MIT/LCS/TM-122

# EFFICIENT SCHEDULING OF TASKS WITHOUT FULL USE OF PROCESSOR RESOURCES

Jeffrey Jaffe

January 1979

# Efficient scheduling of tasks without full use of processor resources

Jeffrey Jaffe *

**Abstract.** The nonpreemptive scheduling of a partially ordered set of tasks on a machine with $m$ processors of different speeds is studied. Heuristics are presented which benefit from selective non-use of slow processors. The performance of these heuristics is asymptotic to $\sqrt{m}$ times worse than optimal, whereas demand driven schedules are unboundedly worse than optimal for any fixed value of $m$.

The algorithms are extended to the situation where functionally dedicated processors must process tasks of a given type. Here, too, the worst case performance of the algorithms improves on the worst case performance of known algorithms. The techniques of analyzing these schedules are used to obtain a bound on a large class of preemptive schedules.

**Keywords.** scheduling, list schedules, worst case performance bounds, preemptive and nonpreemptive schedules

---

* Laboratory for Computer Science, MIT, 545 Tech. Sq., Cambridge, MA 02139
This report was prepared with the support of a National Science Foundation graduate fellowship, and National Science Foundation grant no. MCS77-19754. The computations on MACSYMA were supported by ERDA contract no. E(11-1)-3070 and NASA grant no. NSG1323.

## 1. Introduction

The problem of nonpreemptive job scheduling on a machine with $m$ processors of different speeds was introduced by Liu and Liu [5,6]. They studied a class of schedules known as demand driven or list schedules. The characteristic property of these schedules is that at no time is there an idle processor at the same time that the system has an unexecuted executable task. They showed that any list schedule has a finishing time that is at most $1+(b_1/b_m)-(b_1/(b_1+\ldots+b_m))$ times worse than optimal where $b_i$ is the speed of the $i^{\text{th}}$ fastest processor (in this paper the optimal schedule will always be the one with least finishing time). In addition, examples were presented which showed that demand driven or list schedules did in fact perform as poorly as the bound. This is a discouraging result in that a large gap between the speeds of the fastest and slowest processors implies the relative ineffectiveness of list scheduling, independent of the speeds of the other processors or number of processors. List scheduling has been the prototype of approximation algorithms since its introduction in the identical processor case [1].

One way of avoiding the problem of this unboundedly bad behavior is to use preemptive scheduling. Horvath, Lam, and Sethi studied a "level algorithm" for the preemptive scheduling of tasks [2] which generalizes the algorithms of [8,9]. The worst case performance of this and many other preemptive algorithms is at most $(1/2)+\sqrt{m}$ times worse than optimal [3]. This class of schedules is sufficiently general that any schedule may be easily transformed into a schedule in the class, where the new schedule has a finishing time at least as small as the original schedule.

The focus of this paper is to provide nonpreemptive heuristics which are guaranteed to be no worse than $O(\sqrt{m})$ times worse than optimal, regardless of the speeds of the processors. While $\sqrt{m}$ and $b_1/b_m$ are strictly speaking incomparable (in that either may be smaller for any particular set of processor speeds), the natural way that the heuristics are developed guarantees that the worst case performance for any fixed set of processor speeds is not worse than the bound obtained in [5,6] for arbitrary list schedules. The basic strategy of the heuristic is to use only the fastest $i$ processors for an appropriately chosen value of $i$. After formal definitions are provided in Section 2, Section 3 describes which processors are to be used if $O(\sqrt{m})$ behavior is desired. A bound of $\sqrt{m} + O(m^{1/4})$ is obtained on the performance of the heuristic. Also, exact bounds are computed for small values of $m$ which describe the exact worst case performance of the heuristic. This is significant as $O(m^{1/4})$ is potentially a dominating factor for small values of $m$.

Recently, the scheduling of systems with different types of processors - dedicated to different types of tasks has been studied [4,7]. In [4], the behavior of schedules for such typed systems was analyzed when different processors of the same type may be of different speeds. As in the untyped case, the behavior of list schedules may be unboundedly bad for a fixed specification of the number of processors of each type.

Section 5 generalizes the heuristics of Section 3, and develops a heuristic which in the worst case is at most $k+2\sqrt{\max_j (km_j)}$ times worse than optimal where $m_j$ is the number of processors of type $j$ and $k$ is the number of types. Section 6 discusses the preemptive scheduling of typed task systems. Whereas the techniques of [3] did not generalize directly to provide a speed independent bound for the preemptive scheduling of typed task systems, the techniques used in this paper provide the insight needed to get speed independent bounds for preemptive scheduling of typed task systems.

## 2. Task Systems

A(n *ordinary*) *task system* $(\mathcal{T},<,\mu)$ consists of:

(1) The set $\mathcal{T}=(T_1,\ldots,T_r)$; the elements $T_i \in \mathcal{T}$ are called *tasks*.

(2) A *partial ordering* $<$ on $\mathcal{T}$.

(3) A *time function* $\mu:\mathcal{T}\rightarrow\mathbb{R}$.

The set $\mathcal{T}$ represents the set of tasks or jobs that need to be executed. The partial ordering specifies which tasks must be executed before other tasks. The value $\mu(T)$ is the *time requirement* of the task $T$.

When the set of processors $\mathcal{P}=\{P_i:1\leq i\leq m\}$ are not identical there is a *rate* $b_i$ associated with $P_i$ ($b_1\geq b_2\geq\ldots\geq b_m>0$). If a task $T$ is assigned to a processor $P$ with rate $b$, then $\mu(T)/b$ time units are required for the processing of $T$ on $P$. (When discussing a generic processor "$P$", the associated rate is taken to be "$b$".)

The execution of a task system by processors of a machine is modelled by the notion of a schedule. A *schedule* for $(\mathcal{T},<,\mu)$ *on a set of processors* $\mathcal{P}$ *with rates* $b_1,\ldots,b_m$ is a total function $S:\mathcal{T}\rightarrow\mathbb{R}\times\mathcal{P}$ (where if $S(T)=(t,P)$ then the *starting time* of $T$ is $t$ and the *finishing time* of $T$ is $t+(\mu(T)/b)$) such that

(a) For every task $T$, if $S(T)=(t,P)$, then no other task $T'$ may have $S(T')=(t',P)$ for any $t'\geq t$ which is earlier than the finishing time of $T$.

(*b*) whenever $T<U$ the starting time of $U$ is no less than the finishing time of $T$.

Condition (a) asserts that processor capabilities may not be exceeded. Condition (*b*) forces the obedience of precedence constraints.

If $S(T)=(t,P)$ then the task $T$ is *being executed on processor* $P$ *at time* $t'$ for $t \leq t' < t+(\mu(T)/b)$.

The *finishing time* of a schedule is the maximum finishing time of the set of tasks. An *optimal schedule* is any schedule that minimizes the finishing time. For two schedules $S$ and $S'$, with finishing times $w$ and $w'$ the *performance ratio of S relative to S'* is $w/w'$.

There are schedules that may be arbitrarily worse than the optimal schedule. For example, there may be an interval of time before the finishing time during which no task is being executed. Schedules of interest are not so blatantly wasteful. Thus earlier papers [1,5,6] have restricted attention to schedules conforming to a heuristic (or what may be used as part of a heuristic) that seems to be as useful as possible at each moment in time. Specifically, a (priority) *list* $L=(U_1,\ldots,U_r)$ $(U_i \in \mathcal{T})$ consists of a permutation of all the tasks of $\mathcal{T}$. The *list schedule* for $(\mathcal{T},<,\mu)$ with the list $L$ is defined as follows. At each point in time that at least one processor completes a task, each processor that is not still executing a task chooses an unexecuted executable task. The tasks are chosen by giving higher priority to those unexecuted tasks with the lowest indices. If $n>1$ processors simultaneously look for tasks, then the $n$ highest priority unexecuted, executable tasks are selected for execution. The decision as to which processor gets which task is made arbitrarily (or one may choose to assign higher priority tasks to faster processors). Only if not enough unexecuted tasks are executable do processors remain idle. It is important to note that any schedule that is unwasteful in the sense that processors are never permitted to be idle unless no free tasks are available can be formulated as a list schedule.

When the processors are of different speeds, list schedules may be as bad as $1+(b_1/b_m)-(b_1/(b_1+\ldots+b_m))$ times worse than optimal [5,6]. The reason for this "unboundedly" bad behavior is that an extremely slow processor may bottleneck the entire system by spending a large amount of time on a task. This motivates the following class of heuristics. A *list schedule on the fastest $i$ processors* has a priority list as above. The difference in the execution strategy is that the slowest $m-i$ processors are never used, and tasks are scheduled as if the only processors available were the fastest $i$ processors.

To analyze this class of schedules it will be useful to have the following definitions. A *chain* $C$ is a sequence of tasks $C=(U_1,\ldots,U_l)$ with $U_i \in \mathcal{T}$ such that for all $j$, $1 \leq j < l$, $U_j < U_{j+1}$. $C$ *starts* with task $U_1$. The *length* of $C$ is equal to $\Sigma_{j=1}^{l} \mu(U_j)$. The *height* of a task $T \in \mathcal{T}$ is the maximum over all chains starting with $T$ of the length of the chain. The *height* of $(\mathcal{T},<,\mu)$ is the maximum over all tasks $T \in \mathcal{T}$ of the height of $T$.

While the notion of the height of a task is a static notion which is a property of $(\mathcal{T},<,\mu)$, we also associate a dynamic notion of the height of a task with any schedule for $(\mathcal{T},<,\mu)$. Specifically, let $S$ be a schedule for $(\mathcal{T},<,\mu)$, and let $t$ be less than the finishing time of $S$. Then the *height of the task $T$ at the time $t$* is equal to the height of $T$ in the unexecuted portion of the task system (that is, the maximum over all chains starting with $T$ of the length of the chain, where the length of the chain considers only the unexecuted time requirements). Similarly, the *height of $(\mathcal{T},<,\mu)$ at time $t$* is the maximum over all tasks (not yet completed) $T \in \mathcal{T}$ of the height of $T$. Note that if a portion of a task has been finished at time $t$, then it contributes to the height the proportion of the time requirement not yet completed.

It will often be convenient to analyze portions of the schedule based on whether or not the height is decreasing during an interval of time. One may plot the height of $(\mathcal{T},<,\mu)$ as a function of time (for a given schedule $S$), and observe that it is a nonincreasing function that starts (at $t=0$) at the original height of $(\mathcal{T},<,\mu)$, and ends (at the finishing time) at height 0. If during an interval of time, the height was a monotonically decreasing function of time then that interval is called a *height reducing interval*. If during an interval the height is constant the interval is called a *constant height interval*. Any schedule may be completely partitioned into portions executed during height reducing intervals, and portions executed during constant height intervals.

## 3. List schedules on the fastest i processors

The first portion of this section entails an analysis of the worst case performance of list schedules on the fastest $i$ processors. Given a set of speeds $b_1,\ldots,b_m$, and given $i$, a bound will be obtained in terms of the parameters (the $b_j$'s and $i$). The second portion of this section analyzes this bound more carefully, and indicates why for each set of processor speeds, an easily determined value of $i$ causes the performance ratio to be no worse than $1+2\sqrt{m}$ times worse than optimal. A more complicated analysis then shows that in fact some value of $i$ will permit a ratio of no worse than $\sqrt{m}+O(m^{1/4})$. The final portion of this section provides examples that indicate that the performance bound is the correct order of magnitude. For certain sets of speeds, our heuristic and a class of related heuristics are as bad as $\sqrt{m-1}$ times worse than optimal.

It is easy to get a speed independent bound for one class of schedules, but this bound is not very good. It is not hard to see that if only the fastest processor is used, and it is always used, that the resulting schedule is no worse than $m$ times worse than optimal. The bound of this section (which discusses a natural generalization of using only the fastest processor) is substantially better than this.

## 3.1 Performance bound on list schedules on the fastest i processors

The approach to be used is to obtain two lower bounds on the optimal schedule for a given task system, and to compare them to an upper bound on the effectiveness of the schedule of interest. The resulting ratio is then an upper bound on the performancee ratio of the schedule relative to optimal.

Define $B_i = \Sigma_{j=1}^{i} b_j$. Thus $B_i$ is the *total processing power of the fastest i processors*. $B_m$ is the total processing power of all the processors of the machine, and $B_1 = b_1$.

For a given task system, let $\mu$ denote the sum of the time requirements of the tasks in the system (by abuse of notation) and let $h$ denote the (original) height of the system.

**Lemma 3.1.** Let $(\mathcal{T}, <, \mu)$ be a task system executed on processors of different speeds as above. Let $w_{\text{opt}}$ be the finishing time of an optimal schedule. Then $w_{\text{opt}} \geq \max(\mu / B_m, h / b_1)$.

**Proof.** The most that any schedule can process in unit time is $B_m$ time units of the time requirement of the system. It is thus immediate that $w_{\text{opt}} \geq \mu / B_m$.

If one fixes attention on one chain of length $h$, it is evident that this chain requires at least time $h/b_1$ to be processed, even if the fastest processor is always used on the chain. It follows that $w_{\text{opt}} \geq h/b_1$. □

**Lemma 3.2.** Let $(\mathcal{T},<,\mu)$ as in Lemma 3.1. Let $w_i$ be the finishing time of a list schedule on the $i$ fastest processors. Then $w_i \leq (\mu/B_i)+(h/b_i)$.

**Proof.** To analyze the effectiveness of any list schedule on the fastest $i$ processors, it is convenient to break up the schedule of interest into height reducing intervals and constant height intervals. The sum of the total lengths of these intervals equals $w_i$.

Consider any constant height interval. Throughout the interval all of the fastest $i$ processors are in use. The reason is as follows. Assume that this assertion is false, and time $t$ is a time within a constant height interval when fewer than $i$ processors are in use. Consider the set of tasks that are at maximum height at time $t$. Each of these tasks is executable (i.e. has no unfinished predecessors). Since not all $i$ processors are in use, and the schedule is a list schedule on the fastest $i$ processors, it must be that all of these maximum height tasks are being executed. But then, it follows that the height of the task system in this interval is being reduced, contradicting the fact that this is a constant height interval.

By the above remarks, it follows that during each constant height interval, the processors of the machine are processing at least $B_i$ units of the time requirement of the task system per unit time. Thus the total time spent on constant height intervals is at most $\mu/B_i$.

Next, examine the height reducing intervals. At each point in time, some of the (at most $i$) tasks being executed are at the maximum height (of all remaining tasks). Whichever are at the maximum height are being processed at the rate of at least $b_i$. Since the total height may be reduced by at most $h$ throughout the schedule, it follows the the total amount of time spent on height reducing intervals is at most $h/b_i$. Together with the above bound on the amount of time in a constant height interval, one may conclude that $w_i \leq (\mu/B_i)+(h/b_i)$.

Actually it is easily shown that $w_i \leq ((\mu-h)/B_i)+(h/b_i)$, but this does not substantially improve the performance bound. This improvement follows from the fact that at least $h$ units of the time requirement are executed during height reducing intervals leaving only $\mu-h$ for constant height intervals. It will be important to remember this improved bound for some numerical results in Section 3.2. □

It follows from Lemmas 3.1 and 3.2 that:

$$(1) \qquad \frac{w_i}{w_{\text{opt}}} \leq \frac{(\mu/B_i)+(h/b_i)}{\max(\mu/B_m, h/b_1)}$$

Equation (1) presents us with an opportunity to formally state the schedule that will be used. Given a task system $(\mathcal{T},<,\mu)$, determine the total time requirement of all tasks ($\mu$), and the height of the system ($h$). Compute the right hand side of equation (1) for each value of $i=1,\ldots,m$. The value of $i$ that minimizes the expression is the number of processors that will be used. Devise any list schedule on the fastest $i$ processors.

In Section 3.2 it will be shown that the performance ratio of the above schedule (relative to optimal) is at most $\sqrt{m} + O(m^{1/4})$. Before proceeding to the proof of that fact, a modification in the scheduling algorithm will be suggested. The strategy as stated involves doing a separate calculation for each task system in order to determine how many processors to use. In fact, to get the $\sqrt{m} + O(m^{1/4})$ behavior, it is possible to use the same number of processors independent of the task system (based only on the $b_j$'s). Note that equation (1) also implies:

$$(2) \qquad \frac{w_i}{w_{opt}} \leq \frac{(\mu/B_i)}{(\mu/B_m)} + \frac{(h/b_i)}{(h/b_1)} = \frac{B_m}{B_i} + \frac{b_1}{b_i}$$

If the strategy is to use a list schedule on the fastest $i$ processors where $i$ minimizes this last performance bound, then (as shown in Section 3.2), any resulting schedule is never worse than $\sqrt{m} + O(m^{1/4})$ times worse than optimal irrespective of the value of the $b_j$'s.

Note that if the bound used on $w_i$ is $w_i \leq ((\mu-h)/B_i)+(h/b_i)$ then one may obtain a bound on the algorithm of:

$$(3) \qquad \frac{w_i}{w_{opt}} \leq \frac{B_m}{B_i} + \frac{b_1}{b_i} - \frac{b_1}{B_i}$$

Thus, an alternative scheduling algorithm chooses $i$ on the basis of equation (3). This improved algorithm does not have better asymptotic behavior, but does provide a better algorithm for small values of $m$. This will be discussed further when numerical results are discussed.

## 3.2 Calculation of speed-independent bound

This section analyzes the bound of Section 3.1 in three different ways. The first way provides an indication of which processors to use. Specifically, if the processors used are those that are within a factor of $\sqrt{m}$ of the fastest processor, then it is not too difficult to see that any list schedule is at most $1+2\sqrt{m}$ times worse than optimal. The second approach proves that one may always choose $i$ such that the bound is $\sqrt{m} + O(m^{1/4})$. This complicated proof does not give any intuitive idea how to choose $i$ in general, although it is certainly quite easy to calculate $(B_m/B_i)+(b_1/b_i)$ for each value of $i$ and then to minimize. The third method of analysis is a calculation of the actual numerical bounds for very small values of $m$. These bounds are better than $1+2\sqrt{m}$ and are more exact than a bound with an $O(m^{1/4})$ term.

**Theorem 1.** Consider a set of $m$ processors of different speeds. Then some value of $i$ $(1 \leq i \leq m)$ has the property that for any task system (with optimal finishing time $w_{\text{opt}}$), and any list schedule on the fastest $i$ processors for that task system (with finishing time $w_i$) $w_i/w_{\text{opt}} \leq 1+2\sqrt{m}$.

**Proof.** Recall that $w_j/w_{\text{opt}} \leq (B_m/B_j)+(b_1/b_j)$. Choose $i$ such that $\sqrt{m}\,b_i \geq b_1$ and $\sqrt{m}\,b_{i+1} < b_1$. Certainly some $b_j$ satisfies $\sqrt{m}\,b_j \geq b_1$ (since $b_1$ satisfies it) and if no $b_j$ satisfies $\sqrt{m}\,b_{j+1} < b_1$, then choose $i=m$. Now from equation (2) and the choice of $i$:

$$(4)\qquad \frac{w_i}{w_{opt}} \le 1 + \frac{b_{i+1}+\ldots+b_m}{B_i} + \frac{\sqrt{m}\,b_i}{b_i}$$

This follows from breaking up $B_m$ into $B_i+b_{i+1}+\ldots+b_m$ and using the upper bound on $b_1$.

Now clearly $B_i \ge b_1$. Also using the lower bound on $b_{i+1}$ (which is also a lower bound on $b_j$ for any $j>i$), (4) may be modified to obtain:

$$(5)\qquad \frac{w_i}{w_{opt}} \le 1 + \frac{(m-i)(b_1/\sqrt{m})}{b_1} + \sqrt{m}$$

Since $(m-i) \le m$ one may conclude that $w/w_{opt} \le 1+2\sqrt{m}.$ □

Note that as claimed above, the bound is never worse than the bound of $1+(b_1/b_m)$ even if that bound is small. The reason is that one may always choose $i=m$ if that gives the smallest value of the expression $(B_m/B_i)+(b_1/b_i)$.

The next result to be presented is a more complicated proof which improves the bound of Theorem 1.

**Theorem 2.** Consider a set of $m$ processors of different speeds. Then some value of $i$ $(1 \le i \le m)$ has the property that for any task system (with optimal finishing time $w_{opt}$), and any list schedule on the fastest $i$ processors for that task system (with finishing time $w_i$) $w_i/w_{opt} \le \sqrt{m} + O(m^{1/4})$.

**Proof.** Let $f(m)$ be the supremum of $\min_i ((b_1/b_i)+(B_m/B_i))$ where the sup is taken over all sets of speeds $b_1 \geq b_2 \geq \ldots \geq b_m > 0$. It will be shown that $f(m)$ is actually achieved by a particular set of speeds $b_1,\ldots,b_m$. Also, these speeds have the property that for every $i$ $f(m)=(b_1/b_i)+(B_m/B_i)$. Using this fact one may conclude that $f(m) \leq \sqrt{m} + O(m^{1/4})$.

Define $B \subset \mathbb{R}^m$ by $B=\{(b_1,\ldots,b_m) \in \mathbb{R}^m : b_1 \geq b_2 \geq \ldots \geq b_m \geq 0 \text{ and } B_m=1\}$. Note that $B$ consists of every legal set of processor speeds (normalized to sum to 1), and some illegal sets (e.g. $b_m=0$). For $b \in B$, with $b=(b_1,\ldots,b_m)$, define $E_i(b)=(b_1/b_i)+(1/B_i)$ and $g(b)=\min_i E_i(b)$. (Note that $g(b) \neq \infty$ since $E_1(b)=1+(1/b_1)<\infty$.) If $b_m \neq 0$, then $g(b)$ is a bound on the heuristic with processor speeds $b_1,\ldots,b_m$. Since $B$ is compact and $g$ is continuous, $g$ attains a maximum at some particular point $b^*=(b_1^*,\ldots,b_m^*) \in B$. Note that $g(b^*) \geq f(m)$.

It must be that $g(b^*)=E_m(b^*)$. Otherwise, one could define $b'$ by $b_i'=b_i^*/(1+\epsilon)$ and $b_m'=(b_m^*+\epsilon)/(1+\epsilon)$. Then $E_i(b')>E_i(b^*)$ for $i<m$. If $g(b^*) \neq E_m(b^*)$, then for sufficiently small $\epsilon$, $g(b^*)<E_m(b')$ (contradicting the fact that $b^*$ maximizes $g$). (If $b_m^*=b_{m-1}^*=\ldots=b_{i+1}^*<b_i^*$, then one may similarly define $b_j'=(b_j^*+(\epsilon/(m-i)))/(1+\epsilon)$ for $j=i+1,\ldots,m$ in order to preserve the decreasing nature of the vector $b'$. Essentially the same proof follows.)

To prove that for every $i$ $E_i(b^*)$ is the same, it suffices to consider the case that $g(b^*)=E_m(b^*)=\ldots=E_{k+1}(b^*)<E_k(b^*)$. In that case define $b_k'=b_k^*+\epsilon$ and $b_{k+1}'=b_{k+1}^*-\epsilon$. It is easy to verify that both $E_{k+1}(b')$ and $E_k(b')$ exceed $g(b^*)$ for small $\epsilon$. Note that $b_{k+1}'>b_{k+2}'$ for small $\epsilon$ since $b_{k+1}^*>b_{k+2}^*$. However, $b_k^*=b_{k-1}^*$ is possible. In that case, the $\epsilon$ is not added to $b_k^*$, but rather a total of $\epsilon$ is added to the processors whose speeds equal $b_k^*$. Continuing in this manner, we may define $b''$, $b'''$, ... so that each successive vector has the various $E_i$ values agreeing with $g$ at one less value of $i$. Finally, one gets $E_m(b)>g(b^*)$ for some vector $b$ with each $E_i(b)$ at least as large as $g(b^*)$, and then the previous construction again provides a contradiction.

It follows from the fact that $E_i(b^*)=g(b^*)$ for every $i$ that $b_m^*\neq 0$ and thus $g(b^*)=f(m)$.

The bound of $\sqrt{m}+O(m^{1/4})$ will be proved by using the fact that $f=f(m)=E_i(b^*)$ for each value of $i$. Using $f=E_i(b^*)$ and $f=E_1(b^*)$ one may conclude that $b_i^*=B_i^*/(f-1)(fB_i^*-1)$ since $b_i^*=b_1^*B_i^*/(fB_i^*-1)$ and $b_1^*=1/(f-1)$. This in turn proves that $b_i^*=(1/f(f-1))(1+(1/(fB_i^*-1)))$. From this it follows that:

(6) $b_{i+1}^*+\ldots+b_m^*=(1-B_i^*)\leq(1+\epsilon)(m-i)/f(f-1)$ where $\epsilon=(1/(fB_i^*-1))$.

In order to use the above equation to get a bound on $f$, a few other facts are needed. Note that $f>\sqrt{m}$. This can be shown by considering the vector $b'$ which is a normalized version of the vector $(\sqrt{m-1},1,1,\ldots,1)$ since $g(b')>\sqrt{m}$. Note also that $b_1^*\leq\sqrt{1/m}$. This follows from the fact that $E_1(b^*)=E_m(b^*)$ which implies that $b_1^{*2}=b_m^*$. Since $mb_m^*\leq 1$ the upper bound on $b_1^*$ follows. Note that each processor must therefore have a "relatively" small speed and in particular, the successive sums $B_1^*,B_2^*,\ldots,B_m^*$ are spaced apart by a distance of at most $\sqrt{1/m}$. Thus, if one wants to find an $i$ such that $B_i^*=rm^{-1/4}$ for some $r$ between $\sqrt{2}$ and $1+\sqrt{2}$, this can always be done.

Let $B_i^*=rm^{-1/4}$. Then $1+B_i^*\geq 1+\epsilon$ using the expression for $\epsilon$ in equation (6). This follows from the fact that $(B_i^*/\epsilon)=(fB_i^{*2}-B_i^*)\geq(\sqrt{m}\,r^2m^{-1/2}-B_i^*)=(r^2-B_i^*)\geq(2-1)=1$ and thus $B_i^*\geq\epsilon$. Thus $f(f-1)\leq(m-i)(1+B_i^*)/(1-B_i^*)\leq(m-i)(1+rm^{-1/4})(1+2rm^{-1/4})$ for sufficiently large values of $m$. But then (by further increasing the right hand side) $(f-1)^2 \leq m+4rm^{3/4}+4r^2\sqrt{m}$ which yields $f \leq \sqrt{m}+2rm^{1/4}+1$ for sufficiently large values of $m$. □

While Theorem 2 provides a better asymptotic estimate of the performance of the algorithm than Theorem 1, it does not give a better bound for practical situations. In principle the $O(m^{1/4})$ term may be the dominating factor for the small values of $m$ that typically arise in practice. For that reason, it is important to try to get a more meaningful bound for small values of $m$. A third way of evaluating the heuristic is thus presented, which gives numerical bounds on the algorithm for small values of $m$. This also will give an intuitive idea as to the growth rate of $f(m)$.

Recall that the heuristic takes its worse value at the vector $b^*$ with the property that $E_i(b^*)$ is the same for every $i$. Using $E_1(b^*)=E_m(b^*)$ gives an expression for $b_m^*$ in terms of $b_1^*$. Similarly, using $E_i(b^*)=E_1(b^*)$ gives an expression for $b_i^*$ in terms of $b_1^*, b_m^*, \ldots, b_{i+1}^*$. Inductively, this gives an expression for $b_i^*$ in terms of $b_1^*$. The expression is:

$$(7) \qquad b_i^* = \frac{b_1^{*2}(1-(b_{i+1}^*+\ldots+b_m^*))}{((1-(b_{i+1}^*+\ldots+b_m^*))(b_1^*-1)-b_1^*)}.$$

This is obtained using $B_i^*=1-(b_{i+1}^*+\ldots+b_m^*)$.

Using the expression for $b_i^*$ in terms of $b_1^*$ and using the fact that $b_1^*+\ldots+b_m^*=1$, one obtains an equation for $b_1^*$. Solving this equation and computing $1+(1/b_1^*)$ gives a bound on the algorithm. This calculation was done on the MACSYMA system which generated the expressions to solve and also solved them. The indication of this small sample of data is that $\sqrt{m} + O(\log m)$ might in fact be an accurate bound. The value $f(m)$ for the range of values considered seems to be bounded by $\sqrt{m} + .21(\log_2 m) + 1$. In fact, not only is $f(m)$ bounded by this expression (in the range we considered), but it seems to grow slower. The results are given in Table 1, together with other key quantities for the sake of comparison.

| $m$ | $f(m)$ | $\sqrt{m}$ | $\sqrt{m} + m^{1/4}$ | $1+2\sqrt{m}$ | $(\sqrt{m} + .21\log_2 m + 1)$ |
|---|---|---|---|---|---|
| 2 | 2.62 | 1.41 | 2.60 | 3.82 | 2.62 |
| 3 | 3.06 | 1.73 | 3.05 | 4.46 | 3.06 |
| 4 | 3.41 | 2 | 3.41 | 5 | 3.42 |
| 5 | 3.71 | 2.24 | 3.74 | 5.48 | 3.73 |
| 6 | 3.98 | 2.45 | 4.02 | 5.90 | 3.99 |
| 7 | 4.22 | 2.65 | 4.28 | 6.30 | 4.24 |
| 8 | 4.44 | 2.83 | 4.51 | 6.66 | 4.46 |
| 9 | 4.64 | 3 | 4.73 | 7 | 4.67 |
| 10 | 4.83 | 3.16 | 4.94 | 7.32 | 4.86 |
| 50 | 9.14 | 7.07 | 9.73 | 15.14 | 9.26 |
| 100 | 12.24 | 10 | 13.16 | 21 | 12.40 |
| 500 | 24.98 | 22.36 | 27.09 | 45.72 | 25.24 |
| 1000 | 34.41 | 31.62 | 37.25 | 64.25 | 34.71 |
| 5000 | 73.88 | 70.71 | 79.12 | 142.42 | 74.29 |
| 10000 | 103.33 | 100 | 110 | 201 | 103.79 |

Table 1.

Note that $f(m)$ does seem to be growing faster than $\sqrt{m} + O(1)$ although this can not be proven by such numerical studies.

The above results were obtained using the bound of equation (2). An important purpose of these results is to show how $f(m)$ behaves. An additional reason for this calculational exercise, though, is to get as good a bound as possible on the heuristic for small values of $m$. For the purpose of getting a tight bound on the algorithm for small values of $m$, a better bound is obtained if the algorithm uses the slightly more complicated bound given by equation (3). In our analytic studies the $b_1/B_i$ term was ignored since it does not improve the asymptotic results (in particular it is always less than 1). Nevertheless, for small values of $m$ it is a significant portion of the bound. The next table gives a bound on the algorithm in terms of this better bound. (Incidentally, the same technique was used for generating Table 2 as was used for generating Table 1. To use this technique, it must first be shown that a bound on the algorithm is obtained by analyzing the vector $b^* \epsilon B$ which has the property that $E_i'(b^*)$ is the same for each value of $i$. This is a simple exercise if one just copies the technique used for the simpler bound in the proof of Theorem 2.)

Notice that using this better bound gives a result which is about .7 or .8 better than the other bound for small values of $m$ - a substantial saving for small $m$. For large values of $m$ the improvement is slightly smaller, and less significant due to the large value of the bound.

| $m$ | bound on the algorithm |
|---|---|
| 2 | 1.76 |
| 3 | 2.25 |
| 4 | 2.65 |
| 5 | 2.97 |
| 6 | 3.25 |
| 7 | 3.50 |
| 8 | 3.73 |
| 9 | 3.94 |
| 10 | 4.14 |

Table 2

Intuitively it seems quite wasteful *never* to use processors - no matter how slow they may be. It is an open question to determine how to use the slow processors in order to provide a quantitatively better performance ratio. There are certain simple safe techniques that one may use which do not harm the performance ratio. For example, one may first determine a list schedule on the fastest $i$ processors. Then, if a slow processor is free at a particular time, and an executable task is not being executed, and furthermore the finishing time of the task will be later (with the list schedule) than the time that our slow processor could finish it, then it is safe to assign the task to the slow processor (and possibly to make other improvements based on the earlier finishing time of this task).

The fact that this procedure is not harmful may be easily seen. Since the finishing time of the chosen task is earlier in the new schedule than in the original schedule, no task is finished later than in the original schedule.

While it is easy to determine such safe uses for the slow processors, we have been unable to determine any proofs that guarantee faster behavior.

It is significant to note that the techniques used here are applicable to the preemptive case studied elsewhere. A result of the methods of this section are that the class of schedules studied in [3] (the maximal usage schedules) are never worse than $\sqrt{m} + O(m^{1/4})$ times worse than optimal. This result is not as good as the one in [3], but it demonstrates the same asymptotic behavior for very little extra work. This connection will be elaborated and exploited in Section 6 where the preemptive scheduling of typed task systems is discussed.

## 3.3. Achievability of the performance bound

In this section it is demonstrated that the results of Section 3.2 are asymptotically correct. This is shown by demonstrating that for a certain set of processor speeds and a specific task system, the performance ratio of a list schedule on the fastest $i$ processors (for *any* $i=1,\ldots,m$) may be as large as $\sqrt{m-1}$. The fact that this example shows that any choice of $i$ has the potential of being $\sqrt{m-1}$ times worse than optimal is significant. It tells us that no sophisticated way of choosing $i$ provides better than $\sqrt{m}$ behavior if once $i$ is chosen a list schedule is the only added feature of the heuristic.

Consider the situation where $b_1=\sqrt{m-1}$ and $b_i=1$ for $i>1$. Consider the task system of $2n$ tasks as diagrammed in Figure 1. A node represents a task and an arrow represents a precedence dependence. The time requirement of each of the $n$ tasks in the long chain is $\sqrt{m-1}$. The time requirement of the other $n$ tasks is $m-1$. An asymptotically optimal schedule

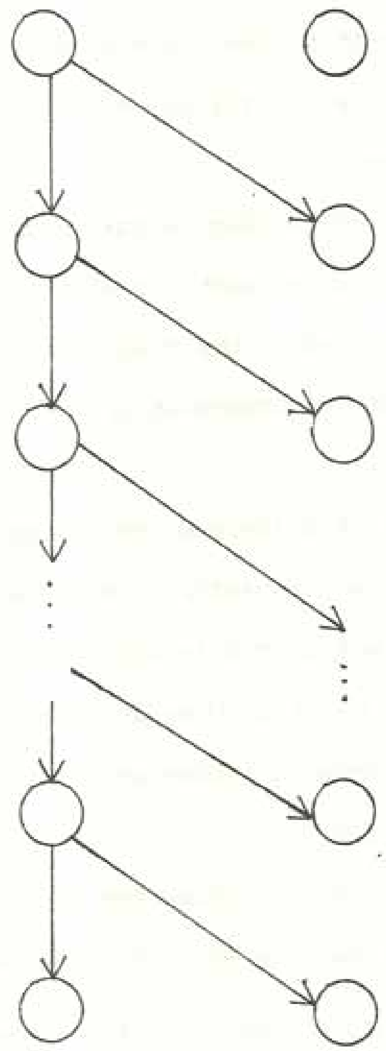


Figure 1

proceeds as follows. $P_1$ executes every task in the long chain. Each task in the long chain requires unit time on $P_1$. Meanwhile, $P_2,\ldots,P_m$ execute the tasks that are not in the long chain. Each of these processors requires time $m-1$ for one of the tasks. If $n=m-1$ then the long chain requires time $m-1$, but $P_m$ will not finish its task until $2m-2$ units of time have passed since its task is not executable until $m-1$ units of time elapse. For any value of $n$ the finishing time is similarly bounded by $n+m-1$.

To discuss the fact that this task system may be executed inefficiently, no matter how many processors are used, consider two situations. The first is the case that one attempts to schedule the system on the fastest processor. The second is the situation that the processing is done on $i$ processors for any $i>1$.

If only the first processor is used, then there is not enough processing power to execute this task system efficiently. Specifically, the total amount of time requirement of this task system is $n(m-1+\sqrt{m-1})$. With only $\sqrt{m-1}$ processing power available, the finishing time must be at least $n(1+\sqrt{m-1})$. For large values of $n$, this provides a performance ratio of approximately $\sqrt{m-1}$ times worse than optimal.

Consider the scheduling of this system on $i$ processors for $i>1$. A "bad" list schedule first tries to use $P_1$ on the "non-long chain" tasks, and $P_2$ on the chain tasks. After time $\sqrt{m-1}$, $P_1$ finishes the first non-chain element, and $P_2$ finishes the first chain element. Repeating this strategy for each pair of tasks requires time $\sqrt{m-1}$ for each pair. Thus the total time for the bad schedule is about $n\sqrt{m-1}$ and the ratio between the finishing times of the "bad" schedule and the optimal schedule approaches $\sqrt{m-1}$ for large $n$. Note that no matter how many processors one attempts to use, a bad list schedule only allows two processors to be used. □

Recall that the upper bound on the algorithm was $\sqrt{m} + O(m^{1/4})$. It has not been shown that a set of processor speeds, $b^*$, exist that have $g(b^*)=\sqrt{m} + O(m^{1/4})$, since that is only an upper bound. However, whatever $f(m)=\max_b (g(b))$ is, a bound on the heuristic of Section 3.1 may be obtained in terms of $f(m)$. Consider a set of $m$ unordered tasks, the $i^{\text{th}}$ having time requirement $b_i^*$. The optimal schedule requires unit time. Using only the fastest processor (a valid choice with the algorithm as presented) requires time $1/b_1^*$. But $f(m)-1=1/b_1^*$. Thus $f(m)$ is almost achievable (whatever $f(m)$ is). This means that an exact bound on the algorithm may be obtained by solving the mathematical problem of determining $f(m)$, without any need to look at more task systems.

Consider the related heuristic of trying to minimize $E_i'(b^*)=(1/B_i^*)+(b_1^*/b_i^*)-(b_1^*/B_i^*)$. In that case $E_1'(b^*)=1/b_1^*$. Recall that at the vector $b^*$ that maximizes $\min_i E_i'(b^*)$, $E_i'(b)$ is the same for every $i$, and using only the fastest processor is a valid choice. In that case, $1/b_1^*$ for this vector $b^*$ is an exact upper and lower bound.

## 4. Typed Task Systems

This section contains additional definitions needed to discuss the case that processors and tasks are of designated "types", and a task of a certain type may be only processed by a processor of the same type.

A *k type task system* $(\mathcal{T},<,\mu,\nu)$ is a task system $(\mathcal{T},<,\mu)$ together with a *type function* $\nu:\mathcal{T}\to\{1,\ldots,k\}$. Intuitively, if $\nu(T)=i$ then $T$ must be executed a processor of type $i$.

The processors of a typed task system are the same as in Section 2, except that each processor has a designated type. Processor $P_{ij}$ is the $j^{th}$ fastest processor of type $i$ (with rate $b_{ij}$). Similarly, a schedule is defined in the same manner except that in addition if $S(T)=(t,P)$, then $P$ must be of type $\nu(T)$.

The definitions of *being executed at time t*, *finishing time* of a schedule, *optimal schedule*, and *performance ratio* generalize in a straightforward manner and are omitted.

A *list schedule* differs in that there is a different list devised for each type, and the processor of a given type chooses tasks based on the list of its type.

In this situation $m_i$ denotes the number of processors of type $i$, and $\mu_i$, the total number of steps required for type $i$ tasks.

The relevant bound for list schedules is approximately $k+\max_i (b_{i1}/b_{im_i})$[4].

## 5. List schedules on the fastest $i_j$ processors of type j (j=1,...,k)

The approach used in this section will closely parallel the approach of Section 3. The one major difference here is that in order to obtain the proper bound, the concept of height of a task must be modified to take into account both the type of the task and the speeds of the processors of that type. Section 5.2 derives a speed independent bound on the heuristic, but the situation is complicated by the fact that there are more parameters to be considered. Section 5.3 again shows that the bounds are fairly close to being achievable although the bound is not as tight as that of Section 3.

### 5.1 Performance bound

The schedules considered will operate as follows. A priority list of tasks will be prepared for each type. Each type will have certain of its fastest processors designated as the "ones to be used", and the others will not be used at all. This section obtains a bound on these schedules in terms of the $b_{ij}$'s.

Define $B_{ji}=\Sigma_{n=1}^{i}b_{jn}$. $B_{ji}$ is the total processing power of the fastest $i$ processors of type $j$.

It remains to discuss the notion of height for these typed task systems. It turns out that it is convenient to have a different definition of height depending on which processors are being used. The following is the definition of height if the fastest $i_j$ processors of type $j$ are used ($j=1,\dots,k$). (While reading the definition consider the following motivation. We would like to be able to say that the total amount of time spent on height reducing intervals is at most the height of the graph. Thus it is convenient to have the height reduced at least one unit of height per unit time during height reducing intervals.)

The *height length* of a task $T$ (of type $j$) is given by $\mu(T)/b_{ji_j}$. (Thus if $P_{ji_j}$ (the slowest processor of type $j$ that will be used) processes $T$, it executes one unit of the height length of $T$ per unit time.) The length of a chain $C$, the height of a task $T$, and the height of $(\mathcal{T},<,\mu,\nu)$ are defined as in Section 3, except that summations are taken of height lengths instead of time requirements of tasks.

The rest of this discussion assumes a fixed task system $(\mathcal{T},<,\mu,\nu)$ executed on a fixed set of processors $\mathcal{P}$. Also, the discussion fixes which processors are to be used, and thus fixes a notion of the height of a task or of $(\mathcal{T},<,\mu,\nu)$. As before $h$ will denote the height of the system. If $h$ is the height than there is some chain of tasks whose height equals $h$. Let $c_i$

denote the sum of the time requirements of all type $i$ tasks along this chain. Then $h=(c_1/b_{1i_1})+\ldots+(c_k/b_{ki_k})$.

**Lemma 5.1.** Let $(\mathcal{T},<,\mu,\nu)$ as above. Let $w_{\text{opt}}$ be the finishing time of an optimal schedule. Then
$w_{\text{opt}}\geq\max((\mu_1/B_{1m_1}),(\mu_2/B_{2m_2}),\ldots,(\mu_k/B_{km_k}),((c_1/b_{11})+\ldots+(c_k/b_{k1})))$.

**Proof.** The first $k$ bounds follow from the fact that at most $B_{jm_j}$ units of the time requirement of type $j$ tasks can be executed in unit time. To get the last bound, consider a chain of height $h$ as above. Then all the type $j$ tasks in the chain require a total of at least $c_j/b_{j1}$ units of time to be processed, and all tasks must be processed separately. The bound follows immediately. □

**Lemma 5.2.** Let $(\mathcal{T},<,\mu,\nu)$ as above. Let $w(\{i_j\})$ be the finishing time of a list schedule on the fastest $i_j$ processors of type $j$ $(j=1,\ldots,k)$. Then $w(\{i_j\})\leq(\mu_1/B_{1i_1})+\ldots+(\mu_k/B_{ki_k})+((c_1/b_{1i_1})+\ldots+(c_k/b_{ki_k}))$.

**Proof.** As in Section 3 consider height reducing intervals and constant height intervals. Any constant height interval must have all of the fastest $i_j$ processors of type $j$ in use for some $j$. The reason is that otherwise the unexecuted task with the greatest height is being executed reducing the height of the task system. The total time spent in a constant height interval when all $i_j$ processors of type $j$ are in use is $\mu_j/B_{ji_j}$. Thus the total time spent on constant height intervals may be at most $(\mu_1/B_{1i_1})+\ldots+(\mu_k/B_{ki_k})$.

Consider height reducing intervals. The greatest height task being executed is being executed at the rate of (at least) 1 unit of height per unit time. Thus the time spent on height reducing levels can be at most $h=(c_1/b_{1i_1})+\ldots+(c_k/b_{ki_k})$. □

Lemmas 5.1 and 5.2 imply:

$$(8)\qquad \frac{w(\{i_j\})}{w_{opt}} \le \frac{(\mu_1/B_{1i_1})+\ldots+(\mu_k/B_{ki_k})+(c_1/b_{1i_1})+\ldots+(c_k/b_{ki_k})}{\max((\mu_1/B_{1m_1}),\ldots,(\mu_k/B_{km_k}),((c_1/b_{11})+\ldots+(c_k/b_{k1})))}$$

One way to choose the set $\{i_j\}$ is to compute the right hand side of equation (8) for each possible choice of processor speeds. Again, since this depends on the task system, this can be quite tedious. Note that this expression depends on which notion of height is used, since the $c_i$'s refer to a maximal chain, but maximal chains may be different with the different definitions of height. It is thus even more desirable in this case to obtain a task system independent choice of which processors to use. Using techniques similar to those in Section 3 equation (9) follows from equation (8).

$$(9)\qquad \frac{w(\{i_j\})}{w_{opt}} \le \frac{B_{1m_1}}{B_{1i_1}} +\ldots+ \frac{B_{km_k}}{B_{ki_k}} + \frac{(c_1/b_{1i_1})+\ldots+(c_k/b_{ki_k})}{(c_1/b_{11})+\ldots+(c_k/b_{k1})}$$

Let $q=\max((b_{11}/b_{1i_1}),(b_{21}/b_{2i_2}),\ldots,(b_{k1}/b_{ki_k}))$. The value $q$ is the analog of the $b_1/b_i$ term in the ordinary task system case. Note that $q\ge(b_{j1}/b_{ji_j})$ for $j=1,\ldots,k$. Thus $(c_j/b_{ji_j})\le(q(c_j)/b_{j1})$. Substituting the right hand side of this inequality for $c_j/b_{ji_j}$ in the last of the $k$+1 summands gives a bound on the last term of $q$. Thus a task system independent way of choosing which processors to use would be to minimize:

$$(10) \qquad \frac{B_{1m_1}}{B_{1i_1}} + \ldots + \frac{B_{km_k}}{B_{ki_k}} + \max(b_{11}/b_{1i_1},\ldots,b_{k1}/b_{ki_k})$$

The heuristic is then to compute equation (10) for each possible set of processor speeds to determine a set of indices $i_j$ (still a somewhat lengthy procedure, but something that needs be done only once), and then use only the fastest $i_j$ of type $j$. The results of Section 5.2 will indicate that such a choice guarantees performance that is no worse than $k+2\sqrt{\max_j (km_j)}$ times worse than optimal. In fact, the proof technique is such that it suggests one simple way of choosing the $i_j$ so that the bound is reached, and as such even one calculation of equation (7) for each possible assignment to $i_j$ is unnecessary.

## 5.2 Speed independent bound

**Theorem 3.** Consider a set of $m$ processors of different speeds with $m_j$ of type $j$ ($j$=1,...,$k$). Then some set of indices $\{i_j$:$1\leq j\leq m\}$ have the property that for any task system $(\mathcal{T},<,\mu,\nu)$ (with optimal finishing time $w_{\text{opt}}$), and any list schedule on the fastest $i_j$ processors of type $j$ for that task system (with finishing time $w(\{i_j\})$), that $w(\{i_j\})/w_{\text{opt}}\leq k+2\sqrt{\max_j (km_j)}$.

**Proof.** Let $r=\sqrt{\max_j (km_j)}$. Choose $i_j$ such that $rb_{ji_j}\geq b_{j1}$ and $rb_{ji_{j+1}}<b_{j1}$. (Let $i_j$=$m_j$ if the second inequality fails for each value of

$b_{ji}$.) Then $(B_{jm_j}/B_{ji_j})=1+((b_{ji_{j+1}}+...+b_{jm_j})/B_{ji_j})$. Each of the $m_j$-$i_j$ terms in the numerator of the fraction is at most $b_{j1}/r$. The denominator is at least $b_{j1}$. Thus $B_{jm_j}/B_{ji_j}$ is at most $1+(m_j/r)$. Now $r\geq\sqrt{km_j}$ for each $j$. Thus $B_{jm_j}/B_{ji_j}\leq 1+(\sqrt{m_j/k})$.

By the choice of $i_j$, $r\geq b_{j1}/b_{ji_j}$ for each value of $j$. Thus $r\geq\max_j (b_{j1}/b_{ji_j})$. Using this, the bound on $B_{jm_j}/B_{ji_j}$ and equation (10) provides:

$$(11)\qquad w(\{i_j\})/w_{\text{opt}} \leq\ k+\sqrt{1/k}\,(\sqrt{m_1}+...+\sqrt{m_k})+r.$$

Increasing $\sqrt{m_i}$ to $\sqrt{\max_j (m_j)}$ in the above expression yields

$$(12)\qquad w(\{i_j\})/w_{\text{opt}} \leq\ k+k\sqrt{\max_j (m_j/k)}+r.$$

From (12) it is immediate that $w(\{i_j\})/w_{\text{opt}} \leq k+2r$ for this choice of $i_j$ irrespective of the task systems used or list schedule used. □

The bound of Theorem 3 is the bound that one would use if each of the $m_j$ were almost equal. For different situations, though, it might be beneficial to use a different choice of which processors to use. A sample of this is contained in the following theorem.

**Theorem 4.** Consider a set of $m$ processors of different speeds with $m_j$ of type $j$ ($j$=1,...,$k$). Then some set of indices $\{i_j$:$1\leq j\leq m\}$ have the property that for any task system $(\mathcal{T},<,\mu,\nu)$ (with optimal finishing time $w_{\text{opt}}$), and any list schedule on the fastest $i_j$ processors of type $j$ for that task system (with finishing time $w(\{i_j\})$), that $w(\{i_j\})/w_{\text{opt}} \leq k+\sqrt{m_1} + \sqrt{m_2} + \ldots + \sqrt{m_k} + \max_j \sqrt{m_j}$.

**Proof.** Let $r_j=\sqrt{m_j}$. Choose $i_j$ such that $r_j b_{ji_j} \geq b_{j1}$ and $r_j b_{ji_j+1} < b_{j1}$. Then $(B_{jm_j}/B_{ji_j}) \leq 1+\sqrt{m_j}$. Also, $\max_j r_j = \max_j \sqrt{m_j}$ exceeds the $k+1^{\text{st}}$ summand in the bound of equation (10). The theorem follows immediately. □

The choice of which bound is better depends on the values of the $m_j$. If all are equal, the first bound is better by a factor of about $(1/2)\sqrt{k}$. If all the $m_j$ equal 1 except for one which has a large number of processors, then the latter bound beats the former by a factor of $\sqrt{k}$.

## 5.3 Achievability of the performance bound

Three achievability results will be presented. The first two discuss the situation where $k$ is quite small relative to the number of processors, and thus the first summand in the bounds of Theorems 3 and 4 may be ignored. One approach is to show that for a fixed value of $k$, and any values of $m_1,\ldots,m_k$ there are speeds for which the bounds of Section 5.2 are achievable within a constant factor. This is quite easy. Let $j$ be the type that has the most processors. Then the same construction used in Section 3.3, using only processors of type $j$ gives an immediate achievability result of $\sqrt{m_j}-1$. While this is a factor of about $2\sqrt{k}$ times worse than the bound of Theorem 3 and about $k+1$ times worse than the bound of Theorem 4, for a fixed value of $k$, it is only a constant factor worse than the bound. □

A different problem is to show that as $k$ is varied and as the values $m_1,\ldots,m_k$ are varied, there is a set of processors speeds, a task system and a list schedule such that the algorithm is as bad as the bound of Theorem 3. This is in fact not true in general. Neither Theorem 3 nor Theorem 4 is tight for all values of the $m_j$ since either may be as much as a factor of $O(\sqrt{k})$ times more than the true bound (as discussed above). Instead it will be shown that for any values of $m$ and $k$ where $m=\Sigma_{j=1}^{k} m_j$, there are values for $m_j$ and speeds for the processors such that the bound of Theorem 3 is achieved and values for $m_j$ and speeds for the processors such that the bound of Theorem 4 is achieved.

To achieve Theorem 4 is easy. Use $m_1=m-k+1$ and $m_i=1$ for $i>1$. Then the construction used in Section 3.3 provides an achievability result of a constant factor. □

To achieve Theorem 3, consider distributions of processors such that $m_j=m/k$ for each value of $j$ (if $m \neq 0 (\mathrm{mod}\ k)$ then some of the $m_j$'s are appropriately rounded off).

The precedence structure of the graph is the same as the graph of Figure 1 (see Figure 2). A node labelled with the integer $j$ indicates that the task represented by the node is of type $j$. In this case, there are $n$ blocks, each made up of $m-k$ pairs of tasks. The tasks of the first $m_1-1$ pairs are of type 1; the tasks of the next $m_2-1$ pairs are of type 2, etc. The time requirement of each task in the long chain is $\sqrt{m-k}$. The time requirement of the other $n(m-k)$ tasks is $m-k$. Type $j$ has one processor with rate $\sqrt{m-k}$ and $m_j-1$ with rate 1.

An asymptotically optimal schedule proceeds as follows. At each point in time, the fastest processor of some type is executing some task on the long chain. Thus the execution of each task on the long chain requires unit time. To finish all tasks on the long chain requires time $n(m-k)$. Meanwhile, the

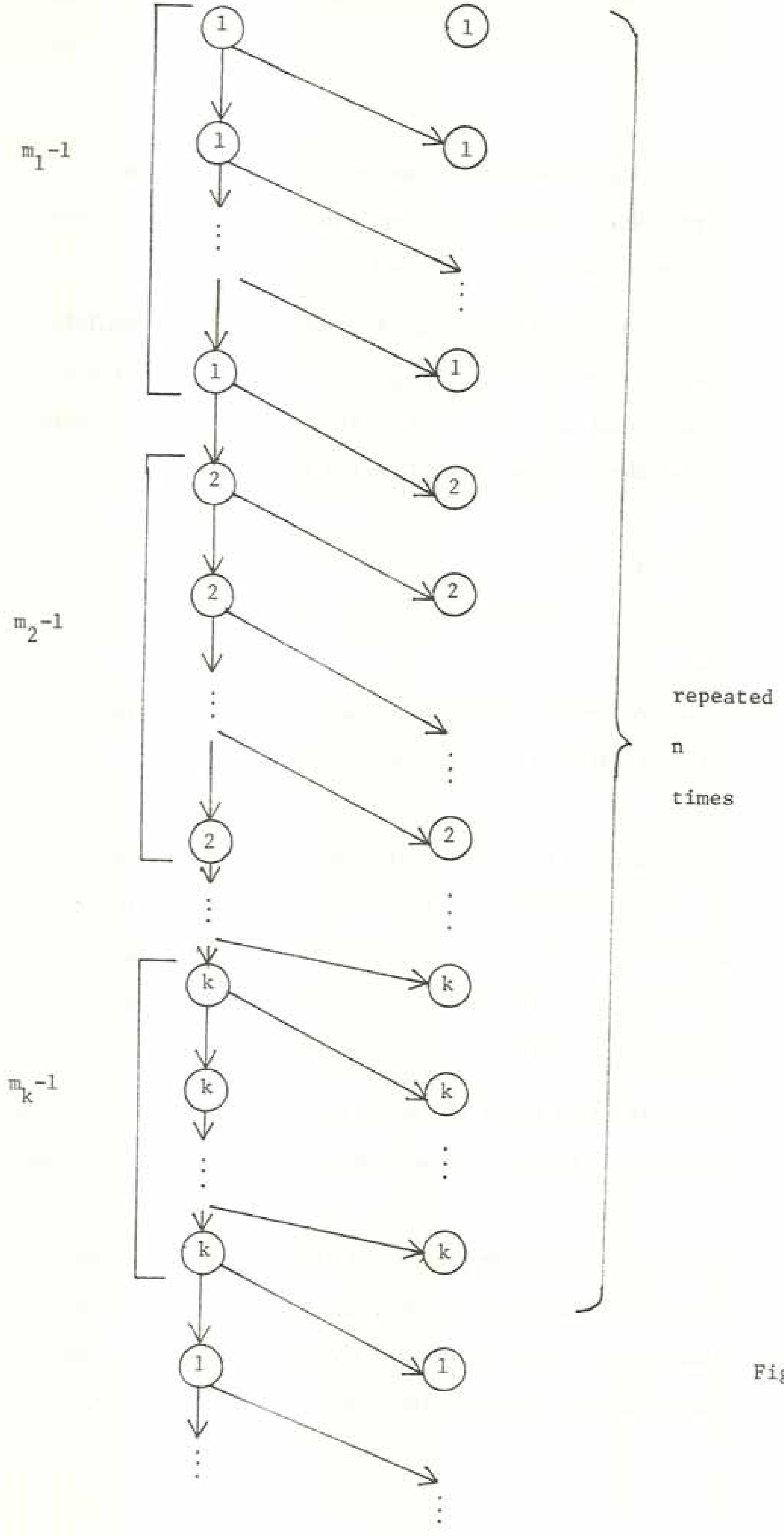


Figure 2

rest of the processors execute the tasks that are not on the long chain. Once a processor begins executing one of these tasks, it takes $m-k$ units of time until it is completed. At that time, the processor begins executing the task that is in the same position in the next block. Thus, after the chain is completed, at most an additional $m-k$ units of time are needed, for a finishing time of at most $(n+1)(m-k)$.

A bad list schedule on the fastest $i_j$ processors of type $j$ ($j=1,\ldots,k$) might proceed as follows. In fact, only the two fastest processors of each type would be used (unless $i_j=1$ for some $j$ in which case only the fastest processor of that type would be used). While executing type $j$ tasks, the schedule assigns the chain task to processor $P_{j2}$ and the non chain task to processor $P_{j1}$. It takes times $\sqrt{m-k}$ to finish both of them (simultaneously). If only one processor of type $j$ is to be used, then it first processes the non chain task, and then the chain task, again requiring at least $\sqrt{m-k}$ time units to finish each pair of tasks. Thus the total time required is at least $n(m-k)(\sqrt{m-k})$. This is $\sqrt{m-k}=\sqrt{k(m_j-1)}$ times worse than optimal. Since $m$ is substantially larger than $k$ Theorem 3 is essentially achieved to within a factor of 2. □

There is a large spectrum of results between the two extremes that have just been considered. It can be shown that Theorem 4 is achieved for a class of processor distributions that have $m_1=cm$ for some $c<1$. Similarly, Theorem 3 is achieved by a class of processor distributions that have $\max_j m_j=cm/k$ for a fixed $c>1$. These added constructions are trivial extensions of the above and are omitted.

The third type of result involves the situation that the $m_i$ are relatively small (compared to $k$). In this case we will show that $k$ times worse than optimal is achievable.

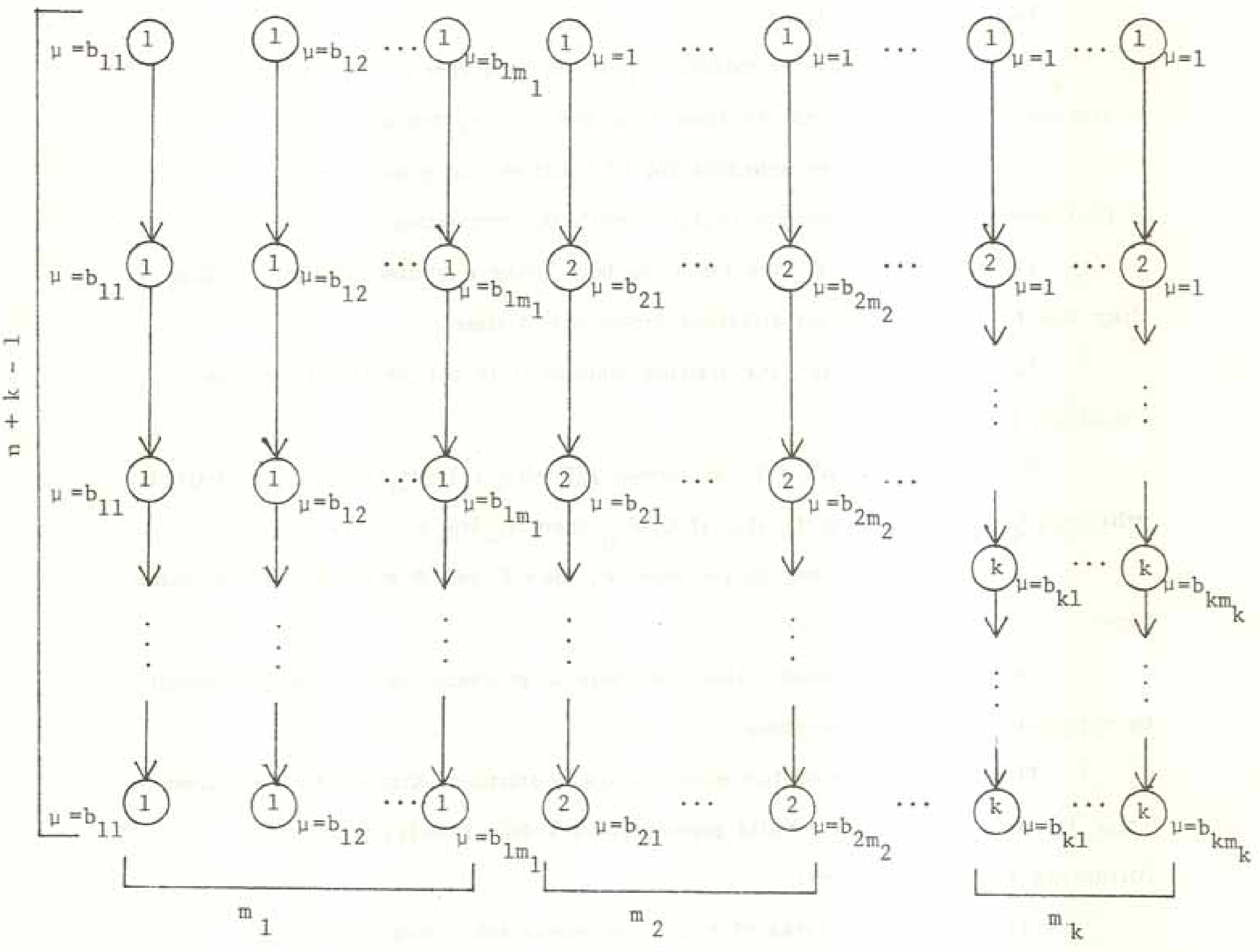
Consider Figure 3. There are $m_i$ columns that informally speaking "correspond to type $i$". Each of these $m_i$ columns contains a chain of $n+k-1$ tasks. The $j^{\text{th}}$ task in each of these columns has $\nu(T)=j$ (for $j \leq i-1$) and $\nu(T)=i$ (for $i \leq j$). The notation $\mu=i$ means that the time required by the indicated task is the value $i$ (note that the only values that appear are the integer 1 or the rate of a processor). An asymptotically optimal schedule first executes the first $k-1$ tasks of each column using an arbitrary schedule. Then only $n$ units of time are required. The $i^{\text{th}}$ remaining task in every column is executed $i$ units of time later.

A bad schedule first executes only those tasks in the first $m_1$ columns. The best that could be done in that case (assuming all processors of type 1 are in use) is that these columns will be finished in time $n$. In a similar manner, it takes a minimum of time $kn$ to finish the entire tasks system. Thus this schedule is at least $k$ times worse than optimal. Thus for the class of processors considered, the example illustrates achievability up to a constant factor. □

## 6. Preemptive scheduling of typed task systems

This section discusses preemptive scheduling of typed task systems. When one permits preemptive scheduling, one permits the temporary suspension of the execution of a task. When the task is continued, only the unexecuted portion needs to be finished, and there is no penalty for the temporary suspension. Formally:

A *preemptive schedule* for $(\mathcal{T},<,\mu,\nu)$ on a set of processors $\mathcal{P}$ is a total function $S$ that maps each task $T \in \mathcal{T}$ to a finite set of interval, processor



$\mu=b_{11}$
$\mu=b_{12}$
$\mu=b_{1m_1}$
$\mu=1$
$\mu=b_{21}$
$\mu=b_{2m_2}$
$\mu=b_{k1}$
$\mu=b_{km_k}$
$n+k-1$
$m_1$
$m_2$
$m_k$

Figure 3

pairs. If $S(T)=\{([i_1,j_1],Q_1),([i_2,j_2],Q_2),\dots,([i_n,j_n],Q_n)\}$ then

(1) $i_p,j_p \in \mathbb{R}$ for $p=1,\dots,n$.

(2) $i_p \le j_p$ for $p=1,\dots,n$ and $j_p \le i_{p+1}$ for $p=1,\dots,n-1$

(3) $Q_p \in \mathcal{P}$ for $p=1,\dots,n$.

For $i_p \le t < j_p$ $T$ is *being executed on processor* $Q_p$ *at time* $t$. The time $i_1$ is the *starting time* of $T$, and the time $j_n$ is the *finishing time* of $T$.

A *valid* preemptive schedule for $(\mathcal{T},<,\mu,\nu)$ *on a set of processors* $\mathcal{P}$ is a preemptive schedule for $(\mathcal{T},<,\mu,\nu)$ with the properties:

(1) For all $t \in \mathbb{N}$, if two tasks are both being executed at time $t$, then they are being executed on different processors at time $t$.

(2) Whenever $T<U$, the starting time of $U$ is not smaller than the finishing time of $T$.

(3) For $T \in \mathcal{T}$ (with $S(T)$ as above), $\mu(T)=((j_1-i_1)/r(Q_1))+\dots+((j_n-i_n)/r(Q_n))$ where $r(Q_i)$ is the rate of $Q_i$ (i.e. if $Q_i=P_{nj}$ then $r(Q_i)=b_{nj}$).

(4) If $T$ is executed on processor $P$, then $T$ and $P$ must be of the same type.

Condition three asserts that each task is processed exactly long enough to complete its time requirement.

The performance of the *maximal usage* heuristic is discussed. A *maximal usage preemptive schedule* is a valid preemptive schedule satisfying the following two requirements.

(1) Whenever $i_j$ tasks of type $j$ are executable, then $\min(m_j,i_j)$ tasks of type $j$ are being executed. (A task is executable if all its predecessors have been finished, but the task itself has not been finished.)

(2) Whenever $i_j$ processors of type $j$ are being used, the fastest $i_j$ processors of type $j$ are in use.

It is easy to see how to transform any schedule $S$ into a maximal usage schedule that has a finishing time at least as small as that of $S$.

Maximal usage preemptive schedules were studied in [3] for ordinary task systems. It was shown that any maximal usage schedule is not worse than $(1/2)+\sqrt{m}$ times worse than optimal. The method used there was to obtain two performance bounds in terms of a parameter $r$, which were inversely related as a function of $r$. A bound on the minimum of the two bounds was $(1/2)+\sqrt{m}$. This method did not generalize to typed task systems for the following reason. When each bound was generalized to the typed task system case, they were no longer inversely related.

In this section a performance bound on maximal usage schedules is obtained by appealing directly to the results of Section 5. Fix a set of processors with speeds $b_{i_j}$. Consider equation (10) in Section 5.1. It suffices to show that equation (10) (when interpreted as a bound on the performance of any maximal usage schedule) applies to any task system, and any maximal usage schedule for the task system, and any set of choices of indices $\{i_j\}$. From this, one may conclude that for any task system and any maximal usage schedule for the task system (with finishing time $w$), $w/w_{\text{opt}} \leq k+2\sqrt{\max_j (km_j)}$ . Similarly, one may conclude that for any task system and any maximal usage schedule $w/w_{\text{opt}} \leq k + \sqrt{m_1} + \ldots + \sqrt{m_k} + \max_j \sqrt{m_j}$. This proof follows by applying the bound of equation (10) for the set of $i_j$'s that minimize equation (10). For this set of $i_j$'s, equation (10) is bounded by the above quantities (as shown in Section 5.2). Note that in this context $w_{\text{opt}}$ represents the finishing time of the optimal *preemptive* schedule.

It suffices to show that $w_{\text{opt}}$ satisfies the lower bound of Lemma 5.1 and $w$ satisfies the upper bound of Lemma 5.2 (using the definition of height relevant to the set of chosen $i_j$'s). The former is immediate, since the lower bound did not consider the fact that non preemptive schedules were used.

To get the upper bound on $w$ given in Lemma 5.2, break up all intervals of any maximal usage schedule into two types of intervals. One type of interval is when $i_j$ processors of type $j$ are being used for some $j$, and the second type is when $i_j$ processors of type $j$ are not being used for any $j$. Clearly, one may use at least $i_j$ processors of type $j$ for at most a total of $\mu/B_{ji_j}$ units of time. Also, the intervals during which $i_j$ processors of type $j$ are not used for any $j$ must be height reducing intervals. These height reducing intervals decrease the height by a rate of at least one per unit time. Lemma 5.2 follows and thus one may conclude:

**Theorem 5.** Let $(\mathcal{T},<,\mu,\nu)$ as above. Let $w$ be the finishing time of any preemptive maximal usage schedule, and let $w_{\text{opt}}$ be the finishing time of an optimal preemptive schedule. Then
$w/w_{\text{opt}} \leq \min(k+2\sqrt{\max_j (km_j)}, k+\sqrt{m_1} + \dots + \sqrt{m_k} + \max_j \sqrt{m_j})$.

Achievability may similarly be obtained by appealing ot the constructions of Section 5.3. The "bad" list schedules on the $i_j$ fastest processors of type $j$ are also bad maximal usage preemptive schedules.

## Conclusion

The algorithms presented in this paper are examples of scheduling algorithms that violate the naive "greedy" heuristic of trying to schedule as many tasks as possible at each point in time. In this context methods have been developed for deciding when to be greedy and how greedy to be. It would be interesting to obtain similar algorithms for other situations, for example where the processors are identical.

## Acknowledgements

The author wishes to express his gratitude to E. Davis and M. Rabin for helpful discussions and in particular for their suggestions on how to approach the proof of Theorem 2. Also, to A. Meyer for several helpful comments. Finally, to D. Kessler for programming the MACSYMA calculations.

## References

1. R. L. Graham, Bounds on Multiprocessing Timing Anomalies, *SIAM J. Appl. Math.*, *17* (1969), 263-269.

2. E. C. Horvath, S. Lam, and R. Sethi, A Level Algorithm for Preemptive Scheduling, *JACM 24*, 1, (1977) 32-43.

3. J. M. Jaffe, An Analysis of Preemptive Multiprocessor Job Scheduling, MIT, Laboratory for Computer Science, Technical Memo No. 110, September 1978. Also, submitted to *Mathematics of Operations Research*.

4. J. M. Jaffe, Bounds on the Scheduling of Typed Task Systems, MIT, Laboratory for Computer Science, Technical Memo No. 111, September 1978. Also, submitted to *SIAM Journal on Computing*.

5. J. W. S. Liu and C. L. Liu, Bounds on Scheduling Algorithms for Heterogeneous Computing Systems, TR No. UIUCDCS-R-74-632 Dept. of Comp. Sci., Univ. of Illinois, June 1974.

6. J. W. S. Liu and C. L. Liu, Bounds on Scheduling Algorithms for Heterogeneous Computing Systems, *IFIP74*, (North Holland Pub. Co.), 349-353.

7. J. W. S. Liu and C. L. Liu, Performance Analysis of Multiprocessor Systems

Containing Functionally Dedicated Processors, *Acta Informatica*, *10*, 1, (1978) 95-104.

8. R. R. Muntz and E. G. Coffman Jr., Optimal preemptive scheduling on two-processor systems, *IEEE Trans. Comptrs.*, *C-18*, 11 (1969) 1014-1020.

9. R. R. Muntz and E. G. Coffman Jr., Preemptive scheduling of real time tasks on multiprocessor systems, *JACM 17*, 2 (1970) 324-338.