A Level Algorithm for Preemptive Scheduling

EDWARD C. HORVATH, SHUI LAM, AND RAVI SETHI

The Pennsylvania State University, University Park, Pennsylvania

ABSTRACT Muntz and Coffman give a level algorithm that constructs optimal preemptive schedules on identical processors when the task system is a tree or when there are only two processors available Their algorithm is adapted here to handle processors of different speeds. The new algorithm is optimal for independent tasks on any number of processors and for arbitrary task systems on two processors, but not on three or more processors, even for trees By taking the algorithm as a heuristic on m processors and using the ratio of the lengths of the constructed and optimal schedules as a measure, an upper bound on its performance is derived in terms of the speeds of the processors It is further shown that 1 $23\sqrt{m}$ is an upper bound over all possible processor speeds and that the 1 $23\sqrt{m}$ bound can be improved at most by a constant factor, by giving an example of a system for which the bound 0 $35\sqrt{m}$ can be approached asymptotically

KEY WORDS AND PHRASES. worst-case analysis, critical path algorithm, minimal length schedules, processor sharing

CR CATEGORIES 5 25, 6 32

1. *Overview*

The time taken to finish a given set of tasks to be executed on a multiprocessor system (the *schedule length*) provides a measure of processor utilization Schedule length has therefore been a popular cost criterion in the design of scheduling algorithms. In this paper we consider *preemptive* schedules in which it is possible to suspend, and at a later stage resume, execution of a task at the point of suspension. Preemption costs are assumed to be negligible.

In general, processors in a computer system may execute at different speeds. Even if the hardware is identical, a processor P may be spending some of its time servicing interrupts from peripheral devices. If the proportion of time processor P spends attending to such tasks is relatively stable, then P can be treated as a slower dedicated processor.

Thus we have a set of m processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of *speeds* $b_1 \geq b_2 \geq \cdots \geq b_m$, where b_m is generally taken to be 1. We use the notation $\mathcal{P} = (b_1, b_2, \ldots, b_m)$ to denote the processors and their speeds. $\mathcal{P}$ is called a system of *identical* processors if $b_1 = b_2 = \cdots = b_m = 1$. A task with service requirement τ takes τ/b_j time units to finish on processor P_j.

Scheduling algorithms tend to fall into two categories: (1) "critical path" or "level" algorithms, and (2) "bin packing" algorithms. The "level" approach incrementally (in time) builds up a schedule on all processors The "bin packing" approach assigns all tasks that will ever be assigned to processor P_1 before considering processor P_2 and so on.

An earlier version of this paper was presented at the Fifth Symposium on Operating Systems Principles, Austin, Texas, November 1975

This work was partially supported by the National Science Foundation under Grant GJ-28290

Authors' present addresses E C Horvath, Computer Science Department, 312 Whitmore Laboratory, The Pennsylvania State University, University Park, PA 16802, S Lam, School of Computer Science, McGill University, Montreal, Canada, R Sethi, Bell Laboratories, Murray Hill, NJ 07974

The bin packing approach has been successfully used by McNaughton [7] and Rothkopf [10] to construct shortest preemptive schedules for independent tasks on $m \geq 1$ identical processors. When processor speeds are not identical, the bin packing approach falters. Liu and Liu [5] as well as Liu and Yang [6] assign independent tasks on a system with one fast processor. Proving that their algorithm constructs optimal schedules is long and involved. They also give an expression for the length of an optimal schedule for a set of independent tasks on an arbitrary processing system $\mathcal{P}$.

In this paper we adapt the "critical path" algorithm used by Muntz and Coffman [8, 9] to the case in which processor speeds are not all identical. Since the algorithm we give is a straightforward generalization of the one due to Muntz and Coffman, we shall use the term "the level algorithm" to refer to either algorithm. It is easy to show that the algorithm determines the shortest schedules for independent tasks on any number of processors. The number of preemptions may, however, be high.

In order to specify the algorithm in question more precisely, we need the notion of "level" of a task. Suppose there is no shortage of identical processors. The *level* of a task T then corresponds to the minimal amount of time it would take to execute T and all tasks that follow it. The *level algorithm* of Muntz and Coffman [8, 9] executes tasks level by level, starting with the highest level tasks

The level algorithm can also be used when there are constraints on the order in which tasks are to be processed. An example of such constraints is shown in Figure 2. We represent *task systems* by $(\mathcal{T}, <)$, where $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ is a set of *tasks* with *service requirements* $\tau_1, \tau_2, \ldots, \tau_n$, respectively, and $<$ is a partial order on $\mathcal{T}$ specifying the precedence constraints. For given tasks T and T', if $T < T'$ then task T must be finished before T' can be started. It follows from Lam and Sethi [4] that the level algorithm constructs shortest schedules when there are two processors in the system. A similar result for identical processors is given by Muntz and Coffman [8]

On three or more processors of different speeds, the level algorithm is not optimal even when the partial order $<$ defines a tree on $\mathcal{T}$ By way of contrast, Muntz and Coffman [9] show that when $<$ defines a tree, the algorithm constructs optimal schedules on any number of identical processors. We therefore study the algorithm as a heuristic on $m \geq 1$ processors.

It is usual to measure scheduling heuristics by looking at the ratio of the constructed to the optimal schedule lengths. The worst-case *performance* of a heuristic is given by the largest value that this ratio can take over all possible task systems.

Lam and Sethi [4] showed that on identical processors $2-2/m$ is an upper bound on the worst-case performance of the level algorithm. Moreover $2-2/m$ is a best bound in that there exist task systems for which this bound can be approached arbitrarily closely. In Section 4 we study the level algorithm and derive an upper bound on its worst-case performance in terms of the speeds of the given processor system. In order to get a feeling for the value of this upper bound over all possible values of processor speeds, we determine the upper bound $\sqrt{(1.5m)} \approx 1.23\sqrt{m}$, which depends only on m, the number of processors. We also give an example of a system with one fast processor for which the ratio $\sqrt{(m)}/2\sqrt{2} \approx 0.35\sqrt{m}$ can be asymptotically approached, thus establishing that the $\sqrt{(1.5m)}$ bound cannot be improved much further.

In summary, this paper is devoted to a study of a critical path algorithm on a system of processors $\mathcal{P}$. Variations of critical path algorithms are the only ways known to construct shortest preemptive schedules in the presence of precedence constraints. A thorough understanding of such algorithms is therefore desirable.

The reader is referred to Coffman [1] for background information on scheduling problems.

2. *The Level Algorithm*

Suppose we are given tasks T_1 and T_2 with service requirements 8 and 7 to be scheduled on $\mathcal{P} = (2, 1)$. T_1 requires 4 time units on P_1. If T_1 is assigned to P_1 and T_2 to P_2 as in

Figure 1(a), we get a schedule of length 7. A shorter schedule can be obtained by averaging the execution of T_1 and T_2, as in Figure 1(b).

Definitions. Since we will be talking of the collective processing rate of a set of processors, define $B_j = \sum_{i=1}^{j} b_i$ for all j, $1 \le j \le m$. Similarly let $X_j = \sum_{i=1}^{j} \tau_i$ for all j, $1 \le j \le n$. □

Consider a set of independent tasks with service requirements $\tau_1 \ge \tau_2 \ge \cdots \ge \tau_n$ to be scheduled on $\mathscr{P} = (b_1, b_2, \ldots, b_m)$. Since each task can be executed by at most one processor at a time, any schedule will be at least τ_1/b_1 units long. Task T_1 would take at least as long on any other processor. If the second processor is significantly slower than the first, it is possible that τ_2/b_2 is longer than τ_1/b_1. As in Figure 1(b), we can average the execution of the two tasks. The total service requirement is $X_2 = \tau_1 + \tau_2$. The processing rate is $B_2 = b_1 + b_2$. Thus the averaged schedule will be at least X_2/B_2 units long. In general we have to consider $\max_j X_j/B_j$ as a lower bound on the schedule length since τ_3/b_3 may be longer than X_2/B_2 and so on. Finally, consider the case in which $n \ge m$. X_n/B_m corresponds to the length of a schedule with no idle time and is clearly a lower bound.

The algorithm we give constructs a schedule of length ω,

$$\omega = \max[\max_{1 \le j \le m} (X_j/B_j), X_n/B_m].$$

We assume that $m \le n$ since when there are fewer tasks than processors, only the fastest n processors need be used. From the above discussion it is clear that this value of ω is optimal. The expression for ω is derived from Liu and Liu [5] as well as Liu and Yang [6]. The algorithm below is a slight modification of the algorithm given by Muntz and Coffman [8, 9] for constructing preemptive schedules on m identical processors.

Definitions. A *chain* is a sequence of tasks $\mathscr{C} = (T_{i_1}, T_{i_2}, \ldots, T_{i_k})$ such that for all j, $1 \le j < k$, $T_{i_j} < T_{i_{j+1}}$. The chain $\mathscr{C}$ is said to *start* with task T_{i_1}. The *length* of $\mathscr{C}$ is given by $\sum_{j=1}^{k} \tau_{i_j}$, where τ_{i_j} is the service requirement of T_{i_j}, $1 \le j \le k$. The *level* of a task T is the maximum over the lengths of all chains starting with T □

The following algorithm executes tasks level by level. Tasks at the same level are executed at the same average rate. Since levels of tasks change as execution proceeds we will tie levels to time in a schedule. Let S be a schedule for a task system $(\mathscr{T}, <)$ $L_t(T)$, the *level of* T *at time* t with respect to schedule S, is the level of task T in the unexecuted portion of the task system.

Level algorithm. Given a task system $(\mathscr{T}, <)$, this algorithm first determines the rates at which tasks are to be executed. These rates are used to determine a preemptive

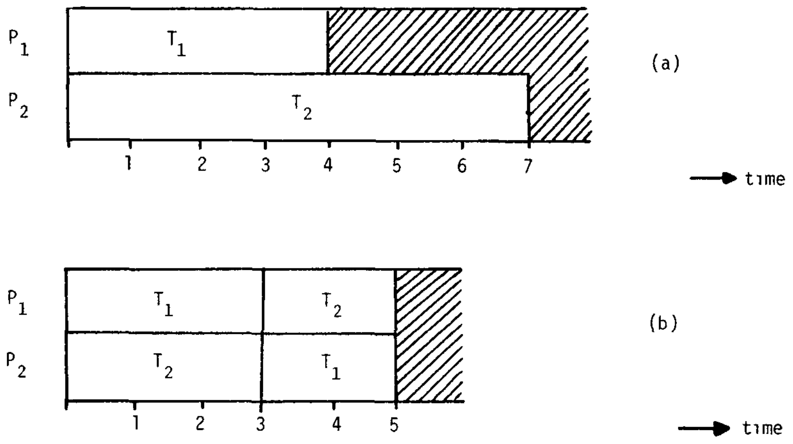


FIG 1 Processor P_1 is twice as fast as processor P_2. The service requirements of T_1 and T_2 are 8 and 7 units

schedule. Let s be the time when assignment of processors is made. Initially $s = 0$.

(1) Let j be the number of unassigned processors at time s, and let k be the number of ready tasks at the highest level at s. If $k \leq j$, assign the k tasks to be executed at the same rate on the fastest k processors. Otherwise assign the k tasks to execute at the same rate on the j processors. If there are any processors left, then consider the tasks at the next highest level that are ready for execution and so on. Continue such an assignment until a time t when one of the following occurs:

Event 1: A task is completed at t.

Event 2: There are two tasks T and T' such that $L_s(T) > L_s(T')$ but $L_t(T) = L_t(T')$. That is, T has "caught up" with T' at time t.

In either case set $s = t$ and reassign all processors to the unexecuted portion of the task system.

(2) In order to construct a preemptive schedule from the *shared* schedule from (1) above, assign the portion of the schedule between every pair of events as follows: If k tasks have been sharing j processors, $k \geq j$, then from (1) each of the k tasks will have the same service requirement in the interval. Divide the interval into k equal subintervals Assign the k tasks so that each task occurs in exactly j subintervals, each time on a different processor.

More precisely let the tasks be $V_1, \dots, V_k$ and let the subintervals be $1, 2, \dots, k$. Then assign task V_h, $1 \leq h \leq k$, onto processors $P_g, P_{g+1}, \dots, P_{g+j-1}$, in subintervals $(h + i) \bmod k$ for $0 \leq i \leq j - 1$, respectively, where $P_g, \dots, P_{g+j-1}$ are the processors in question. □

Figure 2 gives an example illustrating the above algorithm. Note that the shared schedule and the preemptive schedule constructed from it have the same length and the same tasks in each interval. Since we are only concerned with schedule lengths, we will refer to both the shared schedule and the constructed preemptive schedule as *level schedules*.

The implementation of the above algorithm is an interesting exercise. Note that events 1 and 2 can occur at most n times each. Step 2 dominates the time complexity since there may be $O(mn^2)$ preemptions in the preemptive schedule, so it takes $O(mn^2)$ time just to write down the schedule

THEOREM 1. *The level algorithm constructs a minimal length schedule for the set of independent tasks $\mathcal{T}$ with service requirements $\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n$ on the processing system $\mathcal{P} = (b_1, b_2, \dots, b_m)$, $m \leq n$. The schedule length is given by*

$$\omega = \max\left[\max_{1 \leq j \leq m} (X_j/B_j),\ X_n/B_m\right].$$

PROOF. If there is no idle time in the shared schedule constructed by the algorithm, then $\omega = X_n/B_m$, and the schedule is optimal. Therefore assume there is idle time in the shared schedule.

According to the level algorithm, when two tasks have the same remaining service requirement they will be executed at the same rate until they are finished. Therefore with all tasks independent, a task T will never be finished before a task with a lower service requirement than T.

Suppose P_{j+1} is the fastest processor on which there is idle time. Then $P_1, \dots, P_j$ must be executing tasks $T_1, \dots, T_j$ after P_{j+1} becomes idle, and no other tasks share processors with these tasks. Since all tasks are ready for execution at time 0, we conclude that $P_1, \dots, P_j$ have been executing $T_1, \dots, T_j$ from the beginning of the schedule. Since there is no idle time on $P_1, P_2, \dots, P_j$, tasks $T_1, T_2, \dots, T_j$ finish at the same time. It follows that the length of the schedule is X_j/B_j. Note that if any of the tasks $T_1, T_2, \dots, T_j$ is ever assigned to a slower processor than P_j, then the schedule becomes even longer.

It is easy to see that the shared schedule is transformed into a valid preemptive schedule. The algorithm therefore constructs an optimal schedule for $\mathcal{T}$ on $\mathcal{P}$. □

If the task system in question is a tree then the level algorithm does not construct

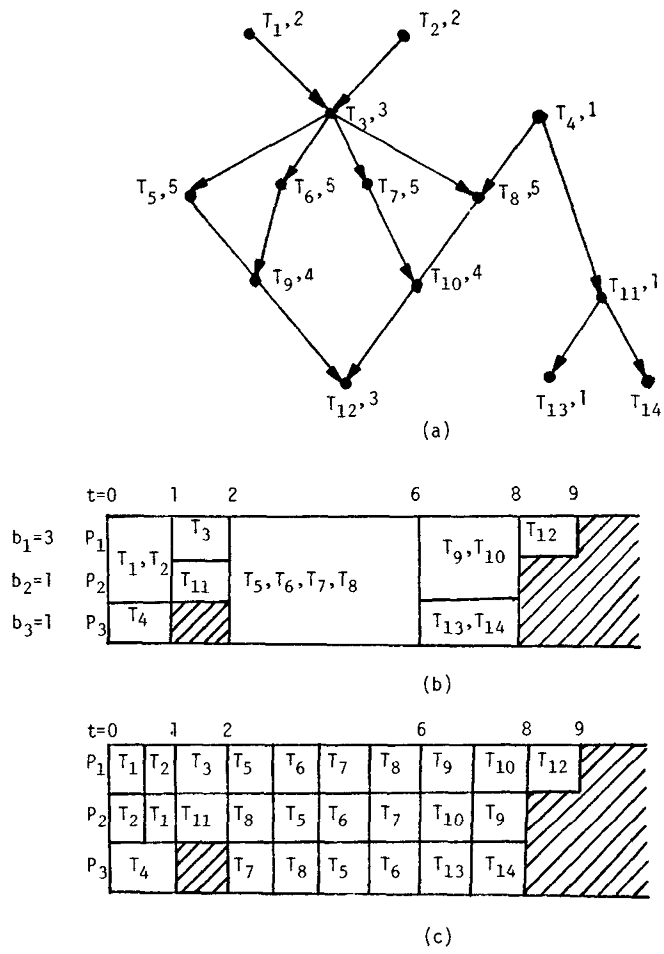


FIG 2 An example for the level algorithm. (a) a task system, (b) the shared schedule for (a), (c) the preemptive schedule transformed from (b)

optimal schedules on three or more processors. Consider a tree with tasks at three levels as in Figure 3, scheduled on a processing system with $b_1 = (\sqrt{m}) + 1$ and $b_i = 1$ for all i, $2 \le i \le m$. The root W has $\sqrt{m}$ predecessors $V_1, V_2, \ldots, V_{\sqrt{m}}$. $V_{\sqrt{m}}$ has m predecessors $U_1, U_2, \ldots, U_m$. In Figure 3, $m = 9$ so $\sqrt{(m)} = 3$. The service requirements are chosen so that the resultant schedules look like the ones in Figure 3.[1] As m increases the ratio of the lengths of the schedule constructed by the algorithm to those of the optimal schedule becomes 2 in the limit.

3. *Background Results*

In this section we review properties of level schedules that will be useful in the sequel. Level schedules will be divided into "segments" such that (a) within each segment, except while the last task in the segment is executing, at least two processors are busy and (b) in any valid schedule all tasks in one segment must be finished before any task in the next segment can be started. The technique used is similar to that used by Coffman and Graham [2] for nonpreemptive schedules. The material in this section follows immediately from Lam and Sethi [4], but has been included for completeness. The importance

[1] For arbitrary m, the service requirements $\tau(V)$ and $\tau(U)$ for the V and U tasks, respectively, should be chosen so that $\tau(V)/\tau(U) = [\sqrt{m}(2m + \sqrt{m} - 1)]/[(m - \sqrt{m} + 1)(\sqrt{m} - 1)]$

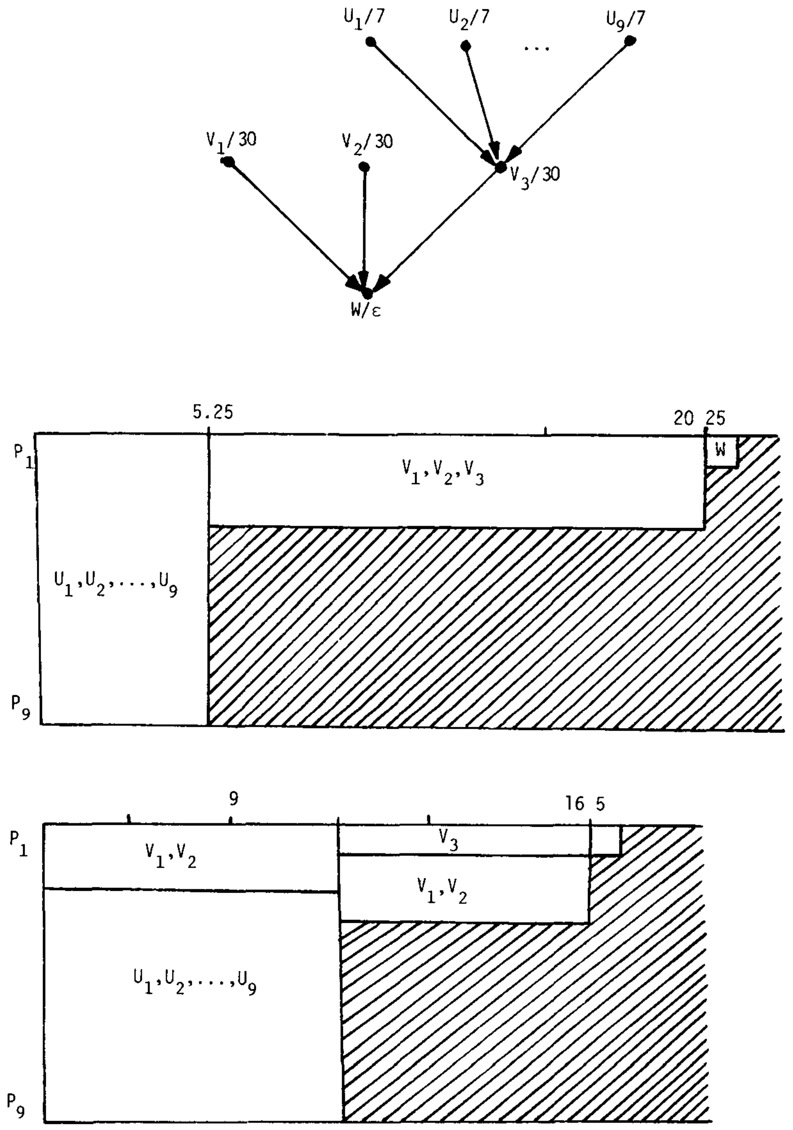


FIG 3 The level algorithm is not optimal for trees on three or more processors In this example $b_1 = 4$, $b_2 = b_3 \quad b_9 = 1$

of the results is that in proving upper bounds on the performance of the level algorithm in the next section, we can restrict attention to individual segments.

In level schedules a task may be split into several pieces each executed in a different interval of time. Given a level schedule S for an n-task system $(\mathcal{T}, <)$, let $U_{i1}, \ldots, U_{in_i}$ for some $n_i \geq 1$ be all the pieces of task T_i in S, where piece U_{ij} starts before U_{ij+1} in S for all j, $1 \leq j < n_i$. Also let $\tau_{i1}, \ldots, \tau_{in_i}$ be the execution requirements of the pieces $U_{i1}, \ldots,$ U_{in_i}, respectively. Then obviously we have $\sum_{j=1}^{n_i} \tau_{ij} = \tau_i$. We define the precedence relation $<'$ among the pieces by $U_{ij} <' U_{ij+1}$ for all $1 \leq j < n_i$, $1 \leq i \leq n$, and $U_{in_i} <' U_{k1}$ if and only if $T_i < T_k$ for $1 \leq i$, $k \leq n$. It is easy to see that each of these pieces is just like a task and S is a level schedule for the task system $(\mathcal{T}', <')$, where $\mathcal{T}' = \{U_{ij}, 1 \leq j \leq n_i,$ $1 \leq i \leq n\}$ and $<'$ is as defined above. Therefore from now on we shall refer to each of these pieces as a task as if S were constructed for the task system $(\mathcal{T}', <')$, formed by the pieces, and use the terms "piece" and "task" interchangeably.

According to step (1) of the level algorithm, if there are fewer tasks at the highest level than there are processors, then tasks at the next lower level are considered. In dividing a schedule into segments as in Figure 4, we shall remove the effect of tasks that are executed ahead of their level.

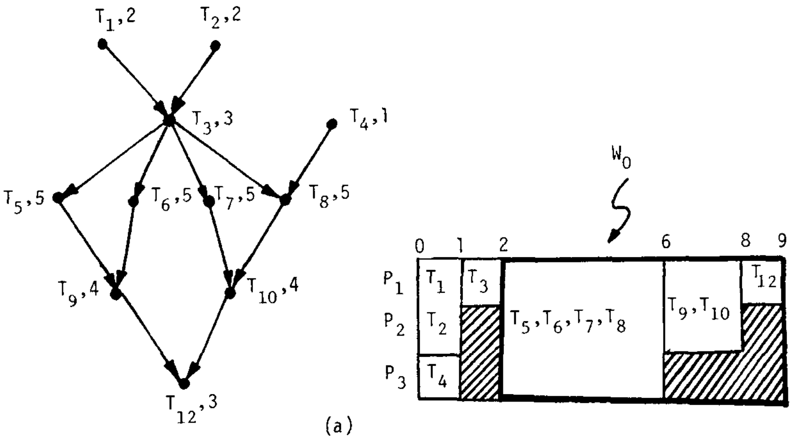
Definition. Let S be a level schedule of length ω, and let $0 = t_1 < t_2 < \cdots < t_k = \omega$ be *the sequence of times at which either event 1 or event 2 occurs A processor idle in some* interval is said to be executing an empty task $\varnothing$ in that interval. Define *segments* W_r, W_{r-1}, … , W_0 as follows:

(1) Set $j = 0$ If the last time interval has only one task executing, then call this task U_0, set $t = t_{k-1}$, and compute $L_t(U_0)$. Otherwise set $U_0 = \varnothing$, $t = t_k$, and $L_t(U_0) = 0$. U_0 is *in* segment W_0.

(2) If $t = 0$ then the entire schedule is divided, so stop. Otherwise let T_i be t for some $i > 1$. For h from i down to 2 do:

(a) Compute $L_{t_h}(T)$ for all tasks T in the interval (t_{h-1}, t_h), with respect to the unremoved tasks.

(b) Remove all T that have' $L_{t_h}(T) < L_t(U_j)$.



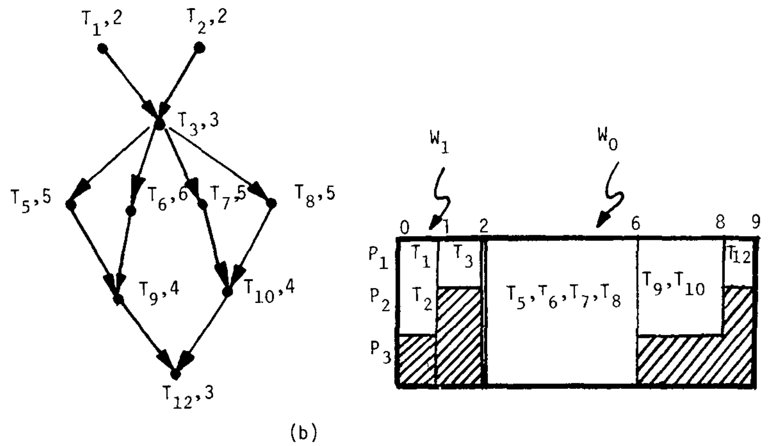


FIG 4 Reduced systems and reduced schedules Using the level schedule for the task system given in Figure 2, (a) gives the reduced task system and the reduced schedule after the removal of tasks in the definition of segment W_0, and (b) gives the reduced task system and the reduced schedule after the removal of tasks in the definition of segment W_1

(3) Find the first interval (t_l, t_{l+1}) before t in which only one processor is assigned a task. Define this task to be U_{j+1} and add the portion of the schedule between U_{j+1} and U_j to W_u. U_{j+1} is *in* W_{j+1} If no such interval (t_l, t_{l+1}) is found, then add the entire portion of the schedule between times 0 and t to W_j, and stop.

(4) Set $j = j + 1$ and $t = t_l$ and go to (2). □

Note that segments are defined from right to left, with W_0 being the rightmost segment. After segments are defined for a level schedule S, we have a *reduced schedule* S' and a *reduced task system.* Since a task is removed from the task system only when it is removed from S', and no edges have been added, it is obvious that S' is a valid schedule for the reduced task system. Let $0 = t_1 < t_2 < \cdots < t_k$ be the sequence of times when event 1 or 2 occurs in S. The reader can verify that, for $1 \le h < k$ and for all tasks T in the reduced task system, $L_{t_h}(T)$ in S' is exactly the same as $L_{t_h}(T)$ in S, thus proving that S' is a level schedule for the reduced task system.

LEMMA 1. *Given a level schedule S of length ω divided into segments $W_r, W_{r-1}, \ldots, W_0$, for $r \ge 0$, it must be true that:*

(*a*) *segments $W_r, \ldots, W_0$ taken together form a level schedule of length ω for the reduced task system,*

(*b*) *in each segment W_j at least two processors are busy except, possibly, while the last task in W_j is being executed, and*

(*c*) *if U_{j+1} is the last task in segment W_{j+1}, then for all other tasks $T \in W_{j+1}$ and all tasks $T' \in W_j$, $T < U_{j+1} < T'$.*

PROOF. From the above discussion, we can see that segments $W_r, \ldots, W_0$ together form a level schedule for the reduced task system. It remains to show that the total length is ω. Consider a task T in interval (t_{h-1}, t_h) for some h that is deleted in the definition of segment W_j. From step (2) of the definition it follows that $L_{t_h}(T) < L_t(U_j)$, where t is the time at which U_j begins execution is S. There must exist a task V executed concurrently with T such that V is a predecessor of U_j. It follows that $L_{t_{h-1}}(V) > L_{t_{h-1}}(T)$, which, according to the level algorithm, implies that V is running shared with other $k - 1$ tasks on k processors for some $k \ge 1$. Thus deleting task T cannot decrease the length of the segments, thereby proving part (a). Part (b) of the lemma follows immediately from the definition of segments.

For part (c) consider tasks T and V in segments W_{j+1} and W_j, respectively. Recall that W_{j+1} occurs to the left of W_j. By definition W_{j+1} ends with task U_{j+1}, and while U_{j+1} is being executed all other processors are idle in the reduced schedule. Since the reduced schedule is a level schedule, it must be true that $U_{j+1} < V$, for otherwise V would have been executed concurrently with U_{j+1}.

To show that $T < U_{j+1}$ for all $T \ne U_{j+1}$ in W_{j+1}, we assume the contrary. That is, there exists a task T in W_{j+1} such that $T < U_{j+1}$ is false. Choose T so that T is the last such task to finish in W_{j+1}. This choice ensures that T has no successors in W_{j+1}. That is, T can only have immediate successors in W_j or some later segment, which implies that $L_{t_h}(T) < L_t(U_{j+1})$, where (t_{h-1}, t_h) is the interval T is in and t is the time at which U_{j+1} starts execution. However, $L_{t_h}(T) < L_t(U_{j+1})$ contradicts the fact that T is not deleted. Consequently we have $T < U_{j+1} < V$ for all $T \ne U_{j+1}$ in W_{j+1} and all V in W_j. □

THEOREM 2. *Level schedules are of minimal length on two-processor systems.*

PROOF. From Lemma 1 it follows that all tasks in a segment W_{j+1} must be finished before any tasks in the following segment W_j are started. With only two processors on hand, within a segment W_j the only idle time occurs while the last task in W_j is being executed. From Lemma 1 it also follows that tasks not in the segments do not affect the length of the schedule, so no task in any segment can be executed during this idle period. The schedule must therefore be optimal. □

4. *Level Schedules on m Processors*

When processor speeds are allowed to be different, Figure 3 shows that, even for a tree, a level schedule on three or more processors need not be optimal. We will first determine an upper bound for the worst-case performance of the level algorithm in terms of the speeds of the processors. From this bound we will show that $\sqrt{(1.5m)}$ is an upper bound, independent of the speeds, over all systems of m processors. We will then give an example of a system with one fast processor for which the ratio of the lengths of a level schedule to those of an optimal schedule approaches $(\sqrt{m})/2\sqrt{2}$

Let ω be the length of a level schedule S for a task system $(\mathscr{T}, <)$, and let ω_0 be the length of an optimal schedule for $(\mathscr{T}, <)$. Let X be the total execution requirement of tasks in $(\mathscr{T}, <)$, and let I be the total idle processor capacity in S. From the definition of X and I, it follows that $\omega = (X + I)/B_m$. Or, $B_m\omega = X + I$. A useful approach in estimating ω/ω_0 is to determine X and I in terms of ω_0 and then substitute the estimates into the equation $B_m\omega = X + I$. Since the optimal schedule can do no better than to have no idle time, $\omega_0 \geq X/B_m$, which yields $X \leq \omega_0 B_m$. The next lemma determines lower bounds on ω_0 in terms of I and the processing system.

Consider the processing system $\mathscr{P}$. If i processors are busy, then the processing rate of the busy processors is B_i and the idle rate is $B_m - B_i$. For identical processors, knowing i we can immediately determine the ratio of the used to the idle rate. With processors of different speeds this ratio depends on the given processing system.

Let Y_i be the used and I_i the idle capacity in the schedule when exactly i processors are busy Lemma 1 shows that, without loss of generality, we can concentrate on parts of the level schedule that have at least two processors busy at all times. Then clearly $X \geq \sum_{i=2}^{m} Y_i$ and $I = \sum_{i=2}^{m-2} I_i$, and $Y_i/I_i = B_i/(B_m - B_i)$ for $2 \leq i \leq m - 1$.

Since $B_2 < B_3 < \cdots < B_m$, we have $Y_i/B_i \geq Y_i/B_2$ for all i, $2 \geq i \geq m$, and $U_i/(B_m - B_i) \leq I_i/(B_m - B_2)$ for $2 \leq i \leq m - 1$. Consequently $X/B_2 \geq I/(B_m - B_2)$. Then, letting $\beta = B_2/(B_m - B_2)$, we get $X \geq I\beta$. Since $\omega_0 \geq X/B_m$, we have established a lower bound on ω_0 in terms of I, namely, $\omega_0 \geq I\beta/B_m$.

Intuitively β represents the worst possible balance between the used and idle processing rates. If the total idle capacity in a schedule is I, then the used capacity will be at least $I\beta$. Since β is minimal, $I\beta$ is as small as can be, thus delivering a lower bound on the used capacity.

As with independent tasks in Section 2, another lower bound on the length of an optimal schedule is provided by the time it takes to execute the longest chain of tasks on the fastest processor. If i tasks are being executed on the fastest i processors, the highest level task is being executed at a rate of at least B_i/i. Define $\alpha_i = (B_i/i)/(B_m - B_i)$, $1 < i < m$. Let α be the least element of the set $\{\alpha_2, \dots, \alpha_{m-1}\}$. We will see that if a level schedule has idle capacity I, then there must be a chain of length at least $I\alpha$.

LEMMA 2 *Consider a level schedule S of length ω in which at least two processors are busy at all times. Let I be the total idle capacity in S. If ω_0 is the length of an optimal schedule, then*

(*a*) $\omega_0 \geq I\beta/B_m$, *and*

(*b*) $\omega_0 \geq I\alpha/B_1$.

PROOF. In the level schedule S, let exactly i processors be executing for l_i units. Since at least two processors are busy at all times, $\omega = \sum_{i=2}^{m} l_i$. By the same token, $I = \sum_{i=2}^{m} l_i(B_m - B_i)$. So for $2 \leq k \leq m - 1$, we have

$$l_k = I/(B_m - B_k) - \sum_{i=2, i\neq k}^{m} [l_i(B_m - B_i)/(B_m - B_k)]. \tag{1}$$

Consider a time interval in which i processors are executing tasks, $i < m$. Since at least one processor is idle, exactly i tasks are being executed. If these i tasks are executed at the same rate for l units, then each task will be of length lB_i/i in the l unit interval If the tasks are not executed at the same rate, then the longest task will be of length at least lB_i/i. Thus in schedule S there must be a chain of length at least $\sum_{i=2}^{m-1} l_i B_i/i$. Since the optimal schedule can do no better than to execute this chain on the fastest processor P_1,

$$\omega_0 \geq \sum_{i=2}^{m-1} l_i B_i / i B_1. \tag{2}$$

Let h be such that $\alpha_h = \alpha$. Substituting the l_h from eq. (1) into eq. (2), we get

$$B_1\omega_0 \geq I\alpha_h + \sum_{i=2, i\neq h}^{m-1} l_i(B_m - B_i)(\alpha_i - \alpha_h).$$

Since $\alpha_i - \alpha_h$ is nonnegative, by choice of $\alpha_h = \alpha$, we get $B_1\omega_0 \geq I\alpha$. So $\omega_0 \geq I\alpha/B_1$, thus proving part (b). □

Lemma 2 provides two ways of measuring ω_0. The measures immediately lead to the following result.

THEOREM 3. *Let ω be the length of a level schedule S for a task system $(\mathcal{T}, <)$. Let ω_0 be the length of an optimal schedule for $(\mathcal{T}, <)$. Then*

$$\omega/\omega_0 \leq 1 + min(1/\beta, B_1/\alpha B_m).$$

PROOF. Let X be the total execution requirement of tasks in $(\mathcal{T}, <)$, and let I be the idle capacity in S. Then $B_m\omega = X + I$. For each individual segment the bound follows by substituting for I in terms of ω_0 from Lemma 2 Since the segments must be processed sequentially from Lemma 1, the theorem follows. □

While Theorem 3 permits us to bound ω/ω_0 in terms of the processing system, we would like a bound that is easier to relate to.

THEOREM 4 *Let ω be the length of a level schedule S for a task system $(\mathcal{T}, <)$. Let ω_0 be the length of an optimal schedule for $(\mathcal{T}, <)$. Then $\omega/\omega_0 \leq \sqrt{(1.5m)}$.*

PROOF. From Theorem 3 we know that $\omega/\omega_0 \leq 1 + 1/\beta$ By definition, $\beta = B_2/(B_m - B_2)$. We therefore get $\omega/\omega_0 \leq B_m/B_2$, which leads to $B_m \geq B_2\omega/\omega_0$.

With X and I representing the execution requirement and idle capacity in S, respectively, we must have $B_m\omega = X + I$. Replacing B_m with a lower bound on its value, we get $B_m\omega = X + I \geq B_2\omega^2/\omega_0$. Rewriting, we get

$$\omega^2/\omega_0 \leq (X + I)/B_2. \tag{3}$$

We know that $X \leq B_m\omega_0$ and, from Lemma 2, $I \leq B_1\omega_0/\alpha$ Substituting into (3), we get

$$\omega^2/\omega_0^2 \leq B_m/B_2 + B_1/\alpha B_2. \tag{4}$$

Since $b_1 \geq \cdots \geq b_m$, we must have $B_2/2 \geq B_m/m$, and therefore $B_m/B_2 \leq m/2$. Since $B_1 < B_2$, $B_1/\alpha B_2 \leq 1/\alpha$. Substituting from the definition of α into (4), we get

$$\omega^2/\omega_0^2 \leq m/2 + \max_{2\leq h\leq m-1} (B_m - B_h)/(B_h/h).$$

B_h/h is the average speed of the fastest h processors Thus $B_m/(B_h/h) \leq m$. We therefore get

$$\omega^2/\omega_0^2 \leq 3m/2,$$

which gives us $\omega/\omega_0 \leq \sqrt{(1.5m)}$. □

Seeing the methods used to obtain it, we do not expect $\sqrt{(1.5m)}$ to be a best bound. It turns out, however, that $(\sqrt{m})/2\sqrt{2}$ can be approached arbitrarily closely, as in Figure 5. Thus improvements in the bound can only be by a constant factor

5. *Conclusions*

In this paper we have analyzed the basic algorithm known for constructing optimal preemptive schedules. We have determined bounds on its performance as a heuristic of

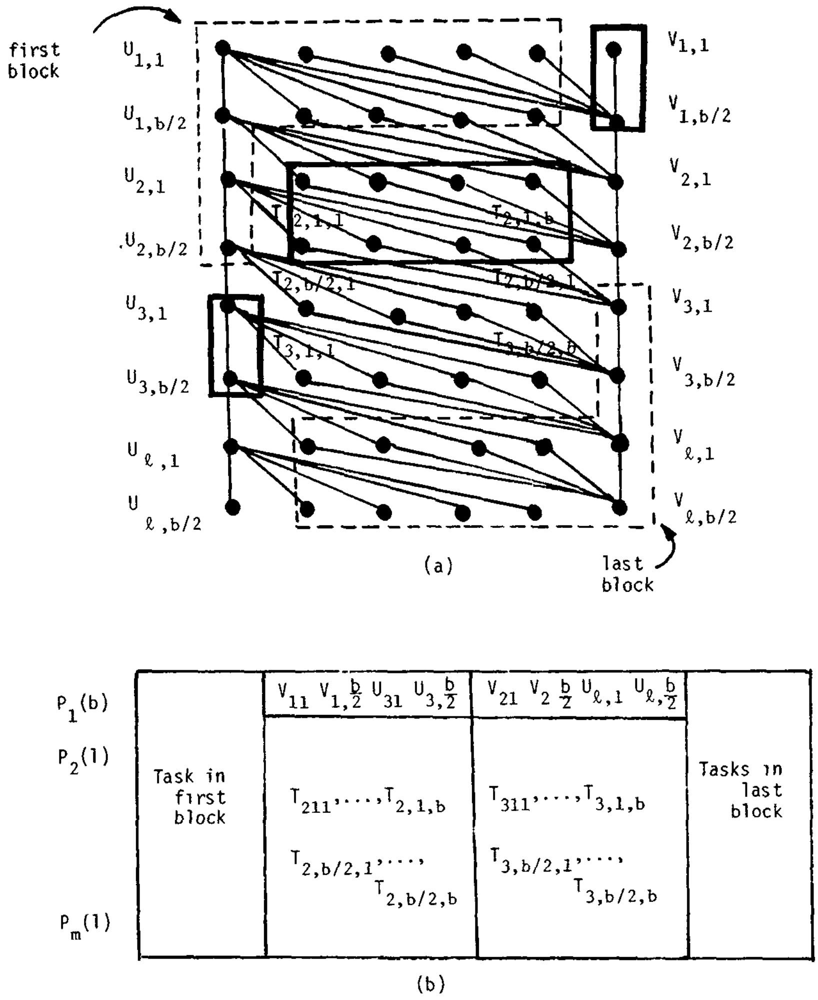


FIG 5 (a) A task system and (b) an optimal schedule for the task system that yields $\omega/\omega_0 \rightarrow (\sqrt{m})/2\sqrt{2}$ as $m \rightarrow \infty$ and $l \rightarrow \infty$ The example has been drawn for $m = 9$ and $l = 4$. $b = \sqrt{(2(m-1))}$, $\tau_i = 1$ for all i There are $b + 2$ tasks at each level in the task system

an m-processor system. The interesting question is whether other reasonable heuristics can be found that fare substantially better than the level algorithm. Heuristics for the nonpreemptive case are given by Gonzalez et al. [3].

REFERENCES

1. COFFMAN, E G JR , Ed *Computer and Jobshop Scheduling Theory* Wiley, New York, 1976
2 COFFMAN, E G JR , AND GRAHAM, R.L Optimal scheduling for two processor systems *Acta Informatica 1*, 3 (1972), 200-213
3 GONZALEZ, T , IBARRA, O , AND SAHNI, S. Bounds for LPT schedules on uniform processors Comptr Sci Tech Rep No 75-1, U of Minnesota, Minneapolis, Minn , 1974
4 LAM, S , AND SETHI, R Worst case analysis of two scheduling algorithms (To appear in *SIAM J Comptg* , 1976)
5 LIU, J.W S , AND LIU, C.L Performance analysis of heterogeneous multiprocessor computing systems In *Computer Architectures and Networks*, E Gelenbe and R. Mahl, Eds , North-Holland Pub Co , Amsterdam, 1974, pp 27-45
6 LIU, J W S , AND YANG, A Optimal scheduling of independent tasks on heterogeneous computing systems Proc ACM 1974 Annual Conf., pp 38-45

7 McNaughton, R Scheduling with deadlines and loss functions *Manage Sci 12,* 1 (Oct 1959), 1-12
8 Muntz, R R, and Coffman, E G Jr Optimal preemptive scheduling on two-processor systems *IEEE Trans Comptrs C-18,* 11 (Nov 1969), 1014-1020
9 Muntz, R R, and Coffman, E G Jr. Preemptive scheduling of real time tasks on multiprocessor systems *J ACM 17,* 2 (April 1970), 324-338
10 Rothkopf, M H Scheduling independent tasks on parallel processors *Manage Sci 12,* 5 (Jan 1966), 437-447

RECEIVED FEBRUARY 1976, REVISED JULY 1976