An Efficient Algorithm for Computing Lower Bounds on Time and Processors for Scheduling Precedence Graphs on Multicomputer Systems

B.S.Panda\textsuperscript{1} and Sajal K. Das\textsuperscript{2}

\textsuperscript{1} Department of Computer and Information Sciences
University of Hyderabad, Hyderabad 500 046, INDIA
bspcs@uohyd.ernet.in

\textsuperscript{2} Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019-0015, USA
das@cse.uta.edu

Abstract. In this paper, we propose new lower bounds on minimum number of processors and minimum time to execute a given program on a multicomputer system, where the program is represented by a directed acyclic task graph having arbitrary execution time and arbitrary communication delays. Additionally, we propose an $O(n^2 + m \log n)$ time algorithm to compute these bounds for a task graph with $n$ nodes and $m$ arcs. The key ideas of our approach include (i) identification of certain points called event points and proving that the intervals having event points at both ends are enough to compute the desired bounds; and (ii) the use of a sweeping technique. Our bounds are shown to be as sharp as the current best known bounds due to Jain and Rajaraman [7]. However, their approach requires $O(n^2 + m \log n + nW_{rel}^2)$ time, where $W_{rel}$ is the earliest execution time of the task graph when arbitrary number of processors are available. Thus, in general, our algorithm performs as good as their algorithm, and exhibits better time complexity for task graphs having $W_{rel} > O(\sqrt{n})$.

1 Introduction

A multicomputer is a multiple instruction stream, multiple data stream (MIMD), distributed memory parallel computer. It consists of several processors each of which has its own memory. There is no global memory and processors communicate via message passing. The objective of multiprocessing is to minimize the overall computation time of a problem (and hence gain speedup compared to a sequential algorithm) by solving it on a multicomputer. However, it is a challenge

\* Part of this work was done while the first author was visiting the Department of Computer Science and Engineering, the University of Texas at Arlington. This work was supported by NASA Ames Research Center under Cooperative Agreement Number NCC 2-5395.
to develop efficient software that takes full advantage of hardware parallelism offered by a multicomputer. A parallel program is a collection of tasks that may run serially or in parallel. These tasks must be optimally placed on the processors with the shortest possible execution time. This is known as the scheduling problem, which is NP-Hard in its general form [13].

A program on a multicomputer can be modeled as a directed acyclic graph \( G(\Gamma, \rightarrow, \mu, \eta) \), where \( \Gamma = \{T_1, T_2, \ldots, T_n\} \) represents the set of tasks constituting the program, \( \rightarrow \) represents the temporal dependencies between the tasks, \( \mu(T_i) \) denotes the execution time for task \( T_i \), and each arc \( T_i \rightarrow T_j \) is associated with a positive integer \( \eta(T_i, T_j) \) representing the number of messages sent from the task \( T_i \) to its immediate successor \( T_j \). To characterize the underlying system, a processor graph \( P(\tau) \) is also introduced such that \( \tau(i, j) \) represents the communication delay (i.e., the time to transfer a message) between processors \( i \) and \( j \). This model is called Enhanced Directed Acyclic Graph (EDAG) [6].

Given a parallel program or equivalently an EDAG, the problem is to assign the tasks to different identical processors of a multicomputer so that the overall execution time or makespan is minimum. As this problem is NP-Hard [13], it is unlikely to design a polynomial time optimal algorithm for scheduling EDAG or precedence graphs on multicomputer systems. Therefore, many heuristic algorithms have been proposed in recent years [3,7,9-12].

This paper deals with two basic problems concerning the theory of scheduling. Given a precedence graph \( G=(\Gamma, \rightarrow, \mu, \eta) \),

1. Find the minimum number of identical processors that are needed in order to execute the program represented by the precedence graph in time not exceeding that taken when arbitrary number of processors are available.
2. Determine the minimum time to process the given precedence graph when a fixed number of processors are available.

These problems being NP-Hard [13], researchers have proposed lower bounds for (1), i.e., minimum number of processors (LBMP), and lower bounds for (2), i.e., execution time (LBMT) for EDAG model [2,7,8].

Al-Mouhamed [2] was the first to suggest LBMT and LBMP. His algorithm runs in \( O(m \log n + n^2) \) time, where \( m \) is the number of edges and \( n \) is the number of tasks in the precedence graph. Jain and Rajaraman [7] have proposed an algorithm which gives tighter LBMP and LBMT than that in [2] but takes \( O(nW_{egr}^2 + m \log n) \) time where \( W_{egr} \) is the earliest completion time of the task graph when arbitrary number of processors are available. Using a graph partitioning technique, the authors [7] also suggested another algorithm which gives the current best LBMP and LBMT in \( O(m \log n + n^2 + nW_{egr}^2) \) time.

In this paper, we propose new LBMP and LBMT needed to execute a given program on a multicomputer system, where the program is represented by an EDAG. We, then, show that these bounds are as sharp as the current known best bounds due to Jain and Rajaraman [7]. We present an algorithm to find these bounds in \( O(m \log n + \min(n^2, W_{egr}^2)) \) time. So, in general, our algorithm is as good as the algorithm due to Jain and Rajaraman [7] and exhibits better time complexity for task graphs having \( W_{egr} > O(\sqrt{n}) \). The key ideas of our approach
include: (i) identification of certain points called event points and proving that the interval having event points as both ends are enough to compute the desired bounds; and (ii) the use of a sweeping technique.

2 Preliminaries

Let $G(T, \rightarrow, \mu, \eta)$ be a precedence graph having $n$ nodes. The earliest time at which a task $T_i$ can be started on any processor, called earliest starting time, is denoted by $est_i$. The earliest time at which a task can be completed on any processor, called earliest completion time, is denoted by $ect_i$. Let $W_{ecl}$ be the earliest completion time of the task graph. Then $W_{ecl} = \max_{1 \leq i \leq n} \{ect_i\}$. Let $d(T_i)$ denote the maximum time (or delay) by which a task $T_i$ can be postponed without increasing $W_{ecl}$ of the task graph. Let $lst_i$, the latest starting time of task $T_i$, be the latest time at which a task $T_i$ can be started without increasing $W_{ecl}$. So, $lst_i = est_i + d(T_i)$. Similarly, the latest completion time of a task $T_i$ is given by $lct_i = ect_i + d(T_i)$.

Let $\Delta t_m$ be the lower bound on the minimum increase in time to execute the precedence graph $G$ over $W_{ecl}$ when $m$ processors are used, and $m_{min}$ be the lower bound on the number of processors required to complete $G$ in $W_{ecl}$ time.

An O$(m \log n)$-time algorithm for finding $est_i$ and $d(T_i)$ for a precedence graph is given in [2].

The load density function $f(T_i, t_{sa}, t)$, associated with a task $T_i$ represents its activity which starts at time $t_{sa}$ and ends at time $t_{sa} + \mu(T_i)$. For $t_{sa} \in [est_i, lst_i]$, this is defined as:

$$f(t_{sa}, t) = \begin{cases} 1, & \text{if } t \in [t_{sa}, t_{sa} + \mu(T_i)] \\ 0, & \text{otherwise} \end{cases}$$

The load density function $F(t_s, t)$ of a graph $G$ is the sum of all the task activities at time $t$ and is given by

$$F(t_s, t) = \sum_{i=1}^{n} f(T_i, t_{sa}, t).$$

Let $\phi(t_1, t_2, t)$ represent the load density function within interval $[t_1, t_2] \subseteq [0, W_{ecl}]$, after all tasks have been shifted to yield minimum overlap in $[t_1, t_2]$. Thus $\phi(t_1, t_2, t)$ can be viewed as the essential activity in time $[t_1, t_2]$ which should be executed within this time interval. Let $(m_{min})_{[a,b]} = \frac{1}{b-a} \int_{a}^{b} \phi(a, b, t) dt$, and $(\Delta t_m)_{[a,b]} = \frac{1}{m} \int_{a}^{b} \phi(a, b, t) dt - (b - a)]$.

The LBMP and LBMT due to Al-Mouhamed [2] can be stated as follows:

$$m_{min} = \max_{1 \leq i \leq n} \{(m_{min})_{[est_i, lct_i]}\} \quad (1)$$

$$\Delta t_m = \max_{1 \leq i \leq n} \{((\Delta t_m)_{[est_i, lct_i]}\} \quad (2)$$

In other words, the author in [2] considered only the $n$ intervals of the type $[est_i, lct_i]$, $1 \leq i \leq n$, to find the bounds. The time complexity to calculate Eqs. (1) and (2) is O$(m \log n + n^2)$. 
Jain and Rajaraman [7] showed that sharper bounds can be obtained if all integer intervals in \([0, W_{er}]\) are considered to compute LBMP and LBMT. They proposed the following sharper bounds:

\[
m_{min} = \max_{t_1 < t_2 \leq W_{er}} \{(m_{min})_{[t_1, t_2]}\} \\
\Delta t = \max_{t_1 < t_2 \leq W_{er}} \{\Delta t_{m_{(t_1, t_2)}}\}
\]

where \(t_1\) and \(t_2\) are integers.

However, \(O(nW_{er}^2 + m \log n)\) time is required to compute Eqs. (3) and (4). To get tighter bounds, the authors [7] have suggested an \(O(n^2)\) time graph partitioning algorithm. They have shown that under certain conditions, the task graph can be partitioned into two or more parts in such a way that the LBMT (the LBMP) for the original task graph is the sum (minimum) of the LBMT (LBMP) of the different parts of the partition. So \(O(m \log n + n^2 + nW_{er}^2)\) time is needed to compute the current best known LBMT and LBMT due to Jain and Rajaraman [7].

3 Proposed Bounds and Algorithm

The heart of the algorithm due to [2] to compute the lower bounds is the calculation of \(\phi(t_1, t_2, t)\) within \([t_1, t_2] \subseteq [0, W_{er}]\). This is calculated by summing up the contribution of all the tasks to this interval. The contribution of the task \(T_i\) is computed by shifting the tasks to yield minimum overlap within this interval. We denote \(\mu(T_i)\) by \(\mu_i\) if no confusion arises.

A task \(T_i\) can start at any time \(t_i \in [est_i, lst_i]\), without increasing \(W_{er}\).

However, we show below that a task gives minimum overlap within an interval if it is scheduled to start at either \(est_i\) or \(lst_i\).

**Lemma 3.1:** Let \([c, d] \subseteq [0, W_{er}]\). A task \(T_i\) yields minimum overlap within \([c, d]\) if it is scheduled to start either at \(est_i\) or at \(lst_i\).

**Proof:** Let \(s_i\) be the time when \(T_i\) is started. So, \(est_i \leq s_i \leq lst_i\). We consider two cases.

**Case I:** \(s_i \leq c\).

If \(s_i + \mu_i \leq c\), then overlap of \(T_i\) in \([c, d]\) = 0 = overlap of \(T_i\) in \([c, d]\) when \(T_i\) is scheduled at \(est_i\). If \(s_i + \mu_i > c\), then Overlap of \(T_i\) in \([c, d]\) = \([c, d]\) \cup [s_i, s_i + \mu_i]\.

\[
= \min\{s_i + \mu_i, d\} - c
\]

\[
\geq \min\{est_i + \mu_i, d\} - c, \text{ since } est_i \leq s_i
\]

\[
= \text{overlap when } T_i \text{ is scheduled at } est_i.
\]

So, overlap of \(T_i\) in \([c, d]\) is minimum if \(T_i\) is scheduled at \(est_i\).

**Case II:** \(s_i > c\).

If \(s_i > d\), then overlap of \(T_i\) in \([c, d]\) = 0 = overlap of \(T_i\) in \([c, d]\) when \(T_i\) is scheduled at \(lst_i\). If \(s_i \leq d\), then overlap of \(T_i\) in \([c, d]\) = \(\min\{s_i + \mu_i, d\} - s_i\).

\[
\geq \min\{lst_i + \mu_i, d\} - lst_i, \text{ since } lst_i \leq s_i
\]

\[
= \text{overlap when } T_i \text{ is scheduled at } lst_i.
\]

Thus, a smaller overlap is achieved when a task \(T_i\) is scheduled either at \(est_i\) or at \(lst_i\) than any other point.

\(\square\)
The heart of our lower bounds is the identification of certain points, called event points. Jain and Rajaraman [7] consider all integer subintervals of [0, W_{cr}] for calculation of lower bounds. We will show that it is enough to consider all subintervals of [0, W_{cr}] having both the ends as event points.

Let us now sort est_i, lst_i, est_i, and lct_i for all i = 1, 2, ..., n in non-decreasing order and eliminate duplicate points to obtain distinct points e_1, e_2, ..., e_p, where p ≤ 4n. Each point, so obtained, is called an event point.

Let I = [c, d] be the interval for which minimum overlap is to be calculated. Let [a, b] ⊆ [e_i, e_{i+1}] for some i, 1 ≤ i ≤ p − 1.

The activity, P_{[a, b]}(I), of [a, b] with respect to I, is defined as the number of tasks which have a nonzero overlap with [a, b] after being shifted to yield minimum overlap within I.

**Lemma 3.2** Let [a, b] ∈ [e_i, e_{i+1}] for some i, 1 ≤ i ≤ p − 1, and let [x, y] ⊆ [0, W_{cr}]. Then P_{[a, b]}([x, y]) = P_{[e_i, e_{i+1}]}([x, y]).

**Proof:** Follows directly from Lemma 3.1.

### 3.1 New LBMP and LBMT

In the following we propose a new LBMP and LBMT and we prove that they are as sharp as that given by Eqs. (2.3) and (2.4), respectively.

\[ m_{min} = \max_{1 \leq i < j \leq p} \{(m_{min})_{[e_i, e_j]}\} \quad (5) \]

\[ \Delta t_m = \max_{1 \leq i < j \leq p} \{(\Delta t_m)_{[e_i, e_j]}\} \quad (6) \]

**Lemma 3.3:** The bounds obtained by Eqs. (5) and (6) are as sharp as those obtained by Eqs. (3) and (4), respectively.

**Proof:** Let I = [c, d] be any integer subinterval of [0, W_{cr}].

**Case I:** I ⊆ [e_i, e_{i+1}] = I', for some i, 1 < i < p.

Let L = e_{i+1} − e_i, and d − c = l. Now by Lemma 3.2, P_I'(f''') = P_{I'}(f') = Q, say, where I' ⊆ I'' ⊆ [0, W_{cr}].

Now, \((m_{min})_{[a, b]} = \left[ \frac{Q - L}{T} \right]\), and \((m_{min})_{[c, d]} = \left[ \frac{Q + L}{T} \right]\). So, \((m_{min})_{[c, d]} = (m_{min})_{[a, b]}\).

Also, \((\Delta t_m)_{[c, d]} = \left[ \frac{Q + L}{T} \right] - l \leq \left[ \frac{Q - L}{T} \right] = (\Delta t_m)_{[a, b]}\).

Hence \((\Delta t_m)_{[c, d]} \leq (\Delta t_m)_{[a, b]}\). In other words, it is better to consider the interval \([e_i, e_{i+1}]\) rather than any of its subinterval for both bounds.

**Case II:** There exists k, 2 ≤ k ≤ p − 1, such that \(e_k < d \leq e_{k+1}\).

Let i be the largest integer for which \(e_i < c\), and let j be the smallest integer for which \(d \leq e_j\). So, \(e_i < c < d \leq e_j\) and \(j ≥ i + 2\). Hence, \(c < e_{i+1} < d\).

**Claim** (i) Max \(\{(m_{min})_{[e_j, e_{j+1}]}, (m_{min})_{[e_i, e_j]}\}\) ≥ \(m_{min})_{[e_i, e_j]}\),

(ii) Max \(\{(\Delta t_m)_{[e_i, e_{j-1}]}, (\Delta t_m)_{[e_i, e_j]}\}\) ≥ \(\Delta t_m)_{[e_i, e_{j-1}]\}.

**Proof of the claim:**

Let \(X = P_{[e_j, e_{j+1}]}(I)\), and \(A = \int_c^{e_j-1} \phi(c, d, t)dt\). Now, \((m_{min})_{[e_i, e_j]} = \left[ \frac{A + X (d - e_{j-1})}{d - c} \right]\),

\( (m_{min})_{[e_j, e_{j+1}]} = \left[ \frac{A + X (d - e_{j-1})}{d - c} \right] \), and \((m_{min})_{[e_i, e_{j-1}]} = \left[ \frac{A}{e_{j-1} - c} \right]\). If \((m_{min})_{[e_i, e_j]} > (m_{min})_{[e_j, e_{j+1}]}, then \( \frac{A + X (d - e_{j-1})}{d - c} > \frac{A}{e_{j-1} - c} \).

So, \(A < X(e_{j-1} - c) \quad (7)\).
If \((m_{min})_{c,d} > \langle m_{min} \rangle_{c,e_i}\), then \[\frac{A + X(d - e_{j-1})}{d - c} \] > \[\frac{A + X(e_{j-1} - e_j)}{e_j - c}\]. So, 
\[A > X(e_{j-1} - c)\] \hspace{1cm} (8) 

Now (7) and (8) cannot hold simultaneously. So, either \((m_{min})_{c,e_{j-1}} \geq \langle m_{min} \rangle_{c,d}\) or \((m_{min})_{c,e_j} \geq \langle m_{min} \rangle_{c,d}\).

So claim (i) is true.

Now, \((\Delta t_m)_{c,d} = \left[\frac{A + X(d - e_{j-1})}{m} \right] - (d - c)\),
\[(\Delta t_m)_{c,e_j} = \left[\frac{A}{m} - (e_{j-1} - c)\right].\]

If, \((\Delta t_m)_{c,d} > (\Delta t_m)_{c,e_j}\), then \(m < P\) \hspace{1cm} (9)

Again, if \((\Delta t_m)_{c,d} > (\Delta t_m)_{c,e_j}\), then \(m > P\) \hspace{1cm} (10)

Now (9) and (10) cannot hold simultaneously. So, either, \((\Delta t_m)_{c,e_{j-1}} \geq \langle \Delta t_m \rangle_{c,d}\) or \((\Delta t_m)_{c,e_j} \geq \langle \Delta t_m \rangle_{c,d}\).

Hence claim (ii) is true.

So, it is sufficient to consider all intervals \([c,d]\), where \(d\) is an event point, to find LBMP and LBMT. Using a similar argument it can be proved that it is sufficient to consider all intervals \([c,d]\) where \(c\) is an event point, to find LBMP and LBMT.

Hence, the lemma is proved. \(\square\)

3.2 Computation of Minimum Overlap in an Interval

Let \(S_i^c = \{T_i| est_i > c\}, S_i^s_1 = \{T_i| est_i \leq c < lst_i\} and est_i > c\}, \text{ and } S_i^s_2 = \{T_i| lst_i < c\} \text{ and } est_i > c\}, \text{ where } c \in [0, W_{err}]\). Note that tasks \(T_i\) with \(est_i \leq c\) do not belong to any of the above sets and gives zero overlap within \([c,d]\), \(c < d \leq W_{err}\).

We define \(b_i\) and \(l_i\) for each task \(T_i\) as follows

**Rule 1:** If \(T_i \in S_i^c\), then \(b_i = lst_i\) and \(l_i = lst_i\).

**Rule 2:** If \(T_i \in S_i^s_1\), then \(b_i = lst_i\) and \(l_i = est_i + lst_i - c\).

**Rule 3:** If \(T_i \in S_i^s_2\), then \(b_i = c\) and \(l_i = est_i\).

**Rule 4:** If \(T_i \notin (S_i^c \cup S_i^s_1 \cup S_i^s_2)\), then \(b_i = 0\) and \(l_i = 0\).

The numbers \(b_i\) and \(l_i\) are the starting and ending time when the task \(T_i\) begins and ends its contribution for minimum overlap within the interval \([e_i, e_j]\) where \(e_i = c\) and \(i < j \leq p\). If \(b_i \geq d\), then clearly the minimum overlap of the task \(T_i\) within \([c,d]\) is 0.

Let \(\text{minOV}\) denote the minimum overlap in a given interval. The following lemma gives a formula for computing the minimum overlap of a task \(T_i\) within \([c,d]\) in terms of \(b_i\) and \(l_i\).

**Lemma 3.4** Let \(d > b_i\) for a task \(T_i\). Then the minimum overlap of a task \(T_i\) within \([c,d]\) is \(\min\{l_i, d\} - b_i\).

**Proof** Let \(ov_1\) and \(ov_2\) be the overlap of task \(T_i\) within \([c,d]\) when scheduled at \(est_i\) and \(lst_i\), respectively. We consider the following cases corresponding to the above four rules.

**Case 1:** \(T_i \in S_i^c\).
So, \( c < \text{est}_i \). Now, \( \text{ov}_1 = \min\{d, \text{ect}_i\} - \text{est}_i = \min\{d - \text{est}_i, \mu(T_i)\} \)
and \( \text{ov}_2 = \min\{d, \text{lct}_i\} - \text{lst}_i = \min\{d - \text{lst}_i, \mu(T_i)\} \).
So, \( \min\{\text{ov}_1, \text{ov}_2\} = \text{ov}_2 = \min\{d - \text{lst}_i, \mu(T_i)\} = \min\{l_i, d\} - b_i \).

**Case II:** \( T_i \in S^2_2 \).
So, \( \text{est}_i \leq c \leq \text{lst}_i \). Considering the subcases (i) \( d \leq \text{ect}_i \), (ii) \( \text{ect}_i < d < \text{ect}_i + \text{lst}_i - c \), (iii) \( d = \text{ect}_i + \text{lst}_i - c \), (iv) \( d > \text{ect}_i + \text{lst}_i - c \), and the computation of \( \text{ov}_1 \) and \( \text{ov}_2 \) as in **Case I**, it is a routine job to show that \( \min\{\text{ov}_1, \text{ov}_2\} = \min\{l_i, d\} - b_i \) in this case.

**Case III:** \( T_i \in S^3_2 \).
So, \( c > \text{lst}_i \). Now, \( \text{ov}_1 = \min\{d, \text{est}_i\} - c \) and \( \text{ov}_2 = \min\{d, \text{lct}_i\} - c \). So, \( \text{ov}_1 \leq \text{ov}_2 \). Hence, \( \text{minOV} = \min\{\text{d,est}_i\} \cdot c = \min\{d,l_i\} - b_i \).

**Case IV:** \( T_i \notin (S^3_1 \cup S^3_2 \cup S^3_3) \).
So, \( l_i = b_i = 0 \). So, \( \text{minOV} = \min\{l_i, d\} - b_i = l_i - b_i = 0 \).

The Lemma is thus proved.

\[\square\]

### 3.3 Algorithm Pseudo-code

Below, we give an algorithm that computes a matrix \( OV_{\text{p} \times \text{p}}[] \), where \( OV[i,j] = \int_{e_i}^{e_j} \phi(e_i,e_j,t)dt \) for \( 1 \leq i < j \leq \text{p} \).

Lemma 3.4 tells that the minimum overlap in an interval can be computed by adding the l’s and subtracting the b’s of the tasks. Again, if we know the minimum overlap in an interval \([c,d]\), then minimum overlap in any other interval \([c,e]\), where \( e > d \) can be computed by making use of the minimum overlap in \([c,d]\) and by adding certain l’s and subtracting certain b’s of tasks. Making use of this fact and a *sweeping* technique, we compute the minimum overlap. We find the overlap of the intervals in the following sequence:

\([e_1, e_2], [e_1, e_3], \ldots, [e_1, e_p], [e_2, e_3], \ldots, [e_2, e_p], \ldots, [e_{p-1}, e_p], [e_{p-2}, e_{p-1}], [e_{p-2}, e_p], [e_{p-1}, e_p]\)

i.e., we fix the left boundary and vary the right boundary till we reach to \( e_p \), then we increment the left boundary and continue the process till the left boundary becomes \( e_p \). So, at the end of the \( i^{th} \) stage we have the overlap of all subintervals \([e_j, e_i]\) for \( j = 1, 2, \ldots, i-1 \). Algorithm MinOverlap, which is given below is a clever use of the Lemma 3.4.

**Algorithm MinOverlap**

**Input:** A task graph \( G(T, \rightarrow, \mu, \eta) \) and a processor graph \( P(\tau) \).

**Output:** A matrix \( [OV]_{\text{p} \times \text{p}} \) such that \( OV[i,j] = \int_{e_i}^{e_j} \phi(e_i,e_j,t)dt \), \( 1 \leq i < j \leq p \) gives minimum overlap within the interval \([e_i, e_j]\).

1. Find \( \text{est}_i, \text{lst}_i, \text{ect}_i, \text{lct}_i, 1 \leq i \leq n \);
2. Sort \( \text{est}_i, \text{lst}_i, \text{ect}_i, \text{lct}_i, 1 \leq i \leq n \) to get distinct event points \( e_1, e_2, \ldots, e_p, p \leq 4n \);
3. Insert all tasks in \( S_1 \);
   - Find corresponding \( b_i \) and \( l_i \) according to Rule (1);
   - Sort all \( b_i \) in non-decreasing order to get sorted list \( B_1 \);
   - Sort all \( l_i \) in non-decreasing order to get sorted list \( L_1 \);
   - \( S_2 = S_3 = B_2 = B_3 = L_2 = L_3 = \phi \).
OV[i,j] = 0, 1 ≤ i < j ≤ p;

e_0 = e_1;

4. for i = 1 to p − 1 do
/* Here c = e_i */
    { Call Update(i);
        Merge sorted lists B_1, B_2, B_3, L_1, L_2, L_3 to get sorted list R = r_1, r_2, . . . , r_q
        (all need not be distinct);
        for j = 1 to q do
            if (r_j ∈ B_k for some k = 1, 2, 3) then type(r_j) = b; else type(r_j) = l;
            count = 0; pos = 1; r_0 = r_1; B = 0; L = 0;
            for j = i + 1 to p do /* Here d = e_j */
                while (e_j > r_pos) do {
                    if (type(r_pos) = ‘b’) then
                        count = count + 1; B = B + r_pos;
                    else count = count − 1; L = L + r_pos; pos = pos + 1;
                }
                OV[i,j] = L−B+count * e_j;
            }
    }

As c varies from e_1 to e_{p−1}, S_1^c, S_2^c, and S_3^c also change. The following procedure
obtains S_1^{c+1}, S_2^{c+1}, and S_3^{c+1} from S_1^c, S_2^c, and S_3^c.

Procedure Update(i) {
    for all tasks T_j ∈ S_1 do
        if (est_j ≥ e_i) then Move_and_Adjust(T_j, S_1, S_2);
    for all tasks T_j ∈ S_2 do
        if (est_j ≤ e_i) then
            Delete T_j from S_2;
            Delete b_j and l_j from B_2 and L_2, respectively;
        else if (lst_j > e_i) then
            Move_and_Adjust(T_j, S_2, S_3);
    for all tasks T_j ∈ S_2 do
        l_j = l_j − (e_i − e_{i−1});
        for each task T_j ∈ S_3 do
            if (est_j ≤ e_i) then { Delete T_j from S_3;
                Delete b_j and l_j from B_3 and L_3, respectively; }
            for each task T_j ∈ S_3 do
                b_j = e_i;

Move_and_Adjust(T_i, S_j, S_k) {
    Move T_i from S_j to S_k;
    Delete b_i and l_i from B_j and L_j, respectively;
    Find new b_i and l_i according to Rule k, and
    insert in B_k and L_k, respectively;
}

We are now in a position to give an algorithm to compute the lower bounds.
Algorithm L_Bound

Input: A task graph $G(\Gamma, \pi, \mu, \eta)$ and processor graph $P(\tau)$ and number of available processors $m$.

Output: $m_{\text{min}}$, the LBMP and $\Delta t_m$, the LBMT.

\begin{enumerate}
    \item Compute matrix $OV[i,j]$ using algorithm MinOverlap.
    \item Compute the following using the matrix $OV$.
        \begin{align*}
            m_{\text{min}} &= \max_{1 \leq i < j \leq p} \left\{ \frac{OV[i,j]}{(j-i)} \right\} \\
            \Delta t_m &= \max_{1 \leq i < j \leq p} \left\{ \frac{OV[i,j]}{m} - (j-i) \right\}
        \end{align*}
\end{enumerate}

4 Proof of Correctness and Complexity

The proof of correctness of Algorithm L_Bound follows from the proof of correctness of MinOverlap, which we shall prove below. Also, we show that it can be implemented in $O((m + n) \log n + \text{Min}\{nW_{eri}, n^2\})$ time.

Let $[c, d] \subseteq [0, W_{eri}]$. If $T_i \in S_i^1 \cup S_i^2 \cup S_i^3$, then clearly $b_i \geq c$. So, if $l_i \leq d$, then $b_i \in [c,d]$. Let $X_1 = \{T_i | l_i < d\}$ and $X_2 = \{T_i | l_i \geq d$ and $b_i < d\}$.

So, Minimum overlap within an interval $[c,d] = \sum_{T_i \in X_1} (l_i - b_i) + \sum_{T_i \in X_2} (d - b_i) = \left( \sum_{T_i \in X_1} l_i - \sum_{T_i \in X_1 \cup X_2} b_i \right) + d_r$, where $r = |X_2|$.

In the MinOverlap algorithm, $L = \sum_{l_i < d} l_i$, and $B = \sum_{b_i < d} b_i$, and count is the difference between the number of $b$’s and number of $l$’s in $[c,d] = [e_i, e_j]$. Hence we have the following results.

Theorem 4.1 Algorithm MinOverlap correctly computes the matrix $OV[i,j]$, $1 \leq i < j \leq p$, such that

\[ OV[i,j] = \int_{e_i}^{e_j} \phi(e_i, e_j, t)dt. \]

Theorem 4.2 Algorithm L_Bound requires $O((n + m) \log n + \text{Min}\{nW_{eri}, n^2\})$ time.

Proof: The complexity of the algorithm L_Bound is dominated by the complexity of algorithm MinOverlap. So, it is enough to show that algorithm MinOverlap runs in $O((m + n) \log n + \text{Min}\{nW_{eri}, n^2\})$ time. Now, we analyze algorithm MinOverlap. Step 1 can be implemented in $O(m \log n)$ time [2]. Step 2 can be implemented in $O(n \log n)$ time by using merge Sort. In Step 3, the procedure Update is invoked $p$ times. In every call of Update, each task $T_i \in (S_i^{e_{i-1}} \cup S_i^{e_i} \cup S_i^{e_{i+1}})$ is scanned once. Each scan takes $O(n)$ time. Again each task is deleted from $S_i^{e_{i-1}}$ and inserted in $S_i^{e_{j+1}}$, $1 \leq j \leq 2$, for some $i$. The lists $B_i$ and $L_i$ can be implemented as height balanced trees [1]. Hence each insertion and deletion takes $O(\log n)$ time. Throughout the $p$ calls of Update, each task is inserted and deleted a maximum of three times. This is because
a task from $S_{2}^{i-1}$ may be moved to $S_{2}^{i}$ and that from $S_{2}^{i-1}$ may be moved to $S_{2}^{i}$ but not vice-versa. Thus $O(n)$ insertions and deletions take place. So Step 3 takes $O(n \log n + n \cdot p)$ time. Since $p \leq 4n$ and $p \leq W_{cr}$, Step 3 takes $O(n \log n + \min\{nW_{cr}, n^2\})$ time. Now for each $i$, Step 4 takes $O(p - i)$ time. So Step 4 takes an overall time $O(p^2 = O(\min\{n^2, W_{cr}^2\})$. Hence, the algorithm LBound takes $O((m + n) \log n + \min\{nW_{cr}, n^2\} + \min\{n^2, W_{cr}^2\})$ time which can be reduced to $O((n + m) \log n + \min\{n^2, nW_{cr}\})$ time.

Jain and Rajaraman [7] have introduced the concept of partitioning technique. They have shown that under certain conditions, the task graph can be decomposed into two or more parts so that the lower bounds for the original task graph is sum of the lower bounds of the parts. If we incorporate the partitioning technique of Jain and Rajaraman [7], we can compute the current best LBMP and LBMT in $O(m \log n + n^2)$ time as partitioning the precedence graph takes $O(n^2)$ time [7].

5 Conclusion

In this paper, we have proposed new lower bounds for time and processors to execute a task graph and have also shown that our bounds are as sharp as those obtained by Jain and Rajaraman [7]. We then proposed an algorithm to compute LBMP and LBMT. Our algorithm is at least as good as the one in [7], but performs better if $W_{cr} > O(\sqrt{n})$, where $W_{cr}$ is the earliest execution time of the task graph having $n$ nodes when arbitrary number of processors are available. More precisely, in this case, our algorithm has an improvement of $O(n)$ over that due to [7].

References