Preemptive Scheduling Of Uniform Processors With Memory*

Ten-Hwang Lai and Sartaj Sahni
University of Minnesota

Abstract

We consider the problem of preemptively scheduling n jobs on a system of m uniform processors. Each job has a release time, a due time, and a memory requirement associated with it. Each processor has a speed and a memory capacity associated with it. Linear programming formulations are obtained for the following optimization criteria: (1) minimize the schedule length and (2) minimize the maximum lateness. We also consider three special cases with the former optimization criteria. For each of these cases, low order polynomial time algorithms are obtained.

Keywords and Phrases

preemptive scheduling, uniform processors, memories, release time, due time, $C_{\max}$, $L_{\max}$, linear programming, complexity.

* This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650.

1. Introduction

Let J = { J_i | 1 ≤ i ≤ n } be a set of independent jobs. Associated with each job, J_i, is a 4-tuple (t_i, m_i, r_i, d_i) where t_i is the job's processing time ($t_i > 0$), m_i is its memory requirement, r_i is its release time, and d_i is its due time. Let P = { P_i | 1 ≤ i ≤ m } be a set of m processors. With each processor, P_i, there is a 2-tuple (s_i, μ_i) associated. s_i is the *speed* of P_i and μ_i is its *memory capacity*. Job J_i (or simply job i) can be run on processor P_j (or simply processor j) iff $m_i \le \mu_j$. Furthermore, it takes P_j t_i/s_j time to completely run job J_i. P is said to be a *uniform processor system with memories*. If $s_1 = s_2 = \cdots = s_m$, then P is a system of *identical processors with memories*. If $s_1 = s_2 = \cdots = s_m$ and $\mu_1 = \mu_2 = \cdots = \mu_m$, P is simply a system of *identical processors*. Finally, if $\mu_1 = \mu_2 = \cdots = \mu_m$, P is called a system of *uniform processors*.

A *feasible preemptive schedule* S for job set J on the processor system P is an assignment of jobs to time slots on processors such that:

(a) No job is processed by more than one processor at any given time.

(b) No processor executes more than one job at any given time.

(c) The total processing assigned for each job equals its processing requirement.

(d) No job is processed by a processor with memory capacity less than the job's memory requirement.

(e) No job is processed before its release time.

S is a *nonpreemptive schedule* iff, in S, each job is processed continuously, on the same processor, from its start to finish.

The *finish time*, f_i, of job i in schedule S, is the time at which its processing is completed. The *length* (or *finish time*), $C_{\max}(S)$, of schedule S is the least time by which all jobs have been processed. I.e.,

$$C_{\max}(S) = \max_i \{f_i\}$$

The $C_{\max}$ *problem* is that of determining a schedule S for which $C_{\max}(S)$ is minimum amongst all feasible schedules. Let S be a feasible schedule. $L_i = f_i - d_i$ is the *lateness* of job i, 1 ≤ i ≤ n. $L_{\max}(S) = \max_i\{L_i\}$ is the maximum lateness of any job in S. The $L_{\max}$ *problem* is that of determining a schedule with minimum $L_{\max}$.

When S is restricted to be a non-preemptive schedule, the $L_{\max}$ problem is known to be NP-hard even when m = 1 [3]. Under this same restriction, the $C_{\max}$ problem is NP-hard even when m = 2, $s_1 = s_2$, $\mu_1 = \mu_2$, and $r_1 = r_2 = ... = r_n$ [3]. In the remainder of this paper, we shall therefore be concerned only with schedules in which preemptions are permitted.

Suppose that P is a system of identical processors (i.e., $s_1 = s_2 = ... = s_m$ and $\mu_1 = \mu_2 = ... = \mu_m$). When all jobs have the same release time ($r_1 = r_2 = ... = r_n$) a schedule that minimizes $C_{\max}$

can be obtained in O(n) time using McNaugton’s algorithm [12]. When the release times are not necessarily the same, a schedule that minimizes $C_{\max}$ can be obtained in O(nm) time using the algorithm of Gonzalez and Johnson [5].

If P is a uniform processor system ($\mu_1 = \mu_2 = ... = \mu_m$) and all jobs have the same release time, a schedule minimizing $C_{\max}$ may be obtained in O(n + mlogm) time using the algorithm of Gonzalez and Sahni [7]. When the release times are not necessarily the same, the algorithm of Sahni and Cho [14] may be used. This algorithm has a complexity of O(m^2n + mnlogn).

Scheduling on processor systems with memories has been studied by Kafura and Shen [8] as well as by Lai and Sahni [11]. Both these papers deal exclusively with a system of identical processors with memory. Kafura and Shen [8] develop an O(nlogm), n ≥ m, algorithm for the $C_{\max}$ problem when all release times are the same. Lai and Sahni [11] consider the $L_{\max}$ problem when all release times are the same. Their algorithm is of complexity O(k^2n + nlogn) where k is the number of distinct due dates and n ≥ m.

In this paper, we consider scheduling systems of uniform processors with memories. In Section 2, we obtain linear programming formulations for the $C_{\max}$ and $L_{\max}$ problems. Three special cases of $C_{\max}$ problem are considered in Section 3. Each of these cases assumes that all jobs are released at the same time (i.e., $r_1 = r_2 = ... = r_n$). In addition, the three cases, respectively, assume:

(1) m = 2

(2) m = 3

(3) The m processors can be partitioned into two classes A and B such that all processors in A have the same speed while all those in B have the same memory size. In addition, all processors in A have a larger memory size than those in B. So, for some ordering of the processors, it is the case that $\mu_1 \geq \mu_2 \geq ... \geq \mu_k > \mu_{k+1} = ... = \mu_m$ and $s_1 = s_2 = ... = s_k$.

For each of the above three cases, we develop low order polynomial time algorithms.

2. Linear Programming Formulations

Let J be a set of n jobs and let P be a set of m processors. Without loss of generality, we assume that $\mu_1 \geq \mu_2 \geq ... \geq \mu_m$. Furthermore, let S be any schedule for J on P. The following notation will prove useful in obtaining the linear programming formulations.

(a) h(i) is the largest index such that job i can be processed on $P_{h(i)}$. I.e., h(i) = max { j | $m_i \leq \mu_j$ }.

(b) $q_1, q_2, ..., q_u$ are the distinct release times in the multiset { $r_1, r_2, ..., r_n$ } and $q_1 < q_2 < ... < q_u$. Let q_{u+1} denote the length of S (i.e., $q_{u+1} = C_{\max}$(S)). Clearly, $q_u < q_{u+1}$.

(c) r(i) is such that $r_i = q_{r(i)}$, 1 ≤ i ≤ n.

(d) x_{ijk} denotes the amount of time for which job i is assigned to procesor j (or P_j) in the interval $[q_k, q_{k+1}]$, $1 \le i \le n$, $1 \le j \le m$, $1 \le k \le u$.

2.1 The $C_{\max}$ Problem

It is clear that for every feasible schedule S the x_{ijk}s satisfy the following constraints ((2.1) - (2.4)):

$$\sum_{j=1}^{m} x_{ijk} \le q_{k+1} - q_k,\ 1 \le i \le n,\ 1 \le k \le u \qquad (2.1)$$

I.e., no job is scheduled in any interval for an amount of time greater than the interval length.

$$\sum_{i=1}^{n} x_{ijk} \le q_{k+1} - q_k,\ 1 \le j \le m,\ 1 \le k \le u \qquad (2.2)$$

I.e., no processor is scheduled in any interval for an amount of time greater than the interval length.

$$\sum_{j=h(i)+1}^{m} x_{ijk} = 0,\ 1 \le i \le n,\ 1 \le k \le u \qquad (2.3a)$$

I.e., no job is assigned to a processor with insufficient memory.

$$\sum_{k=1}^{r(i)-1} x_{ijk} = 0,\ 1 \le i \le n,\ 1 \le j \le m \qquad (2.3b)$$

I.e., no job is processed before its release time.

$$\sum_{j=1}^{m} \sum_{k=1}^{u} x_{ijk} s_j = t_i,\ 1 \le i \le n \qquad (2.3c)$$

I.e., each job is completed.
And

$$x_{ijk} \ge 0,\ 1 \le i \le n,\ 1 \le j \le m,\ 1 \le k \le u \qquad (2.4)$$

I.e., all assignments are for a nonnegative amount of time.

One readily sees that if the x_{ijk}s satisfy the above constraints, then they also satisfy the requirements of the Gonzales-Sahni [6] theorem for open shop scheduling. Hence, their algorithm can be used to obtain a feasible schedule of length q_{u+1} in which job i is scheduled on processor j in interval $[q_k, q_{k+1}]$ for exactly x_{ijk} time. So, a schedule that minimizes $C_{\max}$ can be obtined by first solving the linear programming program:

minimize q_{u+1}

subject to constraints (2.1), (2.2), (2.3a), (2.3b),
(2.3c) and (2.4).

Once the x_{ijk}s and q_{u+1} have been obtained, the schedule can be constructed, interval by interval, using the algorithm for open shop scheduling [6]. It is not too difficult to see that the n(u+m+1) equalities corresponding to (2.3a), (2.3b), and (2.3c) can be replaced by the n equalities:

$$\sum_{j=1}^{h(i)} \sum_{k=r(i)}^{u} x_{ijk} s_j = t_i,\ 1 \le i \le n \tag{2.3}$$

in the above linear programming formulation. In addition, the variables

$$x_{ijk},\ h(i)+1 \le j \le m,\ 1 \le k \le r(i)-1,\ 1 \le i \le n$$

may be set to zero. This transformation reduces the total number of variables and inequalities and so results in a simpler linear program.

The most practical way to solve the formulated linear program is via the simplex method. This is known to have a good expected performance (see Dantzig [1]). The simplex method does however have an exponential worst case performance ([4] and [10]). It should be pointed out that Khachian [9] has obtained a polynomial time algorithm to solve linear inequalities. His algorithm has a worst case complexity that is $O(Ln^2(mn+n^2))$, where m is the number of inequalities; n the number of variables; and L the number of bits needed to represent all the coefficients. Gacs and Lovasz [2] describe how linear programming problems can be reduced to solving systems of inequalities.

From the results of Khachian [9], Gacs and Lovasz [2], Gonzalez and Sahni [6], and our linear programming formulation, it follows that the $C_{\max}$ problem can be solved in polynomial time.

2.2 The $L_{\max}$ Problem

Let S* be a schedule that minimizes $L_{\max}$. Let y* = $L_{\max}$(S*). It is not too difficult to see that y* is the smallest y for which the job set characterized by $(t_i, m_i, r_i, d_i + y)$, $1 \le i \le n$, has a schedule S with $L_{\max}(S) \le 0$.

Let y be arbitrary. We may determine whether or not there is a schedule for $(t_i, m_i, r_i, d_i + y)$, $1 \le i \le n$ that has an $L_{\max} \le 0$ as follows. First sort the distinct release and due times in the multiset

$$\{\ r_i \mid 1 \le i \le n\ \}\ U\ \{\ d_i + y \mid 1 \le i \le n\ \} \tag{2.5}$$

to get the ordered set $(q_1, q_2, ..., q_u)$, $q_1 < q_2 < ... < q_u$. Let r(i) and d(i) be such that $r_i = q_{r(i)}$ and d_i

+ y = $q_{d(i)}$, 1 ≤ i ≤ n. From the development of Section 2.1, it follows that there is a schedule with $L_{\max} \le 0$ iff the following linear system has a feasible solution (x_{ijk} denotes the amount of time for which job i is scheduled on P_j in the interval $[q_k, q_{k+1}]$).

$$\sum_{j=1}^{m} x_{ijk} \le q_{k+1} - q_k,\ 1 \le i \le n,\ 1 \le k < u \tag{2.6}$$

$$\sum_{i=1}^{n} x_{ijk} \le q_{k+1} - q_k,\ 1 \le j \le m,\ 1 \le k < u \tag{2.7}$$

$$\sum_{j=1}^{h(i)} \sum_{k=r(i)}^{d(i)-1} x_{ijk} s_j = t_i,\ 1 \le i \le n \tag{2.8}$$

$$x_{ijk} \ge 0,\ 1 \le i \le n,\ 1 \le j \le m,\ 1 \le k < u \tag{2.9}$$

To determine y* (the least y for which a schedule with $L_{\max} \le 0$ exists), construct the set Δ = { δ | d_i + δ = r_j for some i and j }. Note that | Δ | ≤ n^2 and Δ may contain both positive and negative elements. Let $\delta_1, \delta_2, ..., \delta_p$ be the elements of Δ. We may assume that $\delta_1 < \delta_2 < ... < \delta_p$. Since $\delta_1 = \min_i\{r_i\} - \max_i\{d_i\}$, $d_i + \delta_1 \le r_i$ for every i. Hence, there is no schedule with $L_{\max} \le 0$ when the job characteristics are (t_i, m_i, r_i, $d_i + \delta_1$), 1 ≤ i ≤ n.

The next step in determining y* is to find the largest j such that there is no schedule with $L_{\max} \le 0$ for (t_i, m_i, r_i, $d_i + \delta_j$), 1 ≤ i ≤ n. This can be done by carrying out a binary search over the δ_is and using the formulation (2.6) - (2.9) to determine the existence of a schedule with $L_{\max} \le 0$. Let this largest j be w. If w = p, then we may define $\delta_{p+1} = \infty$ for convenience. It is now clear that $\delta_w < y^* \le \delta_{w+1}$. Let ($q_1$, q_2, ..., q_u) be the ordered set of distinct release and due times obtained when y is set equal to δ_w in (2.5). Let δ be arbitrary. If q_i corresponds to a $d_j + \delta_w$ but no r_k, then replace it by $d_j + \delta$. If q_i corresponds to both an r_k and a $d_j + \delta_w$ then insert $d_j + \delta$ into the sequence of q_is immediately after q_i. Let the new sequence be (q'_1, q'_2, ..., $q'_{u'}$), u' ≥ u. Note that for $\delta_w < \delta < \delta_{w+1}$, $q'_1 < q'_2 < ... < q'_{u'}$. Consequently, for δ in this range, the r and d functions defined by $r_i = q'_{r(i)}$ and $d_i + \delta = q'_{d(i)}$ do not change with δ. This allows us to formulate the following linear program:

minimize δ

subject to $\sum_{j=1}^{m} x_{ijk} \le q'_{k+1} - q'_k$, 1 ≤ i ≤ n, 1 ≤ k < u'

$\sum_{i=1}^{n} x_{ijk} \le q'_{k+1} - q'_k$, 1 ≤ j ≤ m, 1 ≤ k < u'

$$\sum_{j=1}^{h(i)} \sum_{k=r(i)}^{d(i)-1} x_{ijk} s_j = t_i,\ 1 \le i \le n$$

$$\delta_w < \delta < \delta_{w+1}$$

$$x_{ijk} \ge 0,\ 1 \le i \le n,\ 1 \le j \le m,\ 1 \le k < u'$$

From our earlier discussions, it follows that if the above linear program has no feasible solution, then $y^* = \delta_{w+1}$; if it does then y^* is the min δ obtained above. Once y^* has been obtained, a schedule with $L_{\max} = y^*$ is easily constructed.

In conclusion, we note that the $L_{\max}$ algorithm described can be implemented in polynomial time.

3. Special cases

In this section, we assume that all jobs are released at time 0. Since only the $C_{\max}$ problem is considered, we ignore the due times that may be specified. Under the preceding restriction on the release times, the $C_{\max}$ problem is studied for the following cases:

(1) m = 2

(2) m = 3

(3) The m processors can be partitioned into two classes A and B such that all processors in A have the same speed while all those in B have the same memory size. In addition, all processors in A have a larger memory than those in B.

In order to establish the correctness of our algorithms, we shall make use of an important result from Sahni and Cho [13]. Let P_i, $1 \le i \le m$ be m processors with the same memory size. Let s_i(t), $1 \le i \le m$ be m nondecreasing functions; s_i(t) gives the speed at which P_i may operate at time t. $\{P_1, P_2, ..., P_m\}$ is a generalized processor system (GPS) iff $s_i(t) \ge s_{i+1}(t)$, $1 \le i < m$ for all t.

Theorem 1 [Sahni and Cho]: Let t_i, $1 \le i \le n$ be n job times such that $t_1 \ge t_2 \ge ... \ge t_n$. Let $\{P_1, P_2, ..., P_m\}$ be a GPS and let $T_k = \sum_{i=1}^{k} t_i$, $1 \le k < m$ and $T_m = \sum_{i=1}^{n} t_i$ (if n < m, then set $T_{n+1} = T_{n+2} = ... = T_m$). The given n jobs can be scheduled on the given GPS to finish by time d iff:

$$T_i \le \sum_{j=1}^{i} \int_0^d s_j(t)dt,\ 1 \le i \le m. \qquad []$$

One can prove that the preceding theorem is true even if the restriction that s_i(t) is a nondecreasing function of t is removed. (Actually, Sahni and Cho's proof in [13] does not use the property that s_i(t) is nondecreasing.) Suppose that H_1, H_2, and H_3 form a relaxed GPS, A, (i.e., one that satisfies all the requirements of a GPS except that s_i(t) may not be a nondecreasing function of t). Let $L_i = \int_0^d s_i(t)dt$, $1 \le i \le 3$. Let L'_i, $1 \le i \le 3$ be the corresponding values for some other relaxed GPS, B. If $L_1 \ge L'_1$, $L_1 + L_2 \ge L'_1 + L'_2$, and $L_1 + L_2 + L_3 \ge L'_1 + L'_2 + L'_3$, then it follows that every job set that can be scheduled to complete by d on B can also be scheduled to complete by d on A. This observation will be implicitly used in Section 3.2.

3.1 m = 2

Without loss of generality, assume that $\mu_1 \ge \mu_2$ and $m_i \le \mu_1$, $1 \le i \le n$. Let n_1 be such that $m_i > \mu_2$, $1 \le i \le n_1$, and either $n_1 = n$ or $m_i \le \mu_2$, $n_1 < i \le n$ (it may require a job reordering to guarantee such a n_1). So, jobs 1, 2, ..., n_1 must be scheduled on P_1 while jobs n_1+1, n_1+2, ..., n may be scheduled on either P_1 or P_2. Further, assume that $t_{n_1+1} \ge t_i$, $n_1 + 2 \le i \le n$. We first observe that if there is a schedule S for the n jobs such that $C_{\max}$(S) = d, then there is a schedule with the same finish time d in which jobs 1, 2, ..., n_1 are scheduled on P_1 from time 0 to time $\tau = (\sum_{i=1}^{n_1} t_i)/s_1$ (see Figure 1).

Figure 1

To see this, note that the schedule S can be divided into intervals with the property that in each interval the jobs scheduled on P_1 are either all from {1, 2, ..., n_1} or all from {n_1+1, n_1+2, ..., n}. Once this division into intervals has been done, we can easily construct, by interchanging intervals, a new schedule, S', in which all intervals containing jobs from {1, 2, ..., n_1} are at the left. S' has a finish time d.

Hence, the $C_{\max}$ problem when m = 2 reduces to finding a minimum finish time schedule for jobs $\{n_1+1, ..., n\}$ on a two processor system $\{Q_1, Q_2\}$ in which Q_1 operates at speed 0 from time 0 to τ and at speed s_1 after τ; and Q_2 operates at speed s_2 from time 0. Furthermore, we may assume that Q_1 and Q_2 have the same memory size. If $s_2 \geq s_1$, then $\{Q_2, Q_1\}$ forms a GPS. If $s_2 < s_1$, then a GPS $\{R_1, R_2\}$ equivalent to $\{Q_1, Q_2\}$ may be formed by assigning to R_1 a speed of s_2 from 0 to τ and s_1 from τ on; R_2 has a speed of 0 from 0 to τ and s_2 from τ on (see Figure 2).

Figure 2 Forming the GPS

From Theorem 1, we see that the jobs $n_1+1, n_1+2, ..., n$ can be scheduled to complete by d on the resulting GPS iff

$$t_{n_1+1} \leq s_2\tau + \max\{s_1, s_2\}(\mathrm{d} - \tau)$$

and

$$\sum_{i=n_1+1}^{n} t_i \leq s_2\mathrm{d} + s_1(\mathrm{d} - \tau)$$

Rearranging terms and combining, we see that d must satisfy the inequality:

$$d \geq \max\{\frac{t_{n_1+1} - s_2\tau}{\max\{s_1, s_2\}} + \tau, \frac{T_2 + s_1\tau}{s_1 + s_2}\}$$

where $T_2 = \sum_{i=n_1+1}^{n} t_i$. Let $T_1 = \sum_{i=1}^{n_1} t_i$. Substituting $\tau = T_1/s_1$ and adding the requirement that $\mathrm{d} \geq \tau$, we obtain:

$$d \geq \max\{\frac{T_1}{s_1}, \frac{t_{n_1+1} - s_2 T_1/s_1}{\max\{s_1, s_2\}} + \frac{T_1}{s_1}, \frac{T_1+T_2}{s_1+s_2}\} \tag{3.1}$$

We immediately see that the minimum $C_{\max}$ is given by the right hand side of (3.1).

To summarize, a minimum $C_{\max}$ schedule for 2 processors with memories can be obtained by following the steps given below:

Step 1: Determine n_1 as given above (reorder the jobs if necessary).

Step 2: Schedule jobs 1, 2, ..., n_1 on P_1 from 0 to $\tau = \sum_{i=1}^{n_1} t_i / s_1$

Step 3: Compute the right hand side of (3.1). Let this value be δ.

Step 4: Construct the GPS as in Figure 2. Schedule jobs n_1+1, ..., n on this GPS from 0 to δ using the algorithm of [13].

Step 5: Map the schedule obtained in step 4 back onto P_1 and P_2. This is trivial because of the obvious correspondence between the GPS and segments of P_1 and P_2.

Each of the steps above takes O(n) time and the overall complexity of the two processor algorithm is O(n). In an actual implementation, of course, steps 4 and 5 would be combined and the schedule constructed directly for P_1 and P_2.

3.2 m = 3

Assume that $\mu_1 \geq \mu_2 \geq \mu_3$ and that the jobs are ordered such that $m_1 \geq m_2 \geq \ldots \geq m_n$ (without loss of generality, we may assume that $m_i \in \{\mu_1, \mu_2, \mu_3\}$, $1 \leq i \leq n$ and so it takes only O(n) time to order the jobs). Let n_1 and n_2 be such that jobs 1, 2, ..., n_1 are the only jobs that cannot be processed on P_2 or P_3 because of memory requirements and jobs 1, 2, ..., n_2 are the only jobs that cannot be processed on P_3 because of memory requirements. Clearly, $n_1 \leq n_2$. For convenience, we now define $t_{n+1} = t_{n+2} = 0$. Let $T_1 = \sum_{i=1}^{n_1} t_i$ and $T_2 = \sum_{n_1+1}^{n_2} t_i$. Further, assume that $t_{n_1+1} \geq t_i$, $n_1+2 \leq i \leq n_2$ and $t_{n_2+1} \geq t_{n_2+2} \geq t_i$, $n_2+3 \leq i \leq n$.

A finish time d is said to be *feasible* iff there exists a schedule S for the given n jobs such that $C_{\max}(S) \leq d$. We shall proceed to obtain a set of necessary and sufficient conditions for d to be feasible. Once this has been done, we shall see how to find a minimum d satisfying these conditions and to then construct a schedule with minimum finish time.

First, we see that d must satisfy the inequalities of (3.2) below:

$$\sum_{i=1}^{k} T_i \Big/ \sum_{i=1}^{k} s_i \leq d, \; k = 1, 2, 3 \tag{3.2}$$

For the same reasons as in Section 3.1, we may limit our discussion to schedules in which jobs 1, 2, ..., n_1 are scheduled from 0 to d_1, $d_1 = T_1/s_1$ on P_1 (Figure 3(a)). Hence, the scheduling of n_1+1, ..., n_2 must be done on the GPS of Figure 3(b). In this figure, $\sigma_1 = \max\{s_1, s_2\}$ and $\sigma_2 = \min\{s_1, s_2\}$.

Figure 3

Let P_{fast} and P_{slow}, respectively, denote the processor with speed σ_1 and σ_2 (if $\sigma_1 = \sigma_2$, then fast = 1 and slow = 2). Let $C_1 = s_2 d_1$, and $C_2 = \sigma_1(d - d_1)$. Hence, C_1 is P_2´*s* contribution to G_1 from 0 to d_1 while C_2 is P_{fast}´*s* contribution to G_1 from d_1 to d. Thus the total available processing capability on G_1 is $C_1 + C_2$. Let $C_3 = \sigma_2(d_2 - d_1)$ be the available processing on G_2.

We shall consider three cases for T_2: (a) $T_2 \le C_3$; (b) $C_3 < T_2 \le C_1 + C_3$; and (c) $C_1 + C_3 < T_2$. When $T_2 \le C_3$, a possible assignment for jobs n_1+1, ..., n_2 is to schedule them on G_2 from d_1 to d_2, $d_2 = d_1 + T_2/\sigma_2$ (Figure 4(a)). If this is done, the GPS that remains for the remaining jobs is as in Figure 4(b) where $\sigma_3 = \max\{ s_2, s_3 \}$; $\sigma_4 = \max\{ \sigma_1, s_3 \} = \max\{ s_1, s_2, s_3 \}$; $\sigma_5 = \min\{ s_2, s_3 \}$; $\sigma_6 = \min\{ \sigma_1, s_3 \}$; $\sigma_7 = \max\{ \sigma_2, \sigma_6\}$; and $\sigma_8 = \min\{ \sigma_2, \sigma_6 \} = \min\{ s_1, s_2, s_3 \}$. Let $r_1(t)$ and $r_2(t)$, respectively, denote the speed of H_1 and H_2 at time t, 0 ≤ t ≤ d. We immediately see that there is no sceduling of jobs n_1+1, ..., n_2 that yields a larger value for $\int_0^d r_1(t)dt$ or $\int_0^d r_1(t)dt + \int_0^d r_2(t)dt$. Hence, from Theorem 1, it follows that (3.2) together with the following ((3.3) and (3.4)) form a set of necessary and sufficient conditions for d when $T_2 \le C_3$:

$$t_{n_2+1} \le \sigma_3 d_1 + \sigma_4(d - d_1)$$

$$= d_1 \max\{s_2, s_3\} + (d - d_1)\max\{s_1, s_2, s_3\} \tag{3.3}$$

$$t_{n_2+1} + t_{n_2+2} \le (\sigma_3+\sigma_5)d_1 + (\sigma_4+\sigma_6)(d_2-d_1) + (\sigma_4+\sigma_7)(d-d_2)$$

$$= d_1(s_2+s_3) + (d_2-d_1)(s_3+\sigma_1)$$

$$+ (d - d_2)(s_1 + s_2 + s_3 - \min\{s_1, s_2, s_3\}) \quad (3.4)$$

Figure 4

Next, consider the case $C_3 < T_2 \le C_1 + C_3$. If we schedule the jobs $n_1+1, ..., n_2$ on G_2 from d_1 to d and on G_1 from 0 to $(T_2 - C_3)/s_2$ (Figure 5(a)), then we will be left with the GPS of Figure 5(b). The processing capability, K, of H_1 is seen to be :

$$K = s_3(T_2 - C_3)/s_2 + (d_1 - (T_2 - C_3)/s_2)\max\{s_2, s_3\}$$

$$+ (d - d_1)\max\{s_1, s_2, s_3\}$$

We also see that no matter how jobs $n_1+1, ..., n_2$ are scheduled on G_1 and G_2, we cannot create a generalized processor from the remaining idle times on G_1, G_2, and P_3 such that its procesing capability from 0 to d exceeds K. The total processing capability remaining on G_1, G_2, and G_3 will of course be the same no matter how jobs $n_1+1, ..., n_2$ are scheduled. From these observations and Theorem 1, we see that if (3.2) is satisfied and $C_3 < T_2 \le C_1 + C_3$ then it is both necessary and sufficient that $t_{n_2+1} \le$ K in order for d to be feasible.

Now, let us consider the final case, $T_2 > C_1 + C_3$. Let w = $d_1 + (T_2 - C_1 - C_3)/\sigma_1$. From (3.2), it follows that $T_2 \le C_1 + C_2 + C_3$. This together with $T_2 > C_1 + C_3$ implies that $d_1 <$ w $\le$ d. We shall consider the two subcases: (3a) $t_{n_1+1} \le s_2 d_1$ + (w - d_1)σ_1 + (d - w)σ_2 and (3b) $t_{n_1+1} > s_2 d_1$ + (w - d_1)σ_1 + (d - w)σ_2. In subcase (3a), we see from Theorem 1 that jobs $n_1+1, ..., n_2$ can be scheduled on G_1 and G_2 as shown in Figure 6(a). To see this, note that the scheduled portions are equivalent to the GPS of Figure 6(b) and by Theorem 1, jobs $n_1+1, ..., n_2$ can be scheduled on this

Figure 5

GPS.

Figure 6

Once jobs n_1+1, ..., n_2 have been scheduled on G_1 and G_2 in this manner, the remaining idle times on G_1, G_2, and P_3 yield the GPS of Figure 7. From Theorem 1, it follows that jobs n_2+1, ..., n can be scheduled on the GPS of Figure 7 iff $t_{n_2+1} \leq s_3 w + \sigma_4(d - w)$.

Figure 7

In addition, we note that no matter how jobs n_1+1, ..., n_2 are scheduled on G_1 and G_2, the remaining idle times on G_1, G_2, and P_3 cannot yield a generalized processor with capability more than $s_3 w + \sigma_4(d - w)$. Hence, when (3.2) holds, $T_2 > C_1 + C_3$ and $t_{n_1+1} \le s_2 d_1 + (w - d_1)\sigma_1 + (d - w)\sigma_2$, then it is both necessary and sufficient that $t_{n_2+1} \le s_3 w + \sigma_4(d - w)$ in order for d to be feasible.

For subcase (3b), we have $t_{n_1+1} > s_2 d_1 + (w - d_1)\sigma_1 + (d - w)\sigma_2$ (in addition to (3.2) and $T_2 > C_1 + C_3$). First, we observe that it must be the case that $t_{n_1+1} \le C_1 + C_2$ as otherwise, by Theorem 1, t_{n_1+1} cannot be scheduled on G_1 and G_2. Let x be such that $t_{n_1+1} = C_1 + (x - d_1)\sigma_1 + (d - x)\sigma_2$. I.e., x = $d_1 + (t_{n_1+1} - C_1 - C_3)/(\sigma_1 - \sigma_2)$ (note that $\sigma_1 > \sigma_2$ when $s_2 d_1 + (w - d_1)\sigma_1 + (d - w)\sigma_2 < t_{n_1+1} \le C_1 + C_2$). Clearly, w < x ≤ d and t_{n_1+1} can be scheduled on G_1 and G_2 as in Figure 8(a).

Let z = $d_1 + (T_2 - t_{n_1+1})/\sigma_2$. From our earlier definitions of w and x, it follows that z < w < x. Jobs n_1+2, ..., n_2 may be scheduled on G_2 from d_1 to z (Figure 8(b)). Once this is done, the idle times on G_1, G_2, and P_3 are equivalent to the GPS of Figure 9. In Figure 9, $\sigma_{11} = \max\{\sigma_2, s_3\}$ and $\sigma_{12} = \min\{\sigma_2, s_3\}$.

The available capability on H_1 is $s_3 z + \sigma_{11}(x - z) + \sigma_4(d - x)$. One can argue that following any arbitrary scheduling of jobs n_1+1, ..., n_2, there cannot be a generalized processor with greater capability than that of H_1 (Figure 9). Hence, the remaining jobs n_2+1, ..., n can be scheduled iff $t_{n_2+1} \le s_3 z + \sigma_{11}(x - z) + \sigma_4(d - x)$. This additional inequality is both necessary an sufficient for d

Figure 8

Figure 9

to be feasible.

Our discussion on the feasibility of d is summarized in Theorem 2.

Theorem 2: A given d is feasible iff one of the following groups of inequalities holds.

GROUP 1 (case 1):

$$\sum_{i=1}^{k} T_i / \sum_{i=1}^{k} s_i \le d,\ k = 1, 2, 3 \tag{3.5}$$

$$T_2 \le C_3 = \sigma_2(d - d_1) \tag{3.6}$$

$$t_{n_2+1} \le d_1 \max\{s_2, s_3\} + (d - d_1) max\{s_1, s_2, s_3\} \tag{3.7}$$

$$t_{n_2+1} + t_{n_2+2} \le d_1(s_2 + s_3) + (d_2 - d_1)(s_3 + \sigma_1) + \\ (d - d_2)(s_1 + s_2 + s_3 - \min\{s_1, s_2, s_3\}) \tag{3.8}$$

GROUP 2 (case 2):

$$\sum_{i=1}^{k} T_i / \sum_{i=1}^{k} s_i \le d,\ k = 1, 2, 3 \tag{3.5}$$

$$C_3 < T_2 \le C_1 + C_3 \tag{3.9}$$

$$t_{n_2+1} \le s_3(T_2 - C_3)/s_2 + (d_1 - (T_2 - C_3)/s_2) max\{s_2, s_3\} \\ + (d - d_1)\max\{s_1, s_2, s_3\} \tag{3.10}$$

GROUP 3 (case 3a):

$$\sum_{i=1}^{k} T_i / \sum_{i=1}^{k} s_i \le d,\ k = 1, 2, 3 \tag{3.5}$$

$$C_1 + C_3 < T_2 \tag{3.11}$$

$$t_{n_1+1} \le s_2 d_1 + \sigma_1(w - d_1) + \sigma_2(d - w) \tag{3.12}$$

$$t_{n_2+1} \le s_3 w + \sigma_4(d - w) \tag{3.13}$$

GROUP 4 (case 3b):

$$\sum_{i=1}^{k} T_i / \sum_{i=1}^{k} s_i \le d,\ k = 1, 2, 3 \tag{3.5}$$

$$C_1 + C_3 < T_2 \tag{3.11}$$

$$t_{n_1+1} > s_2 d_1 + \sigma_1(w - d_1) + \sigma_2(d - w) \tag{3.14}$$

$$t_{n_1+1} \le C_1 + C_2 \tag{3.15}$$

$$t_{n_2+1} \le s_3 z + \sigma_{11}(x - z) + \sigma_4(d - x) \tag{3.16}$$

(Note that (3.6), (3.9), and (3.11) are mutually exclusive, also (3.12) and (3.14) are mutually exclusive. So, for any d, at most one group of inequalities may hold.) []

Each inequality above (3.5 - 3.16) may be rearranged so as to be of the form

$$d \geq f(\ ),\ d > f(\ ),\ d \leq f(\ ),\ \text{or}\ d < f(\ )$$

where f is a function of the remaining variables. For each group, one can determine in O(n) time, the minimum d (if any) that satisfies all inequalities in that group. If no d does, the minimum may be set to ∞. The minimum of the ds so obtained for the four groups above yields the minimum $C_{\max}$.

Once the $C_{\max}$ has been determined the corresponding schedule may be constructed in O(n) time as discussed.

3.3 Two Class of Processors

Let A and B be two classes of processors such that all processors in A have the same speed, s_A, while all in B have the same memory size, μ_B. Further, assume that the memory size of every processor in A is greater than that of the processors in B. The n jobs to be scheduled may be partioned into two disjoint sets R and T such that the memory requirement of every job in R exceeds μ_B while the memory requirement of every job in T is less than or equal to μ_B.

It is clear that jobs in R can be scheduled only on processors in A. The set of processors in A forms a system of identical processors with memory. Kafura and Shen [8] have developed necessary and sufficient conditions for d to be feasible for a system of identical processors with memory. Let |A| = a and |R| = r. Assume that $\mu_1 \geq \mu_2 \geq ... \geq \mu_a$ and that $m_1 \geq m_2 \geq ... \geq m_r$. Let F_i denote the subset of R consisting only of those jobs with memory requirement more than μ_{i+1}, $1 \leq$ i < a. Let F_a = R. We assume that $m_1 \leq \mu_1$. Let X_i be the sum of the processing times of all jobs in F_i, $1 \leq i \leq a$. Kafura and Shen [8] have shown that the job set R can be scheduled on the processor set A to complete by time d iff d satisfies the inequality:

$$d \geq f^* = \max\{ \max_{1 \leq i \leq r}\{t_i/s_A\}, \max_{1 \leq i \leq a}\{X_i/(s_A i)\} \} \tag{3.17}$$

Furthermore, when d satisfies the above inequality the desired schedule may be obtained by scheduling the jobs in the order 1, 2, ..., r using McNaughton's strategy [12] on the processors in the order 1, 2, ..., a. The resulting schedule fully utilizes processors 1, 2, ..., b where b = $fP(af\sum_{i=1}^{r} t_i/(s_A d)$. *In addition, if b < a then processor b+1 is utilized from 0 to c =* $rem(\sum_{i=1}^{r} t_i,\ s_A d) / s_A$ *where rem(x, y) is the remainder of x divided by y. Hence the schedule is as in Figure 10.*

Let S be any schedule for the jobs in R $\cup$ T. Let $C_{\max}$(S) = d. Clearly, d $\geq$ f*. We shall

Figure 10 Scheduling R on A

show that S can be transformed into a schedule Q with $C_{\max}(Q) = d$ and Q is such that the jobs in R are scheduled as in Figure 10. Suppose that the jobs in R are not scheduled as in Figure 10. Let Δ be such that d/Δ is an integer and there are no preemptions or completion in the interval $(i\Delta, (i+1)\Delta)$, $0 \le i < d/\Delta$. Since the number of preemptions in S is finite and since all the values concerned are rationals, such a Δ clearly exists. Note that, by definition of Δ, it is not possible for two jobs to be scheduled in the same Δ interval on any processor. Figure 11(a) shows a possible configuration for S. The heavy lines represent jobs in R; the light lines represent either jobs in T or idle times.

First, we shift the heavy lines in each Δ interval to the top of the Δ interval. This results in the light lines moving downwards (but remaining on the A processors). The result is the schedule of Figure 11(b). This shifting preserves feasibility as the heavy lines move up to processors with at least as much memory capacity. The light lines represent jobs in T and these can be processed on any of the processors in A. In addition, all procesors in A have the same speed. So, the amount of processing done in a Δ interval is the same for all procesors in A.

The next transformation is to permute the Δ intervals so that the number of heavy lines in each interval is nonincreasing , left to right. This results in the schedule of Figure 11(c). As we

Figure 11

can see, our schedule is still not in the form of that in Figure 10. In order to obtain a schedule in the right form, slots a, b, and c need to be replaced by heavy lines and slots d, e, and f are to be replaced by light lines. This can be done in a systematic manner as discussed below.

Select the rightmost column into which a heavy line is to be introduced. Call this column u. In this column select the uppermost slot containing a light line. In Figure 11(c), this results in slot a being selected. Next, select the leftmost column into which a light line is to be introduced. Call this column v. In this column, select the lowest slot into which there is a heavy line. This yields slot f of Figure 11(c). It should be clear that column u has fewer heavy lines in it than does column v. Hence, at least one of the heavy lines in v represents a job not scheduled in column u. We shall select one such heavy line to be moved into column u. This selection is done by first lining up heavy lines representing jobs common to u and v.

Suppose the job representation for heavy lines for an example u and v are as in Figure 12(a). Suppose job i is in both u and v and that it is in row r_u of u and row r_v of v. If $r_u < r_v$, then i in column u is interchanged with the line in row r_v of column u. This preserves feasibility as $m_i \leq \mu_{r_v}$. If $r_u > r_v$, then i in column v is interchanged with the line in row r_u of column v. This also preserves feasibility. When these interchanges are performed on Figure 12(a), the configuration of Figure 12(b) is obtained.

Figure 12

For movement into column u, we may select any one of the heavy lines in column v that are lined up with a light line in column u (lines 1 and 9 in Figure 12(b)). This guarantees that the new set of heavy lines in column u can be feasibly assigned to the processors. The remaining heavy lines in v may be shifted upwards.

Now, we have to move one of the light lines in u out to column v. Suppose that u and v are as in Figure 13(a). X_1 denotes the jobs from T assigned to rows of u that have light lines in u but heavy ones in v. Let the number of heavy lines in v be w. From the choice of u and v, it follows that $|X_1| = z \geq 1$. If X_1 contains a job that has not been scheduled in column v, then this job may be selected for movement into column v. Otherwise, permute processors w+1, ..., m so that the set of jobs scheduled in column v on processors w+1, w+2, ..., w+z is X_1. Let X_2 denote the set of jobs scheduled in u on processors w+1, ..., w+z. If every job in X_2 is scheduled in column v, then permute processors w+z+1, ..., m so that the set of jobs scheduled on w+z+1, ..., w+2z in column v is X_2. This permuting of processors is continued until we have an X_k such that at least one job in X_k is not scheduled in column v. Such an X_k exists as u has fewer heavy lines than does v. Let $x_0 \ \varepsilon\ X_k$ be such that x_0 is not scheduled in v. Interchange x_0 with the job, x_1, scheduled on the same processor in v. Clearly, $x_1 \ \varepsilon\ X_{k-1}$. Hence x_1 is scheduled in u on one of the processors w + (k-3)z + j, 1 ≤ j ≤ z. Let this processor have index q. Interchange the assignment of x_1 on processor q column u with that of the job, x_2, in column v of q. Now, $x_2 \ \varepsilon\ X_{k-2}$ and x_2 is scheduled on q, in column u for some q, q ε $\{ w+(k-4)z+j \mid 1 \leq j \leq z\}$. We may continue interchanges in this way until we determine $x_{k-1} \ \varepsilon\ X_1$ that needs to be moved out of X_1 from column u. This x_{k-1} is to be moved into the last heavy line spot in column v.

By repeating the interchange process described above a finite number of times, the configuration of Figure 11(c) can be transformed into the form of Figure 10. Having established this result, we see that we need concern ourselves only with schedules in which jobs in R are scheduled as in Figure 10. For any d, the remaining idle times in A and the processors in B have the form given in Figure 14(a). The processors from B are indexed a+1, a+2, ..., m. It is assumed that $s_{a+1} \geq s_{a+2} \geq \ldots \geq s_{a+k} \geq s_A \geq s_{a+k+1} \geq \ldots \geq s_m$. The conditions for d to be feasible on this system of processors are easily obtained by transforming the system into an equivalent GPS (Figure 14(b)).

Assume that the jobs in T are indexed r+1, r+2, ..., n and that $t_{r+1} \geq t_{r+2} \geq \ldots \geq t_n$. Let $V_i = \sum_{j=1}^{i} t_{r+j}$, 1 ≤ i < m-b and $V_{m-b} = \sum_{j=1}^{n-r} t_{r+j}$. If n-r < m-b, then let $V_{n-r} = V_{n-r+1} = \ldots = V_{m-b} = \sum_{j=1}^{n-r} t_{r+j}$. From Theorem 1, we see that d is feasible iff:

$$V_i \leq \begin{cases} \sum_{j=1}^{i} s_{a+j} d, \ 1 \leq i \leq k \\ \sum_{j=1}^{k} s_{a+j} d + (i-k) s_A d, \ k+1 \leq i < k+a-b \\ \sum_{j=1}^{k} s_{a+j} d + (a-b-1) s_A d + s_A (d-c) \\ \quad + \sum_{j=1}^{i-(k+a-b)} d s_{a+k+j} + c s_{b+i+1}, \ k+a-b \leq i \leq m-b \end{cases} \tag{3.18}$$

For a given b, the least d (if any) that satisfies (3.17) and (3.18) can be found in O(n) time. Since as b decreases (increases) this least d increases (decreases), the optimal b can be obtained in O(log a) = O(log m) attempts. Hence, the least feasible d can be obtained in O(n log

Figure 13

m) time. Note that the $X_i s$ (Eq (3.17)) can be obtained in O(r log a) time and that $V_i s$ (Eq (3.18)) can be obtained in O((n-r) + (m-b) log (m-b)) time (as only the largest m-b jobs in T need to be ordered). So, the total time needed to compute $(C_{\max})_{\min}$ is O(n log m) (assume $n \geq m$). The actual schedule can be constructed in an additional O(mn) time using the algorithm of [12] for R and that of [13] for T. The time O(mn) can be reduced to O(n) using a scheme similar to that of [7] for T.

Figure 14

4. Conclusions

We have obtained linear programming formulations for the general $C_{\max}$ and $L_{\max}$ problems for uniform processors with memories. For three special cases, low order polynomial time algorithms for the $C_{\max}$ problem when all jobs are released at time 0 have also been obtained.

References

1. G. Dantzig, "Khachian’s algorithm: A comment," *SIAM News*, Vol. 13, No. 5, Oct. 1980.
2. P. Gacs and L. Lovasz, "Khachian’s algorithm for linear programming," *Math. Prog. Study*, Vol. 14, 1981, PP. 61-68.
3. M. R. Garey and D. S. Johnson, "Computers and intractability, a guide to the theory of NP-Completeness," W. H. Freeman and Co., San Francisco, 1979.
4. S. Gass, "Linear programming," McGraw Hill Book Co., New York, 1969.
5. T. Gonzalez and D. Johnson, "A new algorithm for preemptive scheduling of trees," *JACM*, Vol. 27, No. 2, 1980, pp. 287-312.
6. T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *JACM*, Vol. 23, No. 4, 1976, PP. 665-679.
7. T. Gonzalez and S. Sahni, "Preemptive scheduling of uniform processor systems," *JACM*, Vol. 25, 1978, PP. 92-101.
8. D. G. Kafura and V. Y. Shen, "Task scheduling on a multiprocessor system with independent memories," *SICOMP*, Vol. 6, No. 1, 1977, PP. 167-187.
9. L. G. Khachian, "A polynomial algorithm in linear programming," *Soviet Math. Dokl.*, Vol. 20, No. 1, 1979, PP. 191-194.
10. V. Klee and G. L. Minty, "How good is the simplex algorithm?" in O. Shisha (ed.) *Inequalities III*, Academic Press, New York, 1972, PP. 159-175.
11. T. H. Lai and S. Sahni, "Preemptive scheduling of a multiprocessor system with memories to minimize $L_{\max}$," Report No. 81-20, Computer Science Dept., Univ. of Minnesota, Minneapolis, 1981.
12. R. McNaughton, "Scheduling with deadlines and loss functions," *Manag-Sci*, 12, 7, 1959.
13. S. Sahni and Y. Cho, "Scheduling independent tasks with due times on a uniform processor system," *JACM*, Vol. 27, No. 3, 1980, PP. 550-563.
14. S. Sahni and Y. Cho, "Nearly on line scheduling of a uniform processor system with release times," *SICOMP*, Vol. 8, No. 2, 1979, pp. 275-285.