E. B. Fernández
T. Lang

Computation of Lower Bounds for Multiprocessor Schedules

Abstract: A multiprocessing system composed of identical units is considered. This system is executing a set of partially ordered tasks, with known execution times, using a non-preemptive scheduling strategy. Lower bounds on the number of processors required to compute the tasks before a deadline, and on the minimum time to execute the tasks with a fixed number of processors, are of great value for the determination of the corresponding optimal schedules. In this paper, methods for the efficient computation of the lower bounds obtained by Fernández and Bussell are discussed. Computational improvements for the case of general partial orders are reported, and further reductions of the number of operations are shown to be possible for special graphs (trees, independent chains, independent tasks).

1. Introduction

A significant decrease in cost for digital computer functional units has occurred in the last few years, due to the extensive use of LSI techniques. This has made the design of multiprocessing systems composed of a large number of functional units a practically feasible endeavor, and many proposals for such systems have appeared [1, 2]. Multiprocessor scheduling is thus becoming a more significant problem and several papers dealing with this subject have been published [3, 4, 5].

Two basic problems are of great interest in multiprocessor scheduling theory: 1) a deadline for the execution time of a given set of tasks must be satisfied using a minimum number of processors; 2) a fixed number of processors must be used to execute a set of tasks in a minimum time.

A computation to be scheduled is represented by some precedence graph indicating the relative order in which the tasks must be executed. The graph ranges from an arbitrary partial order to an empty ordering, i.e., a set of independent tasks. The execution times of the tasks can be estimated in a probabilistic way (time probability distributions) or in a deterministic way (average or maximum execution times). In this paper we are concerned with the case for which the task execution times are deterministically known.

In all known algorithmic methods used to solve the two basic problems defined above, it is necessary to have a starting value for either the number of processors or the total execution time [3, 4]. For instance, in the first case the scheduling algorithm tries to find a schedule using the initial assumed number of processors. If no schedule can be obtained, the initial value is incremented by one and the scheduling algorithm is applied again. The whole process is repeated until a schedule satisfying the deadline is found. The actual number of processors required by this schedule is the optimal value for the given deadline. Since the scheduling of general partial orders is basically an enumerative process [6], the amount of extra work can be enormous if the assumed starting value is not close to the actual number of required processors. The same considerations apply to the second problem. The determination of starting values, i.e., lower bounds, is then an important problem to be solved before applying any scheduling algorithm. Moreover, it is essential that the computational work involved in determining these bounds be very small compared with the complexity of applying some scheduling algorithm to produce all the possible schedules for a given value of the bound. Lower bounds of this type are also valuable for evaluating approximate scheduling methods [5].

In a previous paper [7], the problem of obtaining accurate lower bounds was discussed. Two expressions were obtained which represented sharper lower bounds than those obtained previously. In this paper we study the efficient calculation of these expressions. We discuss also how this efficiency can be improved for special partial orders, in particular trees, independent chains, and independent tasks, and for graphs where all the tasks have equal execution time.

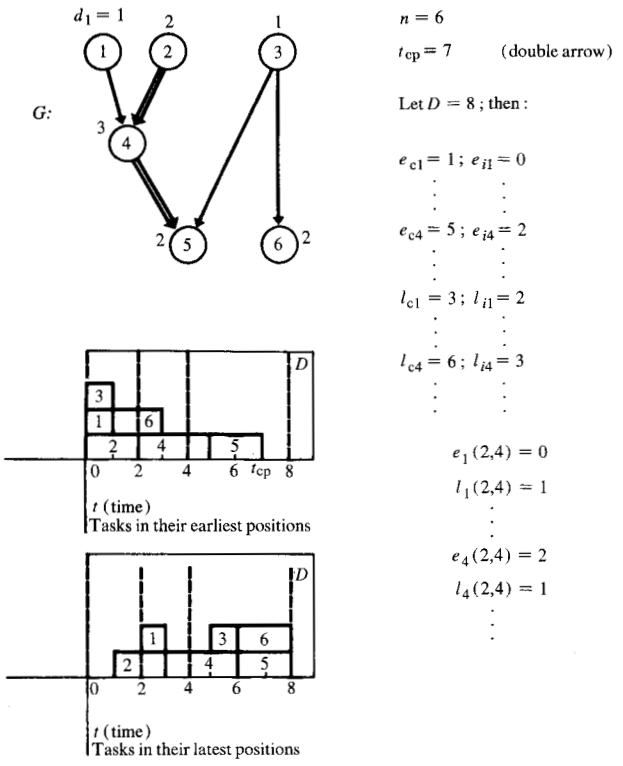

Figure 1 Basic definitions.

In section 2 we present the model, some basic definitions and a summary of the pertinent results from [7]. In section 3 we consider general graphs. The direct determination of the lower bounds requires $O(nD^2)$ operations, where n is the number of tasks and D the deadline for the execution of these tasks. Incremental ways are then proposed to evaluate the expressions and the number of operations becomes $O(D^2)$. The possibility of using parallel computation for this case is also briefly discussed. Since $O(D^2)$ is a natural limit for this expression because of the number of intervals that must be considered, reductions in the number of needed intervals are sought by looking at particular partial orders. In section 4 we discuss the determination of the lower bound on the number of processors for independent tasks. By means of two theorems the number of intervals to consider is reduced to about $D/2$. The total number of operations is $O(D^2)$, which can be reduced by means of incremental methods to $O(D)$ operations. A similar analysis is performed in section 5 for the lower bound on time for independent tasks. Cases of trees and independent chains, where reductions in the number of intervals with respect to general graphs are obtained, are considered in section 6. Finally, in section 7 we evaluate the results obtained.

2. Model, definitions, and previous results

A set of *tasks* $T = \{T_1, T_2, \cdots, T_n\}$ is to be executed by a set of identical *processors* P_i, $i = 1, 2, \cdots, m$. A partial order $<$ is given on T, and a non-negative integer d_j represents the *duration of execution* of task T_j.

The partially ordered set $(T, <)$ is described by a finite, acyclic digraph $G = (V, A)$, where V is a finite set of vertices of cardinality n, and A is a set of arcs represented as vertex pairs. The tasks T_j correspond to the elements of V, and we shall talk interchangeably of tasks or vertices. The arcs in A describe the precedences among the tasks.

Once a processor begins to execute a task it cannot be interrupted until its completion; that is, we have a *non-preemptive* type of scheduling. It is assumed that tasks are scheduled to start only at integer values of time.

The length of the critical path of the graph, t_{cp}, is the minimum time to perform the set of computations. More general, and more useful, is to talk of a deadline, D, i.e., a time within which the set of computations must be finished. Clearly, $D \geq t_{cp}$. According to some possible schedule, for each task T_j we have a specific *completion time* which we denote as c_j. C is the completion time vector, whose jth component is c_j. Of particular interest are the two extreme task completion times defined below.

The *earliest completion time*, e_{cj}, of a task T_j is the minimum time in which this task can be finished, given the precedence constraints of the graph.

The *latest completion time*, l_{cj}, of a task T_j indicates how long the completion time of this task can be delayed without exceeding the deadline.

According to a possible schedule, there exists also an *initiation time* i_j for a given task T_j. In analogy with the definitions above, it is then possible to define for a task T_j an *earliest initiation time*, e_{ij}, and a *latest inititation time*, l_{ij}.

If all the tasks are in their earliest possible positions, the number of units of task T_j that lie in the interval $[t_1, t_2]$ is denoted as $e_j(t_1, t_2)$. Similarly, if all tasks lie in their latest possible positions, the number of units of task T_j in this interval is called $l_j(t_1, t_2)$.

The concepts presented above are illustrated in Fig. 1. By making use of these definitions the lower bounds presented in [7] can be expressed as below.

A lower bound on the minimum number of processors required to perform the computations of G in time D is given by

$$m_L = \left\lceil \max_{[t_1,t_2]} \left\{ \frac{1}{t_2 - t_1} \sum_{j=1}^{n} \min[e_j(t_1, t_2), l_j(t_1, t_2)] \right\} \right\rceil. \qquad (1)$$

If the number of processors, m, is not sufficient for performing the computations of G within time D, a lower

bound on the minimum time to execute G with these m processors is given by

$$t_2 = D + \left\lceil \max_{[t_1, t_2]} \left\{ -(t_2 - t_1) + \frac{1}{m} \sum_{j=1}^{n} \min[e_j(t_1, t_2), l_j(t_1, t_2)] \right\} \right\rceil. \quad (2)$$

Because of its importance for computational purposes, we make the following definition:

$$M(t_1, t_2) = \sum_{j=1}^{n} \min[e_j(t_1, t_2), l_j(t_1, t_2)],$$

which allows rewriting (1) and (2) as

$$m_L = \left\lceil \max_{[t_1, t_2]} \frac{M(t_1, t_2)}{t_2 - t_1} \right\rceil, \quad (3)$$

$$t_L = D + \left\lceil \max_{[t_1, t_2]} \left[-(t_2 - t_1) + \frac{1}{m} M(t_1, t_2) \right] \right\rceil. \quad (4)$$

The operation defined below is used later in this work:

$$a \dot{-} b = \begin{cases} a - b \text{ if } a \geq b, \\ 0 \text{ otherwise.} \end{cases} \quad (5)$$

3. Calculation of the lower bounds for general graphs

• *3.1 Direct computation*

It can be seen from the expressions for m_L and t_L that the key point in their evaluation is the calculation of

$$M(t_1, t_2) = \sum_{j} \min_{[t_1, t_2]} [e_j(t_1, t_2), l_j(t_1, t_2)]$$

for every integer interval.

In this section we shall see the complexity of calculating this expression directly. The number of operations obtained is used as a reference to evaluate alternative procedures. It should be noticed that the number of intervals to consider is $\frac{1}{2}D(D+1)$, and that therefore any computation method will require at least $O(D^2)$ operations.

With reference to Fig. 2, which shows all the possible positions of T_j with respect to $[t_1, t_2]$, it can be seen that $e_j(t_1, t_2)$ and $l_j(t_1, t_2)$ can be expressed as

$$e_j(t_1, t_2) = \min[e_{cj} \dot{-} t_1), d_j, (t_2 - t_1), (t_2 \dot{-} e_{ij})], \quad (6)$$

$$l_j(t_1, t_2) = \min[(t_2 \dot{-} l_{ij}), d_j, (t_2 - t_1), (l_{cj} \dot{-} t_1)], \quad (7)$$

and because $(t_2 \dot{-} e_{ij}) \geq (t_2 \dot{-} l_{ij})$ and $(l_{cj} \dot{-} t_1) \geq (e_{cj} \dot{-} t_1)$, we have that

$$\min[e_j(t_1, t_2), l_j(t_1, t_2)] = \min[(e_{cj} \dot{-} t_1), d_j, (t_2 - t_1), (t_2 \dot{-} l_{ij})]. \quad (8)$$

For a given interval $[t_1, t_2]$, the determination of this minimum requires one subtraction to obtain $(t_2 - t_1)$,

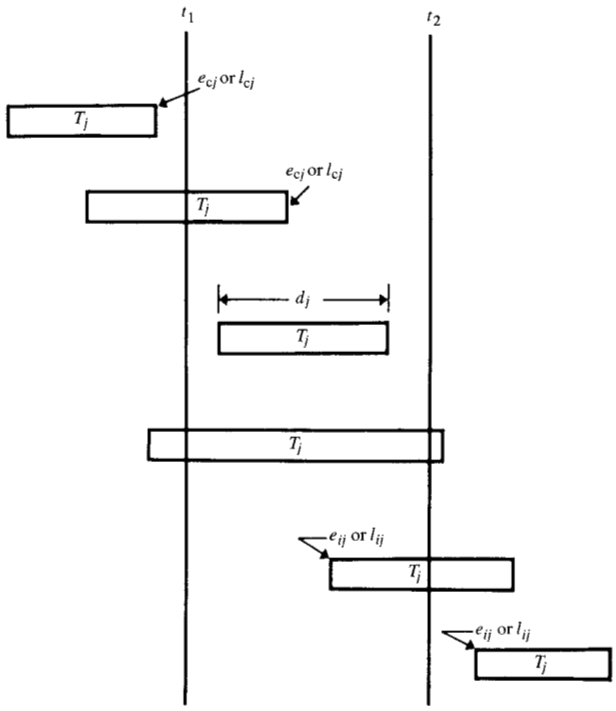

Figure 2 Possible relative positions of t_1, t_2, and T_j.

and then two subtractions and three comparisons per task. Finally, $(n-1)$ additions are required to perform the summation, which gives a total of

$1 + 2n + 3n + (n-1) = 6n$ operations per interval.

For all intervals we have

$6n \times \frac{1}{2}D(D+1) \approx 3nD^2$ operations

to determine $M(t_1, t_2)$.

• *3.2 Incremental computation*

A reduction over the number of steps required for direct computation of the lower bounds can be obtained by using an incremental strategy where $M(t_1, t_2)$ is determined from $M(t_1, t_2 - 1)$. The value of t_1 is varied between 0 and $D-1$, and for each t_1 all the intervals $[t_1, t_2]$ are determined by varying t_2 between $t_1 + 1$ and D.

We show in this section that $M(t_1, t_2)$ can be computed incrementally using the expression

$$M(t_1, t_2) = M(t_1, t_2 - 1) + F_{t_1}(t_2),$$

and therefore only one addition per interval is required to do this computation. For each value of t_1 a new function F_{t_1} has to be computed, and we will see that $4nD$ operations are required for this. We therefore obtain a total of $4nD + \frac{1}{2}D(D+1)$ operations, which is considerably better than the direct method. In fact it can be argued that this is a nearly optimal method because in

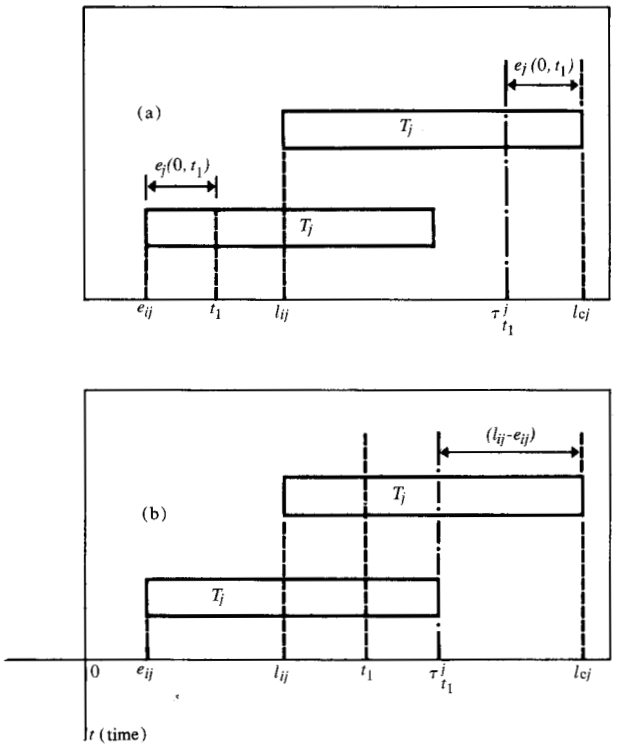

Figure 3 Proof of Lemma 1: (a) $t_1 \leq l_{ij}$; (b) $t_1 > l_{ij}$.

any algorithm all intervals have to be considered and at least one operation has to be performed per interval.

Lemma 1

$$\min[l_j(t_1, t_2), e_j(t_1, t_2)] = \begin{cases} l_j(t_1, t_2) & \text{if } t_2 \leq \tau^j_{t_1} \\ l_j(t_1, \tau^j_{t_1}) & \text{if } t_2 > \tau^j_{t_1} \end{cases}$$

where

$$\tau^j_{t_1} = \begin{cases} l_{cj} - e_j(0, t_1) & t_1 \leq l_{ij} \\ l_{cj} - (l_{ij} - e_{ij}) & t_1 > l_{ij}, \end{cases}$$

that is, for $t_1 \leq l_{ij}$, this minimum is given by $l_j(t_1, t_2)$ if $t_2 \leq l_{cj} - e_j(0, t_1)$, and it is independent of t_2 afterwards. Similarly, for $t_1 > l_{ij}$, the minimum is given by $l_j(t_1, t_2)$ up to $t_2 = l_{cj} - (l_{ij} - e_{ij})$, and it is independent of t_2 afterwards.

The proof of this lemma is evident from Fig. 3.

We now define a density function $f^j_{t_1}$ such that

$$f^j_{t_1}(t) = \begin{cases} 1 \text{ for } l_{ij} \leq t \leq \tau^j_{t_1}, \\ 0 \text{ elsewhere.} \end{cases} \tag{9}$$

By Lemma 1 we have

$$\min[l_j(t_1, t_2), e_j(t_1, t_2)] = \sum_{k=t_1}^{t_2} f^j_{t_1}(k).$$

We can then write

$$M(t_1, t_2) = \sum_{j=1}^{n} \min [l_j(t_1, t_2), e_j(t_1, t_2)]$$

$$= \sum_{j=1}^{n} \sum_{k=t_1}^{t_2} f^j_{t_1}(k),$$

and interchanging the summations and defining

$$F_{t_1}(t) = \sum_{j=1}^{n} f^j_{t_1}(t),$$

we obtain

$$M(t_1, t_2) = \sum_{k=t_1}^{t_2} F_{t_1}(k),$$

and therefore

$$M(t_1, t_2) = M(t_1, t_2 - 1) + F_{t_1}(t_2), \tag{10}$$

with the initial condition $M(t_1, t_1) = 0$.

Consequently, we can compute the M's incrementally by just one addition per interval.

Furthermore, we have to calculate F_{t_1}, also incrementally. From the definition we obtain

$$F_{t_1} - F_{t_1-1} = \sum_{j=1}^{n} (f^j_{t_1} - f^j_{t_1-1});$$

and from the definition of $f^j_{t_1}$,

$$f^j_{t_1} - f^j_{t_1-1} = -\delta[\tau^j_{t_1}] \times (\tau^j_{t_1} - \tau^j_{t_1-1}),$$

where $\delta[a] = 1$ for $t = a$ and 0 elsewhere. Consequently,

$$F_{t_1} = F_{t_1-1} - \sum_{n=1}^{n} \delta[\tau^j_{t_1}] \times (\tau^j_{t_1} - \tau^j_{t_1-1}) \tag{11}$$

with the initial condition

$$F_0 = \sum_{j=1}^{n} f^j_0,$$

which corresponds to the "latest load density function" of section 6.1.

We now present an Algol-like algorithm to compute $M(t_1, t_2)$ incrementally by the use of expressions (10) and (11).

Arrays **M**[0 : D,0 : D] contains $M(t_1, t_2)$;
F[0 : D] contains F_{t_1} (initially latest load density function);
LI[1 : n] contains the latest initiation times;
EI[1 : n] contains the earliest initiation times;
T[1 : n] contains the values of $\tau^j_{t_1}$ (initially l_{c_j});
EC[1 : n] contains the earliest completion times;

```
BEGIN
  FOR t1 = 0 UNTIL D − 1 DO
      BEGIN
        FOR j = 1 UNTIL n DO
          BEGIN 'COMPUTE NEW F';
            IF (t1 > EI(j))∧(t1 ≤ LI(j))∧(t1 ≤ EC(j))
            THEN
              BEGIN
                F(T(j)) = F(T(j)) − 1;
                T(j) = T(j) − 1;
              END;
          END;
        M(t1, t1) = 0;
      FOR t2 = t1 + 1 UNTIL D DO
        M(t1, t2) = M(t1, t2 − 1) + F(t2);
      END;
END.
```

From this algorithm it is possible to see that we need $4nD$ operations for computing the F's, and $\frac{1}{2}D(D+1)$ operations for calculating the M's.

• 3.3 Parallel computation

Parallel computation can reduce the time considerably but at the cost of a large number of processors. In effect, to compute the three subtractions and the three comparisons of the expression (8) in parallel, we could use a processor per task per interval. Then, to perform the additions to consider all tasks, a log-sum procedure [8] can be used.

The number of computation cycles is then

$$6 + \lceil \log_2 n \rceil,$$

requiring a total of $(n/2)D(D+1) \approx (n/2)D^2$ processors.

• 3.4 Completing the calculation

In order to determine m_L and t_L, $M(t_1, t_2)$ must be divided by the interval length $t_2 - t_1$ and the number of processors m, respectively. The number of divisions to be performed in the first case can be minimized by noticing that

$$\max_{[t_1,t_2]} \left\lceil \frac{M(t_1, t_2)}{t_2 - t_1} \right\rceil = \max_{d} \left\{ \frac{1}{d} \left\lceil \max_{(t_2 - t_1 = d)} M(t_1, t_2) \right\rceil \right\}.$$

In other words, by first determining the maximum value of $M(t_1, t_2)$ for each interval size d, only one division per interval size is necessary. This reduces the number of divisions from one per interval of length greater than 1, i.e., $[\frac{1}{2}D(D+1) - D]$, to one per interval size, i.e., $(D-1)$ divisions.

As divisions are usually considerably slower than multiplications and sums, it is possible to obtain a further reduction of divisions by noticing the following fact. Suppose the values of the maxima are obtained in an order corresponding to increasing length of the intervals. Assume $k(d)$ is the maximum obtained up to intervals of length d, and $M(d)$ is the maximum obtained for intervals of length d. Then

$$k(d+1) > k(d) \text{ iff } M(d+1) > k(d) \times (d+1).$$

Consequently, instead of dividing for every length, we perform the multiplication and the comparison, and only divide if the inequality holds.

4. Calculation of the lower bound on the number of processors for independent tasks

We now study the calculation of the lower bound on the number of processors for the case in which the tasks are independent. We show that in this case the number of operations can be reduced considerably because the number of intervals that have to be considered is $\frac{1}{2}D$ instead of $\frac{1}{2}D(D+1)$. The calculation for these intervals can be performed efficiently in an incremental manner.

• 4.1 A simplified expression for the lower bound

To simplfy the notation in the subsequent development we now make the following change of variables to describe the intervals: $[t_1, t_2] = [\alpha, \beta]$, where $\alpha = t_1$ and $\beta = D - t_2$. Then we define

$$b(\alpha, \beta) = M(\alpha, \beta) / (D - \alpha - \beta), \tag{12}$$

which represents the average of the number of tasks that have to be executed in interval $[\alpha, \beta]$, and therefore corresponds to a bound on the number of processors required to execute the tasks in this interval. The lower bound on the number of processors for all intervals is then given by

$$m_L = \left\lceil \max_{[\alpha,\beta]} b(\alpha, \beta) \right\rceil. \tag{13}$$

As the tasks are independent, the earliest completion time of T_j is d_j and the latest completion time is D (Fig. 4). Due to the symmetry of $e_j(\alpha, \beta)$ and $l_j(\alpha, \beta)$ with respect to an axis through $\frac{1}{2}D$, $b(\alpha, \beta) = b(\beta, \alpha)$. Therefore, we calculate $b(\alpha, \beta)$ only for intervals where $\beta \geq \alpha$.

The contributions to $b(\alpha, \beta)$ can be divided into three classes:

a. tasks with $d_j \leq \beta$ do not contribute to the bound because $e_j(\alpha, \beta)$ is zero in that interval;
b. tasks with $\beta < d_j < D - \alpha$ contribute with $(d_j - \beta) / (D - \alpha - \beta)$;
c. tasks with $d_j \geq D - \alpha$ contribute with one unit each because they span the whole interval.

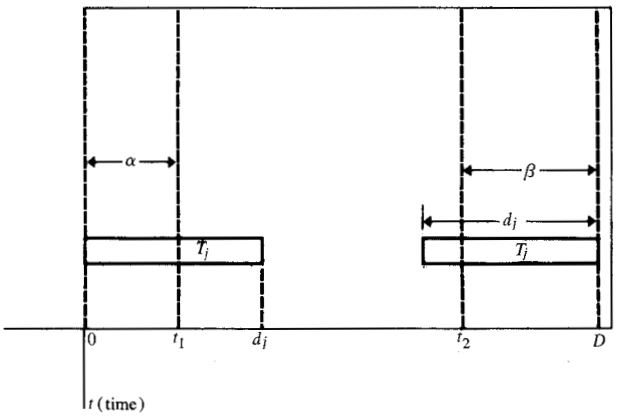

Figure 4 Extreme positions for independent tasks.

Consequently, $b(\alpha, \beta)$ is given by

$$b(\alpha, \beta) = \sum_{L=D-\alpha}^{D} n_L + \sum_{L=\beta+1}^{D-\alpha-1} \frac{n_L(L-\beta)}{D-\alpha-\beta} \tag{14}$$

where n_L is the number of tasks of duration L.

The two theorems below reduce the number of intervals that has to be considered in the evaluation of (13).

Theorem 1 For a graph composed of independent tasks,

$$b(\alpha, \beta) \leq b(\beta, \beta) \text{ for } \beta < \left\lceil \frac{D}{2} \right\rceil \text{ and } \beta \geq \alpha.$$

Proof By (14),

$$b(\beta, \beta) = \sum_{L=D-\beta}^{D} n_L + \sum_{L=\beta+1}^{D-\beta-1} \frac{n_L(L-\beta)}{D-2\beta}, \tag{15}$$

which can be further decomposed into

$$b(\beta, \beta) = \sum_{L=D-\alpha}^{D} n_L + \sum_{L=D-\beta}^{D-\alpha-1} n_L + \sum_{L=\beta+1}^{D-\beta-1} \frac{n_L(L-\beta)}{D-2\beta}.$$

In the same way, $b(\alpha, \beta)$ can be decomposed into

$$b(\alpha, \beta) = \sum_{L=D-\alpha}^{D} n_L + \sum_{L=D-\beta}^{D-\alpha-1} \frac{n_L(L-\beta)}{D-\alpha-\beta} + \sum_{L=\beta+1}^{D-\beta-1} \frac{n_L(L-\beta)}{D-\alpha-\beta}.$$

We now compare term by term both expressions. The first terms are the same; the second term is larger for $b(\beta, \beta)$ because $L - \beta < D - \alpha - \beta$ for that range of values of L, and the third term is also larger for $b(\beta, \beta)$ because $D - 2\beta \geq D - \alpha - \beta$. Consequently, $b(\beta, \beta) \geq b(\alpha, \beta)$ and the theorem is proven.

This theorem indicates that for $\beta < \lceil D/2 \rceil$, only the intervals (β, β) have to be considered to obtain the bound. We now prove another theorem that shows that no additional intervals have to be considered for $\beta \geq \lceil D/2 \rceil$.

Theorem 2 For a graph composed of independent tasks,

$$b(\alpha, \beta) \leq b\left(\left\lceil \frac{D}{2} \right\rceil - 1, \left\lceil \frac{D}{2} \right\rceil - 1\right),$$

for $\beta \geq \left\lceil \frac{D}{2} \right\rceil$ and $\beta \geq \alpha$.

Proof By (14),

$$b\left(\left\lceil \frac{D}{2} \right\rceil - 1, \left\lceil \frac{D}{2} \right\rceil - 1\right) = \sum_{L=\lceil D/2 \rceil}^{D} n_L \text{ for } D \text{ odd};$$

$$= \sum_{L=(D/2)+1}^{D} n_L + \tfrac{1}{2} n_{D/2} \text{ for } D \text{ even.}$$

This expression can be decomposed into

$$b\left(\left\lceil \frac{D}{2} \right\rceil - 1, \left\lceil \frac{D}{2} \right\rceil - 1\right) = \sum_{L=D-\alpha}^{D} n_L + \sum_{L=\lceil D/2 \rceil}^{D-\alpha-1} n_L \text{ for } D \text{ odd}$$

$$= \sum_{L=D-\alpha}^{D} n_L + \sum_{L=(D/2)+1}^{D-\alpha-1} n_L + \tfrac{1}{2} n_{D/2} \text{ for } D \text{ even,}$$

and comparing this expansion with (14), we conclude that

$$b(\alpha, \beta) \leq b\left(\left\lceil \frac{D}{2} \right\rceil - 1, \left\lceil \frac{D}{2} \right\rceil - 1\right) \text{ for } \beta \geq \left\lceil \frac{D}{2} \right\rceil,$$

which proves this theorem.

As a summary, to calculate the lower bound m_L defined by (13), for independent tasks, we compute

$$m_L = \left\lceil \max_{\beta} b(\beta, \beta) \right\rceil, 0 \leq \beta < \left\lceil \frac{D}{2} \right\rceil. \tag{16}$$

Consequently, only $\lceil D/2 \rceil$ intervals need to be considered, instead of the $\frac{1}{2}D(D+1)$ intervals required in the general case.

• *4.2 Number of operations required to compute the lower bound directly*

Consider the expression for $b(\beta, \beta)$ given by (15). To obtain the first term we need β additions. To obtain the second term we perform $(D - 2\beta)$ multiplications, $D - 2\beta - 1$ additions, and one division. Then both terms are added by one addition. This has to be performed for each β, for $0 \leq \beta \leq \lceil D/2 \rceil$. Finally, $\lceil D/2 \rceil - 1$ comparisons are required to obtain the maximum. In total we have $O(D^2)$ additions, $O(D^2)$ multiplications, $O(D)$ divisions, and $O(D)$ comparisons.

This direct calculation does not produce a very significant reduction in the number of operations as compared with the general case. In the following section an incremental method is considered which allows more improvement in this respect.

• *4.3 Incremental calculation*

With use of the notation defined in section 4.1, the expression for $b(\alpha, \beta)$ given by (12) for $\alpha = \beta$ becomes

$$b(\beta, \beta) = \frac{M(\beta, \beta)}{D - 2\beta},$$

so that using (15) we have

$$M(\beta, \beta) = \sum_{L=D-\beta}^{D} n_L(D - 2\beta) + \sum_{L=\beta+1}^{D-\beta-1} n_L(L - \beta), \quad (17)$$

and

$$M(\beta + 1, \beta + 1) = \sum_{L=D-\beta-1}^{D} n_L(D - 2\beta - 2) + \sum_{L=\beta+2}^{D-\beta-2} n_L(L - \beta - 1).$$

This latter expression can be written as

$$M(\beta + 1, \beta + 1) = \sum_{L=D-\beta}^{D} n_L(D - 2\beta - 2) + n_{D-\beta-1}(D - 2\beta - 2) + \sum_{L=\beta+1}^{D-\beta-2} n_L(L - \beta - 1),$$

because $n_{\beta+1}(\beta + 1 - \beta - 1) = 0$.

Consequently,

$$M(\beta + 1, \beta + 1) - M(\beta, \beta) = -2 \sum_{L=D-\beta}^{D} n_L - \sum_{L=\beta+1}^{D-\beta-1} n_L = -h(\beta), \quad (18)$$

and

$$h(\beta + 1) = 2 \sum_{D-\beta-1}^{D} n_L + \sum_{\beta+2}^{D-\beta-2} n_L,$$

so that

$$h(\beta + 1) - h(\beta) = n_{D-\beta-1} - n_{\beta+1}.$$

Consequently,

$$\left.\begin{aligned} &M(\beta + 1, \beta + 1) = M(\beta, \beta) - h(\beta), \\ &\text{and} \\ &h(\beta + 1) = h(\beta) + n_{D-\beta-1} - n_{\beta+1}, \\ &\text{with the initial conditions} \\ &M(0, 0) = \sum_{L=1}^{D} n_L L, \\ &\text{and} \\ &h(0) = n_D + \sum_{L=1}^{D} n_L. \end{aligned}\right\} \quad (19)$$

Since, because of Theorems 1 and 2, we only need to consider intervals $[\beta, \beta]$ for $0 \leq \beta < \lceil D/2 \rceil$, the number of operations required to perform the calculation of m_L in this incremental manner is

1. for calculating $M(0, 0)$, D multiplications and $D - 1$ additions;
2. for calculating $h(0)$, $D - 1$ additions;
3. for calculating $M(\beta + 1, \beta + 1)$ and $h(\beta + 1)$ from $M(\beta, \beta)$ and $h(\beta)$, one subtraction and two additions;
4. for calculating $b(\beta, \beta)$, one division; and
5. for calculating m_L, $\lceil D/2 \rceil - 1$ comparisons.

In total, D multiplications, $7\lceil D/2 \rceil$ additions, $\lceil D/2 \rceil$ divisions, and $\lceil D/2 \rceil - 1$ comparisons.

This is considerably better than the non-incremental calculation. It can still be improved if the incremental calculation is begun with $\beta = \lceil D/2 \rceil - 1$ and this value is decreased until it becomes zero, i.e., we take intervals $[\beta, \beta]$ of increasing size. Similar recurrence relations as for the previous case can be developed, and it can be shown that the total number of operations is decreased by D with respect to the first incremental method. Notice also that the number of divisions can be reduced in the same manner as described for general graphs (section 3.4).

5. Calculation of the lower bound on time for independent tasks

When the tasks are independent it is also possible to reduce considerably the number of operations required to determine the lower bound on time, t_L. In effect, it is shown below that only D intervals have to be considered. It is also possible to calculate this bound in an incremental manner.

If in (4) we define (using the α, β notation)

$$c(\alpha, \beta) = \frac{1}{m} M(\alpha, \beta) - (D - \alpha - \beta),$$

we can express the lower bound on time of (4) as

$$t_L = D + \left\lceil \max_{[\alpha,\beta]} c(\alpha, \beta) \right\rceil. \quad (20)$$

In a similar fashion to what was done for the bound on the number of processors, we can write for independent tasks

$$c(\alpha, \beta) = \frac{1}{m}\left[\sum_{D-\alpha}^{D} n_L(D - \alpha - \beta) + \sum_{\beta+1}^{D-\alpha-1} n_L(L - \beta)\right] - (D - \alpha - \beta), \quad \text{for } \beta \geq \alpha. \quad (21)$$

The following theorem reduces the number of intervals that have to be considered in the calculation of the bound.

Theorem 3 For independent tasks,

$$c(\beta, \beta) \geq c(\beta - \Delta, \beta + \Delta),$$

$$0 \leq \beta \leq \left\lceil \frac{D}{2} \right\rceil - 1, \quad 1 \leq \Delta \leq \beta,$$

and

$$c(\beta, \beta+1) \geq c(\beta-\Delta, \beta+1-\Delta),$$
$$0 \leq \beta \leq \left\lceil \frac{D}{2} \right\rceil - 1,\ 1 \leq \Delta \leq \beta.$$

Proof From (21),

$$c(\beta, \beta) = \frac{1}{m}\left[\sum_{D-\beta}^{D} n_L(D-2\beta) + \sum_{\beta+1}^{D-\beta-1} n_L(L-\beta)\right] - (D-2\beta),$$

and

$$c(\beta-\Delta, \beta+\Delta) = \frac{1}{m}\left[\sum_{D-\beta+\Delta}^{D} n_L(D-2\beta) + \sum_{\beta+\Delta+1}^{D-\beta+\Delta-1} n_L(L-\beta-\Delta)\right] - (D-2\beta).$$

These expressions can be decomposed into

$$c(\beta, \beta) = \frac{1}{m}\left[\sum_{D-\beta+\Delta}^{D} n_L(D-2\beta) + \sum_{D-\beta}^{D-\beta+\Delta-1} n_L(D-2\beta) + \sum_{\beta+\Delta+1}^{D-\beta-1} n_L(L-\beta) + \sum_{\beta+1}^{\beta+\Delta} n_L(L-\beta)\right] - (D-2\beta),$$

and

$$c(\beta-\Delta, \beta+\Delta) = \frac{1}{m}\left[\sum_{D-\beta+\Delta}^{D} n_L(D-2\beta) + \sum_{D-\beta}^{D-\beta+\Delta-1} n_L(L-\beta-\Delta) + \sum_{\beta+\Delta+1}^{D-\beta-1} n_L(L-\beta-\Delta)\right] - (D-2\beta).$$

By comparing corresponding terms it is possible to conclude that

$$c(\beta, \beta) \geq c(\beta-\Delta, \beta+\Delta).$$

An analogous analysis shows that

$$c(\beta, \beta+1) \geq c(\beta-\Delta, \beta+\Delta+1),$$

which proves the theorem.

That is, for all intervals of width $(D-2\beta)$, the largest bound is obtained for the interval (β, β), and for all intervals of width $(D-2\beta-1)$ the largest bound is obtained for the interval $(\beta, \beta+1)$. Therefore, the total number of intervals that have to be considered is only $\lceil D/2 \rceil + \lfloor D/2 \rfloor = D$.

An incremental procedure, similar to the one described for the computation of the lower bound on the number of processors, is also applicable here.

6. Calculation of the lower bound for other types of graphs

• *6.1 The lower bound on the number of processors for trees*

We define trees in the regular way [9]; however, we need the following definitions for our development [7, 10]:

The *activity* of task T_j is defined as

$$f(c_j, t) = \begin{cases} 1, \text{ for } t \in [c_j - d_j, c_j]; \\ 0, \text{ otherwise.} \end{cases} \tag{22}$$

The *load density function* is defined by

$$F(C, t) = \sum_{j=1}^{n} f(c_j, t). \tag{23}$$

Then, $f(c_j, t)$ indicates the activity of task T_j along time, according to some schedule that does not violate the restrictions imposed by the graph, and $F(C, t)$ indicates the total activity as a function of time. Clearly, $\max_t F(C, t)$ indicates the number of processors required for the schedule defined by $F(C, t)$.

If we call E_c the vector of the earliest completion times e_{cj}, then $F(E_c, t)$ is the earliest *load density function*, that is, it corresponds to a schedule where all tasks are processed as early as possible. Similarly, calling L_c the vector of the latest completion times l_{cj}, we can define a latest *load density function*. Also, in this section we shall use $[t_1, t_2]$ instead of $[\alpha, \beta]$.

We consider now the general lower bound on the number of processors defined by (3) and we see that, because of the particular structure of trees, it is not necessary to consider all the possible intervals $[t_1, t_2]$ to calculate this bound. First we need the following lemma.

Lemma 2 For trees, the earliest load density function is monotonically decreasing with time.

Proof The vertices of a tree (with the exception of the root) have exactly one successor. Therefore, each level of the tree can have at most as many vertices as the previous level, until the level of the root is reached, where there is exactly one vertex. As soon as a given vertex ceases its activity, its successor is initiated if possible; that is, at most one new vertex enters in activity. Then, at a given instant in time, there will never be more vertices going into activity than vertices ceasing their activity, i.e., $F(E_c, t)$ is monotonically decreasing along time.

Theorem 4 For trees,

$$b(t_1, D) \leq b(0, D),\ 1 \leq t_1 \leq D-1.$$

Proof For general graphs we have

$$b(t_1, D) = \sum_{j=1}^{n} e_j(\alpha, 0)/(D - t_1), \tag{24}$$

since tasks that are at their earliest positions in the interval $[t_1, D]$ must also be there in their latest positions.

For $t_1 = 0$, (24) becomes

$$b(0, D) = \frac{\sum_{j=1}^{n} e_j(0, D)}{D},$$

which can be decomposed into

$$b(0, D) = \frac{1}{D}\left[\sum_{j=1}^{n} e_j(0, t_1) + \sum_{j=1}^{n} e_j(t_1, D)\right].$$

Since the earliest load density function for trees is monotonically decreasing,

$$\sum_{j=1}^{n} e_j(0, t_1) \geq F(E_c, t_1) \times t_1,$$

and consequently,

$$b(0, D) \geq \frac{1}{D}\left[F(E_c, t_1) \times t_1 + b(t_1, D)]D - t_1]\right]$$

$$\geq \frac{[F(E_c, t_1) - b(t_1, D)]t_1}{D} + b(t_1, D);$$

and again because the earliest density function is monotonically decreasing, we have

$$F(E_c, t_1) \geq b(t_1, D),$$

which results in

$$b(0, D) \geq b(t_1, D).$$

This theorem allows us to drop all intervals of the class $[t_1, D]$, $1 \leq t_1 \leq D - 1$, from consideration in calculating the lower bound on the number of processors, that is, a saving of D intervals.

For trees with tasks of unit length, our bound reduces to Hu's bound [9]. Hu's bound is exact for this type of tree, and since our bound subsumes Hu's bound [1], the bound in (3) is also exact for trees of unit length. In this case the additional terms in (3) are superfluous.

• *6.2 The lower bound on the number of processors for independent chains*

A graph composed of a set of independent chains can be considered as a particular case of a tree (assuming a root of zero duration). Therefore, the reduction in the number of intervals to consider, defined by Theorem 4, still applies. However, because of the particular nature of this type of graph, a further reduction in the number of intervals to consider is possible.

Lemma 3 For independent tasks, $F(E_c, t)$ and $F(L_c, t)$ are the image of each other with respect to an axis through $t = D/2$.

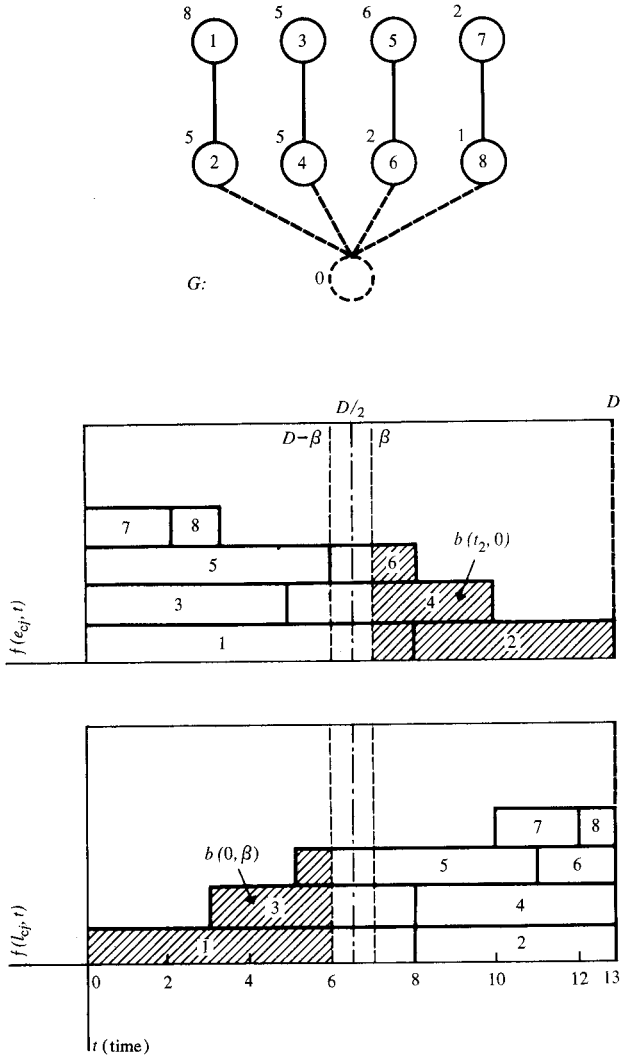

Figure 5 The lower bound for independent chains.

Proof Order chains by increasing length and examine their $f(e_{cj}, t)$ and $f(l_{cj}, t)$, i.e., the activities of the tasks in their earliest and latest positions, respectively. As it can be seen from Fig. 5, for $f(e_{cj}, t)$ chain i starts at $t = 0$ and ends at $t = d_{i_1} + d_{i_2} + \cdots + d_{i_k}$, where the d_i are the durations of the tasks of chain i. In $f(l_{cj}, t)$ chain i starts at $t = D - (d_{i_1} + d_{i_2} + \cdots + d_{i_k})$ and ends at $t = d$. In other words, the two possible positions of each chain are images of each other with respect to $t = D/2$. With the notation $[\alpha, \beta]$ as before, the following theorem can be defined.

Theorem 5 For a graph composed of a set of independent chains, $b(0, \beta) = b(\beta, 0) \leq b(0, 0)$.

Proof From Fig. 5 it can be seen that $b(0, t_2)$ corresponds to the area under the latest density function

$F(L_c, t)$ and $b(\beta, 0)$ corresponds to the area under the earliest density function $F(E_c, t)$. By Lemma 3 these two areas are equal. The inequality holds since this graph is a degenerate tree (Theorem 4).

Hence, for this type of graph, intervals starting at $t_1 = 0$ and intervals ending at $t_2 = D$ do not have to be considered in the calculation of the lower bound on the number of processors, i.e., a saving of $2D$ intervals.

7. Summary

An analysis has been presented of the computational characteristics of lower bound expressions for the number of processors and for the time of optimal schedules. These expressions, proposed in an earlier work [7], imply the consideration of $\frac{1}{2}D(D+1)$ varying-length intervals, and this study has attempted to evaluate and to reduce the number of operations to obtain these bounds.

Direct computation of the expressions for general graphs has been shown to require approximately $3nD^2$ operations (for large n and D). By means of an incremental method this value can be reduced to about D^2 operations. As the incremental method requires only one operation per interval, it represents the best possible way of evaluating these bounds for the general case. Parallel computation can reduce the execution time to $6 + [\log_2 n]$, but at the cost of $(n/2)D^2$ processors.

Reductions in the number of intervals to consider in the determination of the expressions can be obtained for special classes of graphs. For the case of independent tasks, it was shown that it is necessary to consider only $D/2$ intervals for the bound on the number of processors, and D intervals for the bound on time. However, in this case the computational savings are partially offset by the increased number of operations that have to be made in each interval, which results in a total of $\frac{1}{2}D^2$ operations for the bound on the number of processors. Incremental computation in this case decreases this total to only about $(11/2)D$ operations. For trees, the number of intervals to consider can be decreased by D, and for independent chains the saving is $2D$ intervals (for the bound on m in both cases and with respect to $\frac{1}{2}D(D+1)$).

In summary, it can be said that the number of operations required to obtain accurate lower bounds, as the ones represented by the expressions (1) and (2), can be reduced to values that are very small in comparison with the number of additional operations that have to be performed if the lower bound was a poor starting value for the scheduling process.

It should be mentioned also that the lower bound on time can be refined at the cost of extra computation. In effect, in expression (4) t_L can be evaluated recursively, such that at step k,

$$t_L(k) = t_L(k-1) + \left\lceil \max_{[t_1,t_2]} [-(t_2 - t_1) + 1/mM(t_1, t_2)] \right\rceil,$$

$$t_L(0) = t_{cp}.$$

In this case $M(t_1, t_2)$ has to be redefined at every step because of the change in the reference deadline.

References

1. G. H. Barnes et al., "The ILLIAC IV Computer," *IEEE Trans. Comput.* **C-17,** 746 (1968).
2. B. A. Crane et al., "PEPE Computer Architecture," *Proc. IEEE* (COMPCON 1972), 57 (1972).
3. C. V. Ramamoorthy, K. M. Chandy, and M. J. González, "Optimal Scheduling Strategies in a Multiprocessor System," *IEEE Trans. Comput.* **C-21,** 137 (1972).
4. B. Bussell, E. B. Fernández, and H. O. Levy, "Optimal Scheduling for Homogeneous Multiprocessors," *Information Processing 74*, North-Holland Publishing Co., 1974, pp. 286-290.
5. T. L. Adam, K. M. Chandy, and J. R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," *Comm. ACM* **17,** 685 (1974).
6. R. L. Graham, "Bounds on Multiprocessing Anomalies and Related Packing Algorithms," *AFIPS Conf. Proc.* **40,** 205, AFIPS Press, Montvale, NJ, 1972.
7. E. B. Fernández and B. Bussell, "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules," *IEEE Trans. Comput.* **C-22,** 745 (1973).
8. D. J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Trans. Comput.* **C-17,** 758 (1968).
9. T. C. Hu, "Parallel Sequencing and Assembly Line Problems," *Oper. Res.* **9,** 841 (1961).
10. A. B. Barskiy, "Minimizing the Number of Computing Devices Needed to Realize a Computational Process within a Specified Time," *Eng. Cybernetics* (USSR), 59 (1968).

Received January 5, 1975

E. B. Fernández is located at the IBM Scientific Center, Data Processing Division, 1930 Century Park West, Los Angeles, CA 90067; T. Lang is with the Computer Science Department, University of California, Los Angeles, CA 90024.