References

[1] G. Estrin and R. Turn, "Automatic assignment of computations in a variable structure computer system," *IEEE Trans. Electronic Computers*, vol. EC-12, pp. 756–773, December 1963.
[2] D. F. Martin, "The automatic assignment and sequencing of computations on parallel processor systems," Ph.D. dissertation, Department of Engineering, University of California, Los Angeles, 1966.
[3] D. L. Epley, Review R68-5, *IEEE Trans. Computers*, vol. C-17, pp. 191–192, February 1968.
[4] J. L. Baer, "Graph models of computations in computer systems," Ph.D. dissertation, Department of Engineering, University of California, Los Angeles, 1968.
[5] D. F. Martin and G. Estrin, "Models of computational systems—cyclic to acyclic graph transformations," *IEEE Trans. Electronic Computers*, vol. EC-16, pp. 70–79, February 1967.
[6] J. L. Baer and G. Estrin, "Frequency numbers associated with graph models of computations," presented at the 2nd Hawaii Internatl. Conf. on System Sciences, January 1969.
[7] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, pp. 11–12, January 1962.
[8] D. F. Martin and G. Estrin, "Path length computations on graph models of computations," *IEEE Trans. Computers*, vol. C-18, pp. 530–536, June 1969.
[9] E. C. Russell and G. Estrin, "Measurement based automatic analysis of FORTRAN programs," *1969 Spring Joint Computer Conf., AFIPS Proc.*, vol. 34. Montvale, N. J.: AFIPS Press, 1969, pp. 723–733.

Optimal Preemptive Scheduling on Two-Processor Systems

RICHARD R. MUNTZ, MEMBER, IEEE, AND EDWARD G. COFFMAN, JR., MEMBER, IEEE

Abstract—**One of the important potentials of multiprocessor systems is the ability to speed the completion of a computation by concurrently processing independent portions of the job. In this paper we consider the static scheduling of computations for a system containing two indentical processors. The object is to complete the computation in the minimum amount of time. A computation is assumed to be specified as a partially ordered set of tasks and the execution time for each task. A solution for the two-machine case with preemptive scheduling is presented.**

Index Terms—**Computation graphs, multiprocessing, parallel processing, preemptive scheduling, static scheduling.**

Introduction

MULTIPROCESSOR computer systems have been shown to have several advantages [1] over conventional systems. One of great importance to a large class of users is the potential increase in computing speed beyond that achievable with the more conventional single-processor organization. On the other hand, how to best exploit the multiple, independent processor organization in order to obtain maximum performance remains a little-understood problem. It is this problem which has motivated the research reported in this paper.

Informally, the specific question we seek to answer is the following. Given a set of tasks or computations along with a set of operational precedence relationships that exist between certain of these tasks, and given a set of k identical processors, how does one sequence (or schedule) these tasks on the k processors so that they execute in minimum time? In order to describe the extent to which this question has been answered by the present study and by earlier work on this problem, we shall require a number of definitions.

First, a *multiprocessor system* consists quite simply of a set of identical processors capable of independent operation on independent tasks. *Jobs* submitted to the system consist of 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 $n_i > n_j$ (to be read n_i precedes n_j or n_j follows n_i), then n_i must be completed before n_j is started. It is natural to associate a graph with each job such that the nodes correspond to tasks and the arcs correspond to the precedence relations defining the partial ordering. More formally, there is a directed arc from the node representing task n_i to the node representing the task n_j if and only if $n_i > n_j$ and there is no task n_l such that $n_i > n_l > n_j$. We shall use the terms "job," "graph," and "computation graph" interchangeably. Task execution times are represented by node weights.

Informally, a *schedule* or *assignment* for k processors and a given computation graph is a description of the work done by each machine 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. The

Manuscript received September 16, 1968; revised June 2, 1969. This work was supported in part by Bell Telephone Laboratories, Inc., Murray Hill, N. J. This paper was presented at the 1969 IEEE Computer Group Conference, Minneapolis, Minn., June 17–19, 1969.

R. R. Muntz is with the School of Engineering and Applied Science, University of California, Los Angeles.

E. G. Coffman, Jr., was with the Department of Electrical Engineering, Princeton University, Princeton, N. J. Until July, 1970, he will be at the Computing Laboratory, University of Newcastle-upon-Tyne, England.

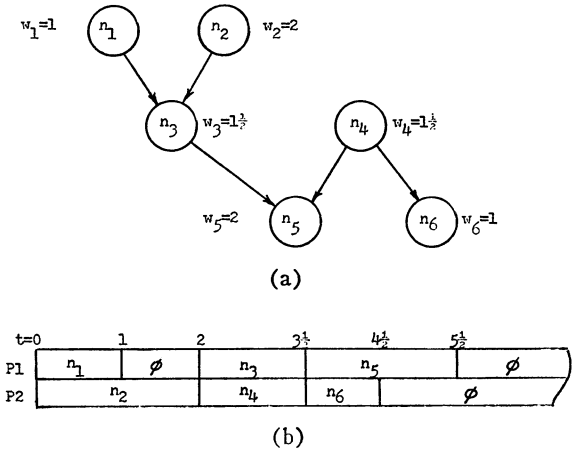


Fig. 1. An example graph and schedule for two processors: P1 and P2.

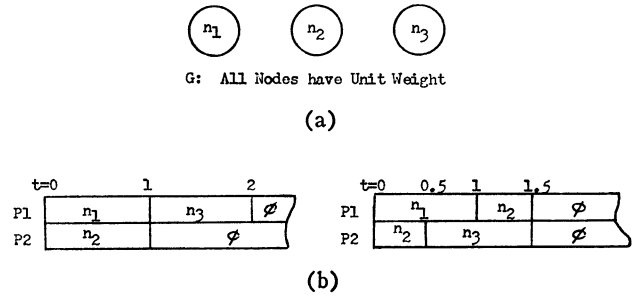


Fig. 2. Illustration of the advantage of preemptive scheduling.

simplest way of specifying a schedule is to use a Gantt chart which consists of a time axis for each processor with intervals marked off and labeled with the name of the task being processed. We use the symbol ϕ to represent an idle period. In Fig. 1(a) we have shown a graph G, and in Fig. 1(b) a schedule for G with two processors. The computation times are shown next to the node representing the task.

In the next section we shall define the principal types of scheduling disciplines of interest to us, after which a preview of the remainder of the paper will be given.

From a more general point of view, it is clear that the static scheduling models we have introduced are most applicable to those cases in which a program or computation is to be executed many times, particularly in real-time situations. In these cases it is worth some effort to investigate the program structure and execution times.

With respect to the task execution times, assumed known in the static model, there are two interpretations which increase the usefulness of the results when these times are not known exactly. First, the execution times may be interpreted as maximum processing times. In this case the schedule length is the maximum time to complete the graph [8]. Second, the execution times may be regarded as expected values of the run times considered as random variables. With this interpretation the length of the schedule produced is an (optimistic) estimate of the mean length of the computation over many runs [9].

The constraint to partial orderings of tasks as a model of computations obviously means that loops and conditional branching cannot be represented. If the tasks are large functional blocks, this may not be a serious problem. Note also that a particular execution of a graph containing a conditional branch will be representable as the execution of a partially ordered set of tasks, since one branch or the other must be taken. This clearly extends to any number of branches and to loops as well. An efficient solution to the problem of optimally scheduling partially ordered tasks can be used as a tool in studying the more general problem. For example, it gives a standard for comparing scheduling heuristics for these graphs rather than having only relative measures of performance.

Preemptive Scheduling Disciplines

In most investigations of the problem of constructing minimum length schedules for computation graphs, it is assumed that once a processor is assigned to a task it must work continuously on this task until it has been completed. (As in Fig. 1(b), for example.) This procedure for assigning processors to nodes will be termed the *basic scheduling* (BS) discipline. Clearly, the problem of finding an optimal BS for any given graph is effectively solvable by exhaustion. However, practical solutions for large problems have consisted only of a result for rooted trees with equally weighted nodes [2], and more recently, an unpublished result for the two-machine case and an arbitrary partial ordering of equally weighted tasks [3].

A more recent study on multiprocessor scheduling [4], which forms the point of departure for the results presented in this paper, has focused on scheduling disciplines in which this run-to-completion constraint has been relaxed. In particular, with the *preemptive scheduling* (PS) discipline it is permitted to interrupt any processor at any time and reassign it to a different task. A solution of the scheduling problem for rooted computation trees has also been found [4] for the PS discipline. Fig. 2 shows how effective use can be made of the preemption capability. Fig. 2(b) shows a minimum length assignment for G with $k=2$ machines and using the BS discipline. The length of this schedule is two time units. By comparison, we see in Fig. 2(c) a PS requiring only 1.5 time units and one preemption. In constructing the PS in Fig. 2, we have assumed that the preemption itself did not require any time. Notice, however, that as long as the total cost of the preemption is less than one unit of time, an optimal PS for G will be less than two 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 permitted. Furthermore, from existing and proposed multiprogramming/multiprocessing systems, there is reason to believe that the costs of preemptions may be

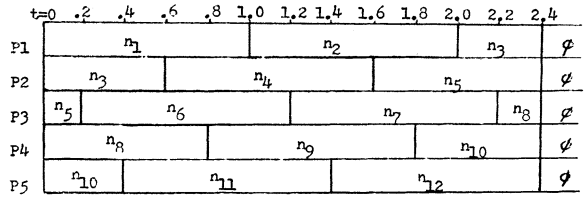


Fig. 3. A PS for twelve independent unit weight nodes with five machines.

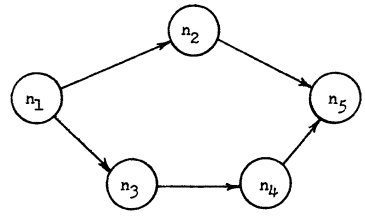


Fig. 4. G: all nodes have unit weight.

quite nominal in some applications. Contributing factors are: 1) the existence of large main memories out of which programs are executed, and 2) the overlapping of transfers between main and auxiliary storage and program execution. In any case, we shall continue to make the assumption that preemption delays are negligible.

Static preemptive scheduling to minimize maximum flow-time has received only slight attention in the literature. A solution for a special case of the problem we have posed has been given by McNaughton [5]. This result is stated in the following lemma.

Lemma 1: For a set of independent tasks with weights $\{w_1, w_2, \cdots, w_n\}$ and k available processors, the optimal PS has length

$$\max\left\{\max_{1\leq i\leq n}\{w_i\}, \frac{\sum_{i=1}^{n} w_i}{k}\right\}.$$

To illustrate the lemma, suppose we have a graph G consisting of twelve independent nodes all of unit weight and $k=5$ available processors. A minimal length PS for this case is shown in Fig. 3. The tasks are assigned, in order, to the first processor until a task (in this case n_3) extends beyond the computed optimal schedule length. When this occurs, the remaining time on the task is assigned to the next machine starting at $t=0$.

McNaughton [5] and Rothkopf [6] have considered preemptive scheduling problems with various objective functions but for independent tasks only.

Subset Schedules

The development of the algorithm for optimal preemptive scheduling with two processors proceeds by first defining a subclass of preemptive schedules which is shown to contain at least one minimal length schedule. We then show how to construct an optimal schedule for this subclass which from the above statement is also an optimal PS. To describe this subclass of preemptive schedules we must first introduce the notion of a *subset sequence.* A subset sequence for a graph G, with equally weighted nodes, is a sequence of disjoint subsets of nodes of G, $S_1, S_2, \cdots, S_m$ such that: 1) if n is a node of G, then $n\in S_i$ for some i, and 2) if n, m are nodes of G, with $n\in S_i$, $m\in S_j$ and $n>m$, then $i<j$. In other words, a subset sequence for G is any partition of the nodes of G satisfying constraint 2) above.

Since G has equally weighted nodes, we may assume without loss of generality that the nodes have unit weight. From the definition we see that each subset S_i consists of a set of mutually independent nodes. It follows from Lemma 1 that if $|S_i| \leq k$[1] we can construct a schedule to execute all nodes in S_i in one unit of time, and if $|S_i| > k$ we can construct a schedule to execute all nodes in S_i in $|S_i|/k$ units of time. From the definition of a subset sequence we know that if we schedule each subset independently we can form a PS for G by using the schedule for S_1 followed immediately by the schedule for S_2, etc. For our purposes we need only consider the schedules formed by this method in which each S_i is scheduled in an optimal way (i.e., achieves the minimum computation times mentioned above). A PS constructed from a subset sequence in the above manner will be called a *subset schedule.*

As an example consider the graph of Fig. 4. One possible subset sequence for G would be $S_1=\{n_1\}$, $S_2=\{n_2, n_3\}$, $S_3=\{n_4\}$, $S_4=\{n_5\}$. Another subset sequence could be formed by making $S_2=\{n_3\}$, $S_3=\{n_2, n_4\}$.

In terms of these definitions, the two major results of this paper are: 1) a proof that an optimal subset schedule for two machines is an optimal PS, and 2) a description of an algorithm for finding the optimal subset schedule for two machines. The corresponding problems for an arbitrary number (k) of machines are as yet unsolved.

Optimal Subset Schedules

In this section we outline the proof that optimal subset schedules for two machines are optimal PSs. The proof is by induction on the number of nodes in a graph. In order to establish the induction step in the proof, we shall need the result of the following lemma. Basically, the lemma states a "normal form" property of PSs for two machines and graphs of equally weighted nodes. The proof is not given here since it is rather long; it can be found in [10].

Lemma 2: Let G be an arbitrary graph all of whose nodes have unit weight. Then any PS for G using two machines can be transformed into a new schedule which is no longer than the first and has one of the two forms shown in Fig. 5 where n, p, and q are nodes of G or ϕ (i.e., an idle period). With this result we can prove the first basic theorem on subset schedules for two machines.

[1] $|S_i|$ = cardinality of S_i.

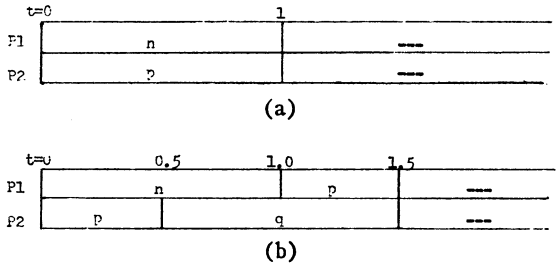


Fig. 5. Schedule forms.

Theorem 1: For $k=2$ machines, an optimal subset schedule for any graph G, with equally weighted nodes, is an optimal PS for G.

Proof: Without loss of generality, assume all nodes of G have weight 1. The proof is by induction on the number of nodes in G. For the graph of one node, the theorem is obvious. Assume the theorem is true for all graphs of less than n nodes, and let G be a graph with n nodes.

First, suppose G has only one or two initial nodes. In a PS it is never advantageous to leave a machine idle when a node is available, and therefore we need only consider schedules which work on these one or two initial nodes from $t=0$ to $t=1$. The remainder of G has less than n nodes, and by the induction hypothesis can be completed in a minimum amount of time by a subset schedule. Putting these two schedules together, we have a subset schedule for G which is clearly optimal.

Now assume G has more than two initial nodes. By Lemma 2 we may assume further that any given PS for G (call this assignment A) has been transformed into a new assignment which is no longer than A and has one of the two forms shown in Fig. 5. Suppose, for instance, that A has been transformed into the second form and n, p, and q are nodes of G. The remainder of the schedule from $t=1.5$ on is a PS for the graph formed by removing n, p, and q from G. Call this graph G'. From the induction hypothesis we know there exists a subset schedule for G' which is an optimal PS for G'. Let the corresponding subset sequence be $S_1, S_2, \cdots, S_m$. It follows that $\{n, p, q\}, S_1, S_2, \cdots, S_m$ is a subset sequence for G whose corresponding subset schedule is no longer than A. Since this is true for any PS for G, it is true for any optimal PS for G. Therefore, there exists a subset schedule for G which is an optimal PS for G. A similar argument holds if we get A into the first form. This completes the induction step and the proof of the theorem. Q.E.D.

It is now desirable to extend the previous result to the case of graphs with arbitrary node weights. Specifically, we shall now consider graphs having *mutually commensurable* node weights; i.e., there exists a real number w such that each node weight is an integer multiple of w. Such graphs are not very restrictive since more general graphs may be approximated arbitrarily closely by ones with mutually commensurable node weights. Before extending Theorem 1 so that it applies to these latter graphs, we must introduce the following definitions.

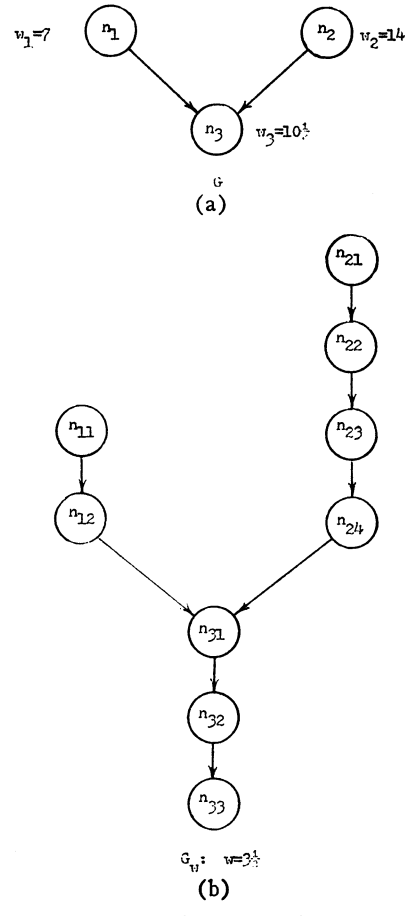


Fig. 6. Transformation of G to G_w.

Let G be a graph with the mutually commensurable node weights $w_1, w_2, \cdots, w_n$, and let w be the largest real number such that w_i/w is an integer for all i. We now define a graph G_w derived from G as follows. We take each node n_i of G with weight w_i and replace it by a series connection of nodes of weight w so that the total weight of these nodes is 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 of the connection. Observe that G and G_w are related by a simple homomorphism. As an example, consider the graph of Fig. 6 for which $w=3\frac{1}{2}$. As shown, a node n_i of G is expanded into nodes $n_{i1}, n_{i2}, \cdots, n_{ij_i}$ of G_w where $j_i=w_i/w$.

From the above definition it follows that a PS for G defines a PS for G_w, and vice versa. Furthermore, an optimal PS for G_w defines an optimal PS for G. Now if we define a subset schedule for G (with unequally weighted nodes) to be a subset schedule of G_w, then we have the following corollary to Theorem 1.

Corollary 1: Let G be any graph with mutually commensurable node weights. Then an optimal subset schedule for G_w is an optimal PS for G with $k=2$ machines.

In the next section we address the problem of finding an optimal subset schedule for a given graph with $k=2$ machines.

An Algorithm for Finding an Optimal Subset Schedule for Two Processors

Let G be a graph with mutually commensurable node weights and let G_w be as defined above. Let L be the length of a longest path in G. We propose the following algorithm for constructing a subset sequence for G_w which corresponds to an optimal subset schedule for G_w (and therefore an optimal PS for G) with $k=2$ machines. The *level* of a node is defined as the length of the longest path from that node to a terminal node. The level of node n is denoted by $l(n)$.

Algorithm:

Step 1) Set index $j=1$.

Step 2) Let Λ_j be the set of all nodes which have not yet been assigned to subsets and are at level $L-j+1$. Assign all nodes in Λ_j to S_j. If $|\Lambda_j|=1$ then go to step 4).

Step 3) If $j=L$ then stop, otherwise, set $j=j+1$ and go to step 2).

Step 4) Let $\Omega=\{q_i\}$ be the set of all nodes which have not been assigned to subsets, but all of whose predecessors are contained in $S_1 \cup S_2 \cup \cdots \cup S_{j-1}$. If $\Omega=\phi$ then go to Step 3). If $\Omega=\phi$ then assign q_u to S_j where q_u is such that $l(q_u)=\max_{q_i \in \Omega}\{l(q_i)\}$, and go to Step 3).

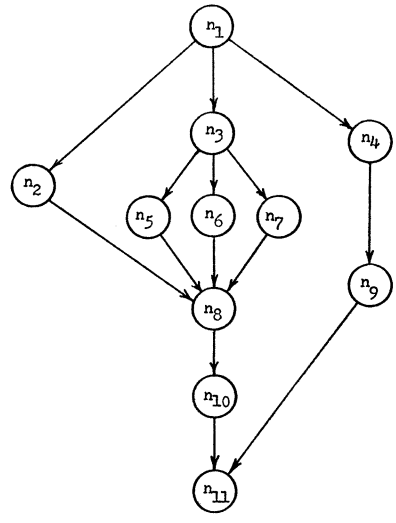
An informal statement of the algorithm is worthwhile at this point. It is proposed that the nodes be assigned to subsets level by level with higher level nodes assigned to subsets first. The only exception to this rule is when it would result in a subset containing only one node while there is at least one other node at a lower level which can be assigned to this subset without violating any precedence relations. In this case a second node is selected which is at the highest possible level.

As an example we have shown in Fig. 7 a graph G and a subset sequence for G constructed by the algorithm. Note that in constructing S_2, the only task at level 5 was n_3. Both n_2 and n_4 are executable but the algorithm states that n_2 is to be assigned to S_2. If n_4 were chosen we could not achieve an optimal subset sequence.

It is an easy matter to show that the construction specified by the algorithm does result in a valid subset sequence. We now prove that the algorithm produces a subset sequence which corresponds to an optimal subset schedule. By Theorem 1 this will also be an optimal PS.

Theorem 2: The subset schedule corresponding to the subset sequence constructed by the algorithm is optimal.

Proof: Let G be any graph with equally weighted



G: All Nodes have Unit Weight

(a)

$\{n_1\}, \{n_3,n_2\}, \{n_5,n_6,n_7\}, \{n_8,n_4\}, \{n_{10},n_9\}, \{n_{11}\}$

Optimal Subset Sequence for G

(b)

Fig. 7. Illustration of the algorithm.

nodes, and $A=S_1, S_2, \cdots, S_m$ be a subset sequence for G with $k=2$ machines. Without loss of generality, assume the nodes of G have unit weight. Suppose subset S_j contains $x_j \geq 2$ nodes. Then by Lemma 1 we can schedule these nodes in $x_j/2$ units of time. If S_j contains only one node, then S_j requires one unit of time. Let W be the total weight of G, let N be the number of subsets containing only one node, and let ϕ be the set of all j such that S_j contains more than one node. Then the schedule for G using this subset sequence requires

$$\left(\sum_{j\in\phi}\frac{x_j}{2}\right)+N=\frac{W-N}{2}+N=\frac{W+N}{2} \text{ units of time.}$$

It follows that a subset schedule for two machines is optimal if its corresponding subset sequence has the least number of subsets with only one element. This is what we shall prove is true of any subset sequence constructed by the algorithm. The proof of the theorem is easier if we prove a somewhat stronger statement. Before giving this latter statement, we need the following notation.

Let G be any graph with equally weighted nodes. Let L be the length of a longest path in G, and $n_1, n_2, \cdots, n_L$ be the nodes, in order, in a maximum length path. Let $A_1=S_1, S_2, \cdots, S_m$ be any subset sequence for G. We know that each of the n_i must appear in one of these subsets and they must be in distinct subsets. Let us consider just those subsets of A_1 that contain an n_i. If S_j contains n_i we shall rename the subset and call it S_{n_i}. Let $S_{n_1}, S_{n_2}, \cdots, S_{nL}$ be this family of subsets. Clearly, the number of subsets in A_1 that have only one element is no less than the number of subsets with one

element in $\{S_{n_1}, S_{n_2}, \cdots, S_{nL}\}$. Let $N_{A_1}(i)$ denote the number of subsets with one element in $\{S_{n_1}, S_{n_2}, \cdots, S_{n_i}\}$. Let $A_2 = S_1', S_2', \cdots, S_L'$ be a subset sequence as constructed by the algorithm. It is clear from the statement of the algorithm that for $1 \leq i \leq L$, $n_i \in S_i$.

We now seek to prove (the stronger statement) that if A_1 is any subset sequence for G, and A_2 is any subset sequence for G constructed in accordance with the algorithm, then $N_{A_1}(i) \geq N_{A_2}(i)$ for $1 \leq i \leq L$. Clearly, $N_{A_1}(L) \geq N_{A_2}(L)$ is all we need in order to prove that A_1 is at least as long as A_2.

The proof is by induction on the number of nodes in G. The statement is obvious for the case of one node, so assume that the theorem is true for all graphs with less than m nodes, and assume that G has m nodes. With G, A_1, and A_2 as above, let $n_L, p_1, p_2, \cdots, p_n$ be the terminal nodes of G, and let G' be the graph remaining when these terminal nodes are removed from G. Clearly, if we delete $n_L, p_1, p_2, \cdots, p_n$ from the subsets in A_1, we are left (after removing any empty subsets) with a valid subset sequence A_1' for G'. Furthermore, it is a straightforward matter to verify that when $n_L, p_1, p_2, \cdots, p_n$ are removed from A_2, we are left with a subset sequence A_2' for G' that can be constructed by the algorithm (the statement must be made this way since the algorithm does not generate a unique subset sequence). Therefore, since G' contains less than m nodes, we have from the induction hypothesis

$$N_{A_1'}(i) \geq N_{A_2'}(i); \qquad 1 \leq i \leq L-1.$$

The two properties below follow easily from the statement of the algorithm and will be used in the induction step of the proof.

Property 1) Any p_i that is assigned to a subset S_j', $j<L$, must be the only node other than n_j assigned to that subset.

Property 2) If S_j' contains only one node, i.e., n_j, then any p_i which is independent of n_j is assigned to a subset of lower index.

Suppose $N_{A_1}(j) < N_{A_2}(j)$ for some j, $1 \leq j < L$. In fact, let j be the first time this happens so that $N_{A_1}(j-1) = N_{A_2}(j-1)$ and S_j' contains one element while S_{n_j} contains at least two nodes. From $N_{A_1}(j) < N_{A_2}(j)$ and $N_{A_1'}(j) \geq N_{A_2'}(j)$ it follows that when $n_L, p_1, p_2, \cdots, p_m$ are removed from A_1 and A_2, the number of subsets in $S_{n_1}, S_{n_2}, \cdots, S_{n_j}$ that become subsets with less than two elements is at least one greater than the number of such subsets in $S_1', S_2', \cdots, S_j'$. This implies that there exists a p_r belonging to one of the $S_{n_1}, S_{n_2}, \cdots, S_{n_j}$ that is either assigned to one of the subsets $S_1', S_2', \cdots, S_j'$ which also contains another node of G(in addition to an n_i), or is not assigned to any of the subsets $S_1', S_2', \cdots, S_j'$. The former contradicts Property 1) above and the latter contradicts Property 2).

For the special case $j=L$ we know that S_L' contains only n_L and it follows from Properties 1) and 2) that in A_2 each p_i is assigned to a subset of cardinality 2 and index less than L. Therefore, $N_{A_2'}(L-1) = N_{A_2}(L-1) + n$. From the fact that at least one of the p_i is an element of S_{n_L} it follows that $N_{A_1'}(L-1) < N_{A_1}(L-1) + n$. Since $N_{A_2}(L-1) = N_{A_1}(L-1)$, we have $N_{A_1'}(L-1) < N_{A_2'}(L-1)$ which is a contradiction. Q.E.D.

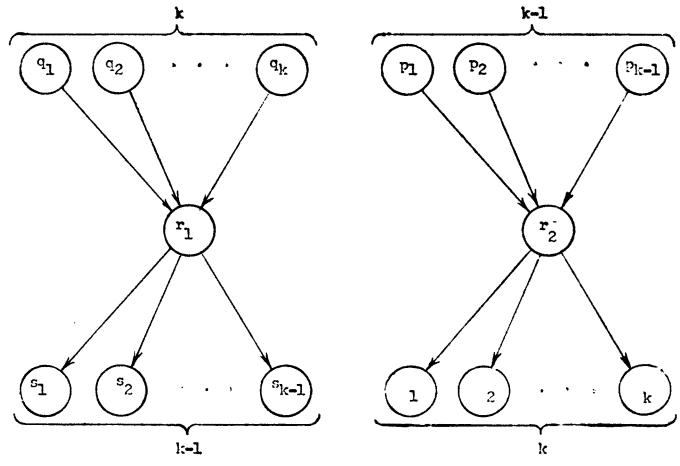


Fig. 8. A counterexample to the algorithm for $k>2$.

There is an obvious generalization of this algorithm to three or more machines. Nodes are again assigned to subsets level by level except when there are $u<k$ nodes at the highest level. In the latter case, as many as possible of the $k-u$ nodes necessary to "fill" the subset are chosen from the executable nodes at lower levels. Of course, if there are more than $k-u$ nodes to choose from, those at the higher levels are selected first.

There is a simple counterexample to show that this algorithm does not produce an optimal subset schedule (and therefore certainly not an optimal PS) for arbitrary graphs when $k>2$.

If we have a graph of the form shown in Fig. 8 and $k \geq 3$ machines, then the subset sequence constructed by the above algorithm would be $\{q_1, q_2, \cdots, q_k, p_1, p_2, \cdots, p_{k-1}\}$, $\{r_1, r_2\}$, $\{s_1, s_2, \cdots s_{k-1}, u_1, u_2, \cdots, u_k\}$. The computation time for the corresponding subset schedule is $2k-1/k+1+2k-1/k = 5-2/k$. However, the subset schedule corresponding to $\{q_1, q_2, \cdots, q_k\}$, $\{p_1, p_2, \cdots, p_{k-1}r_1\}$, $\{s_1, s_2, \cdots, s_{k-1}, r_2\}$, $\{u_1, u_2, \cdots, u_k\}$ has computation time equal to four units which for $k>2$ is less than $5-2/k$.

Unfortunately, Theorem 1 is not true for $k>2$ processors. Consider the graph shown in Fig. 9 when there are k available processors.

We claim that the optimal PS for any value of k has length 3 and that this computation time can be achieved by a subset schedule for $G_{1/(k-1)^2}$ but not for $G_{1/m}$ when

[2] For n an integer, $G_{w/n}$ is formed from G by replacing each node by a series connection of nodes of weight w/n. A node of weight w_i is replaced by $n(w_i/w)$ nodes.

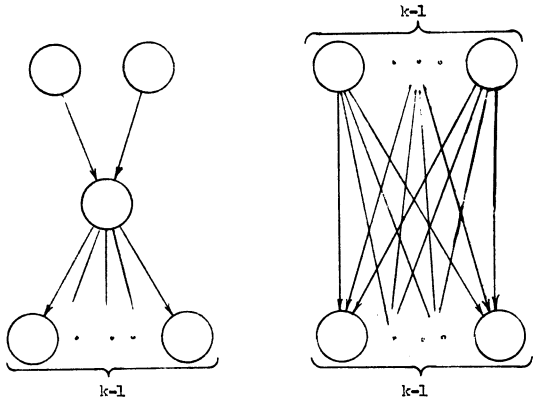


Fig. 9. A counterexample to Theorem 1 when $k>2$.

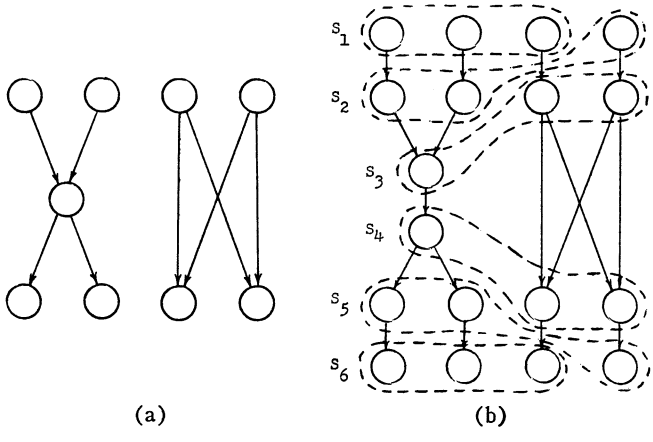


Fig. 10. The counterexample to Theorem 1 when $k=3$. (a) G when $k=3$. (b) $G_{\frac{1}{2}}$ when $k=3$.

$m<k-1$. For example, when $k=3$ we have the graph shown in Fig. 10(a). It is left to the reader to verify that no subset schedule for this graph will achieve a computation time of three units of time. On the other hand, a subset schedule for $G_{\frac{1}{2}}$ does give this result as illustrated in Fig. 10(b). It is conjectured that, in general, an optimal subset schedule for $G_{1/(k-1)!}$ is an optimal PS for G. Incidentally, this is equivalent to the conjecture that an optimal BS for $G_{1/k!}$ with k machines is an optimal PS for G with k machines. This follows from the fact that each task in a subset will be preempted only at integral multiples of $1/k$ times its execution time.

Conclusions

The problems of finding algorithms for determining optimal basic assignments or optimal preemptive assignments for arbitrary computation graphs and using an arbitrary number of processors are as yet unsolved. The present paper represents a positive step towards the preemptive assignment scheduling problem by giving a solution for the two-processor case.

The basic notions leading to the results of the present study are that of the subset sequence and the corresponding subset schedule. The present study is part of a larger research effort in which this relationship between subset assignments and optimal PSs is being explored in depth.

References

[1] A. J. Critchlow, "Generalized multiprocessing and multiprogramming systems," 1963 *Fall Joint Computer Conf., AFIPS Proc.*, vol. 24. Baltimore, Md.: Spartan, 1963, pp. 107–126.
[2] T. C. Hu, "Parallel sequencing and assembly line problems," *Operations Res.*, vol. 9, pp. 841–848. November 1961.
[3] R. L. Graham, private communications.
[4] R. R. Muntz and E. G. Coffman, Jr., "Multiprocessor scheduling of computation graphs," Computer Science Laboratory, Department of Electrical Engineering, Princeton University, Princeton, N. J., Rept. 63, June 1968.
[5] R. McNaughton, "Scheduling with deadline and loss functions," *Management Science*, vol. 6, pp. 1–12, October 1959.
[6] M. H. Rothkopf, "Scheduling independent tasks on parallel processors," *Management Science*, vol. 12. pp. 437–447. January 1966.
[7] R. R. Muntz, "Scheduling of computations on multiprocessor systems: the preemptive assignemnt discipline," Ph.D. dissertation, Princeton University, Princeton, N. J., April 1969.
[8] G. K. Manacher, "Production and stabilization of real-time task schedules," *J. ACM*, vol. 14, pp. 429–465, July 1967.
[9] D. F. Martin, "The automatic assignment and sequencing of computations on parallel processor systems," Ph.D. dissertation University of California, Los Angeles, 1966.
[10] R. R. Muntz and E. G. Coffman, Jr., "The optimization sequencing of computation graphs on two processor systems," Computer Science Laboratory, Department of Electrical Engineering, Princeton University, Princeton, N. J., Rept. 68, July 1968.