L., *General Topology*, Van Nostrand, Princeton, 1955.

* The proofs of this and the preceding result will appear in a later paper.