Memory-Constrained Task Scheduling on a Network of Dual Processors

KEN FUCHS

Iowa State University, Ames, Iowa

AND

DENNIS KAFURA

Virginia Polytechnic Institute, Blacksburg, Virginia

Abstract. One aspect of network design is the extent to which memory is shared among the processing elements. In this paper a model with limited sharing (only two processors connected to each memory) is analyzed and its performance compared with the performance of two other models that have appeared in the literature. One of these is a model of multiple processors sharing a single memory; the other model considers a multiprocessor configuration in which each processor has its own dedicated memory. The tasks processed by these networks are described by both time and memory requirements. The largest-memory-first (LMF) scheduling algorithm is employed and its performance with respect to an enumerative optimal scheduling algorithm is bounded. On the basis of this measure we conclude that memory sharing is only desirable on very small networks and is disadvantageous on networks of larger size.

Categories and Subject Descriptors: C.4 **[Computer Systems Organization]**: Performance of Systems—*modeling techniques*; D.4.1 **[Operating Systems]**: Process Management—*scheduling*; D.4.2 **[Operating Systems]**: Storage Management—*distributed memories, main memory*

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Deterministic scheduling, scheduling algorithms, memory management, largest-memory-first, shared memory, networks of processors

1. *Introduction*

An important element in the design of a computer network is the topology defining the interconnection of processors and memories. In some cases the nature of the network may limit the types of feasible interconnections. For example, in a geographically distributed network, there is usually no opportunity for any sharing of a common memory among separate nodes. On the other hand, a closely coupled network, oriented toward high-speed performance, may demand high-speed shared memory for rapid communication and easy sharing of a universal program and data space. For those networks in which the topology is not predetermined, the extent to which memory should be shared is an interesting and important design

Authors' addresses: K. Fuchs, Computer Science Department, Iowa State University, Ames, IA 50011; D. Kafura, Department of Computer Science, Virginia Polytechnic Institute, Blacksburg, VA 24061.

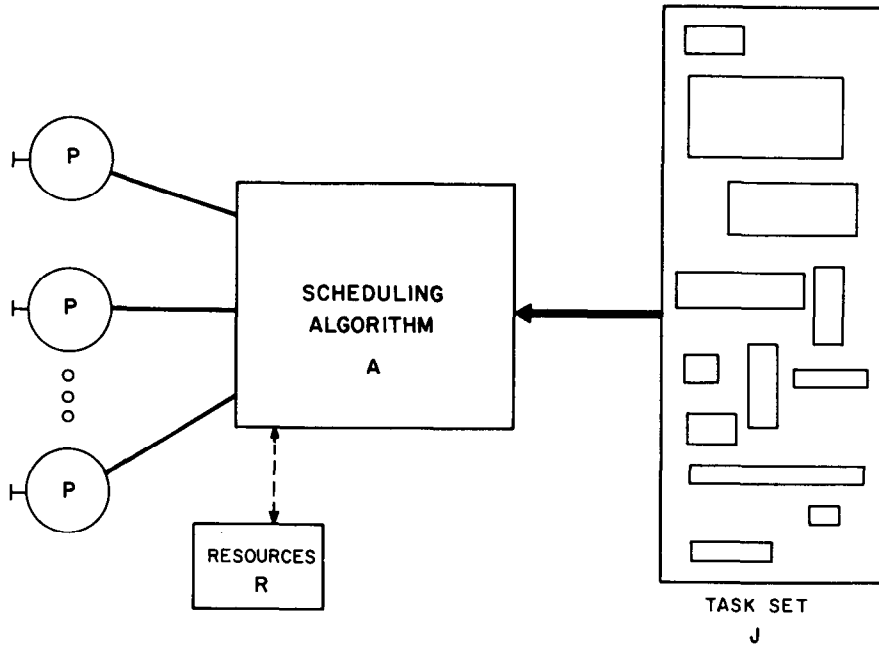


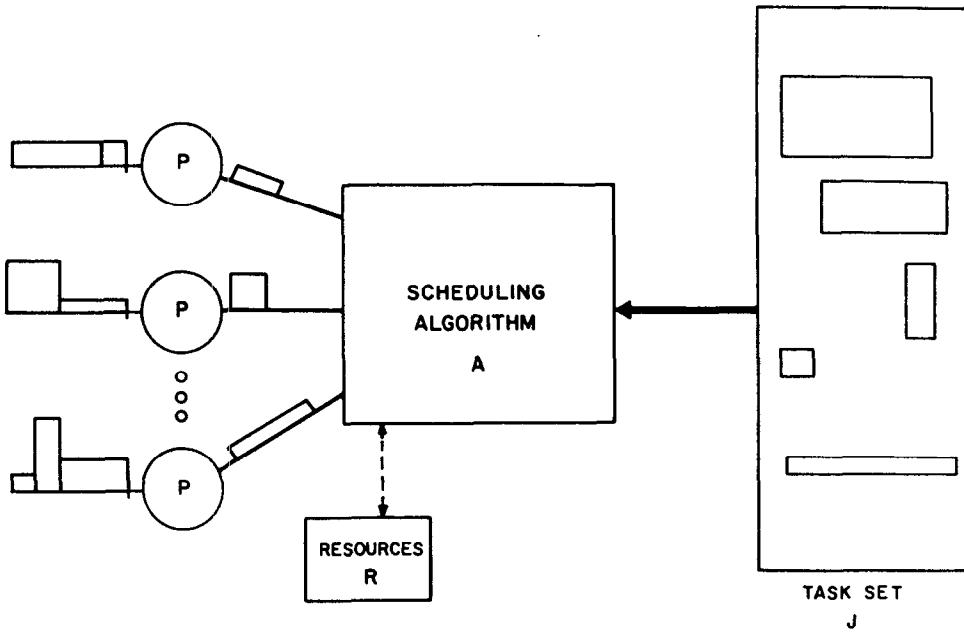
FIG. 1. A general task scheduling model.

problem. In this paper we investigate one aspect of this complex problem, namely, the impact of shared memory on job scheduling. The conclusion derived from this research is that memory should be shared only in the simple network with exactly two processors and should not be shared in any larger network.

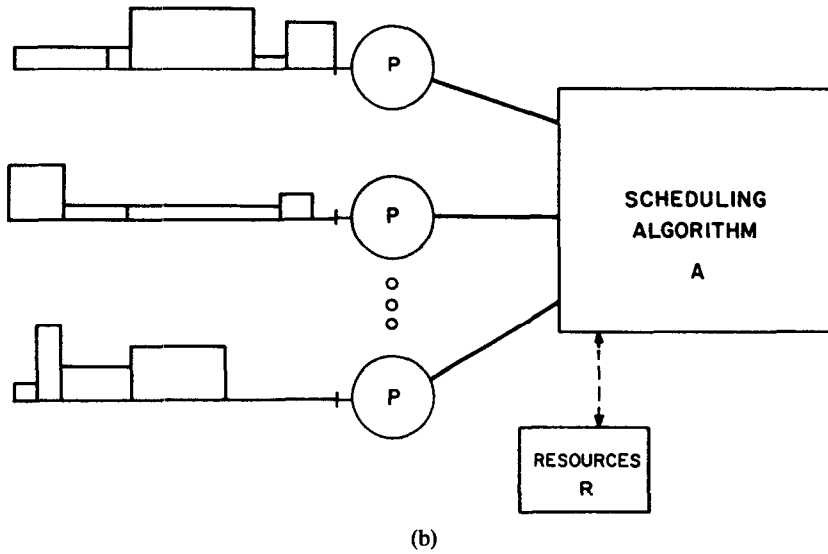
The evaluation of scheduling policies for computer networks may be approached in two basic ways. One approach characterizes the task behavior by simple stochastic processes and yields such performance measures as mean queue lengths, mean response times, and steady-state utilizations [2, 13]. A second approach, referred to as *deterministic scheduling* [1, 5], is used in this paper. It assumes that an arbitrary collection of tasks is initially present for scheduling. The tasks are arbitrary in that, although the exact values of the task parameters are known for this collection of tasks, the task parameter values are not constrained to yield a predetermined mean or fit a prescribed distribution. The performance metric used in this analysis is a bound on the time it takes for the network to complete the processing of the task set. This bound is expressed in relation to the optimal scheduling policy. Except for special cases the optimal scheduling policy is not practical because it requires an enumeration of all possible scheduling alternatives.

A generalized deterministic model is shown in Figure 1. The general model contains an arbitrary collection of known tasks J, a scheduling algorithm A, which determines the assignment of tasks to processors based on the task parameters and other system constraints, and a network Z, consisting of processing nodes P and possible other resources R.

An intermediate state in the execution of the task set is shown in Figure 2a. The scheduling algorithm, governed by the system constraints, removes tasks from the task set and assigns them to the input lines of available processors. When a processor completes a task, the task appears on the processor's output line. Collectively these output lines define a system schedule. The final system state is



(a)



(b)

FIG. 2. The general model. (a) An intermediate state. (b) The final state.

shown in Figure 2b in which all the tasks have been completed. Blank spaces on the output lines of a processor indicate periods when the processor was idle either because no task was available for processing at that time or because an available task could not be scheduled on the idle processor without violating a system constraint.

Within this general framework previous investigations have considered deterministic models with precedence constraints among simple tasks on two or more processors [6–8], preemptible tasks that are either mutually independent or have

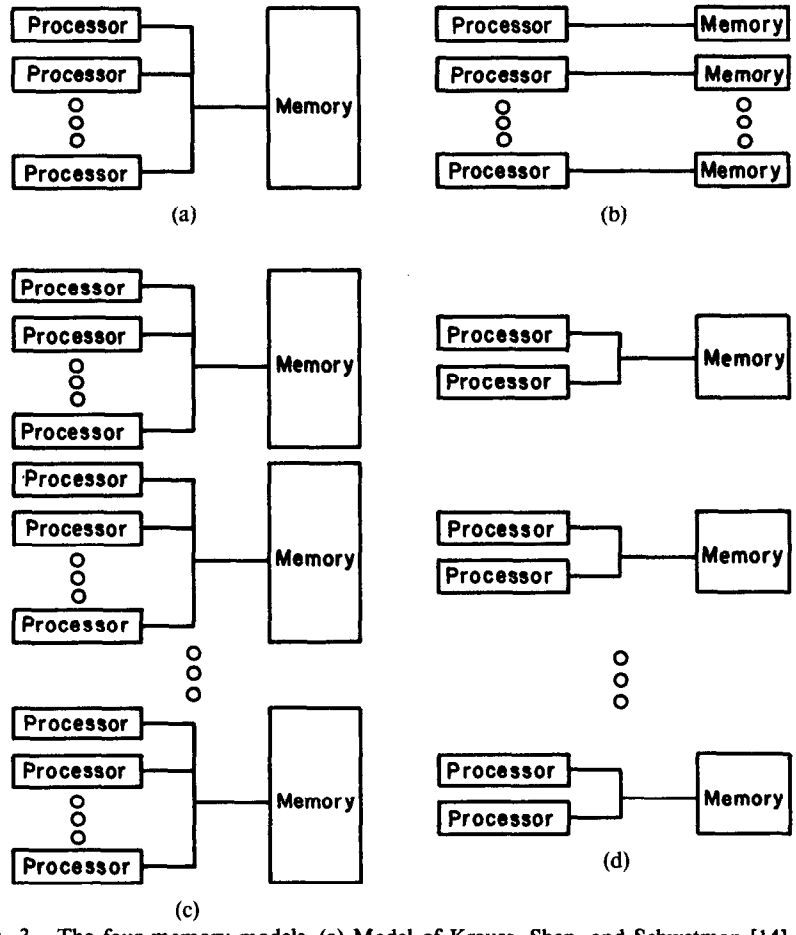


FIG. 3. The four memory models. (a) Model of Krause, Shen, and Schwetman [14]. (b) Model of Kafura and Shen [11]. (c) General composition of (a) and (b). (d) A specific composition of (a) and (b), the model this paper is most concerned with.

special precedence structures [3, 16, 17], and independent tasks with synchronization constraints [9, 10], and two (processor and memory) or more available resources in a multiprocessor network [4, 11, 12, 14]. This paper extends the results for this last class of models.

Two specific models with memory constraints have been analyzed. The first model, from Krause, Shen, and Schwetman [14], is pictured in Figure 3a. This model contains a single memory shared among all the processors in a necessarily local network. In practice such a system offers the advantage of flexible dynamic memory (re)allocation, but suffers from a possible memory contention, which can become severe as the number of processors increases. The second model, from Kafura and Shen [11, 12], is pictured in Figure 3b. This model assigns a private memory of a fixed, and possibly different, size to each processor. Although this configuration eliminates memory contention, it may also lead to low system utilization. This is because idle memory on one processor cannot be used by other processors that may be idle due to lack of sufficient memory to process available tasks with large memory requirements. These two previous models are similar because in each model we assume that the tasks are not preemptible and that each processor can execute only one task at a time.

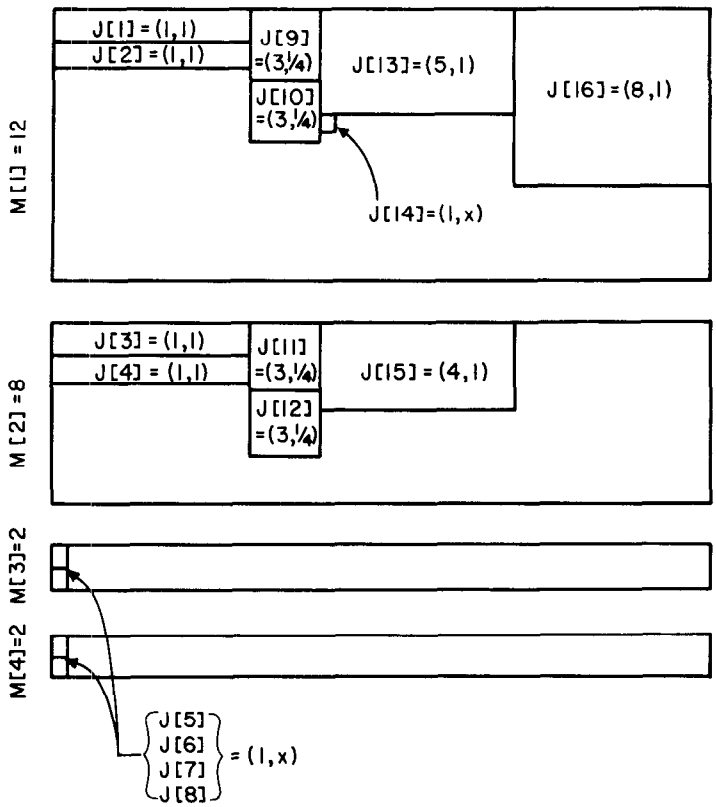


FIG. 4. Example of a random schedule. T[LMF] = $3\frac{1}{4}$, T[OPT] = $1 + x$, R[LMF] = $3\frac{1}{4}$ as $x \to 0$.

Figure 3c shows a network model that generalizes the two specific models described above. In this generalized model each node consists of several processors sharing a common memory. Extending the results of the earlier models to include the general network of multiprocessing stations pictured in Figure 3c would probably be an overly ambitious undertaking. Therefore, the model presented in this paper restricts each station to a dual processor configuration. This model is pictured in Figure 3d. We view this model as interesting in its own right and also as a necessary analytical bridge between the previous models and the more general network of multiprocessors.

The results available from the earlier analysis of models with memory constraints have shown that largest-memory-first (LMF) scheduling policy is the simplest algorithm with good performance. The LMF strategy schedules tasks in decreasing order of memory requirement and, among tasks of equal memory requirement, it uses an arbitrary but fixed selection rule. Other more complicated heuristics either did worse than LMF or resulted in a performance improvement that was not large enough to justify the added complexity of the algorithm and the accompanying analysis. To confirm that some heuristic method, like LMF, was the only scheduling policy of practical interest, we first constructed several example schedules that showed that an arbitrary (random) scheduling method could be ineffective. One of these examples is presented in Figure 4. For this task set, the ratio between the arbitrary schedule and optimal schedule is $3\frac{1}{4}$. The pattern illustrated in this example

can be generalized to show that the bound for an arbitrary schedule is greater than $1 + \log(n)$, where n is the number of dual processors in the network. We also conjectured that if each memory were considered a separate resource in the sense of [4], then the bound could be no worse than $n + 1$. However, bounds in this range compared unfavorably to the results using LMF in previous models. On the basis of this experience the LMF algorithm is used throughout this paper.

It is useful to compare the specific bounds on the LMF algorithm established for the two previous models. In this statement of the results T[LMF] is used to denote the completion time of the LMF algorithm and T[OPT] the completion time of the optimal algorithm. These results are:

Result 1 [14]

$$\frac{T[\text{LMF}]}{T[\text{OPT}]} < 3 - \frac{3}{n},$$

where n is the number of processors sharing a common memory.

Result 2 [11]

$$\frac{T[\text{LMF}]}{T[\text{OPT}]} < 2 - \frac{1}{n},$$

where n is the number of processors each with a private memory.

The next section describes the details of the model considered in this paper, including the notation and certain important definitions used throughout this paper.

2. *Definitions*

A general, deterministic scheduling model with memory constraints is defined here in detail. The definitions used in the lemmas and theorems of Section 3 follow the model definition.

2.1. Model Definition. A network consists of a vector N, of n computers, where each computer $Q[i]$ consists of a vector $P[i]$, of m processors $p[i, j]$, and a private memory of size $M[i]$. The computers are ordered within the vector N by decreasing memory $M[i]$ such that $M[i] \geq M[i + 1]$ for $1 \leq i < n$. The processors $p[i, j]$ are identical; thus the computers are distinct only in $M[i]$. The set of tasks J to be executed by the network consists of k independent tasks $J[r]$, each with resource requirement $(m[r], t[r])$, where $m[r]$ and $t[r]$ represent the memory and processor time requirements, respectively. (Note: The reader should not confuse m, the number of processors per computer, with $m[r]$, the memory requirement of task $J[r]$.) The resource requirements must be specified in integral units of time and memory. This restriction is mild since arbitrarily small units of time or memory may be chosen. This restriction has been used in at least one previous model [4] and the choice of the unit does not affect the results. The vector L, any permutation of the k tasks, represents an arbitrary task list of the tasks.

The various components of the task-scheduling model are summarized as follows:

Network:	$N = (Q[1], Q[2], \ldots, Q[n])$	$n \geq 2.$
Computer:	$Q[i] = (M[i], P[i])$	$1 \leq i \leq n.$
Private memory:	$M[i] =$ [Integral units of memory]	$1 \leq i \leq n;$
	$M[i] \geq M[i + 1]$ for	$1 \leq i < n.$

Processor vector:	$P[i] = (p[i, 1], p[i, 2], \ldots, p[i, m])$ $1 \le i \le n$ and $m \ge 2$.
Processor:	$p[i, j]$ = [Unique processor label] $1 \le i \le n$ and $1 \le j \le m$.
Task set:	$J = \{J[1], J[2], \ldots, J[k]\}$ k mutually independent tasks.
Task list:	$L = (J[i[1]], J[i[2]], \ldots, J[i[k]])$ $\{i[1], i[2], \ldots, i[k]\} = \{1, 2, \ldots, k\}$.
Task:	$J[r] = (m[r], t[r])$ $1 \le r \le k$.
Memory requirement:	$m[r]$ = [Integral units of memory] $1 \le r \le k$ and $m[r] > 0$.
Time requirement:	$t[r]$ = [Integral units of time] $1 \le r \le k$ and $t[r] > 0$.

The model works according to the following rule. When $p[i, j]$ is not executing a task, it instantaneously scans the task list from left to right until it finds an uninitiated task $J[r]$ such that $m[r] \le G[i](T)$, where $G[i](T)$ is the amount of unused memory in computer $Q[i]$ at time T. $J[r]$ is marked as initiated in L, and $p[i, j]$ executes $J[r]$ for $t[r]$ units of time and then marks $J[r]$ in L as terminated. If two or more processors find the same task, the processor with the smallest value $h = im + j$ executes the task and the other processor(s) continues the search through L. If a processor $p[i, j]$ initiates task $J[r]$ at time T, then $G[i](T) = G[i](T) - m[r]$. If a processor $p[i, j]$ terminates task $J[r]$ at time T, then $G[i](T) = G[i](T) + m[r]$. Note that $G[i](T)$ may "change" up to $2m$ times, m terminations, and m initiations, during the instantaneous scan of the task list.

At time $T = 0$, all tasks are uninitiated, all $p[i, j]$ are idle, $G[i](T) = M[i]$ for $1 \le i \le n$, and the previous rule is applies. The task set is executed to completion at time $T > 0$, when all tasks in L are marked terminated. Let this value of T be denoted $T[L]$, the time required to execute task set J using task list L on network N.

The optimal schedule OPT is the schedule that has the minimum $T[L]$ over all possible L. Let $T[\text{OPT}]$ denote the optimal completion time of task set J on network N, where $T[\text{OPT}] = \min(T[L])$ over all L and let $R[L] = T[L]/T[\text{OPT}]$ be the ratio of completion times of schedule L over schedule OPT.

The problem being considered is, "What are the bounds on $R[L]$, with L = LMF and a varying number of computers n, where the number of processors m within each computer is 2?"

2.2. Time Slice and Time Slice Partition Definitions. Having just completed a description of the model, we now begin the analysis of this model by defining several important concepts.

Time slice. A time slice $h(T, Q[i], N, L)$ is the set of tasks executing during the half open interval $[T, T + 1)$ on computer $Q[i]$ of network N, using task list L. Hereafter, $h(T, Q[i], N, L)$ will be designated $h[i]$ where L = LMF, T is specified and in the range $[0, T[L]]$, and N is known by context. Note: Only the memory requirements of the tasks in $h[i]$ are relevant in the use of this definition; also $|h[i]| \le |P[i]| = m$.

Last task. Let s denote the index of the last task to complete on network N using task list L. If more than one task completes at the end of the schedule, let s be the index of the one with the greatest memory requirement.

Partition of the time slices. The time slices on $Q[1]$ are divided into B (big task) slices and D (double task) slices. The D slices are further divided into E (epsilon

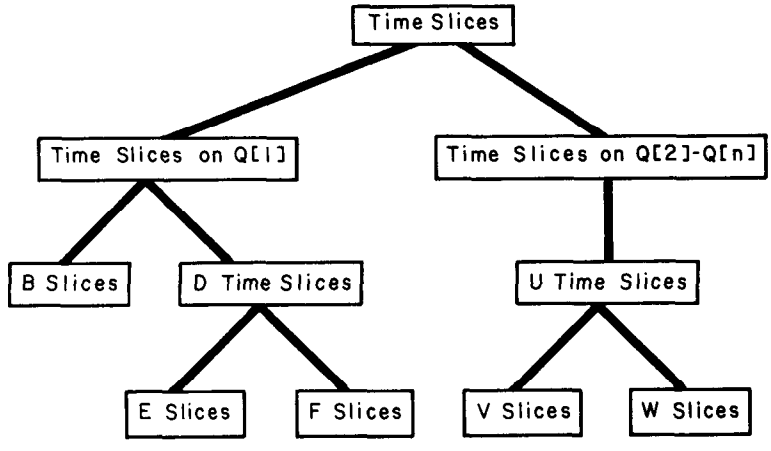


FIG. 5. Time slice partition tree.

task) slices and F (fat task) slices. The time slices on $Q[k]$ for $2 \leq k \leq n$ are called U[k] (Universe of $Q[k]$) slices.

For Lemma 2, V[k] (single task) slices and W[k] (double task) slices form a partition of the U[k] slices. Figure 5 illustrates, in tree form, the partition of the time slices.

Formal definition of the time slice partitions. The following definitions are required for the theoretical analysis which follows in the next section:

$$\begin{aligned}
\mathrm{B} &= \{h[1] \mid J[s] \notin h[1] \text{ and there exists } J[r] \in h[1] \mid m[r] + m[s] > M[1]\},\\
\mathrm{D} &= \{h[1] \mid J[s] \notin h[1] \text{ and for all } J[r] \in h[1],\ m[r] + m[s] \leq M[1]\},\\
\mathrm{E} &= \{h[1] \mid h[1] \in \mathrm{D} \text{ and there exists } J[r] \in h[1] \mid m[r] < m[s]\},\\
\mathrm{F} &= \{h[1] \mid h[1] \in \mathrm{D} \text{ and for all } J[r] \in h[1],\ m[r] \geq m[s]\},\\
\mathrm{U}[i] &= \{h[i] \mid J[s] \notin h[i]\} \qquad \text{where } 2 \leq i \leq n,\\
\mathrm{V}[i] &= \{h[i] \mid h[i] \in \mathrm{U}[i] \text{ and } |h[i]| = 1\} \qquad \text{where } 2 \leq i \leq n,\\
\mathrm{W}[i] &= \{h[i] \mid h[i] \in \mathrm{U}[i] \text{ and } |h[i]| = 2\} \qquad \text{where } 2 \leq i \leq n.
\end{aligned}$$

For each of the seven sets defined above, the small letter is used to denote the size of that set (e.g., $\mathrm{b} = |\mathrm{B}|$). Figure 6 is an example of the various slices in a LMF schedule.

3. *Results*

In this section, various upper bounds on the ratio of the completion times of LMF and OPT schedules, denoted $R[\mathrm{LMF}] = T[\mathrm{LMF}]/T[\mathrm{OPT}]$, are established. The network N contains n computers $Q[i]$, each having $m = 2$ processors $p[i, 1]$ and $p[i, 2]$, and an arbitrary amount of memory $M[i]$. The two general upper bounds are given in Theorems 1 and 2. The cases in which these upper bounds are known to be the least upper bounds are noted and proved through example schedules in the corollaries of the two theorems. For the other cases, where the least upper bounds are unknown, an interval that contains the least upper bound is given. The upper point of the interval is given by one of the theorems and the lower point of the interval is given by example schedules.

The upper bounds on R[LMF], considered here, are functionally dependent on the task set J and the network $N(m = 2)$ on which J is scheduled. Specifically, the bounds are dependent on the number of computers n and the relationship between the memory requirement of the last task to terminate ($m[s]$) and the size of the

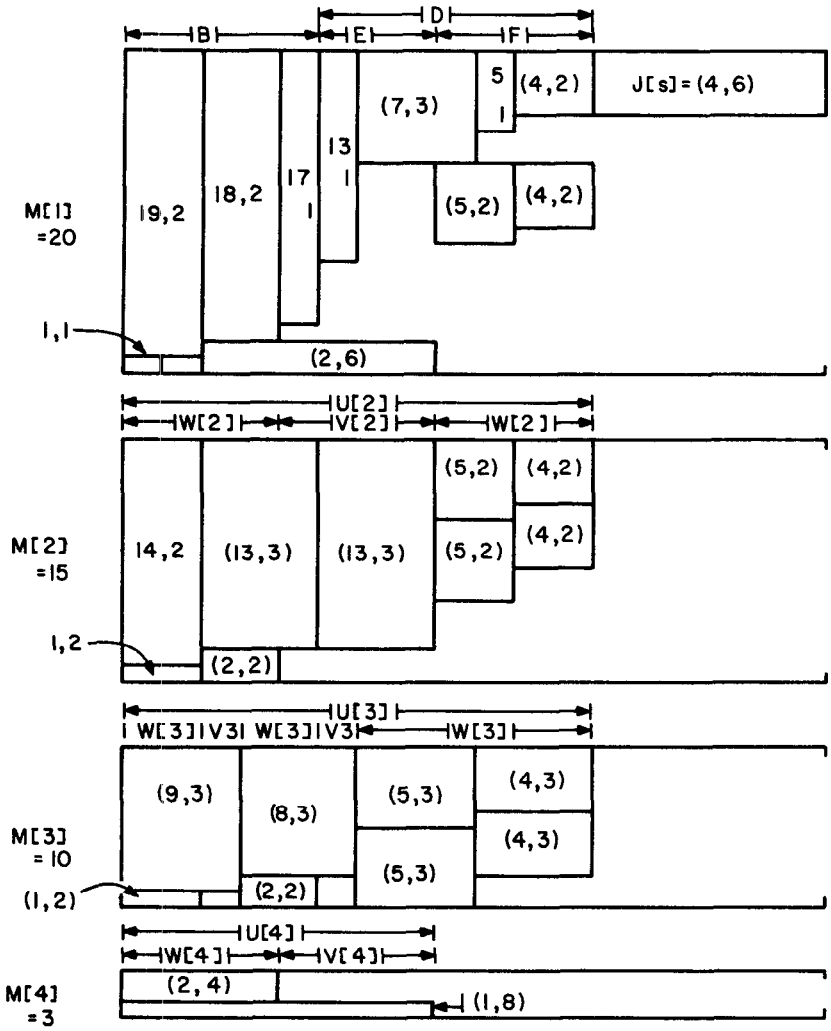


FIG. 6. Example of LMF scheduling and different time slices.

smallest memory in the network ($M[n]$). Let $UB(n, m[s], M[n])$ be the function giving the upper bound on R[LMF]. Therefore, the two theorems show that

$$UB(n, m[s], M[n]) = \begin{cases} \dfrac{3n-2}{n} & \text{if } m[s] > M[n], \\ \dfrac{3n}{n+1} & \text{otherwise.} \end{cases}$$

This section is divided into two parts, corresponding to the two conditions in the above definition of UB. The first part is concerned with results when $m[s] > M[n]$, and the second when $m[s] \leq M[n]$.

The examples used in the corollaries produce a particular value of the function:

$$R[\text{LMF}](N, J) = \frac{T[\text{LMF}](N, J)}{T[\text{OPT}](N, J)}.$$

The function R[LMF] is evaluated for network N and task set J by constructing both the LMF schedule and OPT schedule and taking the ratio of the two schedule lengths, T[LMF]/T[OPT]. In the cases in which the least upper bound of R[LMF] is unknown, the example used in the corollary may not be the best; another example may have a larger value of R[LMF]. If so, the interval containing the least upper bound should be smaller than the one given in this paper.

3.1. General Bound on R[LMF] When $m[s] > M[n]$. Before stating and proving the general bound when $m[s] > M[n]$, we shall prove two lemmas. Lemma 1, is used in the proof of Lemma 2, which itself is used in the proof of Theorem 1. These lemmas are true whether or not $m[s] > M[n]$.

Lemma 1. *In a LMF schedule, B, E, and F are sets of consecutive time slices on Q[1] such that all B slices precede all E slices and all E slices precede all F slices.*

Proof. We first show that all B slices precede all D slices. By definition, each B slice contains a task $J[x]$ such that $m[x] + m[s] > M[1]$. Also by definition, the larger task $J[y]$ on each E slice must have the memory requirement $m[y]$ such that $m[y] + m[s] \leq M[1]$. This implies that $m[x] > m[y]$. That is, each B slice contains a task whose memory requirement is larger than that of any task in each D slice. Thus, in any valid LMF schedule all B slices must precede all D slices.

We now show that, within the D slices, all E slices precede all F slices. Suppose, by contradiction, that an F slice immediately precedes an E slice, both of which precede the last task $J[s]$. By definition, the small task on the E slice has a memory requirement $m[r]$ such that $m[r] < m[s]$. The smaller task on the E slice cannot be one of the two tasks executing during the previous F slice, since the memory requirement of both of those tasks is not less than $m[s]$. The larger task of the E slice can be scheduled with $J[s]$, since any task in a D slice can be scheduled with $J[s]$. However, the smaller task on the E slice was scheduled in preference to $J[s]$ even though $J[s]$ has the larger memory requirement. This is a contradiction to LMF scheduling, thus all E slices precede all F slices.

The time slices in B, E, and F are consecutive since all B slices precede all D slices, all E slices precede all F slices within the D slices, and B + E + F covers $M[1]$ up to $J[s]$. □

Lemma 2. *A single small task $J[e]$ is an element of all E time slices, $e < t[e] \leq T[OPT]$, and $J[e]$ is initiated in a B time slice.*

Proof. By Lemma 1, the E slices are consecutive; thus it is possible that J[e] is an element of all E slices.

The small task in each E slice is initiated in a B slice. Suppose, by contradiction, a small task $J[r]$ in an E slice is initiated in another E slice h[e]. By the definition of an E slice, $J[s]$ can be scheduled with the large task of h[e] and $m[r] < m[s]$. Scheduling $J[r]$ in preference to $J[s]$ is a clear contradiction to LMF scheduling.

Since the small task in each E slice must be initiated in a B slice, there can be only one such task, J[e]. Thus J[e] is an element of all E slices.

Since J[e] is an element of all E slices and at least one B slice, then e < t[e].

T[OPT], must be at least as long as any task scheduled; thus t[e] ≤ T[OPT]. □

With Lemma 2 established, we can now prove the following theorem.

Theorem 1. *Given an LMF schedule of the task set J on N, $m = 2$, satisfying $m[s] > M[n]$, then $R[LMF] < (3n - 2)/n$.*

PROOF. Let k be the largest i such that $M[i] \geq m[s]$. A lower bound on the optimum schedule can be found by determining a lower bound on a schedule of tasks J–S on the first k computers where S, a subset of J, is the set of all tasks whose memory requirement is less than $m[s]$. (For the purpose of constructing this bound, tasks of sizes less than $m[s]$ are deleted from the task set.) The large B task slices clearly use b processor slices; a task larger or equal to $m[s]$ cannot be scheduled with these tasks as explained in the definition of a B slice. The remaining task slices are scheduled as two for every processor slice, giving $(e + 2f + u[2] + u[3] + \cdots + u[k] + t[s])/2$ processor slices (ignoring the second task of a W slice). Adding processor slices, the bound is found to be

$$k \cdot T[\text{OPT}] \geq b + \frac{e + 2f + u[2] + u[3] + \cdots + u[k] + t[s]}{2}. \tag{1}$$

The following bounds on the LMF schedule are used with the bound on $T[\text{OPT}]$ in inequality (1) to prove the theorem directly:

$$T[\text{LMF}] \leq b + e + f + t[s], \quad T[\text{LMF}] \leq u[2] + t[s],$$
$$T[\text{LMF}] \leq u[3] + t[s], \cdots, \quad T[\text{LMF}] \leq u[k] + t[s].$$

Adding and subtracting $(e + k \cdot t[s])/2$ from the right side of inequality (1) and multiplying by 2 gives inequality (2):

$$2k \cdot T[\text{OPT}] \geq 2b + 2e + 2f + 2t[s] - e - t[s] + u[2] + u[3] + \cdots + u[k] + (k-1)t[s] - (k-1)t[s]. \tag{2}$$

Applying the previously mentioned upper bounds on $T[\text{LMF}]$ to inequality (2) results in inequality (3):

$$2k \cdot T[\text{OPT}] \geq 2T[\text{LMF}] - e - t[s] + (k-1)T[\text{LMF}] - (k-1)t[s]. \tag{3}$$

Using the inequality $e < t[e] \leq T[\text{OPT}]$ from Lemma 2 and the fact that $t[s] \leq T[\text{OPT}]$, transform inequality (3) into inequality (4):

$$\begin{aligned} 2k \cdot T[\text{OPT}] &> (k+1)T[\text{LMF}] - (k+1)T[\text{OPT}] \\ (3k+1)T[\text{OPT}] &> (k+1)T[\text{LMF}] \\ \frac{3k+1}{k+1} &> \frac{T[\text{LMF}]}{T[\text{OPT}]} = R[\text{LMF}]. \end{aligned} \tag{4}$$

Combining $(3n-2)/n \geq \max[i = 1, 2, \ldots, n-1]((3i+1)/(i+1)) \geq (3k+1)/(k+1)$ and inequality (4) proves Theorem 1. (The domain of k is the set of integers 1 through $n-1$; this excludes n since a condition of the theorem is $m[s] > M[n]$.) □

The worst case differences between the least upper bound on $R[\text{LMF}]$ and the upper bound of Theorem 1, $(3n-2)/n$, are indicated by Corollaries 1–3 and Table I. Corollary 1 shows that $(3n-2)/n$ is the least upper bound on $R[\text{LMF}]$, for networks with 2 to 4 computers. Corollaries 2 and 3 give an interval containing the least upper bound for networks with 5 to 20 computers. Table I summarizes the intervals and their relative sizes for networks with 5–376 computers. The task sets 1 and 2 in the table are those in Corollaries 2 and 3. The remaining task sets do not appear in the paper because of length considerations.

COROLLARY 1. *For the LMF schedules of all task sets J on N, with $2 \leq n \leq 4$ and $m = 2$, satisfying $m[s] > M[n]$, $(3n-2)/n$ is the least upper bound on $R[LMF]$.*

TABLE I. COMPARISON OF WORST-KNOWN R[LMF]'S AND $(3n - 2)/n$

Task set number	Lower bound on least upper bound on R[LMF] (worst-known R[LMF])	Limitations for bound valid for n	Example value of n	Difference and % difference between column 2 and $(3n - 2)/n$	
				Difference	% difference
1	$(6n + 1)/(2n + 2)$	5, 9	8	1/36	1.010
2	$(12n + 13)/(4n + 8)$	10, 20	16	1/36	0.966
3	$(24n + 49)/(8n + 24)$	21, 43	32	11/560	0.669
4	$(48n + 145)/(16n + 64)$	44, 90	64	15/1088	0.464
5	$(96n + 385)/(32n + 160)$	91, 185	128	7/1064	0.220
6	$(192n + 961)/(64n + 384)$	186, 376	256	15/4192	0.120
k	$(3 \cdot 2^k(n + k - 1)/(2^k(n + k))$	$3 \cdot 2^k - k$, $3 \cdot 2^{(k+1)} - k - 2$	—	—	—

PROOF. An upper bound on R[LMF] is $(3n - 2)/n$ by Theorem 1. This bound may be approached by the R[LMF]'s of the three pairs of schedules in Figure 7 by allowing x to approach infinity.

The sizes of the memories in the three networks are as follows:

Network 1 ($n = 4$): $M[1] = 16$, $M[2] = 8$, $M[3] = 4$, and $M[4] = 1$.
Network 2 ($n = 3$): $M[1] = 8$, $M[2] = 4$, and $M[3] = 1$.
Network 3 ($n = 2$): $M[1] = 4$ and $M[2] = 1$. □

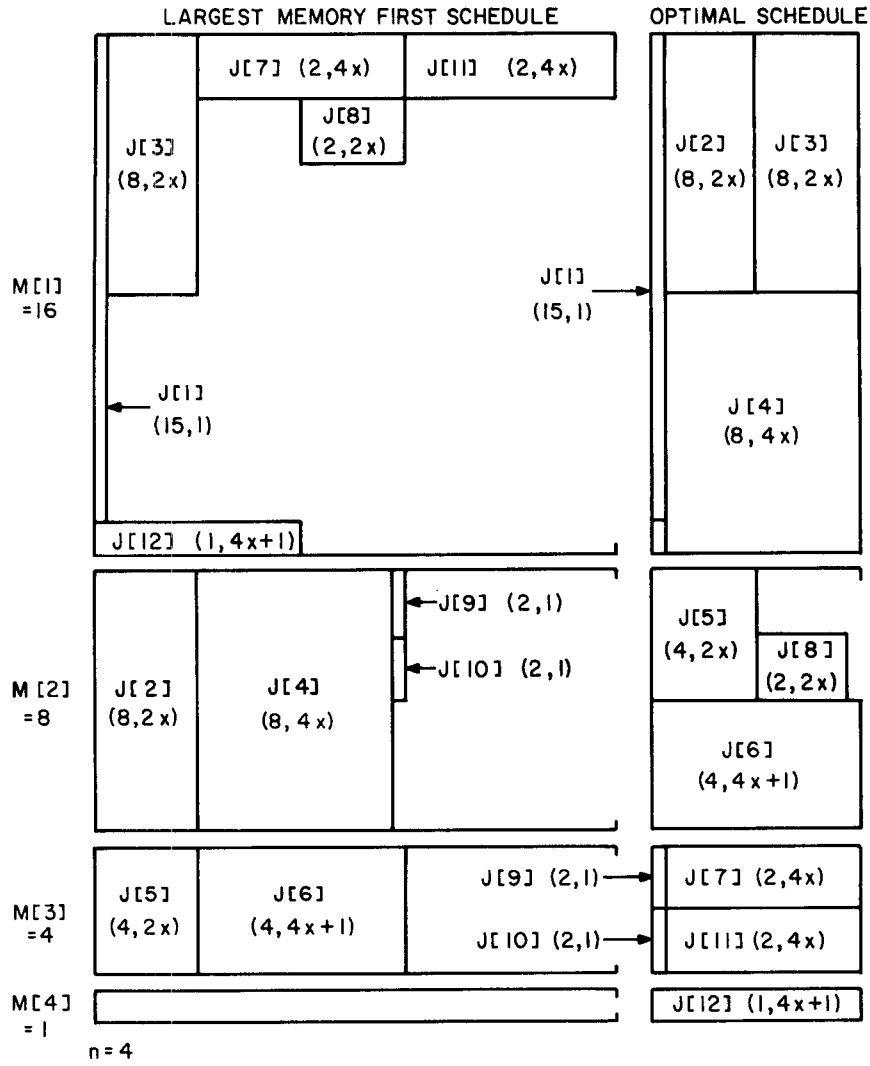
COROLLARY 2. *For the LMF schedules of all task sets J on N, with $5 \le n \le 9$ and $m = 2$, satisfying $m[s] > M[n]$, the least upper bound on R[LMF] lies in the interval*

$$\left[\frac{6n + 1}{2n + 2}, \frac{3n - 2}{n}\right].$$

PROOF. An upper bound on R[LMF] is $(3n - 2)/n$ by Theorem 1. Hence, the least upper bound on R[LMF] is less than or equal to $(3n - 2)/n$, the upper bound of the interval. The pair of schedules in Figure 8 has an R[LMF] that approaches $(6n + 1)/(2n + 2)$ as x approaches infinity. Therefore, the least upper bound on R[LMF] is greater than or equal to $(6n + 1)/(2n + 2)$, the lower bound of the interval.

The sizes of the n memories and $k = 7n - 15$ tasks used in Figure 8 are

$M[i] = 2^{n-i+1}$ for $1 \le i < n$, $M[n] = 1$,
$J[1] = (2^n - 1, 1)$, $J[2] = (2^{n-1}, (n + 1)x)$,
$J[3] = (2^{n-1}, (n + 1)x)$, $J[4] = (2^{n-1}, 2(n + 1)x)$,
$J[5] = (2^{n-2}, (n - 4)x)$,
$J[4 + 2j] = (2^{n-j-1}, (2n - 3)x)$ for $1 \le j \le n - 3$,
$J[5 + 2j] = (2^{n-j-1}, 2(n + 1)x + 1)$ for $1 \le j \le n - 3$,
$J[2n] = (2, 2(n + 1)x)$,
$J[2n + j] = (2, x)$ for $1 \le j \le 5(n - 4)$,
$J[7n - 19] = (2, 1)$, $J[7n - 18] = (2, 1)$,
$J[7n - 17] = (1, 2(n + 1)x)$, $J[7n - 16] = (2, 2(n + 1)x)$,
$J[7n - 15] = (2, (9 - n)x + 1)$. □



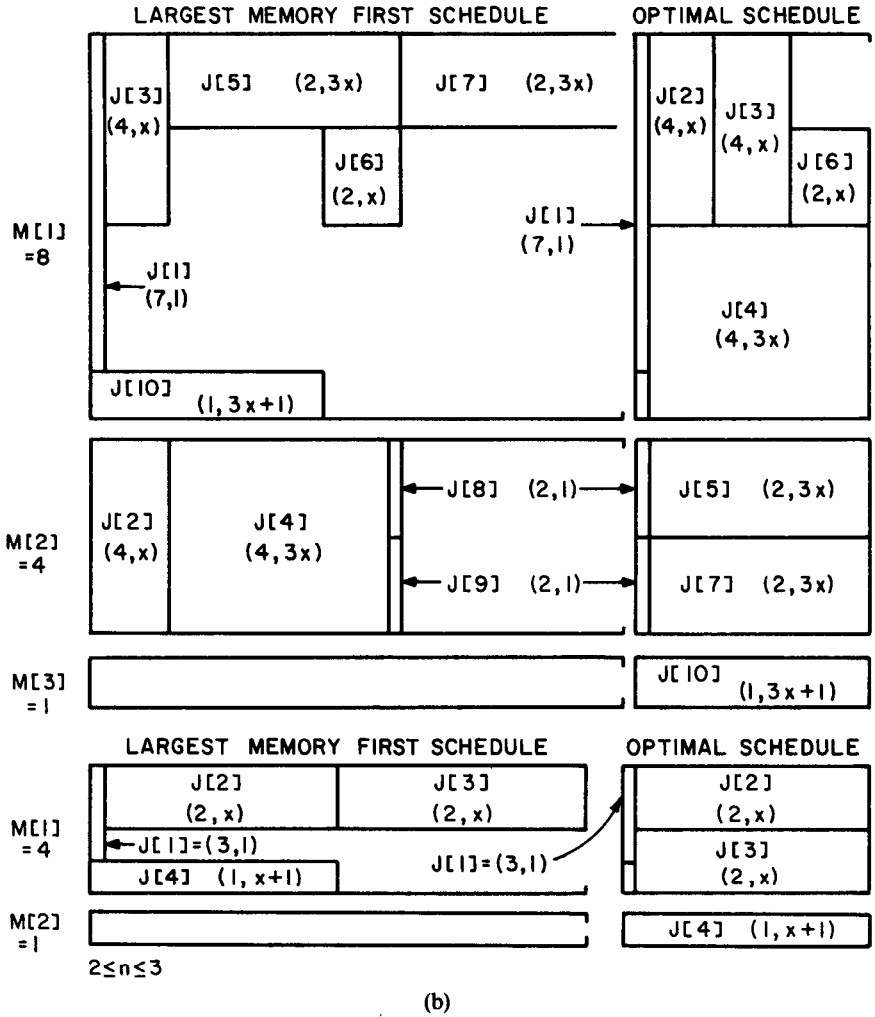
(a)

FIG. 7. Example schedules for Corollary 1.

COROLLARY 3. *For the LMF schedules of all task sets J on N, with* $10 \leq n \leq 20$ *and* $m = 2$, *satisfying* $m[s] > M[n]$, *the least upper bound on* $R[LMF]$ *lies in the interval*

$$\left(\frac{12n + 13}{4n + 8}, \frac{3n - 2}{n}\right).$$

PROOF. An upper bound on R[LMF] is $(3n - 2)/n$ by Theorem 1. Hence, the least upper bound on R[LMF] is less than or equal to $(3n - 2)/n$, the upper bound of the interval. The pair of schedules in Figure 9 has an R[LMF] that approaches $(12n + 13)/(4n + 8)$ as x approaches infinity. Therefore, the least upper bound on R[LMF] is greater than or equal to $(12n + 13)/(4n + 8)$, the lower bound of the interval.



(b)

FIG. 7. Continued.

The sizes of the n memories and $k = 13n - 49$ tasks used in Figure 9 are

$M[i] = 2^{n-i+1}$ for $1 \le i < n$, $M[n] = 1$,
$J[1] = (2^n - 1, 1)$, $J[2] = (2^{n-1}, 2(n+2)x)$,
$J[3] = (2^{n-1}, 2(n+2)x)$, $J[4] = (2^{n-1}, 4(n+2)x)$,
$J[5] = (2^{n-2}, (n+2)x)$, $J[6] = (2^{n-2}, 3(n+2)x)$,
$J[7] = (2^{n-3}, (n-9)x)$, $J[8] = (2^{n-2}, 4(n+2)x + 1)$,
$J[7 + 2j] = (2^{n-j-2}, (4n-3)x)$ for $1 \le j \le n - 4$,
$J[8 + 2j] = (2^{n-j-2}, 4(n+2)x + 1)$ for $1 \le j \le n - 4$,
$J[2n + j] = (2, x)$ for $1 \le j \le 11(n - 5)$,
$J[13n - 54] = (2, 1)$, $J[13n - 53] = (2, 1)$,
$J[13n - 52] = (1, 4(n+2)x)$, $J[13n - 51] = (2, 4(n+2)x)$,
$J[13n - 50] = (2, 4(n+2)x)$, $J[13n - 49] = (2, (20 - n)x + 1)$. □

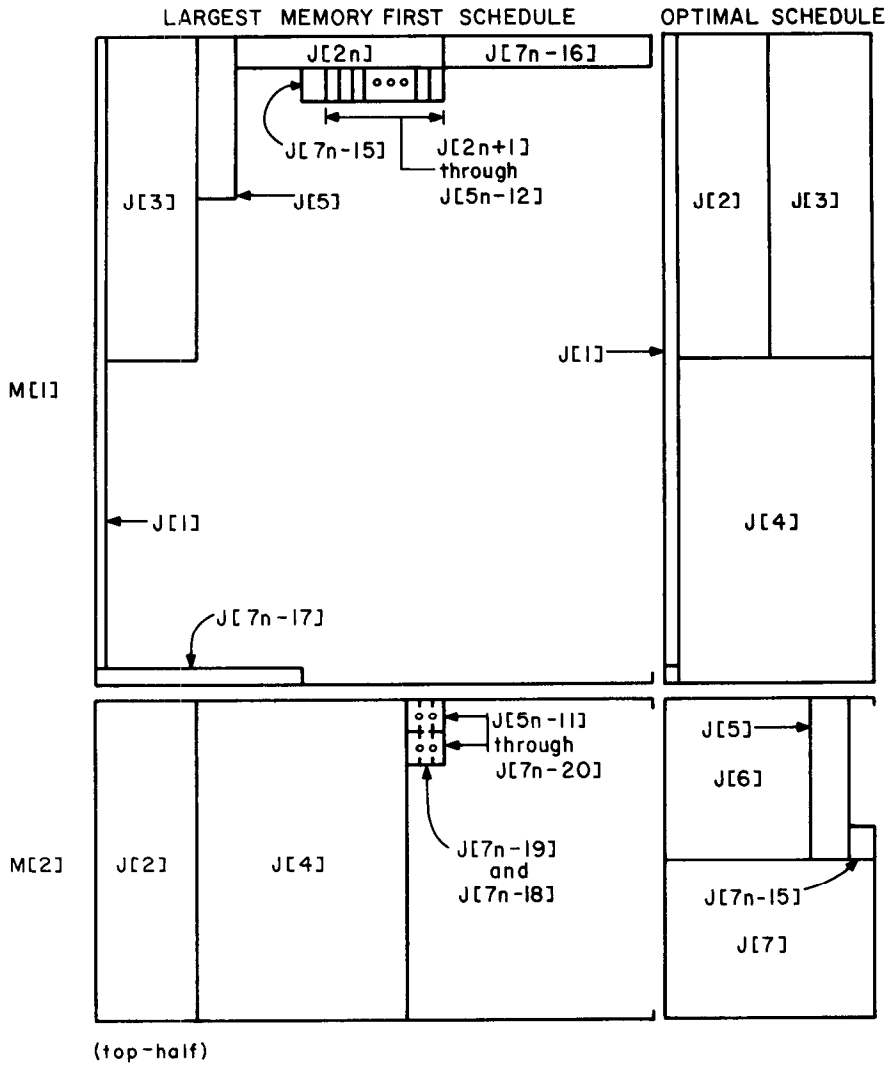


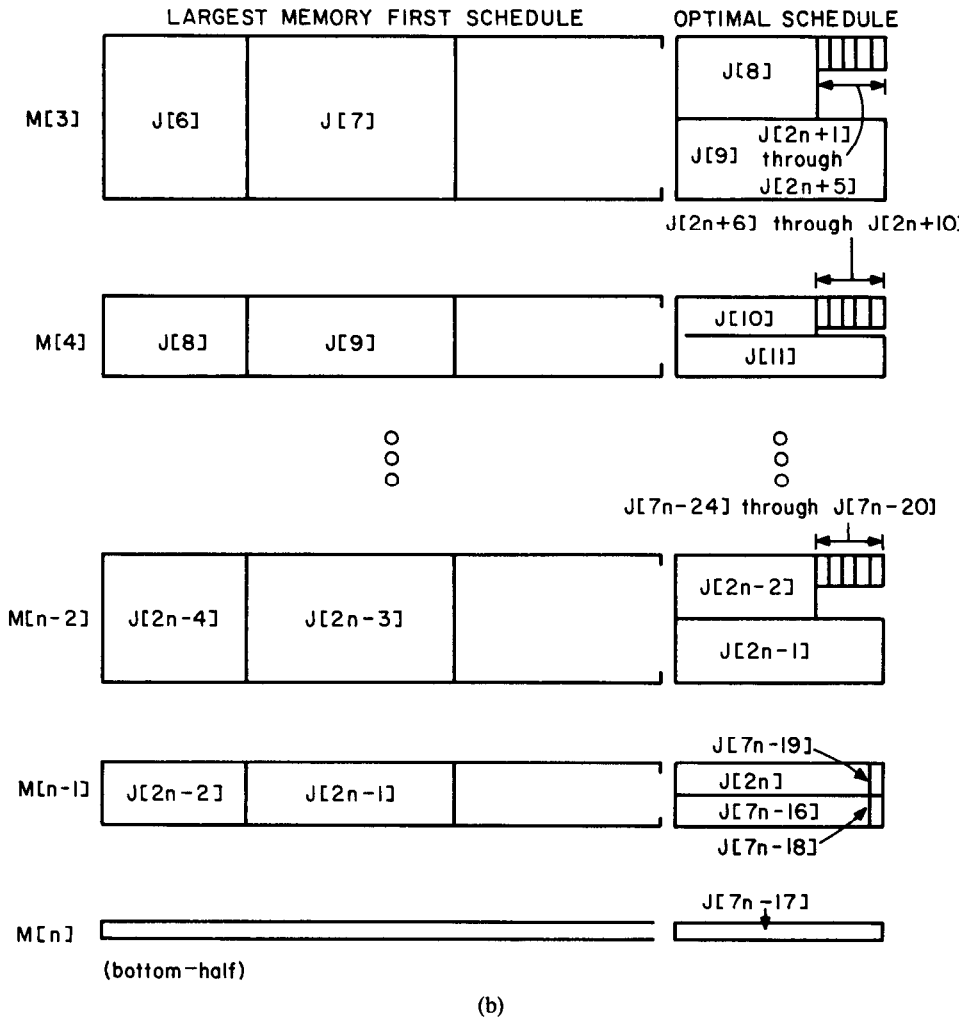
FIG. 8. Example schedules for Corollary 2.

3.2. GENERAL BOUND ON R[LMF] WHEN $m[s] \leq M[n]$. Before stating and proving the general bound when $m[s] \leq M[n]$, we shall prove Lemma 3, which is used in the proof of Theorem 2.

LEMMA 3. *For every LMF schedule on N, with* $m = 2$, *that satisfies* $m[s] \leq M[n]$

$$n \cdot T[OPT] \geq b + d + \frac{u + t[s]}{2}.$$

The general idea of the proof is to determine this lower bound on the optimum schedule by producing an upper bound on the number of task slices that can be scheduled with large tasks of B slices, and assuming the remaining task slices are scheduled two per processor slice.



(b)

FIG. 8. Continued.

PROOF. We show that, in any schedule, the large task in any B slice may only be scheduled with the small task, $J[r]$, of a B, E, or W[i] slice. Since $m[s] \leq M[n]$, the task $J[r]$, which must have $m[r] < m[s]$ by the definition of B slice, may not be the large task in any slice by LMF scheduling. (The large task in any slice must be larger than or equal to $m[s]$.) By the definition of F slice, the small task of an F slice has a memory requirement greater or equal to $m[s]$ and therefore may not be the task $J[r]$ with $m[r] < m[s]$. (The single task of the V[i] slice is excluded, because it is the large task of the V[i] slice, which has already been excluded.)

Following this argument, small B, E, and W[i] task slices may contribute to the bound on the number of task slices that can be scheduled with large tasks of B slices.

We show that the upper bound on the number of task slices that can be scheduled with large tasks of B slices is $z + w$ where z is the number of B slices with a small task. The proof for the case of $e = 0$ is trivial. The proof for the case of $e > 0$ follows.

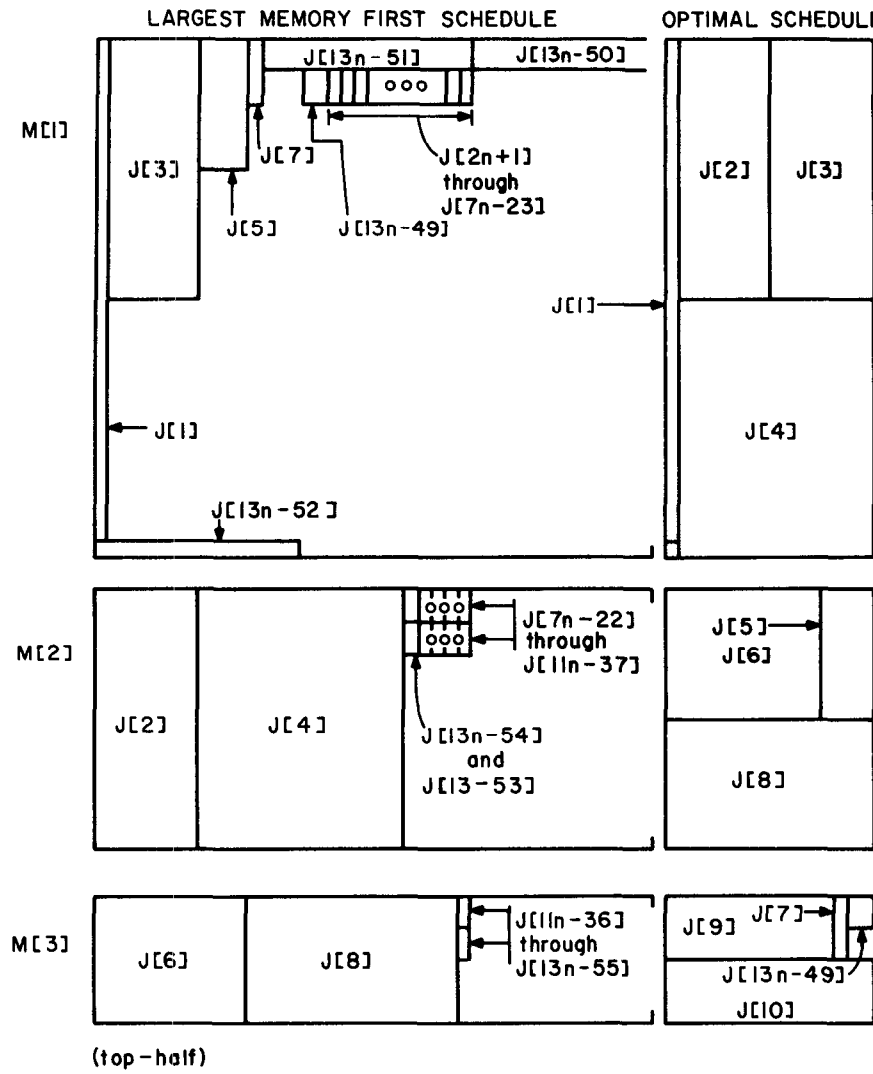


FIG. 9. Example schedules for Corollary 3.

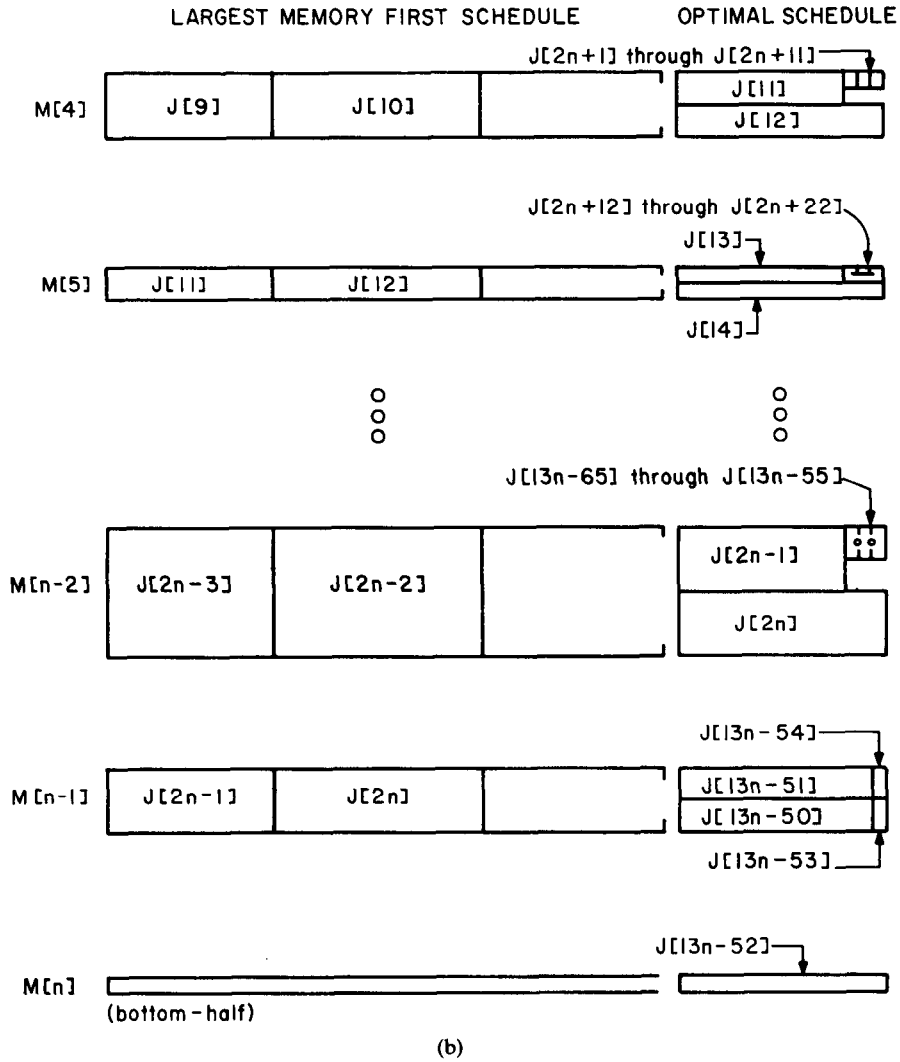
We need the following two new definitions:

$$B[1] = \{h[1] \mid h[1] \in B \text{ and there exists } J[r] \in h[1] \mid m[r] + m[e] > M[1]\},$$
$$B[2] = \{h[1] \mid h[1] \in B \text{ and for all } J[r] \in h[1],\ m[r] + m[e] \leq M[1]\}.$$

The bound $z + w$ is determined by finding the corresponding bounds for large tasks of B[1] and B[2] slices, respectively, then adding the two bounds together.

By the definitions of B[1] and B[2] slices and the LMF scheduling principle, all B[1] slices must precede all B[2] slices.

3.2.1. *Bound for B*[2]. $z[2] = b[2]$, by definition of B[2] slices and the LMF scheduling principle ($z[2] < b[2]$ implies that $J[e]$ does not exist). The upper bound on the number of task slices that can be scheduled with large tasks of B[2] slices is $z[2]$ (the bound cannot be greater than the number of B[2] slices).



(b)

FIG. 9. Continued.

3.2.2 *Bound for B*[1]. The small tasks in B[2] slices must have memory requirements greater than or equal to m[e], since if they are less than m[e], the LMF scheduling principle is violated. (Note: J[e] can coexist with any large task in a B[2] slice.) By the definition of B[1] slices, the small tasks in B[2] or E slices whose memory requirements are known to be greater than or equal to m[e] cannot be scheduled with the large tasks in B[1] slices. This means only small tasks of W[i] and B[1] slices can coexist with large tasks of B[1] slices. Assuming that all small tasks of W[i] and B[1] are scheduled with the large tasks of B[1] slices results in an upper bound of task slices that can be scheduled with large tasks in B[1] slices of z[1] + w.

The upper bound on the number of task slices that can be scheduled with large tasks of B slices is the sum of the upper bound for B[1], z[1] + w, and B[2], z[2], which is (z[1] + w) + (z[2]) = z + w.

The bound on the optimum schedule is formed as follows. The previously mentioned z + w small task slices may be scheduled with the b large task slices of

B slices producing at least b processor slices. The remaining v + 2(e + f) + w + $t[s]$ task slices may at best be scheduled as two for every processor slice, giving at least e + f + (v + w + $t[s]$)/2 processor slices.

Thus, the optimum schedule must have at least b + d + (v + w + $t[s]$)/2 processor slices; therefore, with n processors:

$$n \cdot T[\text{OPT}] \geq \text{b} + \text{d} + \frac{\text{u} + t[s]}{2}. \qquad \square$$

Using Lemma 3, the next theorem is easily proved; this is the sole purpose of the lemma.

THEOREM 2. *Given an LMF schedule of the task set J on N, m = 2, satisfying* $m[s] \leq M[n]$, *then*

$$T[LMF] \leq \frac{3n}{n+1} T[OPT] \cdot$$

PROOF. Assume there exists an LMF schedule such that $T[\text{LMF}] > (3n/(n+1))T[\text{OPT}]$.

The inequalities u[2] + $t[s] \geq T[\text{LMF}]$, u[3] + $t[s] \geq T[\text{LMF}], \ldots,$ u[n] + $t[s] \geq T[\text{LMF}]$, $t[s] \leq T[\text{OPT}]$ and the contradiction hypothesis combine as follows into inequality (5):

$$\begin{aligned} \text{u} + (n-1)t[s] &\geq (n-1)T[\text{LMF}] > (n-1)\frac{3n}{n+1} T[\text{OPT}], \\ \text{u} + \frac{(n-1)(n+1)}{n+1} T[\text{OPT}] &> 3n \frac{n-1}{n+1} T[\text{OPT}], \\ \text{u} &> \frac{2n^2 - 3n + 1}{n+1} T[\text{OPT}]. \end{aligned} \tag{5}$$

Applying inequality (1) to the inequality of Lemma 3, $n \cdot T[\text{OPT}] \geq$ b + d + (u + $t[s]$)/2, and $T[\text{LMF}] \leq$ b + d + $t[s]$ proves the theorem as follows:

$$\begin{aligned} n \cdot T[\text{OPT}] &> \text{b} + \text{d} + \frac{(((2n^2 - 3n + 1)/(n+1))T[\text{OPT}] + t[s])}{2}, \\ \frac{n^2 + n}{n+1} T[\text{OPT}] &> \text{b} + \text{d} + t[s] + \frac{n^2 - 2n}{n+1} T[\text{OPT}], \\ \frac{3n}{n+1} T[\text{OPT}] &> T[\text{LMF}]. \end{aligned}$$

$\square$

The worst case differences between the least upper bound on R[LMF] and the upper bound of Theorem 2, $3n/(n+1)$, are indicated by Corollaries 4–6 and Table II.

COROLLARY 4. *For the LMF schedules of all task sets J on N, with* $1 \leq n \leq 5$ *and* $m = 2$, *satisfying* $m[s] \leq M[n]$, $3n/(n+1)$ *is the least upper bound on* $R[LMF]$.

PROOF. An upper bound on R[LMF] is $3n/(n+1)$ by Theorem 2. This bound may be approached by the R[LMF]s of the three pairs of schedules in Figure 10 by allowing x to approach infinity.

TABLE II. COMPARISON OF WORST-KNOWN R[LMF]'S AND $3n/(n+1)$

Task set number	Lower bound on least upper bound on R[LMF] (worst-known R[LMF])	Limitations for bound valid for n	Example value of n	Difference and % difference between column 2 and $3n/(n+1)$	
				Difference	% difference
1	$(6n+5)/(2n+4)$	6, 12	8	1/60	0.625
2	$(12n+21)/(4n+12)$	13, 27	16	27/1292	0.740
3	$(24n+65)/(8n+32)$	28, 58	32	53/3168	0.533
4	$(48n+177)/(16n+80)$	59, 121	64	26/26720	0.369
5	$(96n+449)/(32n+192)$	122, 248	128	3519/552152	0.212
k	$(3\cdot 2^k(n-k)-2^k+1)/(2^k(n+k+1))$	$2^{(k+2)}-k-1$, $2^{(k+2)}-k-3$	—	—	—

The sizes of the n memories and of the $5n-3$ tasks for the first network are

$M[i] = 2^{n-i+1}$ for $i \le i \le n$,
$J[1] = (2^n - 1, 1)$, $J[2] = (2^{n-1}, (n-2)x)$,
$J[1+2j] = (2^{n-j}, (n-2)x)$ for $1 \le j < n$,
$J[2+2j] = (2^{n-j}, (n+1)x+1)$ for $1 \le j < n$,
$J[2n+j] = (1, x)$ for $1 \le j \le 3(n-2)$,
$J[5n-5] = (1, (n+1)x+1)$, $J[5n-4] = (1, (n+1)x+1)$,
$J[5n-3] = (1, (5-n)x)$.

(Note: When $n = 5$, $J[5n-3]$ should be deleted from the task set.)

The sizes of the memories in the other two networks are $M[1] = 4$ and $M[2] = 2$ for $n = 2$ and $M[1] = 2$ for $n = 1$. □

COROLLARY 5. *For the LMF schedules of all task sets J on N, with $6 \le n \le 12$ and $m = 2$, satisfying $m[s] \le M[n]$, the least upper bound on R[LMF] lies in the interval*

$$\left[\frac{6n+5}{2n+4}, \frac{3n}{n+1}\right].$$

PROOF. An upper bound on R[LMF] is $3n/(n+1)$ by Theorem 2. Hence, the least upper bound on R[LMF] is less than or equal to $3n/(n+1)$, the upper bound of the interval. The pair of schedules in Figure 11 has an R[LMF] that approaches $(6n+5)/(2n+4)$ as x approaches infinity. Therefore, the least upper bound on R[LMF] is greater than or equal to $(6n+5)/(2n+4)$, the lower bound of the interval.

The sizes of the n memories and $k = 9n-15$ tasks used in Figure 11 are

$M[i] = 2^{n-i+1}$ for $1 \le i \le n$,
$J[1] = (2^n - 1, 1)$, $J[2] = (2^{n-1}, (n+2)x)$,
$J[3] = (2^{n-1}, (n+2)x)$, $J[4] = (2^{n-1}, 2(n+2)x)$,
$J[3+2j] = (2^{n-j-1}, (2n-3)x)$ for $1 \le j \le n-2$,
$J[4+2j] = (2^{n-j-1}, 2(n+2)x+1)$ for $1 \le j \le n-2$,
$J[2n+j] = (1, x)$ for $1 \le j \le 7(n-3)$,
$J[9n-20] = (1, 1)$, $J[9n-19] = (1, 1)$,
$J[9n-18] = (1, 2(n+2)x)$, $J[9n-17] = (1, 2(n+2)x)$,
$J[9n-16] = (2^{n-2}, (n-5)x)$, $J[9n-15] = (1, (12-n)x+1)$. □

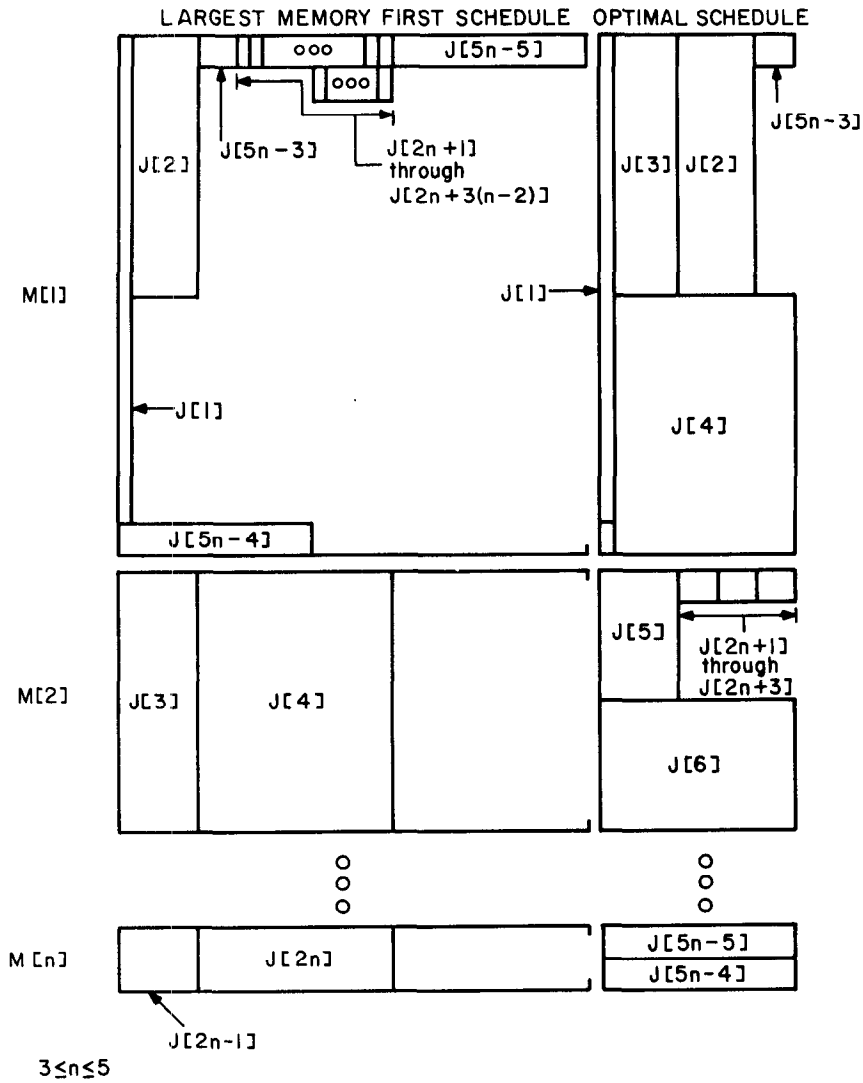
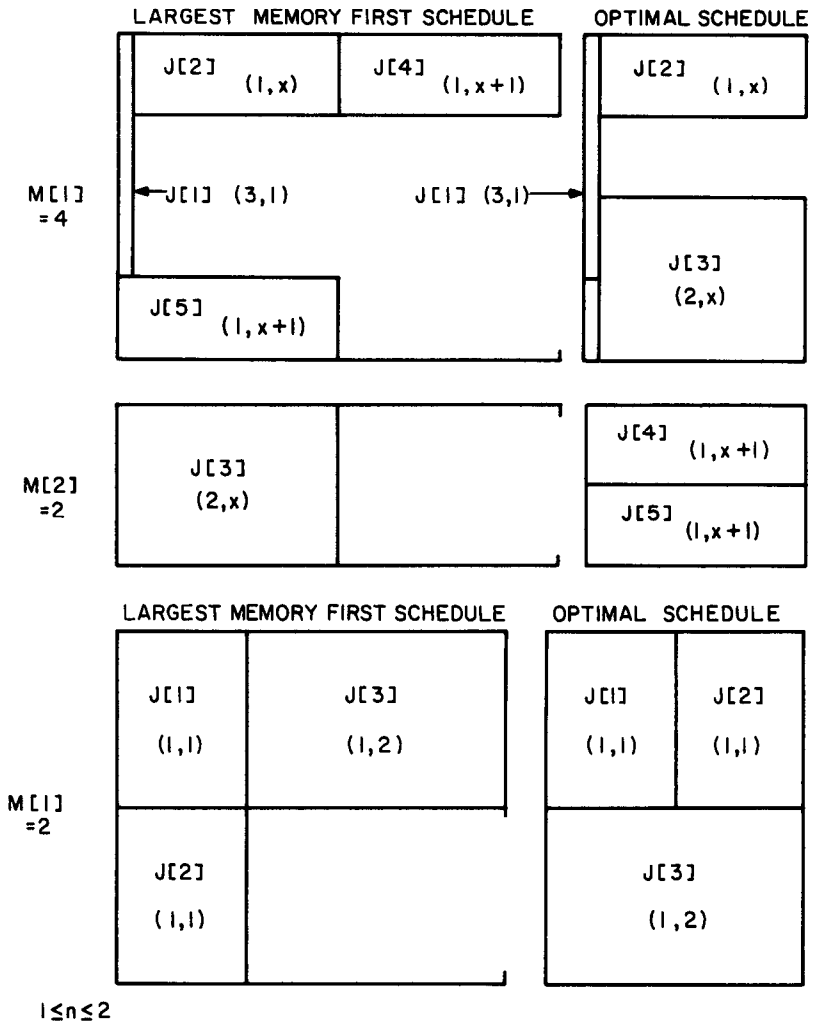


FIG. 10. Example schedules for Corollary 4.

COROLLARY 6. *For the LMF schedules of all task sets J on N, with* $13 \leq n \leq 27$ *and* $m = 2$, *satisfying* $m[s] \leq M[n]$, *the least upper bound on* $R[LMF]$ *lies in the interval*

$$\left[\frac{12n + 21}{4n + 12}, \frac{3n}{n + 1}\right].$$

PROOF. An upper bound on R[LMF] is $3n/(n + 1)$ by Theorem 2. Hence, the least upper bound on R[LMF] is less than or equal to $3n/(n + 1)$, the upper bound of the interval. The pair of schedules in Figure 12 has an R[LMF] that approaches $(12n + 21)/(4n + 12)$ as x approaches infinity. Therefore, the least upper bound on R[LMF] is greater than or equal to $(12n + 21)/(4n + 12)$, the lower bound of the interval.

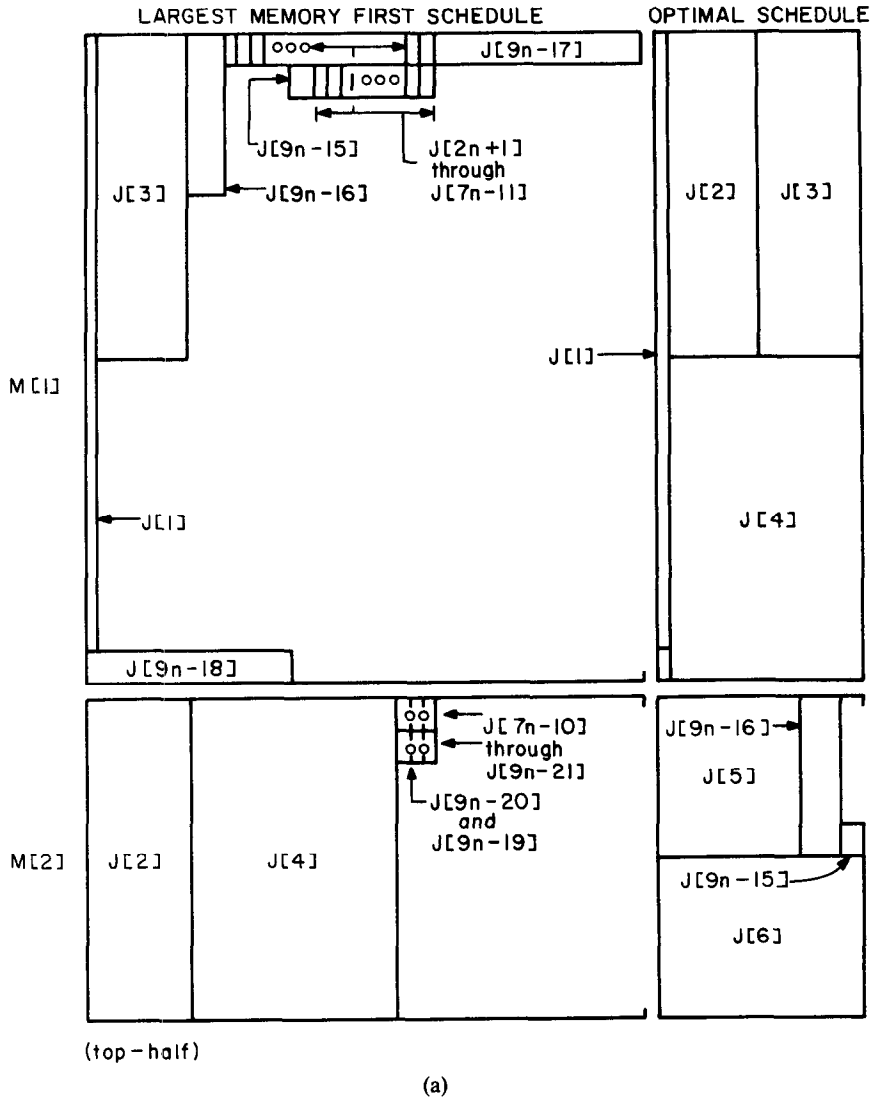


(b)

FIG. 10. Continued.

The sizes of the n memories and $k = 17n - 53$ tasks used in Figure 12 are

$M[i] = 2^{n-i+1}$ for $i \le i \le n$,
$J[1] = (2^n - 1, 1)$,
$J[2] = (2^{n-1}, 2(n + 3)x)$,
$J[3] = (2^{n-1}, 2(n + 3)x)$,
$J[4] = (2^{n-1}, 4(n + 3)x)$,
$J[5] = (2^{n-2}, 3(n + 3)x)$,
$J[6] = (2^{n-2}, 4(n + 3)x + 1)$,
$J[5 + 2j] = (2^{n-j-2}, (4n - 3)x)$ for $1 \le j \le n - 3$,
$J[6 + 2j] = (2^{n-j-2}, 4(n + 3)x + 1)$ for $1 \le j \le n - 3$,
$J[2n + j] = (1, x)$ for $1 \le j \le 15(n - 4)$,
$J[17n - 59] = (1, 1)$,
$J[17n - 58] = (1, 1)$,
$J[17n - 57] = (1, 4(n + 3)x)$,
$J[17n - 56] = (1, 4(n + 3)x)$,
$J[17n - 55] = (2^{n-2}, (n + 3)x)$,
$J[17n - 54] = (2^{n-3}, (n - 12)x)$,
$J[17n - 53] = (1, (27 - n)x + 1)$. □



(a)

FIG. 11. Example schedules for Corollary 5.

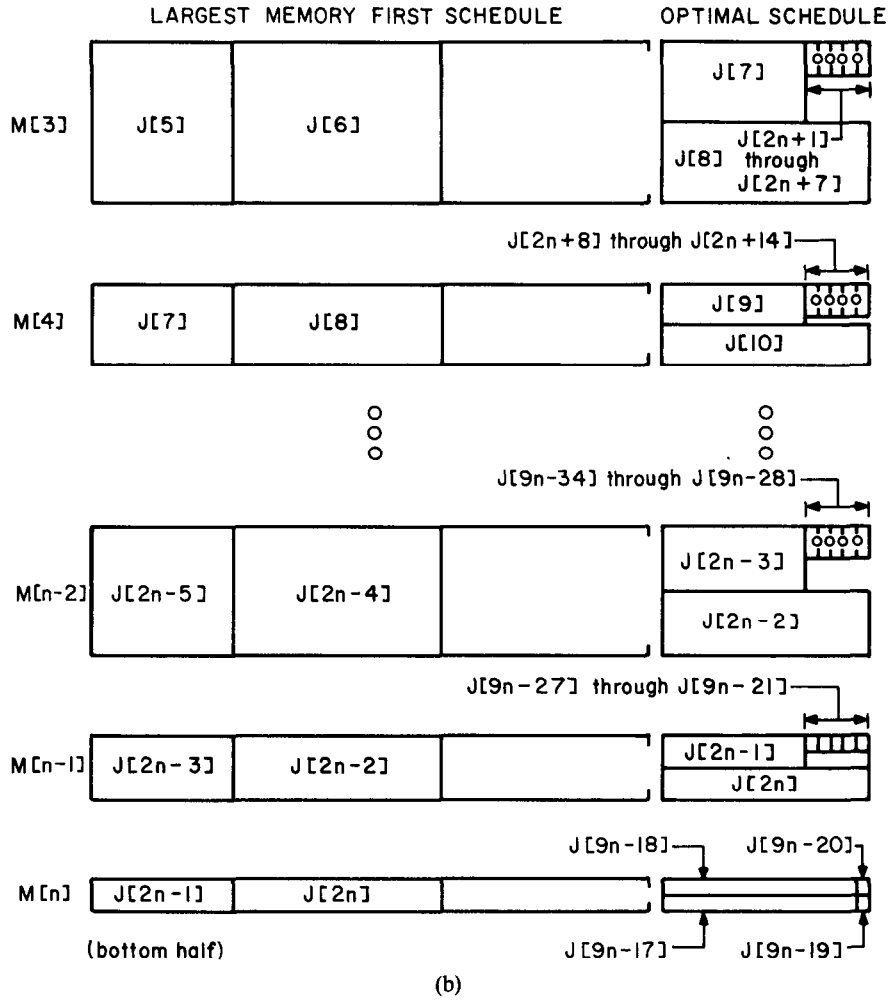
4. *Conclusions and Future Work*

This section compares the bounds of the dual processor model, the model of Krause et al. [14], and the model of Kafura and Shen [12] for a given level of concurrency. To avoid lengthy notation, let model I denote the model of Krause et al., model II the model of Kafura and Shen, and model III the model analyzed in Section 3.

First, we list the established bounds for each of the three models:

I. $T[\text{LMF}] < (3 - 3/m)T[\text{OPT}]$,
II. $T[\text{LMF}] \leq (2 - 1/n)T[\text{OPT}]$,
IIIa. $T[\text{LMF}] < (3 - 2/n)T[\text{OPT}]$ for $m[s] > M[n]$,
IIIb. $T[\text{LMF}] \leq (3 - 3/(n + 1))\ T[\text{OPT}]$ for $m[s] \leq M[n]$.

Recall Figure 3, which illustrates the basic differences between the three models.



(b)

FIG. 11. Continued.

Since the ratio of processors to memories is different in each of the three models, we choose as a common point of comparison the bound of each model for a given maximum level of concurrency. That is, we determined the bound for each model where the configuration of each model would allow the same given number of tasks to be executed concurrently. To obtain a concurrency level of $2c$ in each of the models,

(1) the number of processors m in model I must be $2c$;
(2) the number of processors and memories n in model II must be $2c$;
(3) the number of computers n in model III must be c, since each computer contains two processors.

Making the above substitutions in the bounds previously listed, we obtain

I. $T[\text{LMF}] < (3 - 3/(2c))T[\text{OPT}]$,
II. $T[\text{LMF}] \leq (2 - 1/(2c))T[\text{OPT}]$,
IIIa. $T[\text{LMF}] < (3 - 2/c)T[\text{OPT}]$,
IIIb. $T[\text{LMF}] \leq (3 - 3/(c + 1))T[\text{OPT}]$.

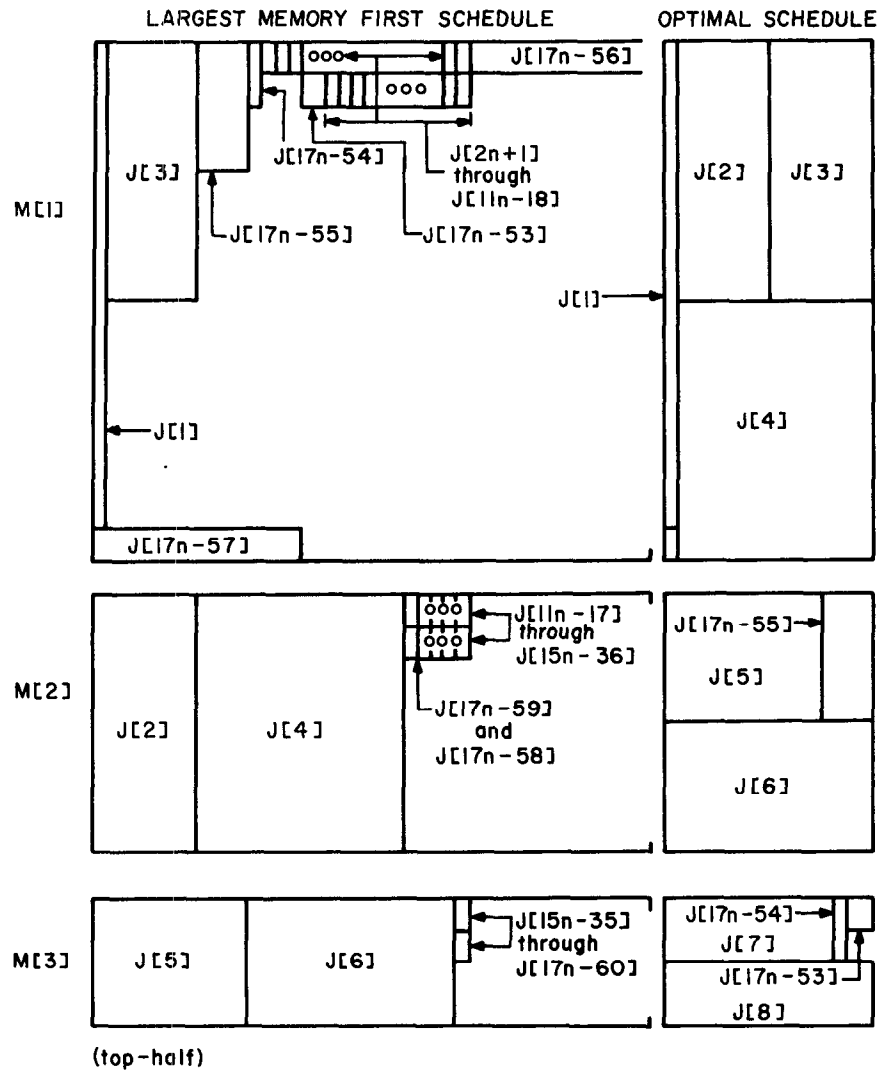


FIG. 12. Example schedules for Corollary 6.

By the above list, the ordering of the bounds on the models from best to worse is II, IIIb, IIIa, I.

Two important observations should be made about this ordering.

(1) For a concurrency level of 2 ($c = 1$), result IIIa is not applicable and all the other bounds are the same (= 3/2).

(2) As the size of the network increases, the bound of II asymptotically approaches 2, whereas the other bounds all approach 3 as a limit. This difference is apparent even in relatively small networks. For example, at $c = 10$ the bound of II is 1.95, whereas the other bounds are 2.73, 2.8, and 2.85, respectively. Thus, for the simple network with exactly two processors there is no difference in the performance of the systems with shared and nonshared memory. However, as the size of the network grows, even a limited amount of shared memory (memory

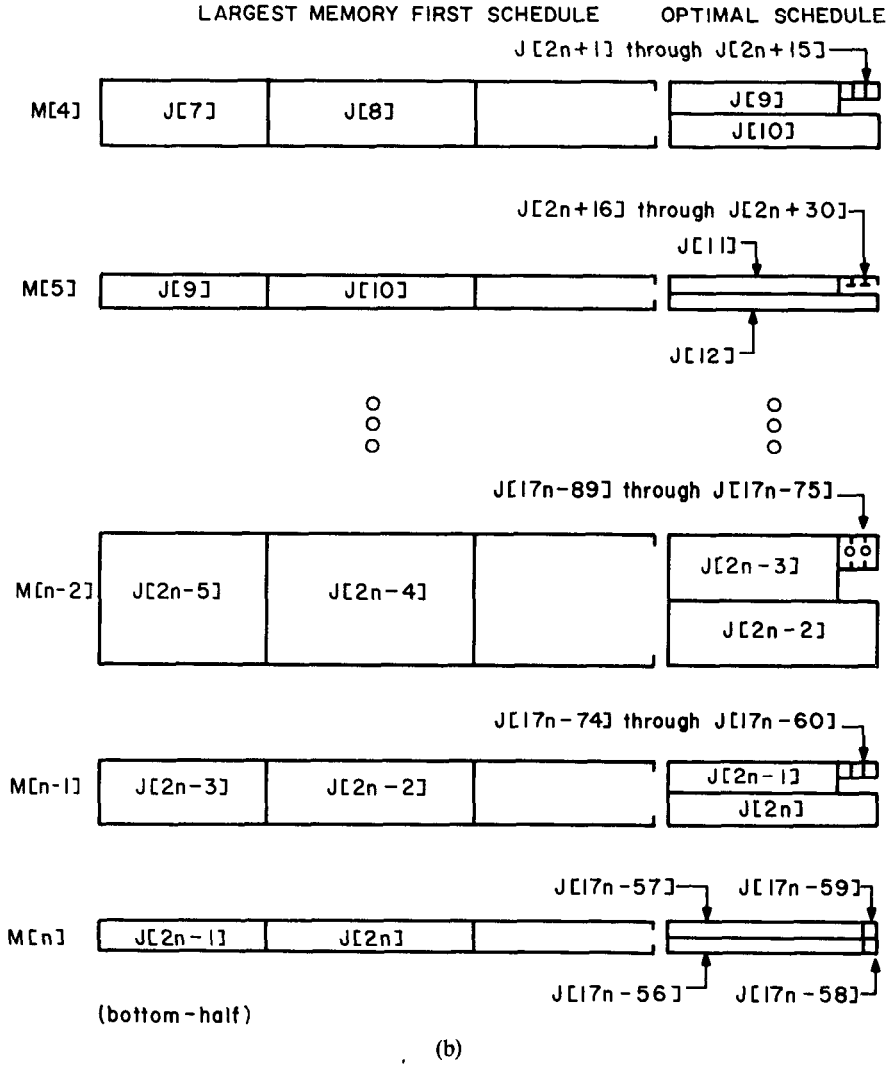


FIG. 12. Continued.

shared by only two processors) becomes as disadvantageous as memory shared among all the processors. This result is the primary conclusion of this paper.

Five other observations are also worth noting.

(1) It may be argued that shared memory is important for high-speed communication in a tightly coupled network. The results previously given would then indicate that the memory should be partitioned with a shared portion for communication and a private portion for each processor. This private memory should be used for the address space of the tasks to be executed.

(2) The use of private memories eliminates the possibility of maintaining a single copy of segments shared among independent processes. However, studies of the MULTICS system have shown that such sharing is only lightly used [15]. Although this may not be true for all systems, it at least shows that, for a large class of local networks, the use of private rather than shared memory is feasible.

(3) Particularly for the model that we have analyzed, the decision as to whether bound IIIa or IIIb is applicable for a given task set cannot be made until the LMF schedule has been constructed. Since these two bounds are very close for reasonably sized networks and since both approach the same limit, we do not view the problem of deciding which bound to use as a serious difficulty.

(4) We see no practical reason to extend the analysis to the general model pictured in Figure 3c in which each multiple node contains an arbitrary number of processors sharing a common memory. The bound on such a system is worse than the bounds already known and would not yield additional insights into the fundamental value of memory sharing. We know that the bound for the more general system is worse than that of the known systems because we have constructed examples of a system with two computers each having a multiprocessor level of 7 where $R[\text{LMF}] > 3$. The possibility exists, however, that an algorithm more sophisticated than LMF would exhibit improved performance on the more general network. The difficulty of the proofs for LMF on the Krause model and for our own model suggests that the analysis of this more complex strategy, using deterministic methods, could well be intractable. Other methods, such as simulation, might prove more profitable.

(5) We have noted a general similarity in the proof technique used in this paper and by Krause et al. [14]. Generally, both proofs consist of the following steps:

(a) Assume that each task is an integral number of a basic time slice.
(b) Divide the LMF schedule into successive pieces equal in length to the basic time slice and characterize different groups of slices (e.g., memory bound).
(c) Determine the possible valid sequencing of these groups to each other.
(d) Use the size and ordering of the groups to derive the overall bound.

Within this general pattern, however, we did not find any similarity in the definition of the types of slices or in the relations between groups of slices. This lack of similarity also leads us to believe that an analysis of the more general model is not practical.

The research we have reported in this paper can be extended in two ways. First, as previously mentioned, other performance analysis techniques, such as simulation or queuing theory, may be applied to these same models in order to confirm or contrast the results we have presented in terms of the impact of memory sharing on network performance. Second, the total amount of memory used in the network may be an important, although not the sole, design constraint. In this case, the "memory-sizing problem" explored in [12] for the private memory model may be extended to a more general network model. However, given the apparent difficulty of analyzing these more general models, we expect that such an extension might prove difficult.

REFERENCES

1. COFFMAN, E. G., JR *Computer and Job/Shop Scheduling Theory.* Wiley, New York, 1976.
2. COFFMAN, E. G., JR, AND DENNING, P. J. *Operating Systems Theory.* Prentice-Hall, Englewood Cliffs, N. J., 1973.
3. COFFMAN, E. G., JR, AND GRAHAM, R. L. Optimal scheduling for two processor systems, *Acta Inf. 1* (1972), 200–213.
4. GAREY, M. R., AND GRAHAM, R. L. Bounds on scheduling with limited resources. In *Proceedings of the 4th Symposium on Operating System Principles* (Yorktown Heights, N.Y., Oct. 15–17). ACM, New York, 1973, pp. 104–111.
5. GONZALES, M. J., JR. Deterministic processor scheduling. *Comput. Surv. 9*, 3 (Sept. 1977), 173–204.

6. GRAHAM, R. L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math. 17*, 2 (Mar. 1969), 416–429.
7. GRAHAM, R. L. Bounds on multiprocessing anomalies and related packing algorithms. In *Proceedings of the Spring Joint Computer Conference*. AFIPS Press, Reston, Va., 1972, pp. 205–217.
8. HU, T. C. Parallel sequencing and assembly line problems. *Oper. Res. 9*, 6 (1961), 841–848.
9. KAFURA, D. G. Task scheduling with critical section constraints. In *Proceedings of the IFIP-77* (Toronto, Ont., Canada, Aug. 8–12). Elsevier-North Holland, New York, 1977, pp. 553–557.
10. KAFURA, D. G. Scheduling tasks with critical sections. In *Proceedings of the 1977 Annual Conference of the Association of Computing Machinery* (Seattle Wash., Oct. 16–19). ACM, New York, pp. 381–385.
11. KAFURA, D. G., AND SHEN, V. Y. Task scheduling on a multiprocessor system with independent memories. *SIAM J. Comput. 6*, 1 (Mar. 1977), pp. 167–187.
12. KAFURA, D. G., AND SHEN, V. Y. An algorithm to design the memory configuration of a computer network. *J. ACM 25*, 3 (July 1978), 365–377.
13. KLEINROCK, L. *Queueing Systems*. Wiley, New York, 1975.
14. KRAUSE, K. L., SHEN, V. Y., AND SCHWETMAN, H. D. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *J. ACM 22*, 4 (Oct. 1975), 522–550.
15. MONTGOMERY, W. A. Measurements of sharing in MULTICS. In *Proceedings of the 6th Symposium on Operating System Principles*, (West Lafayette, Ind., Nov. 16–18). ACM, New York, 1977, pp. 85–90.
16. MUNTZ, R. R., AND COFFMAN, E. G., JR. Optimal preemptive scheduling on two processor systems. *IEEE Trans. Comput. C-18*, 11 (Nov. 1969), 200–213.
17. MUNTZ, R. R., AND COFFMAN, E. G., JR. Preemptive scheduling of real-time tasks on multiprocessor systems. *J. ACM 17*, 2 (Apr. 1970), 324–338.

RECEIVED NOVEMBER 1978; REVISED JUNE 1984; ACCEPTED JULY 1984