MIT/LCS/TM-111

BOUNDS ON THE SCHEDULING OF TYPED TASK SYSTEMS

Jeffrey M. Jaffe

September 1978

Bounds on the scheduling of typed task systems

Jeffrey M. Jaffe, MIT *

Abstract. We study the scheduling of different types of tasks on different types of processors. If there are k types of tasks and m_i identical processors for tasks of type i, the finishing time of any demand driven or list schedule is at most $k+1-(1/\max(m_1,\ldots,m_k))$ times worse than the optimal schedule. This bound is best possible. If the processors execute at different speeds then the performance ratio of any list schedule (relative to the optimal schedule) is bounded by k plus the maximum ratio between the speeds of any two processors of the same type.

Keywords. scheduling, list scheduling, typed task systems, data flow computation, worst case performance bounds

1. Introduction

The problem of job scheduling on multiprocessor systems has been extensively studied (for a current survey see [1,7]). The conventional approach has been to consider a system where each processor may handle any job or task. These systems are referred to as "ordinary" task systems. In some systems it may be useful to have certain tasks processed only by designated processors for those tasks. Examples of these include data flow models of computation [2,8] where primitive operations are computed by different processors. Similarly, in

---

* This report was prepared with the support of a National Science Foundation graduate fellowship, and National Science Foundation grant no. MCS77-19754.

machines such as the CDC6600, there are several specialized functional modules. Also, in a system where I/O tasks and arithmetic tasks are handled by different processor units, such an assumption may be relevant. In this paper we analyze some of the properties of schedules for systems with different *types* of tasks. Many of the results for ordinary task systems immediately generalize to the typed case. The NP completeness results of [12] clearly carry over directly, as do approximate solutions for certain special cases (for example when the tasks are independent [3]).

The complexity of determining the optimal schedule is NP-complete in very simple cases. It is shown in [5], that the problem of determining whether a schedule exists for a given typed task system that requires fewer than a given number of steps is NP-complete even if there are only two processors, one of each type. Also, if the number of types of processors varies, the problem is NP-complete even if the precedence constraint is restricted to being a forest. The techniques used there are adaptations of those found in [1,4].

The focus of this paper is to extend the results of Graham [6], which provide general bounds for non-preemptive list scheduling strategies which satisfy fundamental "no-waste" requirements. The performance criterion is that we attempt to minimize the finishing time of the system. In ordinary task systems, any list schedule is at most $2-(1/m)$ times worse than optimal where m is the number of processors. For typed task systems as defined in Section 2 a similar bound is obtained. With a similar definition of "unwasteful schedules", it is shown in Section 3 that any such schedule is at most $k+1-(1/\max(m_1,\ldots,m_k))$ times worse than optimal, where k is the number of types of tasks and m_i is the number of processors of type i. This bound is achievable for any value of k and any values of $m_1,\ldots,m_k$ as shown in Section 4.

The results of [9,10] which provide general bounds for list schedules on machines with processors of different speeds are also extended. In ordinary task systems with processors of different speeds, any demand driven schedule is at most (approximately) $(f/s)+1$ times worse than optimal where f is the speed of the fastest processor and s the speed of the slowest processor. It is shown (in Sections 6 and 7) that the bound for typed task systems is (approximately) $k+\max(f_1/s_1,\ldots,f_k/s_k)$ where f_i is the speed of the fastest processor of type i and s_i is the speed of the slowest processor of type i. Section 5 introduces additional definitions needed to discuss the case where processors run at different speeds.

2. Typed Task Systems

An *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* $\mu:\mathcal{T}\rightarrow\mathbb{N}$.

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 number of *steps* required by the task T. For simplicity we have assumed that the range of μ is $\mathbb{N}$. This corresponds to the assumption that each operation requires an integral number of steps.

The preceding is the conventional definition of task systems. We now extend the model to the situation where there are many types of processors.

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

The execution of a task system by processors of a machine is modelled by the notion of a schedule. A *schedule* for $(\mathcal{T},<,\mu,\nu)$ is a total function $g:\mathcal{T}\to\mathbb{N}$. We refer to $g(T)$ as the *starting time* of the task T and $g(T)+\mu(T)$ as the *finishing time* of the task T. We also say that $T\in\mathcal{T}$ is *being executed at time* t for $g(T)\le t<g(T)+\mu(T)$. We often refer to time t as the t^{th} *step of execution*.

A *valid schedule* for $(\mathcal{T},<,\mu,\nu)$ *on a set of equally fast processors* $\mathcal{P}=\{P_{ij}: 1\le i\le k \text{ and } 1\le j\le m_i\}$ is a schedule for $(\mathcal{T},<,\mu,\nu)$ with the properties:

(1) For all $i=1,\ldots,k$ and all $t\in\mathbb{N}$ the number of tasks of type i being executed at time t does not exceeds m_i.

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

Condition one asserts that processor capabilities may not be exceeded. Condition two forces the obedience of precedence constraints.

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

There are schedules that may be arbitrarily worse than the optimal schedule. For example, there may be a time before the finishing time at which no task is being executed, but such trivially improvable schedules are not interesting. Schedules of interest are not so blatantly wasteful. We restrict attention to schedules conforming to a heuristic (or what may be used as part of a heuristic) that has attracted attention for ordinary task systems [6,9,10]. 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 $(\mathcal{T},<,\mu,\nu)$ with the list L is defined as follows. At each step that at least one processor completes a task, each processor that is not still executing a task chooses an unexecuted executable task of its type. The tasks are chosen by giving higher priority to those unexecuted tasks with the lowest indices. If $n>1$ processors of the same type 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. Only if no executed tasks of its type are executable does a processor remain idle. List scheduling is unwasteful in the sense that if at time t all predecessors of a task T have been finished, and there are free processors of type $\nu(T)$ (that is at most $m_{\nu(T)}-1$ tasks of type $\nu(T)$ are being executed at time t) then the starting time of T is no later than t. 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.

The motivation for this heuristic comes from several sources. The primary motivation emanates from the optimality of some list schedule in the unit execution time case. It can be shown that when each task requires an equal amount of time at least one list schedule is an optimal schedule. While optimality does not occur if tasks have different time requirements, we feel that it is nonetheless worthwhile to evaluate list schedules. Other sources of interest include the fact that it is simple to implement, and due to its simplicity, it is a good starting place for building other heuristics.

(*Notation*: The total number of steps required by all the type i tasks will be denoted by μ_i.)

3. Performance bounds for list scheduling

In this section a bound is obtained on the performance ratio of any list schedule relative to an optimal schedule. It is shown that in a k type task system, the ratio is at most $k+1-(1/\max(m_1,\ldots,m_k))$. A naive bound on the performance ratio of any list schedule relative to an optimal schedule is given by $m_1+\ldots+m_k$. This follows from the fact that an optimal schedule may use at most $m_1+\ldots+m_k$ processors at each step, and the fact that any list schedule uses at least one processor at every step. The result of this section is that any list schedule is far better than the naive bound. Note that the comparison of list schedules to optimal schedules is applicable even to the situations that no optimal schedule is a list schedule.

We first prove the theorem for "unit execution time" task systems. A task system $(\mathcal{T},<,\mu,\nu)$ is a *unit execution time* task system if $\mu(T)=1$ for every $T\in\mathcal{T}$. As indicated in Section 2, this case is particularly interesting due to the optimality of some list schedule for unit execution time systems. The result for non unit execution time task systems will then follow directly from the unit execution time case.

Theorem 1. Let $(\mathcal{T},<,\mu,\nu)$ be a k type unit execution time task system. Then the performance ratio of any list schedule relative to an optimal schedule is at most $k+1-(1/\max(m_1,\ldots,m_k))$.

It will be convenient to break up the proof into two parts. First, a lower bound is derived on the finishing time of any optimal schedule for $\mathcal{T}$. Then an upper bound on the finishing time of any list schedule is obtained. The ratio between these bounds is an upper bound on the performance ratio of

any list schedule relative to an optimal schedule.

A notion that is needed in both parts of the proof is the notion of the *height* of tasks in $(\mathcal{T},<,\mu,\nu)$. A height of 1 is assigned to any task T that has no successors. Inductively, the height assigned to a task T is one plus the maximum height of all the immediate successors of T. The *height of* $(\mathcal{T},<,\mu,\nu)$, denoted h, is the maximum height of the set of tasks $T\in\mathcal{T}$.

Lemma 1. Let $(\mathcal{T},<,\mu,\nu)$ be a k type unit execution time task system with g_0 an optimal schedule for $(\mathcal{T},<,\mu,\nu)$. Then the finishing time, w_0, of g_0 satisfies: $w_0\geq\max(h,\mu_1/m_1,\ldots,\mu_k/m_k)$.

Proof. Clearly at most m_i total steps of type i tasks may be executed during each time unit (for every i). Thus at least $\lceil\mu_i/m_i\rceil$ units of time must be spent on the execution of $\mathcal{T}$ for every i. A conservative lower bound is thus $\max(\mu_1/m_1,\ldots,\mu_k/m_k)$.

Also, consider a chain of length h in the graph specified by $(\mathcal{T},<)$. At least one such chain must exist by the way height is defined. SInce each of these tasks must be executed at different steps, we conclude that h is a lower bound on the finishing time. Thus $w_0\geq\max(h,\mu_1/m_1,\ldots,\mu_k/m_k)$. □

Lemma 2. Let $(\mathcal{T},<,\mu,\nu)$ be a k type unit execution time task system and g a list schedule for $(\mathcal{T},<,\mu,\nu)$. Then the finishing time, w of g satisfies $w\leq(\mu_1/m_1)+(\mu_2/m_2)+\ldots+(\mu_k/m_k)+h(1-(1/\max(m_1,\ldots,m_k)))$.

Proof. The basic idea is to analyze two types of steps that occur in list schedules. The first type of step occurs when progress is made in finishing

one of the h "height levels" in the graph specified by $<$. The second type of step occurs when no progress is made towards completing a level of the graph.

We say that level j of $(\mathcal{T},<,\mu,\nu)$ is *finished at time t* if t is the smallest time with the property that $t'>t$ implies that no task of height j is started at time t'. Time t is said to be a *level finishing step* if there exists a j, such that level j is finished at time t.

To prove the lemma, the steps of any list schedule are broken into two parts. The first part consists of the level finishing steps. There are at most h of these steps. This follows from the fact that the height (or number of levels) of $(\mathcal{T},<,\mu,\nu)$ is only h and the fact that there is one "last step" per level. The goal is now to show that there are no more than $(\mu_1/m_1)+\ldots+(\mu_k/m_k)-(h/\max(m_1,\ldots,m_k))$ "non-level finishing steps".

Note that at any step t at which a level is not finished, m_i tasks of type i must be executed for some i. The reasoning is as follows. Let j be the greatest level that did not finish before time t. Note that if a task is at height j all predecessors of the task have been finished. Thus if a task of height j has not yet executed, the task must be executable at time t. Assume that at this step we do not use all m_i processors for any i. That is, for each i, at least one processor is unused at this step. Such a "potential waste" of processors may occur only if *all* executable tasks are being executed since otherwise one of the unused processors would be used.. In particular all tasks at height j are being executed and level j is finished. Thus time t is a level finishing step contradicting the assumption.

Let h_i denote the number of steps of type i tasks executed during level finishing steps. Clearly $\Sigma_{i=1}^{k} h_i \geq h$ since for each level, at least one task at the level is executed during a level finishing step. Now an upper

bound on the number of non level finishing steps is obtained. Note that m_i tasks of type i are executed at no more than $\lfloor(\mu_i-h_i)/m_i\rfloor$ non-level finishing steps. Thus, executing m_i tasks of type i at one step for some i may occur during at most $\lfloor(\mu_1-h_1)/m_1\rfloor+\ldots+\lfloor(\mu_k-h_k)/m_k\rfloor$ distinct steps. That is, there are at most

$(\mu_1/m_1)+\ldots+(\mu_k/m_k)-((h_1/m_1)+\ldots+(h_k/m_k))\leq(\mu_1/m_1)+\ldots+(\mu_k/m_k)-(h/\max(m_1,\ldots,m_k))$

non-level finishing steps. A bound on w is thus given by:

$w\leq(\mu_1/m_1)+\ldots+(\mu_k/m_k)+h(1-(1/\max(m_1,\ldots,m_k)))$.□

We may now put together the upper bound on list schedules and the lower bound on optimal schedules. The two results combine to show that the worst possible performance ratio is $k+1-(1/\max(m_1,\ldots,m_k))$.

Proof of Theorem. Fix a k type unit execution time task system $(\mathcal{T},<,\mu,\nu)$, and let $p=\max(\mu_1/m_1,\ldots,\mu_k/m_k,h)$. Then a lower bound on the optimal schedule is p. A conservative upper bound on any list schedule is $(k+1-(1/\max(m_1,\ldots,m_k)))p$. Thus the performance ratio of the list schedule relative to the optimal schedule is bounded by $k+1-(1/\max(m_1,\ldots,m_k))$. □

The next result is an explanation of how Theorem 1 may be easily generalized to apply to non unit execution time task systems. The method is to reduce an arbitrary task system to a unit execution time task system and then apply Theorem 1.

Theorem 2. Let $(\mathcal{T},<,\mu,\nu)$ be a k type task system. Then the performance ratio of any list schedule relative to an optimal schedule is at most $k+1-(1/\max(m_1,\ldots,m_k))$.

Proof. For a given task system $(\mathcal{T},<,\mu,\nu)$ define an *induced* unit execution time task system $(\mathcal{T}',<',\mu',\nu')$ as follows. For a task $T\in\mathcal{T}$ there are $\mu(T)$ corresponding unit execution time tasks in $\mathcal{T}'$. These $\mu(T)$ tasks are linearly ordered by $<'$ and are of the same type as T. The rest of $<'$ is defined as follows. If $S,T\in\mathcal{T}$ and $S<T$ then each of the $\mu(S)$ tasks that correspond to S must precede each of the $\mu(T)$ tasks that correspond to T.

Note that an optimal schedule for $(\mathcal{T},<,\mu,\nu)$ has at least as large a finishing time as an optimal schedule for $(\mathcal{T}',<',\mu',\nu')$. This follows from the fact that any valid schedule for $(\mathcal{T},<,\mu,\nu)$ may be easily transformed into a valid schedule for $(\mathcal{T}',<',\mu',\nu')$ merely by executing the $\mu(T)$ tasks in $\mathcal{T}'$ (that correspond to $T\in\mathcal{T}$) during the same $\mu(T)$ steps that T is executed. Thus an optimal schedule for $(\mathcal{T},<,\mu,\nu)$ corresponds to some schedule for $(\mathcal{T}',<',\mu',\nu')$. In a similar manner, any list schedule for $(\mathcal{T},<,\mu,\nu)$ may be easily transformed into a list schedule for $(\mathcal{T}',<',\mu',\nu')$. Thus the "worst" list schedule for $(\mathcal{T},<,\mu,\nu)$ (i.e. the one with the greatest finishing time) is no worse than the "worst" list schedule for $(\mathcal{T}',<',\mu',\nu')$.

Let w_0 and w be the finishing times of an optimal schedule for $(\mathcal{T},<,\mu,\nu)$ and the "worst" list schedule for $(\mathcal{T},<,\mu,\nu)$ respectively. Let w'_0 and w' be the finishing times of an optimal schedule for $(\mathcal{T}',<',\mu',\nu')$ and the "worst" list schedule for $(\mathcal{T}',<',\mu',\nu')$ repectively. We have that

$$w'_0\le w_0\le w\le w'.$$

Since $(w/w'_0)\le(k+1-(1/\max(m_1,\ldots,m_k)))$

we conclude that $(w/w_0)\le(k+1-(1/\max(m_1,\ldots,m_k)))$. □

We remark that in the classical case where $k=1$, the result of this section reduces to the bound of $2-(1/m)$ obtained in [6].

4. Achievability results for list scheduling strategies

In this section it is shown that the bound of Section 3 is achievable. Specifically, for any k and any values of $m_1,...,m_k$ there is a k type task system and a list schedule for the system with the property that the schedule is $k+1-(1/\max(m_1,...,m_k))$ times worse than optimal.

The set of task systems used for this proof are sketched below (Figure 1). Each node in the graph represents one task. Arrows specify the partial ordering and the labels of the nodes represent the type of the tasks. Each task has unit execution time. (Assume without loss of generality that $m_k=\max(m_1,...,m_k)$.)

In the task systems, there are m_i columns of tasks that informally speaking "correspond to type i" ($1\leq i\leq k-1$). Each of these m_i columns contains a chain of $n+k-1$ tasks (n arbitrary). The j^{th} task in each of these columns has $\nu(T)=j$ (for $j\leq i-1$) and $\nu(T)=i$ (for $i\leq j$).

There are m_k+1 columns that "correspond to type k". In each of these columns the j^{th} task (for $j\leq k-1$) has $\nu(T)=j$. For the first m_k of these m_k+1 columns there is a chain of $n-(n/m_k)$ additional tasks, with $\nu(T)=k$ for each task in the chain. For the $(m_k+1)^{st}$ column there is a chain of n additional tasks, with $\nu(T)=k$ for each task in the chain.

The following is an asymptotically optimal strategy. The first $k-1$ tasks of each column are executed using an arbitrary list schedule. For fixed values of k and the m_i's this may be done in constant time. Now, only n steps are required to complete the entire system. It is clear that only n steps are required to finish the columns corresponding to each of the first $k-1$ types of

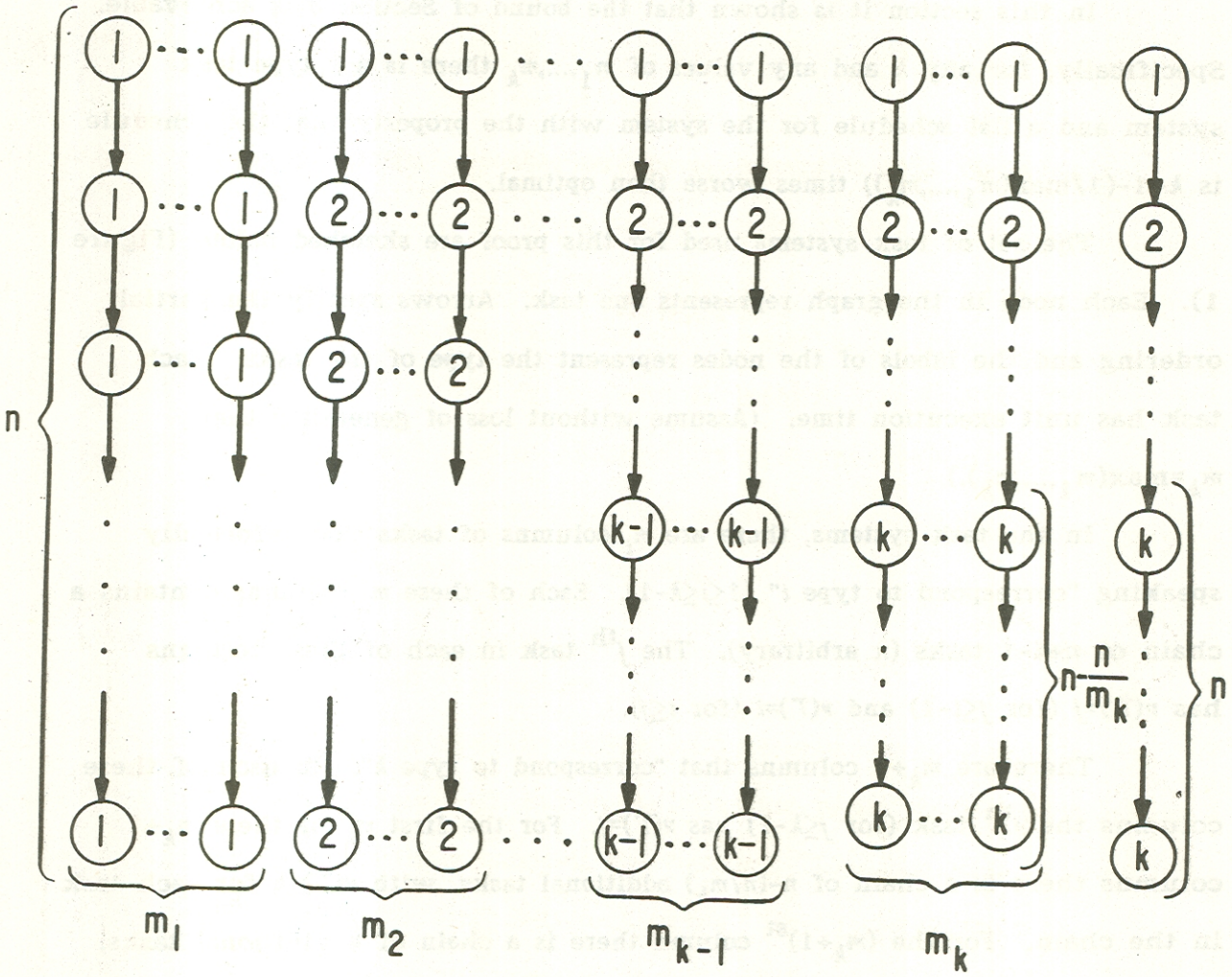

Figure 1

processors. During these same n steps the columns corresponding to the k^{th} type of processor may be completed as follows: During each of the n steps, one of the m_k processors of type k is used on the $(m_k+1)^{st}$ of these columns finishing this column in n steps. The other m_k-1 processors are used on the other m_k columns in rotation. Thus during the first step, no task is executed from the first column, during the second step, no task is executed from the second column, etc. Thus, the total number of steps for this procedure is $n+O(1)$.

An inefficient list schedule is now presented. The schedule first handles all type 1 tasks, then all type 2 tasks, etc. For the first $n+k-1$ steps only tasks from columns that correspond to type 1 are executed. At the next $n+k-1$ steps all tasks from columns that correspond to type 2 are executed, stripping off type 1 tasks from the tops of the rest of the columns in the process. In this manner, about $(k-1)n$ steps are required to finish all of the columns that correspond to the first $k-1$ types of processors.

Now the last m_k+1 columns of the program are executed. Using a list schedule, only the first m_k of them are processed for the next $n-(n/m_k)$ steps, completing these columns in their entirety. Another n steps are required just to process the last of these m_k+1 columns. The total number of steps with this schedule is thus $n(k+1-(1/m_k))$ and the performance ratio between this and the optimal schedule is $n(k+1-(1/m_k))/(n+O(1))$. As n goes to infinity, the ratio approaches $k+1-(1/m_k)$. □

A few remarks may be made about the nature of the construction. First, all tasks take unit time in the example. Thus, although the general bound on performance (Theorem 2) applies to any k type task system, it is achievable even in the special case where each task requires only unit time.

This is particularly significant in light of the fact that for this special case some list schedule is guaranteed to be optimal. Thus, even in the situations where nothing is "lost" by restricting attention to list schedules, the bound is still achievable. Another feature of interest is that the system used is a disjoint union of chains. Each chain may be viewed as one large task, and each task within the chain may be viewed as a subtask of the larger task. We thus overcome the objection that the example is a contrived, complicated system which is unlikely to occur in practice. Finally, the "bad" schedule was an uncontrived type of schedule. It is a schedule that operates on a LIFO (or last in first out) principle. That is, the schedule executes those tasks which most recently became executable.

5. Uniform non-identical processors

We now analyze the situation where each processor runs at a different rate. This is of particular interest due to the fact that in the models of high speed computation that partially motivate this research [2,8], the idea is to use many processors of potentially different speeds. This generalization is also the natural extension of the work of [9,10] which considered processors of different speeds for ordinary task systems.

The processors of each type are assumed to be *uniform* in this analysis. That is, the relative speeds of the processors are independent of the tasks being executed. The more complicated situation in which certain processors handle some tasks relatively quickly, but others relatively slowly is not even very well understood for ordinary task systems. Also, this situation is less likely to occur in a system where the tasks have already been subdivided into different types. In this regard typed task systems may be viewed as a special

case of non-uniform ordinary task systems. If a processor is of a different type than a task, then the speed for the processor on the task is infinity.

When the set of processors $\mathcal{P}=\{P_{ij}: 1\leq i\leq k \text{ and } 1\leq j\leq m_i\}$ are not equally fast, there is an associated *rate function* $r:\mathcal{P}\rightarrow\mathbb{N}$. Informally, the rate function specifies the speed of a processor. If a task T is assigned to a processor P then $\mu(T)/r(P)$ time units are required for the processing of T on P.

Since the speeds of the processors are not the same, the schedule must now specify which task is assigned to which processor. Thus, a *valid schedule* for $(\mathcal{T},<,\mu,\nu)$ *on a set of uniform non-identical processors* $\mathcal{P}$ with rate function r is a total function $g:\mathcal{T}\rightarrow\mathbb{N}\times\mathcal{P}$ (where if $g(T)=(t,P)$ then the *starting time* of T is t and the *finishing time* of T is $t+(\mu(T)/r(P))$) such that

(a) If $g(T)=(t,P_{ij})$ then $\nu(T)=i$.

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

(c) whenever $S<T$ the starting time of T is no less than the finishing time of S.

Informally, if $g(T)=(t,P)$ then the task T is processed by the processor P from time t to time $t+(\mu(T)/r(P))$. There is no loss of generality in assuming that each task is assigned an integer starting time as one may always take a common rational divisor of $\{1/r(P):P\in\mathcal{P}\}$ as the unit of time.

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* may be generalized in two ways. One way is to only insist that at no point in time may a task of a certain type be executable

while a processor of the same type remains unused. A second potential generalization is to further insist that when tasks become executable they are assigned to the fastest available processors. The bounds obtained are applicable to either generalization.

The definition of μ_i, the total number of steps required for type i tasks is the same as in Section 2.

The total processing power of processors of type i, denoted r_i is defined by:

$$r_i = \sum_{j=1}^{m_i} r(P_{ij})$$

This represents the total number of steps of type i tasks that may be processed in unit time.

Let f_i denote the rate of the fastest processor of type i. That is, $f_i=\max\{r(P_{ij}):1\leq j\leq m_i\}$. Similarly, s_i denotes the rate of the slowest processor of type i. Also, $q=\max(f_1/s_1,\ldots,f_k/s_k)$ denotes the greatest ratio (or quotient) between processor rates for any single type of processor. Finally, $d=\min(f_1/r_1,\ldots,f_k/r_k)$, denotes the smallest percentage contribution made by the fastest processor of a particular type.

6. Performance bounds for list schedules on machines with uniform non-identical processors

Following the general outline of Section 3, we obtain a lower bound on the performance of an optimal schedule and an upper bound on the performance of any list schedule. A comparison of the two results provides an upper bound on the performance ratio of any list schedule relative to the optimal schedule. The main result of this section is:

Theorem 3. Let $(\mathcal{T},<,\mu,\nu)$ be a k type task system on a set of uniform non-identical processors. Then the performance ratio of any list schedule relative to the optimal schedule is at most $k+q-d$.

Proof. It will again be helpful to define a notion of the *height* of a task. The notion of height differs slightly from the notion used when the processors were equally fast. Intuitively, the notion of height does not generalize directly from the identical processor case for the following reason. A task T always requires $\mu(T)$ time units to be processed with identical processors. However, in the non-identical situation the amount of time required is a function of which processor is used for the task.

In deciding how many "levels" should be assigned to each task, the important idea is to insure that at every step each non-idle processor executes at least one level of the task it is processing. Thus, if a task T of type j requires time $\mu(T)$, then T is assigned $\mu(T)/s_j$ *levels*. Similarly, if T has no successors then it is at *height* $\mu(T)/s_j$. Otherwise, the height of T is $\mu(T)/s_j$ plus the maximum height of the set of immediate successors of T. The *height* of $(\mathcal{T},<,\mu,\nu)$ is the maximum height of the set of tasks $T\in\mathcal{T}$. If T is of height h' and $g(T)=(t,P)$ then for $i=0,\ldots,\mu(T)/r(P)-1$ *the half-open interval* $(\; h'-(r(P)/s_j)(i+1) \; , \; h'-(r(P)/s_j)(i) \;]$ *of* T *is executed* at time $t+i$. *Level* j of T is *executed at time* $t+i$ if j is a point in the interval of T that is executed at time $t+i$. Note that at least one level of a task is finished at each step that the task is executed. *Level* j is *finished at time* t if time t is the last step during which level j of some task is executed.

We first obtain the upper bound on list schedules. Fix a list schedule, g, and let p denote the number of level finishing steps used by

the list schedule (note $p \leq h$ since each level contributes to at most one level finishing step).

At a non-level finishing step a list schedule must use all m_i processors of type i for some i. The reasoning here is the same as in Section 3; otherwise the greatest unfinished level must be completed. Since there are at most $\lfloor \mu_i / r_i \rfloor$ steps during which all processors of type i may be used, an upper bound on the number of steps during which a level is not finished is given by $(\mu_1 / r_1)+\ldots+(\mu_k / r_k)$. As in Section 3, this may be tightened. Recall that there were p level finishing steps. Let p_i denote the sum of the sizes of intervals of type i tasks executed during level finishing steps. (Note $p_1+\ldots+p_k \geq p$ since the size of each interval is at least one.) By the definition of the intervals at least $p_i s_i$ units of the time requirement of type i tasks are executed during level finishing steps since executing an interval of type i and size 1 corresponds to executing s_i units of the time requirement of type i. Thus there are at most $(\mu_1 - p_1 s_1)/r_1+\ldots+(\mu_k - p_k s_k)/r_k$ non-level finishing steps.

To obtain a lower bound on the finishing time of any optimal schedule, note that only r_i steps of type i tasks may be executed in one time unit. Thus a lower bound is given by $\max(\mu_1 / r_1, \ldots, \mu_k / r_k)$.

Also, consider a set of tasks $T_1 < T_2 < \ldots < T_y$ where, informally speaking the whole path is of height h. (Formally, $h = \Sigma_{i=1}^{y} \mu(T_i)/s_{\nu(T_i)}$.) Some such path must exist by the way that height is defined. Let c_i be the sum of the $\mu(T)$'s for tasks of type i along this path. (Note $h = c_1/s_1+\ldots+c_k/s_k$.) At any point in time at most one of these tasks may be executed. Thus, a lower bound is given by $c_1/f_1+\ldots+c_k/f_k$.

Let w be the finishing time of an arbitrary list schedule and let w_O

be the finishing time of an optimal schedule. A bound on the performance ratio between any list schedule and the optimal schedule is given by:

$$(1) \qquad \frac{w}{w_0} \leq \frac{(\mu_1 - p_1 s_1)/r_1 + \ldots + (\mu_k - p_k s_k)/r_k + p}{\max(\mu_1/r_1, \ldots, \mu_k/r_k, (c_1/f_1) + \ldots + (c_k/f_k))}$$

where the $c_1/f_1 + \ldots + c_k/f_k$ term in the denominator represents the sum of the $\mu(T)'s$ for any path of "height" h as above. Rewriting (1) provides

$$(2) \qquad \frac{w}{w_0} \leq \frac{(\mu_1/r_1) + \ldots + (\mu_k/r_k) + p - ((p_1 s_1/r_1) + \ldots + (p_k s_k/r_k))}{\max(\mu_1/r_1, \ldots, \mu_k/r_k, (c_1/f_1) + \ldots + (c_k/f_k))}$$

To obtain an upper bound the value of the numerator may be increased. Recall that $q = \max(f_1/s_1, \ldots f_k/s_k)$ and $d = \min(f_1/r_1, \ldots f_k/r_k)$. Since $q \geq f_i/s_i$ we may replace s_i with f_i/q. Then, f_i/r_i may be replaced with d (the minimum of the $f_i/r_i's$). Finally, using $p \leq p_1 + \ldots + p_k$ results in:

$$(3) \qquad \frac{w}{w_0} \leq \frac{(\mu_1/r_1) + \ldots + (\mu_k/r_k) + (1 - (d/q))p}{\max(\mu_1/r_1, \ldots, \mu_k/r_k, (c_1/f_1) + \ldots + (c_k/f_k))}$$

Since $p \leq h$ and $h=(c_1/s_1)+\ldots+(c_k/s_k)$ (3) may be rewritten as:

$$(4) \qquad \frac{w}{w_0} \leq \frac{(\mu_1/r_1)+\ldots+(\mu_k/r_k)+(1-d/q)((c_1/s_1)+\ldots+(c_k/s_k))}{\max(\mu_1/r_1,\ldots,\mu_k/r_k,(c_1/f_1)+\ldots+(c_k/f_k))}$$

Again using $q \geq f_i/s_i$ for every i yields

$$(5) \qquad \frac{w}{w_0} \leq \frac{(\mu_1/r_1)+\ldots+(\mu_k/r_k)+q(1-d/q)((c_1/f_1)+\ldots+(c_k/f_k))}{\max(\mu_1/r_1,\ldots,\mu_k/r_k,(c_1/f_1)+\ldots+(c_k/f_k))}$$

The ratio between the sum of $k+q-d$ terms and the maximum of the same terms is at most $k+q-d$. □

Note that when the processors of each type are equally fast then $q=1$ and $d=1/\max(m_1,\ldots,m_k)$ and the bound matches that of Section 3. Also, if $k=1$, then the bound of $1+(f_1/s_1)-(f_1/r_1)$ matches the bound of [9,10].

7. Achievability results for list scheduling on machines with uniform non-identical processors

To obtain a lower bound on the performance of any list schedule for uniform non-identical task systems we combine the construction of Section 4 with a construction used in [9,10]. The result used from [9,10] is as follows. Fix a set of uniform non-identical processors $\mathcal{P}$, of one type. Then there are a set of ordinary task systems for $\mathcal{P}$ (with empty precedence relation) with the property that the performance ratio of list schedules relative to optimal

schedules over this set of task systems is arbitrarily close to $1+(f/s)-(f/r)$ where f is the speed of the fastest processor, s the speed of the slowest and r the total processing power of the processsors of the machine.

Consider the task system of Figure 2. Diagramming conventions are as in Section 4. The notation $\mu=r(P_{ij})$ means that the time required for the task equals the rate of the processor P_{ij}. A node labelled with B denotes a copy of one of the task systems in the set used to obtain the lower bound in [9,10] with the type of each task in this task system being m_k. The interpretation of an arrow between two nodes labelled with B indicates a precedence dependence of each task at the destination of the arrow on each task at the source of the arrow. The class of task systems described in the figure is parameterized by the variable n and the class of task systems described in [9,10]. Let n' denote the time required to execute B using an optimal schedule. Assume without loss of generality that $\max(\{(f_i/s_i)-(f_i/r_i):i=1,..,k\})$ is achieved by processors of type k.

An asymptotically optimal schedule first executes the first $k-1$ tasks of each column using an arbitrary list schedule. Then only n more steps are required. It is clear how to finish the columns that correspond to the first $k-1$ types of tasks in n steps. By using the optimal schedule for each occurrence of B each occurrence of B requires only n' steps. Since there are n/n' copies of B only n steps are required.

A bad list schedule spends $(k-1)n$ steps completing the tasks that correspond to the first $k-1$ types of processors. It then spends arbitrarily close to $(n/n')(n')(1+f_k/s_k-f_k/r_k)$ steps to complete the columns that correspond to the k^{th} type of processor using the bad list schedule from

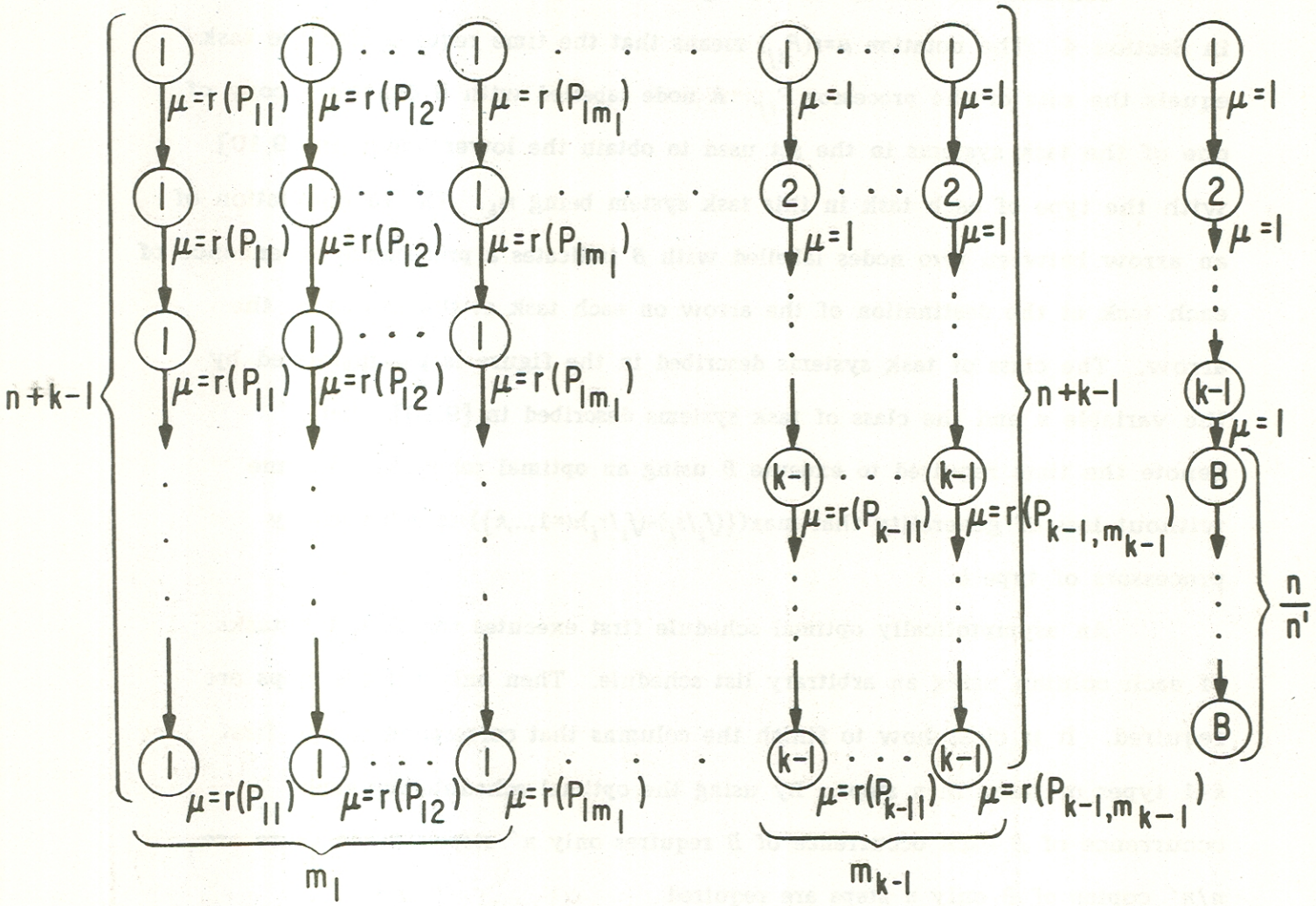
n+k-1
1
2
k-1
B
μ=1
μ=r(P11)
μ=r(P12)
μ=r(P1m1)
μ=r(Pk-11)
μ=r(Pk-1,mk-1)
m1
mk-1
n+k-1
n/n'

Figure 2

[9,10]. The exact number of steps depends on which task system is used for the nodes labelled with B in the task system sketched in Figure 2. The ratio thus approaches $k+f_k/s_k-f_k/r_k$ for large n and B's whose variation in execution speed approaches a ratio of $1+(f_k/s_k)-(f_k/r_k)$. □

The gap between our upper and lower bounds on performance ratios is not very large. The gap is between $k+\max(\{f_i/s_i:i=1,..,k\})-\min(\{f_i/r_i:i=1,..,k\})$ and $k+\max(\{(f_i/s_i)-(f_i/r_i):i=1,..,k\})$. Since both are between $k+q$ and $k+q-1$, for all practical purposes the result is tight. Also, for certain important subcases that have been considered, the result does reduce to a tight result. In particular, if $k=1$ or if processors of each type run at the same rate then the bound on performance ratio obtained in Theorem 3 is achievable.

Also note that the notion of list schedule considered in [9,10] is the version where whenever tasks become executable, they are assigned to the fastest available processors. Thus, as mentioned in Section 5, our results are applicable even to the more restrictive notion of list schedule.

8. Conclusion

We have presented a generalization of the ordinary task systems that are used to model scheduling problems. This generalization is a more effective model of the scheduling problem found on certain types of machines. We have presented general bounds on scheduling strategies for these systems.

There are many extensions to our analysis that should be considered. The application of approximation techniques used in traditional scheduling theory would be a worthwhile endeavor, though we suspect that many of the special case results that have been obtained (for example when the precedence relation is empty) would be directly applicable to this case. Similarly, a study of typed

task systems with different optimality criteria would be of interest (see [7] for a survey of variations).

Finally, it would be instructive to settle the gap between upper and lower bounds for typed task systems executed on a set of processors of different speeds. While the results presented give a fairly tight description of the worst case performance of demand driven schedules, we would like to have the result tightened from the mathematical standpoint.

Acknowledgements

The author would like to express his thanks to Alan Baratz, Errol Lloyd Michael Loui, and Albert Meyer for helpful readings of this paper.

References.

1. E. G. Coffman, *Computer and Job Shop Scheduling Theory*, J. Wiley and Sons, NY 1976.

2. J. B. Dennis, First Version of a Data Flow Procedure Language, *Lecture Notes in Computer Science 19* (G. Goos and J. Hartmanis eds.), pp 362-376. Also *Symposium on Programming*, Institut de Programmation, Univ. of Paris, Paris, France, April 1974, pp 241-271. Also MIT LCS TM61, May 1975.

3. M. R. Garey and R. L. Graham, Bounds for Multiprocessing Scheduling with Resource Constraints, *SIAM J. Comput. 4*, 2, June 1975, pp 187-200.

4. M. R. Garey and D. S. Johnson, Complexity Results for Multiprocessor Scheduling under Resource Constraints, *Proceedings of the Eighth Annual Princeton Conference on Information Sciences and Systems*, 1974.

5. D. K. Goyal, Scheduling Processor Bound Systems, *Proceedings of the Sixth Texas Conference on Computing Systems* 1977.

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

7. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, *Discrete Optimization, 1977.*

8. R. M. Karp and R. E. Miller, Properties of a model for parallel computations: determinacy, termination, queueing, *SIAM J. of Applied Math., 14*, 6, Nov. 1966, pp 1390-1411.

9. 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.

10. J. W. S. Liu and C. L. Liu, Bounds on Scheduling Algorithms for Heterogeneous Computing Systems, *IFIP74*, North Holland Pub. Co., pp349-353.

11. J. E. Thornton, *Design of a Computer - The Control Data 6600*, Scott, Foresman College Division, (1971).

12. J. D. Ullman, *NP*-complete scheduling problems, *JCSS 3*, June 1975, pp 384-393.