An Efficient Parallel Algorithm for Scheduling Interval Ordered Tasks

Yoojin Chung[1], Kunsoo Park[2⋆], and Hyuk-Chul Kwon[3⋆]

[1] Research Institute of Computer and Information & Communication,
Pusan National University, Pusan, 609-735, Korea
chungyj@pusan.ac.kr

[2] School of Computer Science and Engineering,
Seoul National University, Seoul 151-742, Korea
kpark@theory.snu.ac.kr

[3] Division of Computer Science and Engineering,
Pusan National University, Pusan, 609-735, Korea
hckwon@pusan.ac.kr

Abstract. We present an efficient parallel algorithm for scheduling n unit length tasks on m identical processors when the precedence graphs are interval orders. Our algorithm requires $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter. By choosing $v = \sqrt{n}$, we obtain an $O(\sqrt{n} \log n)$-time algorithm with $O(n^2)$ operations. For $v = n/\log n$, we have an $O(\log^2 n)$-time algorithm with $O(n^3/\log^2 n)$ operations. The previous solution takes $O(\log^2 n)$ time with $O(n^3 \log^2 n)$ operations on the CREW PRAM. Our improvement is mainly due to a reduction of the m-processor scheduling problem for interval orders to that of finding a maximum matching in a convex bipartite graph.

1 Introduction

The m-processor scheduling problem for a precedence graph $\boldsymbol{G}$ is defined as follows. An input graph $\boldsymbol{G}$ has n vertices each of which represents a task to be executed on any one of m identical processors. Each task requires exactly one unit of execution time on any processor. At any timestep at most one task can be executed by a processor. If there is a directed edge from task t to task t', then task t must be completed before task t' is started. An m-processor schedule for $\boldsymbol{G}$ specifies the timestep and the processor on which each task is to be executed. The length of a schedule is the number of timesteps in it. A solution to the problem is an optimal (i.e., shortest length) schedule for $\boldsymbol{G}$.

The m-processor scheduling problem for arbitrary precedence graphs has been studied extensively. When $m = 2$, there are polynomial-time algorithms for the problem [6,3,9,7], and when m is part of the input, the problem is known to be NP-hard [20]. When m is part of the input, several researchers have considered restrictions on the precedence graphs. Polynomial-time algorithms for the m-processor scheduling problem are known for the cases that the precedence graphs

⋆ This work was supported by the Brain Korea 21 Project.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 100–111, 2000.

are trees [12] and interval orders [15]. A survey of results on other special cases of the problem can be found in [13].

In parallel computation, the two processor case has been studied mostly. When $m = 2$, Helmbold and Mayr [11] gave the first NC algorithm and Vazirani and Vazirani [21] presented an RNC algorithm. Jung, Serna and Spirakis [16] developed an $O(\log^2 n)$-time algorithm using $O(n^3 \log^2 n)$ operations on the CREW PRAM. When $m = 2$ and the precedence graphs are interval orders, Moitra and Johnson [18] and Chung, Park and Cho [2] gave NC algorithms, and the one in [2] requires $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter.

When m is part of the input and the precedence graphs are interval orders, Sunder and He [19] developed the first NC algorithm for the scheduling problem, which takes $O(\log^2 n)$ time using $O(n^5 \log^2 n)$ operations or $O(\log^3 n)$ time using $O(n^4 \log^3 n)$ operations on the priority CRCW PRAM. Mayr [14] gave an $O(\log^2 n)$-time algorithm using $O(n^3 \log^2 n)$ operations on the CREW PRAM.

In this paper, we present an efficient parallel algorithm for the m-processor scheduling problem when the precedence graphs are interval orders. Our algorithm takes $O(\log^2 v + (n \log n)/v)$ time using $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter. By choosing $v = \sqrt{n}$, we obtain an $O(\sqrt{n} \log n)$-time algorithm with $O(n^2)$ operations. For $v = n/\log n$, we have an $O(\log^2 n)$-time algorithm with $O(n^3/\log^2 n)$ operations.

We briefly compare Mayr's algorithm and ours. A parallel algorithm that computes the length of an optimal m-processor schedule for an interval order will be called an *m-LOS algorithm*. Mayr's algorithm basically consists of two parts. The first part uses an m-LOS algorithm to compute the lengths of optimal schedules, which takes $O(\log^2 n)$ time using $O(n^3 \log^2 n)$ operations on the CREW PRAM. The second part computes an actual scheduling, which takes $O(\log^2 n)$ time using $O(n^3 \log^2 n)$ operations on the CREW PRAM. Our algorithm also consists of two parts and its first part is an m-LOS algorithm, but our algorithm is quite different from Mayr's as follows.

- We give an efficient m-LOS algorithm that takes $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM by generalizing the techniques used for two-processor scheduling in [2].
- After computing the lengths of optimal schedules, we reduce the m-processor scheduling problem for interval orders to that of finding a maximum matching in a convex bipartite graph using the lengths to compute an actual scheduling. Therefore, the part of computing an actual scheduling in our algorithm takes $O(\log^2 n)$ time using $O(n \log^2 n)$ operations on the EREW PRAM.

The remainder of this paper is organized as follows. The next section gives basic definitions and a sequential scheduling algorithm. Section 3 describes the reduction of m-processor scheduling to maximum matching in a convex bipartite graph. Section 4 describes our efficient m-LOS algorithm.

2 Basic Definitions and Sequential Algorithm

In this section we describe basic definitions and a sequential m-processor scheduling algorithm. An instance of the m-processor scheduling problem is given by a precedence graph $\boldsymbol{G} = (V, E)$. A *precedence graph* is an acyclic and transitively closed digraph. Each vertex of $\boldsymbol{G}$ represents a task whose execution requires unit time on one of m identical processors. If there is a directed edge from task t to task t', then task t must be completed before task t' is started. In such a case, we call t a *predecessor* of t' and t' a *successor* of t. We use $\langle t, t' \rangle$ to denote a directed edge from t to t'. A *schedule* is a mapping from tasks to timesteps such that at most m tasks are mapped to each timestep and for every edge $\langle t, t' \rangle$, t is mapped to an earlier timestep than t'. The length of a schedule is the number of timesteps used. An optimal schedule is one with the shortest length.

Let $I = \{I_1, \ldots, I_n\}$ be a set of intervals with each interval I_i represented by $I_i.l$ and $I_i.r$, where $I_i.l$ and $I_i.r$ denote the left and right endpoints of interval I_i, respectively. Without loss of generality, we assume that all the endpoints are distinct. We also assume that the intervals are labeled in the increasing order of right endpoints, i.e., $I_1.r < I_2.r < \cdots < I_n.r$ because sorting can be done in $O(\log n)$ time using $O(n \log n)$ operations on the EREW PRAM [4]. Given a set I of n intervals, let $\boldsymbol{G_I} = (V, E)$ be a graph such that

- $V = I = \{I_1, I_2, \ldots, I_n\}$ and
- $E = \{\langle I_i, I_j \rangle \mid 1 \leq i, j \leq n \text{ and } I_i.r < I_j.l\}$.

Such a graph $\boldsymbol{G_I}$ is called an *interval order*. Note that $\boldsymbol{G_I}$ is a precedence graph. Given a set I of n intervals, the *interval graph* G_I is an undirected graph such that each vertex corresponds to an interval in I and two vertices are adjacent whenever the corresponding intervals have at least one point in common. Therefore, an interval graph G_I is a complement of the interval order $\boldsymbol{G_I}$. We say that two vertices are *independent* if they are not adjacent in a graph. Note that overlapping intervals are adjacent in G_I and they are independent of each other in $\boldsymbol{G_I}$. In what follows, we use the words *tasks* and *intervals* interchangeably.

A schedule of length r on m processors for an interval order $\boldsymbol{G_I}$ can be represented by an $m \times r$ matrix M, where the columns are indexed by $1, \ldots, r$ and the rows are indexed by $1, \ldots, m$. Let $P_1, \ldots, P_m$ denote the m identical processors. If task x is scheduled on processor P_i at timestep τ, then x is assigned to a slot $M[i, \tau]$. No two tasks are assigned to the same slot in M. A slot of M to which no task is assigned is said to have an *empty task*. We assume that the right endpoint of an empty task is larger than all right endpoints in I. A column of M is called *full* if it does not have an empty task. Let $opt(I)$ be the length of an optimal schedule for an interval order $\boldsymbol{G_I}$.

Algorithm m-seq(I, m)

Input: intervals in I
Output: $m \times opt(I)$ matrix M_s

```
begin
τ ← 1;
S_τ ←the list of intervals in I sorted in the increasing order of right endpoints;
while S_τ ≠ φ do
        S' ← {};
        Extract the first interval from S_τ and insert it to S';
        repeat
                Scan S_τ from left to right. When interval w is scanned,
                        if w is overlapping every interval in S'
                        then extract w from S_τ and insert it to S' fi;
        until (S' contains m intervals or all intervals of S_τ are considered)
        Schedule the intervals of S' in column τ of M_s
                in the order of the elements in list S';
        S_{τ+1} ← S_τ;
        τ ← τ + 1;
od
Output the schedule M_s constructed;
end
```

Fig. 1. Sequential scheduling algorithm

The sequential algorithm [15] in Figure 1 solves the m-processor scheduling problem for an interval order $\boldsymbol{G_I}$, which runs in $O(n \log n)$ time. Let $I(1, j)$ denote $\{I_1, \dots, I_j\}$, $1 \le j \le n$. Note that m-seq computes an optimal schedule for $\boldsymbol{G_{I(1,j)}}$. We can easily get the following facts from algorithm m-seq.

Fact 1 *All the intervals in the same column of M_s overlap each others.*

Fact 2 *In each column τ of M_s in* m-seq*, $M_s[1, \tau].r \le M_s[2, \tau].r \le \dots \le M_s[m, \tau].r$.*

Fact 3 *In the first row of M_s, $M_s[1, 1].r < M_s[1, 2].r < \dots < M_s[1, m].r$.*

Proof. It follows from the fact that for every τ, $M_s[1, \tau]$ is the first ending interval in S_τ and $M_s[1, \tau']$ with $\tau' > \tau$ is in S_τ.

3 Constructing an Optimal Schedule

In this section we describe our parallel m-processor scheduling algorithm for interval orders. We first describe characteristics of maximal cliques in interval graphs. A set of intervals form a *clique* if each pair of intervals in the set has a nonempty intersection. If we scan any given interval x from its left endpoint to its right, we can meet all those maximal cliques to which x belongs. This yields the Gilmore-Hoffman theorem [10].

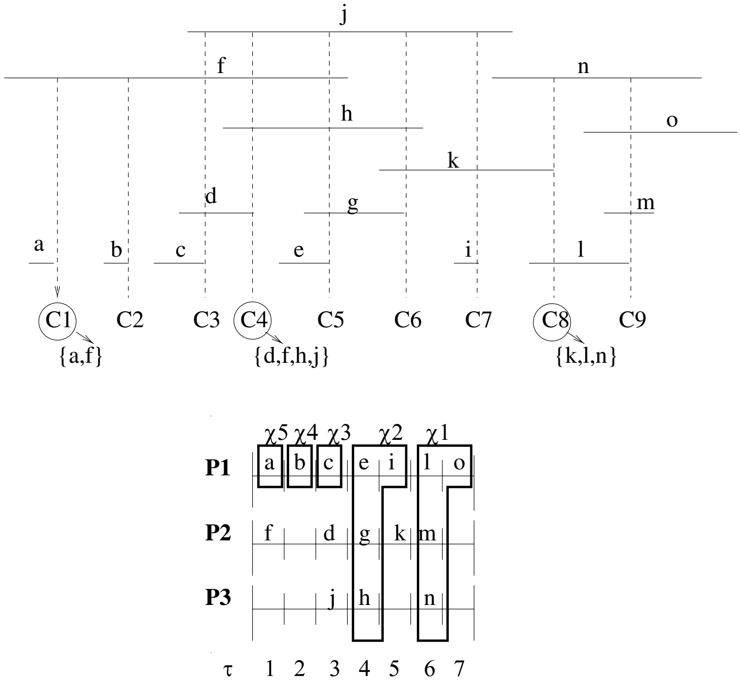

Fig. 2. An interval set I and $\boldsymbol{G_I}$'s optimal schedule when $m = 3$.

Theorem 1. [10] *The maximal cliques of an interval graph can be linearly ordered so that for any given interval x, the set of cliques in which x occurs appear consecutively in the linear order.*

Let k be the number of maximal cliques in G_I. Let $C_1, \ldots, C_k$ be the maximal cliques of G_I in the ordering of Theorem 1. Given an interval set, we can find the maximal cliques of the interval graph G_I using Lemma 1. In Figure 2, dotted vertical lines mark the right endpoints of Lemma 1, i.e., there are nine maximal cliques in G_I and they are $C_1 = \{a, f\}$, $C_2 = \{b, f\}$, $C_3 = \{c, d, f, j\}$, etc.

Lemma 1. [2] *In an interval set I, a right endpoint represents a maximal clique of G_I if and only if its previous endpoint in the sorted list of left and right endpoints is a left endpoint.*

For each interval $x \in I$, let s_x and l_x be the smallest and the largest j, respectively, such that x belongs to C_j. In Figure 2, $s_h = 4$ and $l_h = 6$ because interval h is in C_4, C_5 and C_6. Let $sltask(i, j)$, $1 \leq i, j \leq k$, be the set of intervals x such that $i \leq s_x$ and $l_x \leq j$. In Figure 2, $sltask(1, 5) = \{a, b, c, d, e, f\}$. Note that algorithm m-seq(I, m) in Figure 1 computes an optimal schedule for $\boldsymbol{G_{sltask(1,j)}}$, $1 \leq j \leq k$, because m-seq computes an optimal schedule for $\boldsymbol{G_{I(1,t)}}$, $1 \leq t \leq n$, and maximal cliques $C_1, \ldots, C_k$ of G_I are labeled by scanning endpoints of I

from left to right using Lemma 1. Let $len(i, j)$ be the minimum number of time-steps required to schedule all tasks in $sltask(i, j)$, i.e., $opt(sltask(i, j))$.

Lemma 2. [2] *For two intervals $x, y \in I$, $l_x < s_y$ if and only if $x.r < y.l$.*

We now describe our parallel m-processor scheduling algorithm for interval orders. Our algorithm consists of two parts. The first part is an m-LOS algorithm m-length, which will be described in Section 4. Algorithm m-length computes $len(1, j)$ for all $1 \leq j \leq k$. The second part computes an optimal schedule by reducing the m-processor scheduling problem for an interval order to that of finding a maximum matching in a convex bipartite graph.

We first describe the definition of a convex bipartite graph. A convex bipartite graph G is a triple (A, B, E) such that $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ are disjoint sets of vertices and the edge set E satisfies the following properties:

(1) Every edge of E is of the form (a_i, b_j).
(2) If $(a_i, b_j) \in E$ and $(a_i, b_{j+t}) \in E$, then $(a_i, b_{j+r}) \in E$ for every $1 \leq r < t$.

Property (1) is a bipartite property while property (2) is a convexity property. It is clear that every convex bipartite graph $G = (A, B, E)$, where $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$, is uniquely represented by a set of triples: $T = \{(a_i, g_i, h_i) \mid 1 \leq i \leq n\}$, where $g_i = \min\{j \mid (a_i, b_j) \in E\}$ and $h_i = \max\{j \mid (a_i, b_j) \in E\}$. Dekel and Sahni [5] developed an $O(\log^2 n)$-time convex bipartite maximum matching algorithm using $O(n \log^2 n)$ operations on the EREW PRAM.

Our m-processor scheduling algorithm is as follows.

Algorithm m-schedule

- Step 1: Compute s_x and l_x for every $x \in I$.
- Step 2: Let $L_0 = 0$. Let $L_j = len(1, j)$ for $1 \leq j \leq k$ and compute L_j.
- Step 3: Construct a convex bipartite graph $G_b = (A_b, B_b, E_b)$, where $A_b = I$, $B_b = \{1, 2, \ldots, mL_k\}$ and E_b is computed from L_j, $j \leq k$, as follows. If an interval $x \in I$ is in a maximal clique C_t in G_I, then x is adjacent to all j in B_b such that $mL_{t-1} + 1 \leq j \leq mL_t$. Since an interval x is in every C_t such that $s_x \leq t \leq l_x$ by Theorem 1, G_b is represented by $T = \{(x, mL_{s_x-1} + 1, mL_{l_x}) \mid x \in I\}$.
- Step 4: Find a maximum matching in G_b. Then an optimal schedule for $\boldsymbol{G_I}$ is represented by an $m \times L_k$ matrix M_b, whose j-th column consists of the tasks in A_b matched with $m(j-1)+1, \ldots, mj$ in B_b in the maximum matching of G_b.

We now prove the correctness of algorithm m-schedule.

Lemma 3. *All the intervals in the same column of M_b are independent of each other in $\boldsymbol{G_I}$.*

Proof. By definition of G_b, all intervals that are adjacent to one of $mL_{j-1}+1,\ldots,mL_j$ in G_b, $1 \le j \le L_k$, are also adjacent to all of $mL_{j-1}+1,\ldots,mL_j$ and they are all in the same maximal clique in G_I. Therefore, the intervals matched with $mL_{j-1}+1,\ldots,mL_j$ in the maximum matching of G_b are independent of each other in $\boldsymbol{G_I}$. Since all the intervals in columns $L_{j-1}+1,\ldots,L_j$, $1 \le j \le k$, in M_b are independent of each other in $\boldsymbol{G_I}$, we have the lemma.

Lemma 4. *The convex bipartite graph $G_b = (A_b, B_b, E_b)$ has a maximum matching of size n, i.e., all intervals in A_b are matched in a maximum matching of G_b.*

Proof. Construct an edge set $E' \subseteq A_b \times B_b$ from M_s constructed by algorithm m-seq in Figure 1 as follows. $E' = \{(x,j) \mid x \in A_b$ is the j-th element of M_s in the column-major order$\}$. Then every edge (x,j) in E' satisfies $m(\tau-1)+1 \le j \le m\tau$, where τ is the column number in M_s at which x is. We first show that $E' \subseteq E_b$. Note that $\tau \le L_{l_x}$ because m-seq produces an optimal schedule for $\boldsymbol{G_{sltask(1,l_x)}}$. And we have $\tau > L_{s_x-1}$ by the following.

- If x is in the first row in M_s, then $\tau > L_{s_x-1}$ because $x \notin sltask(1, s_x-1)$ and the task in the first row uses a new time unit after time L_{s_x-1}.
- If x is in row r such that $r \ge 2$, i.e., $x = M_s[r,\tau]$, then $M_s[1,\tau] \notin sltask(1, s_x-1)$ because $M_s[1,\tau]$ and x overlap by Fact 1, and thus $\tau > L_{s_x-1}$.

Hence $L_{s_x-1}+1 \le \tau \le L_{l_x}$. Since x is adjacent to all t such that $mL_{s_x-1}+1 \le t \le mL_{l_x}$ in E_b, every edge (x,j) in E' is also in E_b. Since j's are distinct, E' is a maximum matching of size n in G_b.

Lemma 5. *The $m \times L_k$ matrix M_b is an optimal schedule for $\boldsymbol{G_I}$.*

Proof. Consider tasks x and y of $\boldsymbol{G_I}$ such that y is a successor of x. Let τ and τ' be the columns of M_b at which x and y are, respectively. Note that M_b has L_k columns, which is $opt(I)$, and all tasks are in M_b by Lemma 4. Since all the tasks in the same column of M_b are independent of each other in $\boldsymbol{G_I}$ by Lemma 3, we can prove that M_b is an optimal schedule for $\boldsymbol{G_I}$ by showing that $\tau' > \tau$.

Let t and t' be integers matched with x and y, respectively, in the maximum matching of G_b. Then $t \le mL_{l_x}$ and $mL_{s_y-1}+1 \le t'$ by definition of G_b. Since y is a successor of x, we have $l_x < s_y$ by Lemma 2, which implies that t' is greater than t. Since y must be in a different column of M_b with that of x by Lemma 3, we have $\tau' > \tau$.

Theorem 2. *An optimal schedule for $\boldsymbol{G_I}$ on m processors can be solved in $O(\log^2 v + (n \log n)/v)$ time with $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \le n$ is a parameter.*

Proof. The correctness of algorithm m-schedule follows from Lemma 5. We will show that m-schedule takes $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM. Step 1 takes $O(\log n)$ time using $O(n \log n)$ operations as follows. In a sorted endpoints sequence, put 1 at the right endpoints of Lemma 1 and 0 in other endpoints and compute s_x and l_x using a prefix sum, i.e., the prefix sum at $x.l$ is $s_x - 1$ and the prefix sum at $x.r$ is l_x. Since we can compute all L_j $(= len(1, j))$ for $1 \leq j \leq k$ by running algorithm m-length in Section 4 only once, Step 2 takes $O(\log^2 v + (n \log n)/v)$ time with $O(nv^2 + n^2)$ operations. Step 3 takes constant time using $O(n)$ operations. Step 4 takes $O(\log^2 n)$ time with $O(n \log^2 n)$ operations using Dekel and Sahni's algorithm [5].

4 Computing the Length of an Optimal Schedule

We now describe our m-LOS algorithm. We obtain our m-LOS algorithm in Figure 3 by generalizing the 2-LOS algorithm in [2].

Algorithm m-length
for all i, j with $1 \leq i \leq j \leq k$ **do in parallel**
    compute $|sltask(i, j)|$
    $len_0(i, j) = \lceil |sltask(i, j)|/m \rceil$
od
for $r = 1$ to $\lceil \log n \rceil$ **do**
    for all i, j with $1 \leq i \leq j \leq k$ **do in parallel**
        $len_r(i, j) = \max_{i \leq x \leq j} \{len_{r-1}(i, x) + len_{r-1}(x + 1, j)\}$
    od
od
print $len_{\lceil \log n \rceil}(1, k)$
end

Fig. 3. An efficient m-LOS algorithm

We now prove the correctness of algorithm m-length. We first define sets $\chi_1, \ldots, \chi_z$ of tasks for an interval order such that:

- all tasks in any χ_{i+1} are predecessors of all tasks in χ_i and
- the length of an optimal schedule equals $\sum_i \lceil |\chi_i|/m \rceil$.

Our sets χ_i's for m-processor scheduling are the generalization of those for two-processor scheduling [3] tailored to the special case of interval orders. We do not explicitly compute these sets in algorithm m-length in Figure 3; we only make use of them for the proof of its correctness.

We define the sets $\chi_1, \ldots, \chi_z$ of tasks from the schedule M_s computed by algorithm m-seq in Figure 1 as follows. We recursively define tasks v_i and w_i for $i \geq 1$. Let v_1 be the last task executed by processor P_1 (i.e., v_1 is $M_s[1, opt(I)]$) and w_1 is (a possibly empty task) $M_s[m, opt(I)]$. Given v_i, we define w_{i+1} and v_{i+1} as follows. Suppose that v_i is $M_s[1, \tau]$. Let τ' be the largest column number less than τ in M_s such that $M_s[m,\tau'].r > v_i.r$ or $M_s[m,\tau']$ is an empty task. Then w_{i+1} is $M_s[m,\tau']$ and v_{i+1} is $M_s[1,\tau']$. In Figure 2, $v_1 = o$, and thus w_2 is an empty task and $v_2 = i$. Also $w_3 = j$ and $v_3 = c$. Note that each column τ'' such that $\tau' < \tau'' < \tau$ is full. Let z be the largest index for which w_z and v_z are defined. We assume that v_{z+1} is a special interval β whose right endpoint is smaller than all endpoints in I and $l_{v_{z+1}} = 0$. Let τ_i, $1 \le i \le z$, denote the timestep at which v_i is executed. Define χ_i to be $\{x|x$ is in column τ'' such that $\tau_{i+1} < \tau'' < \tau_i\} \cup \{v_i\}$. In Figure 2, sets χ_i's for $\boldsymbol{G_I}$ are marked by thick lines in the schedule. The characteristics of χ_i's are as follows.

Lemma 6. *In $\boldsymbol{G_I}$, every task $x \in \chi_i$ satisfies $x.r \le v_i.r$.*

Proof. Since τ_{i+1} is the largest column number less than τ_i such that $M_s[m,\tau_{i+1}].r > v_i.r$, we have $M_s[m,\tau''].r < v_i.r$ for $\tau_{i+1} < \tau'' < \tau_i$. Note that we assume that an empty task has the largest right endpoint in I. Since the task in the last row in each column has the largest right endpoint in the column by Fact 2, every task x in column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ satisfies $x.r < v_i.r$. Therefore, every $x \in \chi_i$ satisfies $x.r \le v_i.r$.

Lemma 7. *In $\boldsymbol{G_I}$, all tasks in χ_{i+1} are predecessors of all tasks in χ_i.*

Proof. Let y be a task in χ_i. Since every $x \in \chi_{i+1}$ satisfies $x.r \le v_{i+1}.r$ by Lemma 6, we can prove the lemma by showing that $v_{i+1}.r < y.l$. Since $y.r \le v_i.r$ and $v_i.r < w_{i+1}.r = M_s[m,\tau_{i+1}].r$, we have $y.r < M_s[m,\tau_{i+1}].r$. Since y is at one of columns $\tau_{i+1}+1,\ldots,\tau_i$, we have $M_s[1,\tau_{i+1}].r < y.r$ by Facts 2 and 3. Hence $M_s[1,\tau_{i+1}].r < y.r < M_s[m,\tau_{i+1}].r$. If y overlaps $M_s[1,\tau_{i+1}] = v_{i+1}$, then y should be assigned to column τ_{i+1} in m-seq in Figure 1, which is a contradiction. Therefore, $v_{i+1}.r < y.l$.

Theorem 3. *The length of an optimal schedule for $\boldsymbol{G_I}$ is $\sum_{1\le i\le z}\lceil|\chi_i|/m\rceil$.*

Proof. Since each column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ is full and $v_i = M_s[1,\tau_i]$ is in χ_i, we get $\lceil|\chi_i|/m\rceil = \tau - \tau'$. Therefore, $\sum_{1\le i\le z}\lceil|\chi_i|/m\rceil$ is the number of columns in M_s, which is *opt*(I).

When $m = 2$, Chung et al. [2] showed that χ_i equals $sltask(l_{v_{i+1}}+1, l_{v_i})$ for $1 \le i \le z$ and that $len_{\lceil\log n\rceil}(i,j)$ equals $len(i,j)$ for $1 \le i \le j \le k$. Similarly, we can prove the correctness of algorithm m-length as follows.

Lemma 8. *In $\boldsymbol{G_I}$, $\chi_i \subseteq sltask(l_{v_{i+1}}+1, l_{v_i})$ for $1 \le i \le z$.*

Proof. Let x be a task in χ_i. Since $v_{i+1} \in \chi_{i+1}$ is a predecessor of x by Lemma 7, we have $v_{i+1}.r < x.l$, which implies $l_{v_{i+1}} < s_x$ by Lemma 2. Since $x.r \le v_i.r$ by Lemma 6, we have $l_x \le l_{v_i}$. Therefore, x is in $sltask(l_{v_{i+1}}+1, l_{v_i})$.

Corollary 1. *In $\boldsymbol{G_I}$, $\bigcup_{i\le t\le j}\chi_t \subseteq sltask(l_{v_{j+1}}+1, l_{v_i})$ for $1 \le i \le j \le z$.*

Corollary 2. *In* $\boldsymbol{G_I}$*, all tasks in* $sltask(l_{v_{j+1}}+1, l_{v_i})$*,* $i \leq j$*, are successors of all tasks in* $\bigcup_{j+1 \leq t \leq z} \chi_t$ *and predecessors of all tasks in* $\bigcup_{1 \leq t \leq i-1} \chi_t$*.*

Lemma 9. *Every task in* $sltask(l_{v_{i+1}}+1, l_{v_i})$ *is in one of columns* $\tau_{i+1}+1, \ldots, \tau_i$*.*

Proof. Let y be a task in $sltask(l_{v_{i+1}}+1, l_{v_i})$. Note that y satisfies $M_s[1, \tau_{i+1}].r < y.l$ by Lemma 2 and $y.r < M_s[1, \tau_i + 1].l$ by Lemma 7. Therefore, y must be in one of columns $\tau_{i+1}+1, \ldots, \tau_i$ by the way algorithm m-seq in Figure 1 works.

Lemma 10. *In* $\boldsymbol{G_I}$*,* $\sum_{i \leq t \leq j} \lceil |\chi_t|/m \rceil = len(l_{v_{j+1}}+1, l_{v_i})$ *for* $1 \leq i \leq j \leq z$*.*

Proof. The proof for the case $m = 2$ is in Lemma 8 in [2] and the proof of the lemma is similar.

Lemma 11. *In algorithm* m-length*,* $len_r(i,j) \leq len(i,j)$ *for* $0 \leq r \leq \lceil \log n \rceil$*.*

Proof. It is similar to the proof of Lemma 9 in [2].

Lemma 12. *In algorithm* m-length*,* $len_{\lceil \log n \rceil}(i,j) \geq len(i,j)$ *for* $1 \leq i \leq j \leq k$*.*

Proof. We show that $len_{\lceil \log n \rceil}(1,k) \geq len(1,k)$. We prove by induction on r that for $i \leq 2^r$,

$$len_r(l_{v_{x+i}}+1, l_{v_x}) \geq \sum_{x \leq t < x+i} \lceil |\chi_t|/m \rceil \qquad (1)$$

When $r = 0$, (1) holds as follows. Since each column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ is full, $\lceil |sltask(l_{v_{x+1}}+1, l_{v_x})|/m \rceil \geq \lceil |\chi_x|/m \rceil \geq \tau_i - \tau_{i+1}$ by Lemma 8. Since $\lceil |sltask(l_{v_{x+1}}+1, l_{v_x})|/m \rceil \leq \tau_i - \tau_{i+1}$ by Lemma 9, we have $\lceil |sltask(l_{v_{x+1}}+1, l_{v_x})|/m \rceil = \tau_i - \tau_{i+1}$. Therefore, $len_0(l_{v_{x+1}}+1, l_{v_x}) = \lceil |sltask(l_{v_{x+1}}+1, l_{v_x})|/m \rceil = \lceil |\chi_x|/m \rceil$. Assume that (1) holds after r iterations of the main loop. In the $(r+1)$st iteration for $2^r < i \leq 2^{r+1}$,

$$\begin{aligned} len_{r+1}(l_{v_{x+i}}+1, l_{v_x}) &\geq len_r(l_{v_{x+i}}+1, l_{v_{x+2^r}}) + len_r(l_{v_{x+2^r}}+1, l_{v_x}) \\ &\geq \sum_{x+2^r \leq t < x+i} \lceil |\chi_t|/m \rceil + \sum_{x \leq t < x+2^r} \lceil |\chi_t|/m \rceil \\ &\geq \sum_{x \leq t < x+i} \lceil |\chi_t|/m \rceil \end{aligned}$$

Since each χ_i contains at least one task, there are at most n χ_i's. Thus,

$$\begin{aligned} len_{\lceil \log n \rceil}(l_{v_{z+1}}+1, l_{v_1}) &\geq \sum_{1 \leq t \leq z} \lceil |\chi_t|/m \rceil \\ &\geq len(l_{v_{z+1}}+1, l_{v_1}) \quad \text{by Lemma 10.} \end{aligned}$$

Since $l_{v_{z+1}}+1 = 1$ and $l_{v_1} = k$, we get $len_{\lceil \log n \rceil}(1,k) \geq len(1,k)$. Similarly, we can prove that $len_{\lceil \log n \rceil}(i,j) \geq len(i,j)$ for $1 \leq i \leq j \leq k$ by using sets of χ_i's for $\boldsymbol{G_{I'}}$, where I' is $sltask(i,j)$ in I.

Theorem 4. *There is an m-LOS algorithm that requires $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM, where v is a parameter such that $v \le n$. Furthermore, it also computes the length of an optimal schedule for $\boldsymbol{G_{sltask(1,j)}}$, $1 \le j \le k$.*

Proof. The correctness of algorithm m-length follows from Lemmas 11 and 12. Algorithm m-length has a straightforward implementation using $O(\log^2 n)$ time and $O(n^3)$ processors on the CREW PRAM. It can be improved to $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations using Galil and Park's reduction technique [8], which is similar to the proof of Theorem 3 in [2].

References

1. M. Bartusch, R. H. Mohring, and F. J. Radermacher, "M-machine unit time scheduling: A report of ongoing research," *Lecture Notes in Economics and Mathematical Systems* **304**, Springer-Verlag (1988) 165–212.
2. Y. Chung, K. Park, and Y. Cho, "Parallel maximum matching algorithms in interval graphs," *Int'l. J. Foundations of Comput. Science* **10**, **1** (1999) 47–60.
3. E. Coffman and R. Graham, "Optimal scheduling for two processor systems," *Acta Informatica* **1** (1972) 200–213.
4. R. Cole, "Parallel merge sort," *SIAM J. Comput.* **17**, **4** (1988) 770–785.
5. E. Dekel and S. Sahni, "A parallel matching algorithm for convex bipartite graphs and applications to scheduling," *J. Parallel Distrib. Comput.* **1** (1984) 185–205.
6. M. Fujii, T.Kasami, and K. Ninomiya, "Optimal sequencing of two equivalent processors," *SIAM J. Appl. Math.* **17** (1969) 784–789.
7. H.N. Gabow, "An almost-linear algorithm for two-processor scheduling," *J. ACM* **29**, **3** (1982) 766–780.
8. Z. Galil and K. Park, "Parallel algorithms for dynamic programming recurrences with more than $O(1)$ dependency," *J. Parallel Distrib. Comput.* **21**, (1994) 213–222.
9. M. R. Garey and D. S. Johnson, "Scheduling tasks with nonuniform deadlines on two processors," *J. ACM* **23** (1976) 461–467.
10. M.C. Golumbic, *Graph theory and perfect graphs*, Academic Press, New York, 1980.
11. D. Helmbold and E. Mayr, "Two processor scheduling is in *NC*," *SIAM J. Comput.* **16**, **4** (1987) 747–759.
12. T. C. Hu, "Parallel sequencing and assembly line problems," *Oper. Res.* **9** (1961) 841–848.
13. E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Technical report*, Centrum voor Wiskunde en Informatica, 1989.
14. E. Mayr, "Scheduling interval orders in parallel," *Parallel Algorithms and Applications* **8** (1996) 21–34.
15. C. H. Papadimitriou and M Yannakakis, "Scheduling interval-ordered tasks," *SIAM J. Comput.* **8** (1979), pp. 405–409.
16. H. Jung, M. Serna and P. Spirakis, "A parallel algorithm for two processors precedence constraint scheduling," *Proc. Int'l Colloquium on Automata, Languages and Programming* (1991) pp. 417–428.
17. D. Kozen, U.V. Vazirani and V.V. Vazirani, "*NC* algorithms for cocomparability graphs, interval graphs, and unique perfect matchings," *Proc. Foundations of Software Technology and Theoretical Computer Science* (1985) pp. 496–503.

18. A. Moitra and R. Johnson, “A parallel algorithm for maximum matching on interval graphs,” *Proc. Int’l Conference on Parallel Processing* **3** (1989) pp. 114–120.
19. S. Sunder and X.He, "Scheduling interval ordered tasks in parallel,” *J. Algorithm* **26** (1998), pp.34–47.
20. J. D. Ulman, Complexity of sequencing problems, *in* "Computer and job scheduling theory” (E. G. Coffman, Ed.), Wiley, 1976.
21. U.V. Vazirani and V.V. Vazirani, “The two processor scheduling is in random *NC*,” *SIAM J. Comput.* (1989) pp. 1140–1148.