Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems

R. R. MUNTZ* AND E. G. COFFMAN, JR.

Princeton University, Princeton, New Jersey

ABSTRACT. The use of multiprocessor systems consisting of identical and autonomous processors is a promising approach to the practical solution of problems arising in real-time applications and large compute-bound problems in general. However, finding techniques for obtaining efficient solutions to the related multiprocessor scheduling problems is a little-understood problem. The authors study the problem of scheduling a set of tasks whose operational precedence structure is representable as an acyclic directed graph. In scheduling tasks it is assumed that preemptions are allowed. The major results consist of the statement and proof of an efficient algorithm for finding the minimal-length preemptive schedule for tree-structured computations.

KEY WORDS AND PHRASES: multiprocessor scheduling, scheduling computation graphs, computation graphs, multiprocessing, scheduling real-time tasks, preemptive scheduling, sequencing tasks

CR CATEGORIES: 3.80, 4.32

1. *Introduction*

The basic characteristic of a multiprocessing system is the existence of several processors which can operate independently. The efficient utilization of such a system can be very effective in decreasing the computation times of many programs. This is particularly important for real-time applications but also for lengthy computations [1], such as weather forecasting [2], when the results are needed more quickly than they can be provided by a single processor. In this paper we report results from some investigations into the problem of making optimum use of a multiprocessing system in reducing the computation times of programs which are available for analysis and scheduling prior to execution.

We consider systems in which all processors are identical and assume that the computations submitted to the system have been specified as a set of tasks and a partial ordering of these tasks. The interpretation of the partial ordering is that if n_i and n_j are tasks and $n_i > n_j$ (read n_i precedes n_j or n_j follows n_i) then n_i must be completed before n_j is started. Accordingly, with each computation we associate a graph, G, whose nodes correspond to the set of tasks. There is a directed arc from the node representing task n_i to the node representing n_j if and only if $n_i > n_j$ and there is no task n_l such that $n_i > n_l > n_j$. A *weight* is associated with each node of the graph and is a function of the execution time of the corresponding task. Several possible interpretations of this weight are discussed shortly. Thus, in graph theoretic terminology our computation graphs are acyclic, weighted, directed graphs satisfying

* Present address: Computer Science Department, University of California, Los Angeles, California

the (nontransitivity) constraint just given. From this point on we use the terms "graph," "job," and "computation" as synonyms. Also, we use the terms "node" and "task" interchangeably.

Informally, a schedule or assignment for k processors and a given computation is a description of the work done by each processor as a function of time. Of course the schedule must not violate the given precedence relations and we are not permitted to assign more than one machine to a task at any time. Also, the total processor time devoted to a task must equal the task weight. The simplest way of specifying a schedule is to use a Gantt chart [3], which consists of a time axis for each processor with intervals marked off and labeled with the name of the task being computed. We use the symbol ϕ to represent an idle period.

In Figure 1(a) we have shown a simple graph, G, and in Figure 1(b) a schedule for G with $k = 2$ processors. The task weights are shown next to the node representing the task.

The computational model just described has been studied by others [4–6]. More sophisticated models describing program behavior would include, for example, conditional branching and cycles. For these more general models only heuristic scheduling techniques have proven fruitful [2, 7, 8]. The restriction to partial orderings that we have assumed may be reasonable if the tasks represent large functional blocks so that the graph is a "coarse" description of the computation. We also note that any particular execution of a computation containing conditional branches and cycles is representable as a partially ordered set of tasks. Efficient solutions to optimally scheduling partially ordered tasks may be useful in the study of heuristics for the more general models; e.g. it would be possible to compare heuristics against known minimal solutions for particular runs.

The weight associated with a task may have several interpretations, although it is always treated as a deterministic execution time in the construction of schedules.

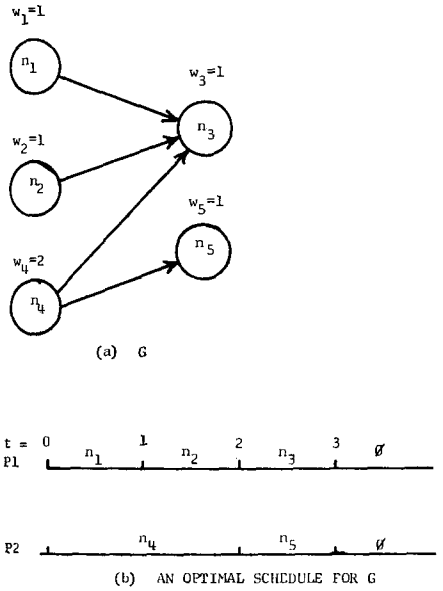

FIG. 1. An example graph and schedule

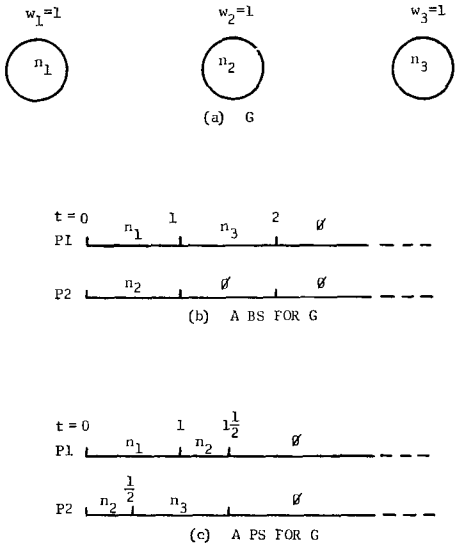

FIG. 2. Illustration of the PS advantage

The interpretation of the task weight then determines the meaning of the schedule constructed. If the task weights represent the maximum execution times of the tasks, then, with certain precautions [6], the schedule length may be taken as the maximum execution time of the computation. Therefore, minimizing the schedule length corresponds to minimizing the maximum execution time of the computation. This interpretation is particularly applicable to "hard" real-time problems with fixed deadlines.

A second interpretation is that the task weight is the mean value of the task execution time considered as a random variable. A schedule such as that shown in Figure 1(b) may be interpreted as defining an *assignment* of tasks to processors (e.g. tasks n_1, n_2, and n_3 are assigned to P1) and a *sequence* of tasks for each processor (e.g. on P1, n_1 precedes n_2 precedes n_3). Then if a schedule is constructed and used to define the assignments and sequences as described above the schedule length is an (optimistic) estimate of the mean length of the execution time [2]. Constructing a minimal-length schedule is then a simple heuristic procedure for minimizing the mean computation time of the job.

We are mainly concerned in this paper with the problem of constructing minimal-length schedules. In most investigations of this problem it is assumed that once a processor is assigned to a task it must work continuously on this task until it has been completed. This we term the Basic Scheduling (BS) discipline. Clearly, the problem of finding an optimal Basic Schedule for any given graph is effectively solvable by exhaustion. However, an efficient solution is known only when the graph has a rooted tree structure and all nodes are equally weighted [9].

We shall consider the effect of replacing the BS condition to allow processors to be interrupted before a task is completed and reassigned to a new task; this is the Preemptive Scheduling (PS) discipline. The potential benefits derivable from preemptions can be seen by the example shown in Figure 2. The schedule shown in Figure 2(b) is a minimal-length schedule for G with $k = 2$ machines using the BS discipline. In Figure 2(c) we compute G in 1.5 time units with one preemption.

In constructing the PS in Figure 2 we have assumed that the preemption itself did not require any time. Note, however, that as long as the cost of the preemption is less than $\frac{1}{2}$ when n_2 is saved and $\frac{1}{2}$ when n_2 is restored, an optimal PS for G will be less than 2 units in length. So we see that even with relatively large costs for preemptions there are jobs which can be completed in less time when preemptions are allowed. From existing and proposed multiprogramming/multiprocessing systems there is reason to believe that the costs of preemptions may be quite nominal in some applications. Two contributing factors are (1) the existence of large (main) memories out of which programs are executed and (2) the latency of transfers between main and auxiliary storage arising from the overlapped execution and input/output obtained with efficient storage management procedures. In any case, to obtain the theoretical results presented later we will continue to make the assumption that preemption costs are negligible. If we do not make this assumption then the task weights (including the cost of preemptions) become schedule dependent and this class of problems is much more difficult to solve in general.

There is another variation of the BS discipline that we want to introduce. We consider that the k processors in a system comprise a certain amount of computing capability rather than being discrete units. We assume further that this computing capability can be assigned to tasks in any amount between 0 and the equivalent

of one processor. If we assign α, $0 < \alpha \leq 1$ computing capability to a task then we assume that the computation time of the task is increased by a factor of $1/\alpha$. For example, if we assign $\frac{1}{2}$ of a machine to task n_i, which has weight w_i, then it will take $2w_i$ units of time to complete n_i. We allow the amount of computing capability assigned to a task to change before the task is completed (we include the case where the task is not worked on at all for some interval). This is the General Scheduling (GS) discipline. It should be clear that in allowing the value of α for a given node to change before the completion of the node we have effectively introduced the preemptive capability into the GS discipline. In particular, a PS is also a GS.

We do not try to justify the GS discipline in the sense of finding practical situations to which it corresponds. However, the GS discipline is extremely useful in studying preemptive scheduling techniques in Section 3.

We pause now to briefly outline the remainder of the paper. Our first concern will be to prove the effective equivalence of the PS and GS disciplines. Following this we exhibit an algorithm for constructing a minimal-length GS(PS) for any computation whose graph is a rooted tree, i.e. each node has one immediate successor except for a unique *root* node which has no successors. The proof that the algorithm produces a minimal-length schedule constitutes the remainder of the paper.

2. *Equivalence of the GS and PS Disciplines*

We need the following terminology. If G is a computation graph and D_1 is a scheduling discipline, then the *minimum* computation time of G with k processors and using D_1 is denoted $CT_{D_1}(G, k)$. We proceed now to verify the effective equivalence of the GS and PS disciplines in the following theorem.

THEOREM 1. *For all G and k, $CT_{GS}(G, k) = CT_{PS}(G, k)$.*

Note that this theorem implies that if preemptions are permitted then the "processor-sharing" capability is not needed for an optimal schedule.

PROOF. Let G be an arbitrary graph with node weights $\{w_i\}$ and let the number of available processors be k. Since a PS is also a GS it follows immediately that $CT_{GS}(G, k) \leq CT_{PS}(G, k)$.

Let S be any GS for G with k processors. We claim that it is possible to construct from S a PS for G with k processors, call it S', which is no longer than S. This claim implies $CT_{PS}(G, k) \leq CT_{GS}(G, k)$. From this and the previous observation the theorem follows easily. The claim is verified as follows.

Consider the sequence of times $(t_1, t_2, \cdots, t_m)$, at which nodes of G are completed in S, where $t_i < t_j$ if $i < j$. The intervals $(0, t_1)$, (t_1, t_2), $\cdots$, (t_{m-1}, t_m) have the property that any two nodes, n_i, n_j, which are assigned processor time in the same interval are *independent* (i.e. $n_i \not> n_j$ and $n_j \not> n_i$). For if this is not true then one of the tasks must complete in the interval, and this contradicts the definition of the sequence $\{t_i\}$.

Let node n_j be assigned processor time in (t_i, t_{i+1}). If n_j is assigned over an interval of length l_j and is computed at a rate α_j, then it would take one processor $\alpha_j l_j$ units of time to compute the same amount of n_j. If we sum $\alpha_j l_j$ over all intervals in which n_j is assigned in (t_i, t_{i+1}), we get the amount of time, Δ_j, it would take one processor to do the total amount of work on n_j in (t_i, t_{i+1}). Since in any interval $\alpha_j \leq 1$ it follows that

$$\Delta_j \leq (t_{i+1} - t_i). \tag{1}$$

Over any finite interval the amount of computing capability assigned to tasks is less than or equal to k machines so that

$$\sum_{n_j} \Delta_j \leq k(t_{i+1} - t_i). \tag{2}$$

In the interval (t_i, t_{i+1}) the processor assignments can be made such that the same fractional part of each node is computed but without processor sharing. For convenience we assume the processors are labeled P1, $\cdots$, Pk. Assign processor P1 to node n_{i_1}, from $t = t_i$ to $t = t_i + \Delta_{i_1}$. From (1) we know that $t_i + \Delta_{i_1} \leq t_{i+1}$; so we have not exceeded the interval length. If $t_i + \Delta_{i_1} < t_{i+1}$ then let P1 start computing n_{i_2}. If we reach the end of the interval before completing work on n_{i_2} we assign P2 to n_{i_2} starting at $t = t_i$. We continue assigning the nodes to the processors in order and each time we reach the end of the interval we proceed to the next processor starting at the beginning of the interval.

From (1) we know that we will not violate the condition that only one processor can be assigned to a task over any finite interval. From (2) we know that we will not run out of processor time in the interval before completing the construction. From the definition of the intervals we know that we will not violate any precedence relations. Thus, it is clear that if we carry out this procedure for each interval we will have the required PS. Q.E.D.

We shall have occasion to make use of the theoretical value of this theorem; that is, we shall make use of the ability to speak about PS's and GS's interchangeably.

Example. One possible GS for the graph in Figure 3(a) is shown in Figure 3(b) as a modified Gantt chart. The sequence of times as defined in the theorem is $(1, 3\frac{1}{3}, 4\frac{1}{2}, 5\frac{1}{2})$. Intervals $(1, 3\frac{1}{3})$ and $(3\frac{1}{3}, 4\frac{1}{2})$ must be altered to transform this schedule into a PS. For the interval $(1, 3\frac{1}{3})$ we have $\Delta_{n_4} = 2$, $\Delta_{n_3} = \Delta_{n_2} = 1\frac{1}{3}$. The same amount of processor time is assigned to each of these nodes during $(1, 3\frac{1}{3})$ in the PS shown in Figure 3(c). (We have assigned the processor time to the nodes in the order n_2, n_3, n_4. The theorem leaves this ordering arbitrary.) A similar transformation was performed on the interval $(3\frac{1}{3}, 4\frac{1}{2})$.

3. *Minimal-Length GS's (PS's) for Computation Trees*

Algorithms for finding minimal-length PS's are as rare as results for the BS discipline. One special case for which a solution has been known for some time [10] occurs when the computation is a set of independent tasks. We state this result in the following theorem.

THEOREM. *Let G be a graph which consists of a set of independent nodes with weights $\{w_1, \cdots, w_n\}$, and let k be the number of available processors. Then*

$$CT_{PS}(G, k) = max\left\{w_1, \cdots, w_n, \sum_{i=1}^{n} w_i/k\right\}.$$

It is clear that this computation time cannot be improved upon since the schedule must be at least as long as the largest task and cannot be more efficient than to keep all the processors continuously busy.

In this section we present an algorithm for constructing an optimal GS(PS) for any number of processors when the computation graph is a rooted tree and the

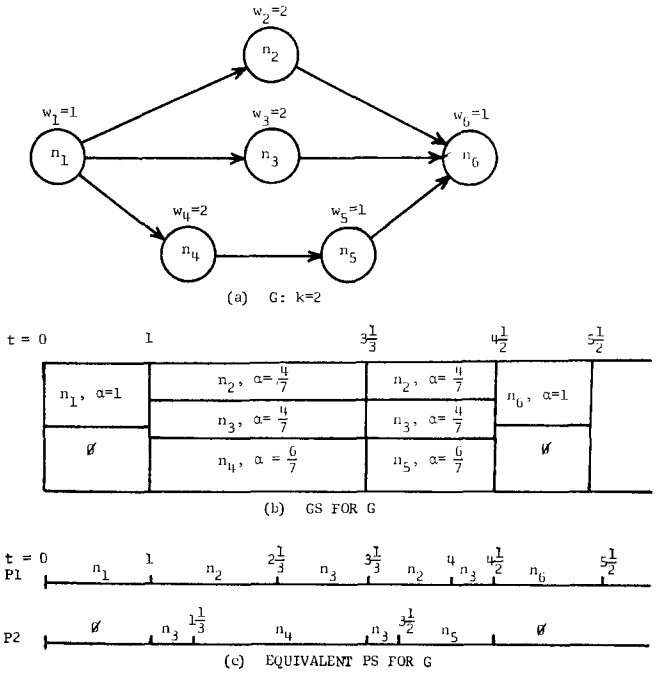

FIG. 3. GS to PS transformation

node weights are *mutually commensurable* (i.e. there exists a real number, w, such that all node weights are integer multiples of w). This last requirement is not very restrictive since we can still approximate a given set of node weights arbitrarily closely.

Before we proceed it will be helpful to introduce the following terminology connected with trees. Consider a tree, T, and two nodes n_i, $n_j \in N(T)$ such[1] that $n_i < n_j$. The *distance* between n_i and n_j is the sum of the node weights for each node (including both n_i and n_j) in the (unique) path connecting n_i and n_j. If n_i is the root of T then this distance is called the *height* of the node n_j. Finally, at any point in the execution of a schedule for a tree the executable nodes are the initial nodes of the tree remaining to be processed. These are the (independent) nodes that may be processed without violating any precedence constraints.

Let T be a tree with mutually commensurable node weights and let k be the number of available processors. Then an algorithm for constructing an optimal GS for T is as follows.

Algorithm. Assign one processor ($\alpha = 1$) to each of the k nodes farthest from the root of T. If there is a tie among b nodes (because they are at the same level) for the last a machines then assign a/b of a processor to each of these b nodes. Each time either one of the two situations described below occurs we reassign the processors to the tree that remains to be computed according to this rule. The two situations are:

Event 1. A node of T is completed.

[1] $N(T)$ is the set of nodes in T.

Event 2. We reach a point where, if we continued the present assignment, we would be computing some nodes at a faster rate than we are computing other nodes which are farther from the root of T. This is further clarified by an example given below.

For convenience we refer to a GS constructed according to the above rules as an *M-schedule*. The computation time of a tree, T with k processors when using this algorithm, is denoted by $CT_M(T, k)$.

Example. Consider the computation tree shown in Figure 4(a) and $k = 3$ processors. In Figure 4(b) we have shown a modified Gantt chart illustrating the M-schedule for T. Observe that at $t = \frac{1}{2}$ there is an occurrence of event 2 because the heights of nodes n_1, n_2, and n_3 have been reduced (prior to completion) to the height of n_5 (viz. 10). At this point the three processors are shared equally by nodes n_1, n_2, n_3, and n_5. The advantages of sharing and preemption are immediately clear from a comparison of processor idle time in Figures 4(b) and (c).

The proof that the above algorithm is optimal is based on the notion of "task splitting." Let G be a graph and let $\{w_1, \cdots, w_n\}$ be the weights of the nodes of G. Assume the w_i are mutually commensurable and let w be the largest real number such that all w_i are integer multiples of w. We now define a graph $G_{w/n}$, where n is any integer, which is derived from G. Informally, we take each node, n_i, of G with weight w_i and replace it by a series connection of nodes of weight w/n so that the total weight of these nodes equals w_i. We require that all edges that were incident into n_i are incident into the first node in the series connection. Similarly, all edges that were incident out of n_i now leave the last node in the series connection.

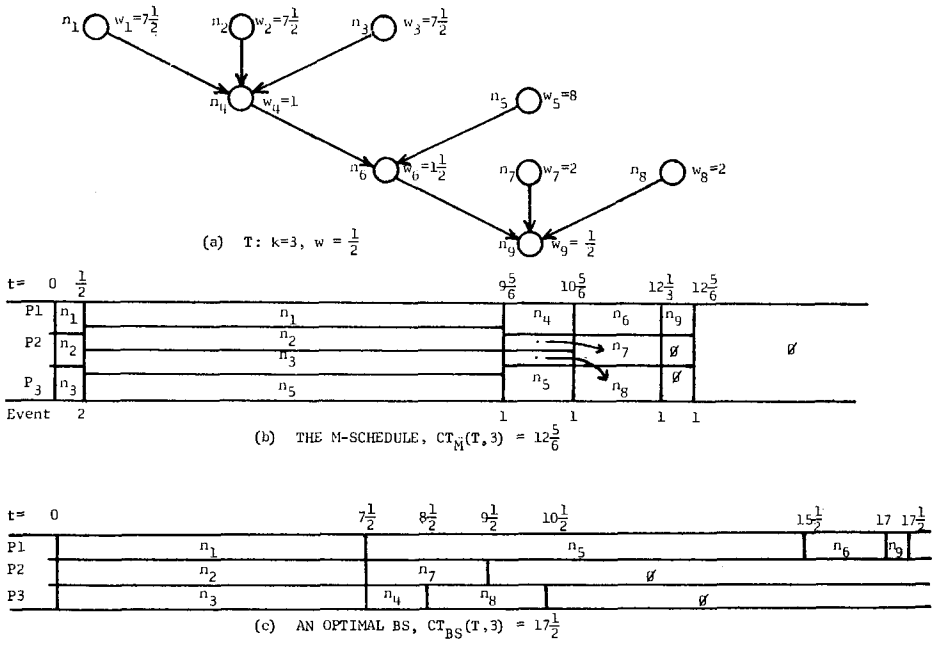

FIG. 4. Example of an M-schedule and optimal BS

Example. Consider the graph shown in Figure 5(a) for which $w = 1$. If we choose $n = 2$ we have the graph $G_{\frac{1}{2}}$ shown in Figure 5(b).

For completeness we give the following formal definition of $G_{w/n}$.

Definition. For any integer n and a given graph G let $G_{w/n}$ be a graph for which there exists a homomorphic mapping, H, of $G_{w/n}$ onto G where H and $G_{w/n}$ satisfy the following conditions.

(a) If $n_{i_1}, \cdots, n_{i_l}$ are nodes of $G_{w/n}$ that are mapped onto $n_i \in N(G)$ by H, then these nodes are a totally ordered subset. Let n_{i_1} be the maximum element and n_{i_l} be the minimum element. Let n_j be an element of $N(G)$ and $n_{j_1}, \cdots, n_{j_h}$ be the nodes of $G_{w/n}$ that are mapped onto n_j by H. If $n_j > n_i$ then $n_{i_1} < n_{j_p}$, $1 \leq p \leq h$ or if $n_j < n_i$ then $n_{i_l} > n_{j_p}$, $1 \leq p \leq h$.

(b) All nodes of $G_{w/n}$ have weight w/n.

(c) If $n_{i_1}, \cdots, n_{i_l}$ are related to n_i as above then n_i has weight $w_i = l(w/n)$.

A BS for $G_{w/n}$ can be considered a PS for G where preemptions can occur only at integer multiples of w/n. It is intuitively clear that as n approaches infinity we can approximate arbitrarily closely any PS for G by a BS for $G_{w/n}$. A formal statement and proof of this fact are given by the following lemma.

LEMMA 1. *Let G, $\{w_i\}$, and w be defined as above and assume there are k processors available. Then we can find a constant K (a function of G and k) such that*

$$CT_{PS}(G, k) \leq CT_{BS}(G_{w/n}, k) \leq CT_{PS}(G, k) + K/n, \qquad n = 1, 2, \cdots.$$

PROOF. A BS for $G_{w/n}$ is a PS for G in the obvious sense and therefore $CT_{PS}(G, k) \leq CT_{BS}(G_{w/n}, k)$. Therefore, it remains to prove $CT_{BS}(G_{w/n}, k) \leq CT_{PS}(G, k) + K/n$, $n = 1, 2, \cdots$, for some fixed K.

We define an *assignment interval* as a time interval (t_1, t_2) during which no processor assignment changes but there is at least one processor assignment change at t_1 and at least one at t_2. During each assignment interval the work being done can be described by a k-tuple, $(a_1, a_2, \cdots, a_k)$, where a_i is n_j $(n_j \in N(G))$ if processor P_i is working on n_j and a_i is ϕ if P_i is idle during the interval. Based on the con-

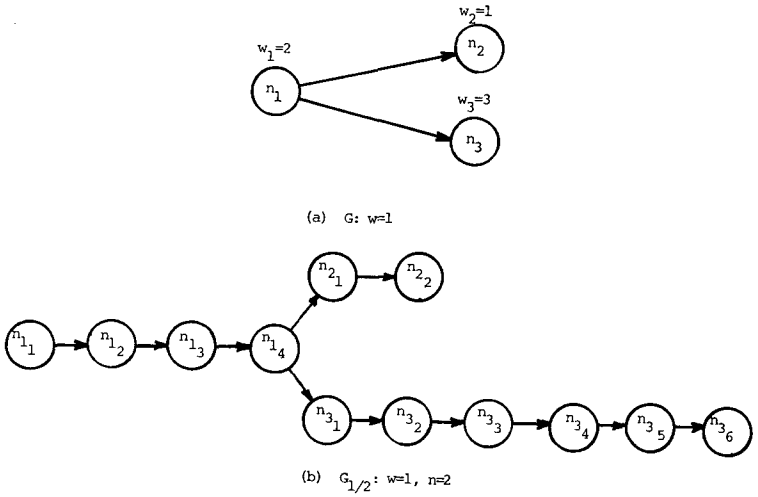

FIG. 5. Example for $G_{w/n}$

struction in Theorem 1, we can now show that any PS for G with k processors can be transformed into an equivalent PS (i.e. one of the same length) which has less than C assignment intervals where C is a constant depending on G and k. Since a PS is also a GS the construction of Theorem 1 can be performed. It is not difficult to verify that in each interval (t_i , t_{i+1}) of the schedule formed, each assignment interval corresponds to a distinct k-tuple. Since there are only a finite number of intervals (t_i , t_{i+1}) and a finite number of distinct k-tuples, it is easy to bound the number of possible assignment intervals. For example, we can let

$$C = | N(G) | [| N(G) | + 1]^k$$

since the number of intervals (t_i , t_{i+1}) is bounded by $| N(G) |$ and the number of distinct k-tuples is bounded by $[| N(G) | + 1]^k$.

We now assume that the schedule has been transformed by the construction of Theorem 1. Suppose we increase the length of each of these assignment intervals, if necessary, so that their lengths are multiples of w/n. The total length of the assignment is not increased by more than K/n where $K = Cw$. In general this will result in more processor time assigned to a node than is actually required, but this is taken care of below.

In this new schedule processor reassignments take place only at times which are multiples of w/n so we can think of the schedule as being divided into intervals of this length. If node n_i has weight mw then n_i has been assigned to a processor in at least mn of these intervals. Let $n_{i_1} , \cdots , n_{i_{mn}}$ be the nodes in $G_{w/n}$ corresponding to n_i. We construct a BS for $G_{w/n}$ by assigning $n_{i_1} , \cdots , n_{i_{mn}}$, in order, to the first mn intervals to which n_i is assigned in the expanded PS. The remaining intervals to which n_i is assigned can be made idle. It is easy to check that this is a BS for $G_{w/n}$ and therefore $CT_{\text{BS}}(G_{w/n} , k)$ cannot be greater than the length of this schedule. We have then $CT_{\text{BS}}(G_{w/n} , k) \leq CT_{\text{PS}}(G, k) + K/n$. Q.E.D.

Example. To illustrate the second half of Lemma 1 we continue the example given after Theorem 1. For convenience we repeat in Figure 6 the PS we constructed in that example and mark off the boundaries of assignment intervals with the symbol Δ.

Now if we refer back to the graph, G, in Figure 3 we see that $w = 1$. We now convert the PS above to a BS for $G_{w/n}$ with $n = 4$ $(G_{w/n} = G_{\frac{1}{4}})$. First, we expand the assignment intervals, where necessary, so that they all have length equal to a multiple of $\frac{1}{4}$. Doing this we get the schedule in Figure 6(b). Now we partition this schedule into intervals of length $\frac{1}{4}$ and we obtain the schedule shown in Figure 6(c). In this schedule there are 9 intervals of length $\frac{1}{4}$ which are assigned to n_2. Since n_2 has weight 2 there are 8 nodes $(n_{2_1} , n_{2_2} , \cdots , n_{2_8})$ of weight $\frac{1}{4}$ in $G_{\frac{1}{4}}$ corresponding to n_2. So we assign $n_{2_1} , \cdots , n_{2_8}$ to the first 8 of these intervals and change the ninth to idle. Treating the rest of the schedule in the same way we get the BS for $G_{\frac{1}{4}}$ shown in Figure 6(d).

Using the definition of $G_{w/n}$ we can outline the method of proving the GS algorithm is optimal. Given a tree T we can construct $T_{w/n}$ for all n. A solution to the problem of finding an optimal BS for $T_{w/n}$ is known and is given below. Using this result it is shown that there exists an integer v such that $CT_{\text{BS}}(T_{w/nv} , k) = CT_M(T, k)$ for all n. But by the previous lemma $CT_{\text{BS}}(T_{w/nv} , k) \to CT_{\text{PS}}(T, k)$ as $n \to \infty$ and it follows that $CT_{\text{PS}}(T, k) = CT_M(T, k)$.

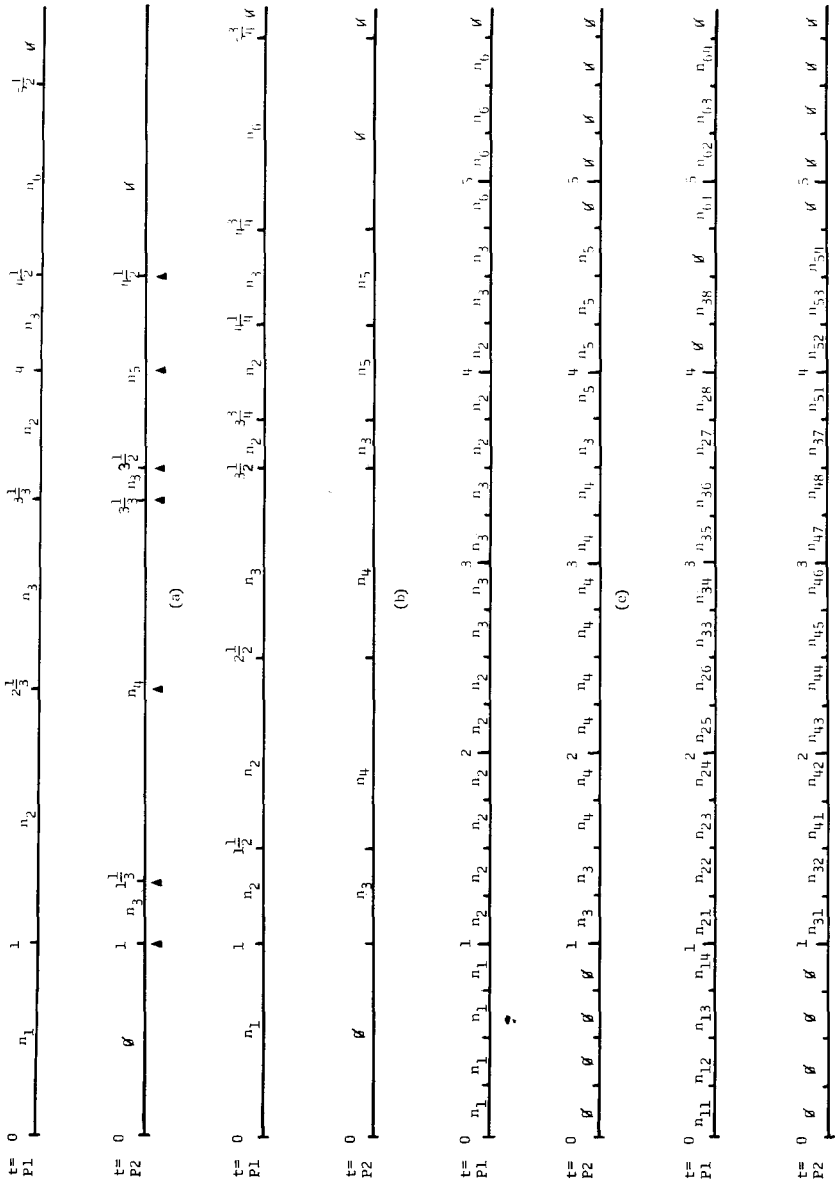

FIG. 6. Illustration for Lemma 1

The algorithm for constructing a minimal-length BS for a tree with equally weighted nodes is as follows.

Algorithm (Hu[5]). Let T be a tree with equally weighted nodes and let k be the number of processors available. Then an optimal BS for T is obtained by assigning the processors to the k nodes farthest from the root of T each time the processors become available for reassignment. If there is a tie among several nodes it is immaterial which are chosen.

We shall refer to this as Hu's algorithm or rule.

The existence of the integer v such that $CT_{\text{BS}}(T_{w/nv}, k) = CT_{\text{PS}}(T, k)$ for all n is proved by induction (Lemmas 4 and 5) on the number of occurrences of events 1 and 2 in an M-schedule. We must first prove that we can indeed use an inductive argument. This is done by proving that the number of occurrences of events 1 and 2 is finite (Lemma 2) and that the node weights remaining at each occurrence of events 1 and 2 are mutually commensurable (Lemma 3).

LEMMA 2. *The number of occurrences of event* 1 *and event* 2 *is finite if* T *has a finite number of nodes.*

PROOF. First, the number of occurrences of event 1 is no greater than $|N(T)|$ and therefore is finite. It remains to show that the number of occurrences of event 2 is also finite.

Let T_t be the tree which remains to be computed at time t when we have constructed an M-schedule for T that starts at $t = 0$. We put the executable nodes of T_t in subsets so that two nodes are in the same subset if and only if they have the same height. Let these subsets be $S_1, S_2, \cdots, S_n$. Since each subset by definition contains at least one node we know that $1 \leq n \leq |N(T)|$. Let the subsets be listed in order of decreasing height in T_t. First, assume that $n > 2$ and that there exists an integer $m < n$ such that $\sum_{i=1}^{m} |S_i| = \chi \leq k$ and $\sum_{i=1}^{m+1} |S_i| = y > k$. Then it is clear from the rule for assigning processor capacity that all nodes in S_1 through S_m are being worked on at full capacity (or at a rate of $\alpha = 1$) and nodes in S_{m+1} are being worked on at a rate equal to $(k - \chi)/|S_{m+1}|$. Let $k - \chi = a$ and $|S_{m+1}| = b$ so that this rate is a/b. If $m < n - 1$, nodes in S_{m+2} through S_n are not being computed. We will have an occurrence of event 2 if either the height of nodes in S_m becomes equal to the height of nodes in S_{m+1} or the height of nodes in S_{m+1} becomes equal to the height of nodes in S_{m+2} (where S_{m+2} is nonempty only if $m < n - 1$). Clearly these events are possible in general because nodes in S_m are being computed at a faster rate than nodes in S_{m+1} and these in turn are being computed at a faster rate than nodes in S_{m+2}. (These events are illustrated in the previous example of Figure 5.) Each time such an event takes place we must stop and reassign the processors. Along with a reassignment of the processors we have a merging of two of the subsets. It follows that there will be no more than $|N(T)| - 1$ of these events before all executable nodes are in the same subset. The next reassignment of processor capacity must take place because of an occurrence of event 1. From this argument we see that between each occurrence of event 1 we can have at most $|N(T)| - 1$ occurrences of event 2. Therefore the number of reassignments of processors is no greater than $|N(T)|[|N(T)| - 1] + |N(T)| = |N(T)|^2$. Finally, this bound obviously holds also for the special cases when the number of sets S_i is $n = 1$ or 2. Q.E.D.

LEMMA 3. *Let* T, $\{w_i\}$, *and* w *be as above. Let* t_1 *be the first time an event* 1 *or event* 2 *occurs in the* M*-schedule for* T. *Then* T_{t_1} *has node weights which are mutually com-*

mensurable. Also, if we let $\{w_i'\}$ *be the node weights of* T_{t_1}, *then* $w' = \gamma w$ *where* γ *is a rational number.*

PROOF. Let $S_1, S_2, \cdots, S_m, \cdots, S_n$ be the subsets of executable nodes of T as defined in Lemma 2 for T_t. We note that the executable nodes of T are being computed at no more than three distinct rates. These rates are 1, a/b, and 0, where a and b are integers.

First we show that $t_1 = rw$, where r is a rational number.

Case 1. At t_1 we have an occurrence of event 1. Let n_i be a node which is completed at t_1. Then n_i must be computed at a rate of 1 or a/b from 0 to t_1. Suppose $n_i = lw$, where l is an integer. Then,

$$t_1 = \frac{[\text{weight of } n_i]}{[\text{rate of computation of } n_i]} = \frac{lw}{1} \quad \text{or} \quad \frac{lw}{a/b}.$$

In either case $t_1 = rw$ with r rational.

Case 2. At t_1 we have an occurrence of event 2. There are only two possibilities. Either the nodes in S_m "catch up" to the nodes in S_{m+1} or the nodes in S_{m+1} "catch up" to the nodes in S_{m+2}. For either possibility it is an easy matter to solve for t_1 and verify that $t_1 = rw$ for some rational r.

It is clear that for any of the above cases at t_1 each node of the original tree is either unchanged or reduced in weight by a rational multiple of t_1 (or equivalently a rational multiple of w), since nodes were computed only at rates 1, a/b, and 0. A node with weight w_i at $t = 0$ has weight $w_i - rw$, $w_i - (ar/b)w$, or w_i at $t = t_1$. Since w_i is an integer multiple of w it follows that each node weight in T_{t_1} is a rational multiple of the corresponding node weight in T. The lemma follows easily from this observation. Q.E.D.

We have shown that a proof based on induction on the number of occurrences of events 1 and 2 is possible. We now state a result which constitutes the major part of the induction step in Lemma 5. The proof of this lemma is contained in the Appendix.

LEMMA 4. *Let* T, T_{t_1}, *and all related quantities be as defined in Lemmas* 2 *and* 3. *Then there exists an integer,* s, *such that*:

(1) $(T_{t_1})_{w/ns}$ *exists for all integers,* n.

(2) *If* T' *is a tree that remains at* $t = t_1$ *when Hu's theorem is used to construct an optimal schedule for* $T_{w/ns}$, *then* T' *is isomorphic to* $(T_{t_1})_{w/ns}$.

With the help of Lemmas 2–4 we are able to prove the following result, from which the basic theorem follows quite easily.

LEMMA 5. *Let* T *be as above. Then there exists an integer* v *such that all trees,* $T_{w/nv}$, $n = 1, 2, \cdots$, *have a minimum computation time using BS equal to the computation time of an* M*-schedule for* T. *That is,*

$$CT_{BS}(T_{w/nv}, k) = CT_M(T, k), \qquad n = 1, 2, \cdots. \tag{3}$$

PROOF. The proof is by induction on the number of occurrences of event 1 or event 2 in the schedule for T. Clearly, if there is only one such event then this must be an occurrence of event 1 at the completion of the schedule. It follows that T consists of a single node. In this case any positive integer will suffice for v.

Assume the lemma is true if there are $p - 1$ occurrences of event 1 or event 2 and assume that there are p occurrences of these events in the M-schedule for T.

Let t_1 be the time at which the first event takes place and let T_{t_1} be the tree which remains to be computed at t_1.

It should be clear that the schedule generated for T_{t_1} is the schedule generated for T for $t = t_1$ to the end. Therefore,

$$CT_M(T, k) = t_1 + CT_M(T_{t_1}, k). \tag{4}$$

We know from Lemma 3 that T_{t_1} has node weights that are mutually commensurable and that $w' = (q/u)w$ with q, u integers. Also, the schedule for T_{t_1} has $p - 1$ occurrences of events 1 and 2 and therefore, by the induction hypothesis, we can find an integer v' such that

$$CT_{BS}([T_{t_1}]_{w'/n'v'}, k) = CT_M(T_{t_1}, k); \qquad n' = 1, 2, \cdots.$$

If the above is true for $n' = 1, 2, \cdots$ then it is true for $n' = q, 2q \cdots$. Thus we have, after substituting qw/u for w' and nq for n',

$$CT_{BS}([T_{t_1}]_{w/nv'u}, k) = CT_M(T_{t_1}, k); \qquad n = 1, 2, \cdots. \tag{5}$$

Let s be as defined in Lemma 4 and let v be the least common multiple of s and $v'u$. Then, again from Lemma 4, $(T_{t_1})_{w/nv}$ is the tree which remains at $t = t_1$ when we compute $T_{w/nv}$ using Hu's rule. From this we have that

$$CT_{BS}(T_{w/nv}, k) = t_1 + CT_{BS}([T_{t_1}]_{w/nv}, k); \qquad n = 1, 2, \cdots. \tag{6}$$

Substituting on the right from (5) we obtain

$$CT_{BS}(T_{w/nv}, k) = t_1 + CT_M(T_{t_1}, k); \qquad n = 1, 2, \cdots. \tag{7}$$

From (4) and (7) the result (3) follows directly. Q.E.D.

Finally, as pointed out previously, the existence of the integer v such that $CT_{BS}(T_{w/nv}, k) = CT_{PS}(t, k)$ for all n together with Lemma 1 implies the result we have been seeking.

THEOREM 2. *The M-schedule for a computation tree, T, is a minimal-length GS(PS).*

Remark. It should be observed that the M-schedule is also applicable to *forests* (or sets) of computation trees. To obtain an optimal GS for a set of trees one simply applies the algorithm to the tree constructed by joining the roots of each of the given trees to a new node of weight zero which will be the root of a new composite tree.

4. *Conclusion*

In this paper our goal has been to extend the rather limited theory so far developed for the study of multiprocessor scheduling. The major point of departure for our study has been the work of Hu in BS scheduling of computation trees. The principal generalizations introduced are those of preemptive (PS) and processor sharing (GS) assignments in which computations are allowed to be interrupted and shelved in favor of other tasks whose earlier computation leads to greater efficiency, and in which processors may be shared by more than one task at a cost in computation rate for each task. The basic results have been:

(1) a proof that for general graphs scheduling with preemptions is not improved by allowing processor sharing, and

(2) the statement and proof of an optimal PS(GS) algorithm for tree-structured computations.

Contrasting lines of research are those reported by Martin and Estrin [7, 8] in which more general computation models are introduced and studied through extensive simulation, and the theoretical studies of Coffman [11] and Kleinrock [12] in which probability models of multiprocessor systems have been analyzed. For further studies of multiprocessor systems we reference the survey by Critchlow [13].

Appendix. Proof of Lemma 4

LEMMA 4. *Let* T, T_{t_1}, *and all related quantities be as defined in Lemmas* 2 *and* 3. *Then there exists an integer,* s, *such that*:

(1) $(T_{t_1})_{w/ns}$ *exists for all integers* n.

(2) *If* T' *is a tree that remains at* $t = t_1$ *when Hu's theorem is used to construct an optimal schedule for* $T_{w/ns}$, *then* T' *is isomorphic to* $(T_{t_1})_{w/ns}$.

PROOF. $(T_{t_1})_{w/ns}$ is defined for all integers n if for each n, $w/ns = w'/n'$ for some integer n'. We have from Lemma 3 that $w' = (q/u)w$ for some integers q, u and $t_1 = (c/d)w$ for c, d integers. If we let $s = dbu$ then $w/ns = w/ndbu = w'/n'$ with $n' = ndbq$. Note that $s = u$ is sufficient for the first part of the lemma; however, for the second part we need the additional factors.

To show that $s = dbu$ satisfies condition (2) we first note that T' is a subtree of $T_{w/ns}$ and that $(T_{t_1})_{w/ns}$ is isomorphic to a subtree of $T_{w/ns}$. If n_{i_j} is a node of $T_{w/ns}$ let n'_{i_j} be the corresponding element of $(T_{t_1})_{w/ns}$ if it exists. Let $\xi_i = \{n_{i_1}, \cdots, n_{i_l}\}$ be the nodes of $T_{w/ns}$ corresponding to n_i in T. Let $\xi' = \{n'_{i_j} \mid n'_{i_j} \in N((T_{t_1})_{w/ns})$ and $n_{i_j} \in \xi_i\}$ and let $\xi'' = \{n_{i_1}, n_{i_2}, \cdots, n_{i_l}\} \cap N(T')$. If we can show that $|\xi_i'| = |\xi_i''|$ for all i then clearly T' and $(T_{t_1})_{w/ns}$ are isomorphic. We proceed to do this now.

No nodes of T are completed before t_1 in the schedule constructed by the algorithm. Therefore if n_i is not an initial node of T it is not worked on in $(0, t_1)$ and it must be that $|\xi_i'| = |\xi_i|$. Now consider the initial nodes of T. Let $S_1, S_2, \cdots, S_m, S_{m+1}, \cdots, S_n$ be as defined in Lemma 3. In the interval $(0, t_1)$ nodes in S_1 through S_m are executed at a rate $\alpha = 1$. Therefore, if n_i is a node in one of these sets its weight is reduced by $t_1 \cdot 1 = w(c/d) = nsc/d \cdot w/ns$. Therefore ξ_i' has nsc/d fewer elements then ξ_i. Substituting $s = dbu$ we have $|\xi_i'| = |\xi_i| - ncbu$, $n_i \in \bigcup_{j=1}^{m} S_j$.

In $(0, t_1)$ nodes in S_{m+1} are computed at a rate of $\alpha = a/b$. Therefore if $n_i \in S_{m+1}$ its weight is reduced by $t_1 \cdot a/b$ in this interval. Thus, from $t_1 \cdot a/b = ac/bd \cdot w = acns/bd \cdot w/ns$ we have $|\xi_j'| = |\xi_j| - acun$, for $n_j \in S_{m+1}$.

If $n_i \in \bigcup_{j=m+2}^{n} S_j$ then it is not computed at all in $(0, t_1)$ in the M-schedule. Clearly then $|\xi_i'| = |\xi_i|$ for all $n_i \in \bigcup_{j=m+2}^{n} S_j$. We now know how many nodes are in each ξ_i' in relation to how many were in ξ_i. We have yet to show that these same expressions hold for the ξ_i''.

Let $S_1', S_2', \cdots, S_m', \cdots, S_n'$ be those sets of subsets of nodes of $T_{w/ns}$ such that $\xi_i \in S_j'$ at $t = 0$ if and only if $n_i \in S_j$. We let S_j' change with time by removing a node as soon as it is assigned to a processor. We note that, since all nodes of $T_{w/ns}$ have weight w/ns, the k processors finish nodes and are reassigned at discrete time intervals: $0, w/ns, 2w/ns, \cdots$. In $(0, t_1)$ there are exactly $t_1/(w/ns) = ncbu$ such intervals. Now it can be shown that if we use Hu's rule then in each w/ns interval

in $(0, t_1)$ we must compute exactly one node from each subset in S_1' through S_m' and one node from a of the subsets in S_{m+1}'. (Of course, no nodes will be done from subsets in S_{m+2}' through S_n'.) This intuitive result can be shown using an inductive argument which involves ruling out all other possibilities for node assignments in $(0, t_1)$. However, the details of this are lengthy and unrewarding and will be omitted.

It follows from the above observations that if $\xi_j \in S_i'$ $(1 \leq i \leq m)$ at $t = 0$ then at t_1 there are $ncbu$ fewer nodes in ξ_j'', i.e. $|\xi_j'| = |\xi_j| - ncbu$ for $\xi_j \in \bigcup_{l=1}^{m} S_l$. Since the executable nodes in S_{m+1}' are never more than w/ns apart in height it follows that $nbuc \cdot a/b = nauc$ nodes are done from each subset in S_{m+1}'. Therefore $|\xi_i''| = |\xi_i| - nauc$, $\xi_i \in S_{m+1}'$. The subsets in S_{m+2}' through S_n' are unchanged. Checking with the earlier results we see that $|\xi_i'| = |\xi_i''|$ for all i. Q.E.D.

REFERENCES

1. ESTRIN, G. The organization of computer systems—the fixed plus variable structure computer. *Proc. 1960 Western Joint Comput. Conf.*, Vol. 17, pp. 33–40.
2. MARTIN, DAVID. The automatic assignment and sequencing of computations on parallel systems. Doctoral Diss., U. of California at Los Angeles, 1966.
3. HELLER, J. Sequencing aspects of multiprogramming. *J. ACM 8*, 3 (July 1961), 426–439.
4. RICHARDS, P. Timing properties of multiprocessor systems. Rep. No. TD-B60-27, Tech. Operations, Inc., Burlington, Mass., Aug. 1960.
5. GRAHAM, R. L. Bounds for certain multiprocessing anomalies. *Bell Syst. Tech. J. 45* (1966), 1563–1581.
6. MANACHER, G. K. Production and stabilization of real-time task schedules. *J. ACM 14*, 3 (July 1967), 439–465.
7. MARTIN, D., AND ESTRIN, G. Experiments on models of computations and systems. *IEEE Trans. EC-16*, 1 (Feb. 1967), 59–69.
8. MARTIN, D., AND ESTRIN, G. Models of computational systems—cyclic to acyclic graph transformation. *IEEE Trans. EC-16*, 1 (Feb. 1967), 70–79.
9. HU, T. C. Parallel sequencing and assembly line problems. *Oper. Res. 9*, 6 (Nov. 1961), 841–848.
10. MCNAUGHTON, R. Scheduling with deadlines and loss functions. *Manage. Sci. 6*, 1 (Oct. 1959), 1–12.
11. COFFMAN, E. G. Bounds on parallel-processing of queues with multiple-phase jobs. *Naval Res. Logistics Quart. 14*, 3 (Sept. 1967), 345–366.
12. KLEINROCK, L. Sequential processing machines (S.P.M.) analyzed with a queuing theory model. *J. ACM 13*, 2 (April 1966), 179–193.
13. CRITCHLOW, A. J. Generalized multiprocessing and multiprogramming systems. Proc. AFIPS 1963 Fall Joint Comput. Conf., Vol. 24, pp. 107–126.

RECEIVED SEPTEMBER, 1968; REVISED OCTOBER, 1969