

Parallel Algorithms for Maximum Matching in Interval Graphs*

Marilyn G. Andrews[†] Mikhail J. Atallah[‡] Danny Z. Chen[§] D. T. Lee[¶]

Abstract

Given a set of n intervals representing an interval graph, the problem of finding a maximum matching between pairs of disjoint (nonintersecting) intervals has been considered in the sequential model. Here we present parallel algorithms for computing maximum cardinality matchings among pairs of disjoint intervals in interval graphs in the EREW PRAM and hypercube models. For the general case of the problem, our algorithms compute a maximum matching in $O(\log^3 n)$ time using $O(n/\log^2 n)$ processors on the EREW PRAM and using $O(n)$ processors on the hypercubes. For the case of proper interval graphs, our algorithm runs in $O(\log n)$ time using $O(n)$ processors if the input intervals are not given already sorted and using $O(n/\log n)$ processors otherwise, on the EREW PRAM. On n -processor hypercubes, our algorithm for this case takes $O(\log n \log \log n)$ time for unsorted input and $O(\log n)$ time for sorted input. Our parallel results also lead to optimal sequential algorithms for computing maximum matchings among disjoint intervals. We also present an improved parallel algorithm for maximum matching between overlapping intervals in proper interval graphs.

1 Introduction

Consider an interval set $I = \{I_1, I_2, \dots, I_n\}$ on the x -axis, where interval $I_i = [l(i), r(i)]$ is specified by its two endpoints: the left endpoint, $l(i)$, and the right endpoint, $r(i)$, with $l(i) < r(i)$. Two intervals $I_i = [l(i), r(i)]$ and $I_j = [l(j), r(j)]$ are *disjoint*

if $r(i) < l(j)$ or $r(j) < l(i)$; otherwise they *overlap*. A graph G is called an *interval graph* if there exists a set I_G of intervals such that there is a one-to-one correspondence between the vertices of G and the intervals in I_G , and that any two vertices in G are connected by an edge if and only if their corresponding intervals in I_G overlap. Such an interval set I_G is called an *interval model* of G . An interval graph G is said to be *proper* if and only if there is an interval model I_G of G such that no interval in I_G is contained within any other interval in I_G . Interval graphs find applications in many areas, such as VLSI design, scheduling, biology, traffic control, and archeology [15]. In this paper, we assume that an interval model of the corresponding interval graph is already given. We will refer to interval I_i , interval i , interval $[l(i), r(i)]$, and vertex i (corresponding to interval i) interchangeably.

A *matching* in a graph G is a subset M of edges of G such that no two distinct edges in M are incident to the same vertex. The problem of computing maximum matchings in graphs has many applications and has received a lot of attention [10]. However, there is no known deterministic parallel algorithm for computing maximum matchings in general graphs that takes polylogarithmic time using a polylogarithmic number of processors [17].

We consider the following matching problem on a set I of n intervals: Find a maximum cardinality matching M for I such that two intervals can be matched in M only if they are disjoint. This problem, in fact, is that of computing a maximum cardinality matching in the complement graph of the corresponding interval graph G of I , and such a complement graph is called a *comparability graph*. An $O(n \log n)$ time sequential algorithm for this matching problem is given by Andrews and Lee [3]. A related problem for matching in interval graphs was considered by Moitra and Johnson [24], who give a sequential and a parallel algorithm for finding maximum cardinality matchings in interval graphs where two *overlapping* intervals can be matched. To the best of our knowledge, there was no efficient parallel algorithm (i.e., in polylogarithmic time using a polylogarithmic number of processors) previously known for finding maximum matchings in comparability graphs.

We present the first efficient parallel algorithms for computing maximum cardinality matchings in interval models in which only disjoint intervals can be matched. For the general case of the problem, our al-

*Part of this research was done while the authors were visiting the Leonardo Fibonacci Institute in Trento, Italy.

[†]AT&T Bell Laboratories, 2000 N. Naperville Road, Naperville, IL 60566, USA. E-mail: m.g.andrews@att.com.

[‡]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA. E-mail: mja@cs.purdue.edu. This author's research was supported in part by the National Science Foundation under Grant CCR-9202807.

[§]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. E-mail: chen@cse.nd.edu.

[¶]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA. E-mail: dtlee@eecs.nwu.edu. This author's research was supported in part by the National Science Foundation under Grants CCR-8901815 and CCR-9309743.

gorithms compute a maximum matching in $O(\log^3 n)$ time using $O(n/\log^2 n)$ processors on the EREW PRAM and using $O(n)$ processors on the hypercubes. For the case of proper interval graphs, our algorithm runs in $O(\log n)$ time using $O(n)$ processors if the intervals are not given already sorted and using $O(n/\log n)$ processors otherwise, on the EREW PRAM. On n -processor hypercubes, our algorithm for this case takes $O(\log n \log \log n)$ time for unsorted input and $O(\log n)$ time for sorted input. The approaches used by our parallel algorithms are very different from the seemingly inherently sequential *plane sweeping* method used in [3], and are based on new characterizations of this matching problem. In fact, by simulating sequentially our EREW PRAM algorithm, we can immediately give an optimal sequential $O(n \log n)$ time algorithm for computing maximum matchings among disjoint intervals. Furthermore, if the endpoints of the input intervals are given sorted, we can make our sequential algorithm for computing maximum matchings among disjoint intervals run in linear time, as follows: A key subproblem in our parallel algorithms for computing maximum matchings among disjoint intervals is the problem of finding maximum matchings in convex bipartite graphs, which can be solved sequentially in linear time [13, 23] by using the optimal sequential algorithm for computing maximum matchings in convex bipartite graphs [13, 23]. By simulating sequentially the rest of our EREW PRAM algorithm, a maximum matching among disjoint intervals can be obtained in linear time. Such a sequential algorithm is very different from the plane sweeping algorithm of Andrews and Lee [3] (which still requires $O(n \log n)$ time for sorted input intervals). We also give an EREW PRAM algorithm for maximum matching between *overlapping* intervals in proper interval graphs, improving the processor complexity of the previously best known solution for this problem [24] by a factor of $n/\log n$.

The computational models we use are the EREW PRAM and n -processor hypercube [22].

The rest of the paper is organized as follows. Section 2 gives some notation and preliminary results we need. In Section 3, we present parallel algorithms for the disjoint matching problem for general interval graphs. In the interest of space, the algorithms for the disjoint matching problem for proper interval graphs and our improved parallel algorithm for the matching problem on overlapping intervals are omitted. See the full paper [2].

2 Preliminaries

The input consists of a set of n intervals $I = \{I_1, I_2, \dots, I_n\}$. To avoid cluttering the exposition, we assume that no two input intervals have the same

endpoint (i.e., the $2n$ endpoints are distinct). Our algorithms can easily be modified for the general case.

We first sort the $2n$ endpoints in I from left to right if they are not given sorted. This sorting can be done in $O(\log n)$ time using $O(n)$ processors on the EREW PRAM [9] and in $O(\log n \log \log n)$ time on n -processor hypercubes [11]. From now on, we assume that the $2n$ endpoints in I are available in this sorted order. On the EREW PRAM, these endpoints are stored in an array; on an n -processor hypercube, each processor PE_i stores 2 endpoints, with the sorted order of the endpoints corresponding to the increasing order of the processor indices. Without loss of generality (WLOG), we also assume that the intervals in I have been relabeled such that $i < j$ implies $l(i)$ occurs before $l(j)$ in the sorted array of endpoints. In the case of proper intervals, $i < j$ also implies $r(i)$ occurs before $r(j)$ in the sorted array of endpoints. This relabeling can be easily implemented by a parallel prefix computation. The parallel prefix operation can be performed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM [20, 21] and in $O(\log n)$ time on n -processor hypercubes [22].

Given a matching M for I , we say that an interval *matches left* (resp., *right*), denoted by a left (resp., right) arrow, if it is matched with an interval to its left (resp., right). An interval is *free* if it is unmatched with respect to M . That interval i is matched with j in M is denoted as $mate_M(i) = j$.

An interval i is said to be *to the left* (resp., *right*) of a vertical line L if $r(i) < x(L)$ (resp., $l(i) > x(L)$), where $x(L)$ denotes the x -coordinate of L .

Definition 2.1 A middle line V is a vertical line that divides the set of $2n$ endpoints of I into two subsets, with one subset to each side of V , such that every subset has exactly n endpoints. Those intervals of I that are cut by a middle line V are called cut intervals.

Lemma 2.1 For any middle line V , the number b of intervals not cut by V lying to the left of V is the same as the number of intervals not cut by V lying to the right of V . Furthermore, $b \leq |M|$, where M is a maximum matching of I .

Proof: See the full paper. \square

The following problem, defined by Kim [18], plays an important role in our algorithms.

Definition 2.2 Given a set of points on the x -axis, some colored red and the other colored blue, a red point r can be matched with a blue point b if $x(r) < x(b)$. The red-blue matching problem is to find a maximum matching between the red and blue points.

Kim [18] presents an EREW PRAM algorithm for the red-blue matching problem with sorted input which takes $O(\log n)$ time using $n/\log n$ processors. On n -processor hypercubes, the red-blue matching

problem with sorted input can be solved in $O(\log n)$ time, as follows: First apply Kim's reduction [18] to reduce the problem to the *all nearest smaller values* problem [4] (this reduction takes $O(\log n)$ time on the hypercubes since it mainly performs parallel prefix); then use the optimal hypercube algorithms by Chen [8] and Kravets and Plaxton [19] to solve the all nearest smaller values problem in $O(\log n)$ time.

Our algorithms also make use of convex bipartite graphs which are defined next.

Definition 2.3 A convex bipartite graph $G = (A, B, E)$ is a bipartite graph where A and B are respectively sequences of vertices (a_1, a_2, \dots, a_m) and (b_1, b_2, \dots, b_n) , and E is the set of edges. $(a, b) \in E$ implies that $a \in A$ and $b \in B$, and furthermore, $(a_i, b_j) \in E$ iff $f_i \leq j \leq l_i$, where f_i (resp., l_i) is the index of the first (resp., last) element in B with which a_i is connected.

Maximum cardinality matchings in convex bipartite graphs can be computed sequentially by using the linear time algorithm of Gabow and Tarjan [13] that is based on the nearly linear time solution of Lipski and Preparata [23], or by using the $O(n \log n)$ time algorithm of Gallo [14]. The next lemma is needed by our PRAM algorithms.

Lemma 2.2 For every convex bipartite graph $G = (A, B, E)$ with $|A| \leq n$ and $|B| \leq n$, a maximum cardinality matching M_{cb}^* of G can be obtained in $O(\log^2 n)$ time using $O(n/\log n)$ EREW PRAM processors.

Proof: The basic idea is to apply Brent's theorem [6] to simulate the parallel algorithm of Dekel and Sahni [12] for computing maximum cardinality matchings in convex bipartite graphs. Dekel and Sahni's algorithm [12] computes a maximum cardinality matching in a convex bipartite graph in $O(\log^2 n)$ time using $O(n)$ EREW PRAM processors. Note that the algorithm in [12] consists of two passes, each pass traversing a complete binary tree of n leaves level by level. At each level, the algorithm in [12] essentially performs a constant number of parallel merges; hence the number of operations performed by this algorithm at each level is $O(n)$. The total number of operations taken by the algorithm of Dekel and Sahni [12], therefore, is $O(n \log n)$. By applying Brent's theorem [6] and by using the parallel merge algorithms on the EREW PRAM [5, 7, 16], we can easily simulate the algorithm [12] in $O(\log^2 n)$ time using $O(n/\log n)$ EREW PRAM processors. \square

A maximum cardinality matching in convex bipartite graphs can be computed in $O(\log^2 n)$ time using $O(n)$ processors on an n -processor hypercube [1].

3 Disjoint Matching for General Interval Graphs

In the next subsection (3.1), we give the observations for our algorithms. Subsection 3.2 presents the idea and main steps for finding a maximum matching between disjoint intervals in general interval graphs. We refer you to the full paper for details of our EREW PRAM and hypercube algorithms.

3.1 Useful Observations

Our algorithms for the general case of disjoint matching are based on the following observations.

Lemma 3.1 There exists an optimal matching M^* such that either the matched intervals in M^* that are to the left of a middle line V all have right arrows (i.e., they match right), or the matched intervals in M^* that are to the right of V all have left arrows (i.e., they match left).

Proof: See the full paper. \square

Lemma 3.2 There exists an optimal matching M^* such that the middle line V cuts all free intervals with respect to M^* .

Proof: See the full paper. \square

Based on Lemmas 3.1 and 3.2, there exists an optimal matching such that all the non-cut intervals on either the left of V match right or *vice versa*. For all the cut intervals, some of them (say, l) match left and some of them (say, r) match right. WLOG, assume that all the non-cut intervals to the right of V match left; this implies that $r \geq l$. Then there are precisely $|r - l|$ intervals lying to the left of V which are paired with each other. Denote this set as C' . Since the intervals of C' are paired, this implies that $|r - l|$ must be a multiple of 2. Then, the form of M^* is as shown in Figure 1.

Lemma 3.3 Let M' be an optimal matching in the form defined in Lemmas 3.1 and 3.2. WLOG, assume that the number of intervals cut by V which match right, r , is greater than the number of intervals cut by V which match left, l . Let i be the unmatched interval such that $l(i)$ is closest to V among all the unmatched intervals. Let j be the interval cut by V which matches right, such that $l(j)$ is closest to V among all the cut intervals that match right. Let $z = \max\{l(i), l(j)\}$ (the closest endpoint to V). Let C' denote the set of intervals to the left of V which are paired together. Then there exists an optimal matching M^* such that for any interval $c \in C'$, $z \leq r(c) \leq x(V)$. (For the case in which $l > r$, i is the unmatched interval such that $r(i)$ is closest to V . Let j be the interval cut by V which matches left and whose $r(j)$ is closest to

V . Let $z = \min\{r(i), r(j)\}$. Then there exists an optimal matching M^* such that for any interval $c \in C'$, $x(V) \leq l(c) \leq z$.

Proof: See the full paper. □

Based on Lemmas 3.1 and 3.2, there is an optimal matching M^* that has four different types of matched pairs, as illustrated in Figure 1.

Type 1 matches an interval to the left of V with one to the right of V .

Type 2 matches an interval to the left of V with an interval cut by V .

Type 3 matches an interval to the right of V with an interval cut by V .

Type 4 matches two intervals on the same side of V .

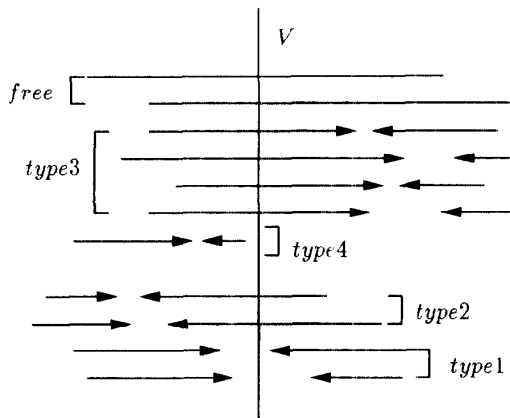


Figure 1: The four types of matched pairs in an maximum matching.

From Lemma 2.1, we know that there are b intervals to the left and to the right of V . We number the intervals to the left (resp., right) of V from 1 to b (resp., from $b + 1$ to $2b$) by decreasing x -coordinates of the right (resp., left) endpoints and store them in an array X (resp., Y). For example, the interval b has its right endpoint furthest from V and the interval $2b$ has its left endpoint closest to V (see Figure 2). Let U denote the set of intervals cut by V . Now consider an interval $u \in U$. The set of possible candidates for u to “match left” is a range of intervals $k, k + 1, \dots, b$, where k is the interval whose right endpoint is closest to $l(u)$ on the left (if $l(u) \leq r(b)$, then this set is empty). Similarly, the set of possible candidates for u to “match right” is a range of intervals $b + 1, b + 2, \dots, l$, where l is the interval whose left endpoint is closest to $r(u)$ on the right (if $r(u) \geq l(b + 1)$, then this set is

empty). Note that we can combine these two sets into a single range of intervals $(k, k + 1, \dots, b, b + 1, \dots, l)$. We can also represent the range as $(first, last)$, where $first = k$ and $last = l$. Note that this representation of all possible matching pairs between U and $(X \cup Y = I - U)$ is that of a convex bipartite graph.

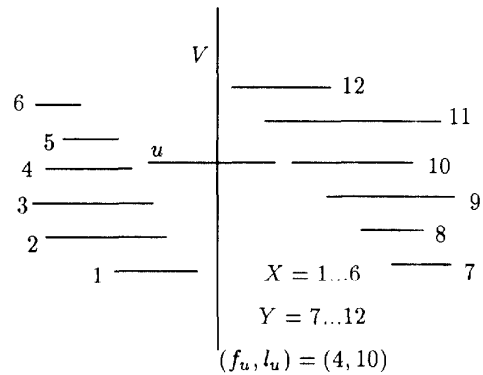


Figure 2: Candidates for matching with $u \in U$.

In order to determine the subset of intervals in U which are candidates for M^* , we construct a convex bipartite graph G such that $A = U$, $B = I - U$, and the edge set E such that $(u, b_u) \in E$ iff $k \leq b_u \leq l$, with k and l determined as discussed earlier. Note that A is convex on B . The maximum cardinality matching M_{cb}^* of G is useful to us.

Lemma 3.4 Let M_{cb}^* be a maximum cardinality matching of the convex bipartite graph G . Then any interval $u \in U$ not matched in M_{cb}^* is not a candidate for matching in the maximum matching M^* as defined in Lemmas 3.1 and 3.2.

Proof: See the full paper. □

The following operation involving red-blue matching is needed by our algorithms. Given a set S_R of red points and a set S_B of blue points on the x -axis, we first perform a red-blue matching to obtain a maximum assignment M_{RB} of the red points to the blue points. We then include more red points (S'_R) and rerun the red-blue matching algorithm to get a new optimal matching M'_{RB} on the sets $S_R \cup S'_R$ and S_B . We claim that it is possible to convert M'_{RB} to another optimal matching M''_{RB} (for $S_R \cup S'_R$ and S_B) such that every red point in S_R that is matched in M_{RB} is also matched in M''_{RB} .

Lemma 3.5 For every red point $r \in S_R$ which is matched in M_{RB} but not in M'_{RB} , there exists a distinct red point $p(r) \in S'_R$ that is to the right of r and is matched in M_{RB} .

Proof: See the full paper. □

Therefore, to convert M'_{RB} to M''_{RB} such that every $r \in S_R$ which is matched in M_{RB} is also matched in M''_{RB} , we must find, for every r which is matched in M_{RB} but unmatched in M'_{RB} , a corresponding point $p(r) \in S'_R$ that is matched in M'_{RB} and is to the right of r . There are two sets of points, (i) $r \in S_R$ such that $r \in M_{RB}$ but $r \notin M'_{RB}$, and (ii) $r' \in S'_R$ such that $r' \in M'_{RB}$. We want to find a pairing (matching) between r and r' such that r is to the left of r' . Clearly, this is an instance of a red-blue matching with the first set of points as red and the second set of points as blue.

3.2 General Matching Algorithm for Disjoint Intervals

We begin with a general discussion of the basis for the algorithms, and then give the main steps.

Let M^* be an optimal matching in the form defined by Lemmas 3.1 and 3.2. Then all intervals not cut by V will be in M^* . Hence we can start with an initial matching M which consists of the b matched pairs formed by matching each interval to the left of V with a distinct interval to the right of V . (In fact, any interval to the left of V can match with any interval to the right of V .) We then try to increase the size of the matching by pairing intervals cut by V with intervals in M , as described below. Let U be the set of unmatched intervals with respect to M .

From Lemma 3.2, we know that all free intervals in M^* will be cut by V . Thus, if we match the maximum number of cut intervals (i.e., the set U) while simultaneously retaining all non-cut intervals in the matching, then we will have the optimal matching M^* . In order to increase the cardinality of the matching (from the size of M), for every $j \in U$ which is a candidate to match right (resp., left) with a non-cut interval i , we must find another interval which can match left (resp., right) with $mate_M(i)$. That is, by finding a new mate for both i and $mate_M(i)$, we increase the cardinality of the matching (from the size of M) by one pair. Precisely, we can increase the size of the matching in one of two ways.

1. For a $j \in U$ which matches left (resp., right), we find another $j' \in U$ which matches right (resp., left). Or,
2. For two intervals $j, j' \in U$ which both match left (resp., right), we find a type-4 interval pair (see Figure 1) to the right (resp., left) of V . Note that it takes two intervals of U to match left (resp., right) for every type-4 pair to the right (resp., left) of V . Consider the pairs formed with j and j' (say, $(i, j), (i', j')$, with both i and i' to the left of V). These pairs can possibly belong to M^* if we are able to find two intervals from the right of V to create a type-4 pair (such a type-4 pair does not have to be $(mate_M(i), mate_M(i'))$)

because any interval to the left of V can match any interval to the right of V).

For all the intervals in U , we determine the maximum subset of U which can be matched with the intervals in M , and let U' denote this subset. We partition U' into two subsets: L (resp., R), the set of intervals of U' which are paired with some intervals of M to the left (resp., right) of V . Each interval in $L \cup R$ is paired with a unique interval in M . We refer to any interval in M not paired with an interval in $L \cup R$ as *unassigned*. If L and R are not equal in size, then we attempt to make the sizes of the two sets equal by finding from the larger set those intervals which can be paired with unassigned intervals of M on the opposite side of V ; i.e., for such intervals, we change the direction of their matching.

If L and R cannot be made equal in size, then on the side of V which has fewer candidates (e.g., if $|L| > |R|$, then the right side of V), we look for type-4 pairs. WLOG, assume that $|L| > |R|$ (the reasoning is identical for the case in which $|R| > |L|$). In order to utilize in the matching as many intervals in L as possible, we must find intervals of M to the right of V such that their matching direction can be reversed (i.e., match right instead of left). Note that we need to find at most $\lfloor (|L| - |R|)/2 \rfloor$ type-4 pairs from the intervals to the right of V . Hence when $|L| - |R|$ is not a multiple of 2, we delete one interval from L . We then compute the set of intervals to the right of V which are candidates for type-4 pairs. For all such candidates, their endpoints are in the range specified by Lemma 3.3. Let these candidate intervals belong to a set C of intervals. (C may also contain some intervals from R ; this will be made clear.) For the set C , we compute its optimal matching M^*_C . Let m denote the number of type-4 pairs in M^*_C . If $m \geq \lfloor (|L| - |R|)/2 \rfloor$, then we take $\lfloor (|L| - |R|)/2 \rfloor$ type-4 pairs from M^*_C and add them to M^* ; otherwise, we add all the type-4 pairs in M^*_C to M^* , and delete $(|L| - |R|) - 2m$ intervals from L . For each remaining $l \in L$, there is a unique interval to the left of V with which l is paired, and we add these $|L|$ pairs to M^* . For each $r \in R$, there is a unique interval to the right of V with which r is paired, and we add these $|R|$ pairs to M^* . Finally, there are s unassigned intervals remaining on both the left and right sides of V ; form s pairs from those intervals and add them to M^* . M^* thus obtained is an optimal matching in the input interval set I .

We now discuss in more detail the various steps involved. To determine the initial matching M , we sort the $2n$ endpoints and determine the location of the middle line V . Then by Lemma 2.1, the set of intervals to the left of V is of the same size as the set of intervals to the right of V . Identify these intervals and store them in the array M of size $2b$. All the remaining intervals are cut by V and are stored in the array U .

We next determine, for all the intervals in U , whether they can be matched with intervals in M . First, for every $u \in U$, we find the range of the intervals in M with which u is disjoint, as follows. Store the intervals to the left of V in an array X in decreasing order of their right endpoints and assign a new index to each interval from 1 to b ; also, store the intervals to the right of V in an array Y in decreasing order of their left endpoints and assign a new index to each interval from $b + 1$ to $2b$. For each $u \in U$, find from the set X the right endpoint, f_u , closest to $l(u)$ on the left, and from the set Y find the left endpoint, l_u , closest to $r(u)$ on the right. Then using the new indices $1, 2, \dots, 2b$, express the set of intervals with which u is disjoint as a range (f_u, l_u) of intervals. Construct a convex bipartite graph $G = (A, B, E)$, with $A = U$, $B = M = I - U$, and the edge set $E = \{(a_i, b_j) \mid a_i \in U, b_j \in I - U, f_{a_i} \leq b_j \leq l_{a_i}, f_{a_i} \in M\}$. Compute a maximum matching M_{cb}^* in G . As a result, each $u \in U$ which is a candidate for M^* (Lemma 3.4) will be assigned to an $i \in M$. Based on whether i is to the left or right of V , u either matches left or matches right in M_{cb}^* . Denote the subset of U which matches left (resp., right) in M_{cb}^* as L (resp., R), and let $k_l = |L|$ and $k_r = |R|$. The intervals of M which are paired (resp., not paired) in M_{cb}^* with intervals in $L \cup R$ are referred as *assigned* (resp., *unassigned*). For any $j \in U$ which is not matched in M_{cb}^* (call these ‘free’), there is no unassigned interval of M with which j is disjoint (otherwise, M_{cb}^* would not be optimal); therefore j need not be considered any further. Furthermore, any ‘balancing’ step (e.g., moving an interval from L to R) will not unassign an interval of M with which a ‘free’ interval could be paired because this also implies that M_{cb}^* is not optimal. The convex bipartite matching algorithm (the algorithms [1] and [12] and Lemma 2.2) that we use assigns matches using the lowest indexed vertices of B first. Therefore, if $k_l > k_r$, we must determine whether any of $l \in L$ can match right instead (so as to make $k_l = k_r$). If $k_r > k_l$, knowing that no interval in R can be ‘moved’ to L because of the way matches are made by the algorithm in Lemma 2.2 and [1] and [12], we proceed to compute the set C that contains the intervals to the left of V which are candidates to match with intervals on the same side of V (Lemma 3.3). If $k_l = k_r$, then the size of the optimal matching is known and we proceed with creating the representation of the optimal matching M^* .

If $k_l > k_r$, then we first attempt to ‘balance’ the sizes of L and R . To determine the subset of intervals of L which can also match right, we perform a matching between the intervals in $L \cup R$ and the intervals in Y . In this matching, $j \in L \cup R$ can be matched with $i \in Y$ if $r(j) < l(i)$. Thus, we can transform this into an instance of a red-blue matching. We color the right endpoints of the intervals in $L \cup R$ red and the

left endpoints of the intervals in Y blue, and then perform a red-blue matching. Let M_{RB}^* be the maximum red-blue matching so resulted. If $|M_{RB}^*| > k_r$, then some intervals of L which formerly matched left are now matched right. When M_{RB}^* is computed, it does not necessarily include all the intervals of R . However, as shown in Lemma 3.5, we can convert M_{RB}^* to M_{RB}' (both optimal) by performing another red-blue matching with the red points being the right endpoints of those intervals of R not in M_{RB}^* and the blue points being the right endpoints of the intervals of L which are in M_{RB}^* . If $|M_{RB}'| - k_r \geq \lfloor (k_l - k_r)/2 \rfloor$, then L and R can be made equal in size by deleting $\lfloor (k_l - k_r)/2 \rfloor$ intervals of L that match in M_{RB}' from L and adding them to R . Otherwise ($|M_{RB}'| - k_r < \lfloor (k_l - k_r)/2 \rfloor$), we move all $l \in L$ in M_{RB}' from L to R and still have $k_l > k_r$. As stated earlier, when computing type-4 pairs, we want $k_l - k_r$ to be a multiple of 2, so that if it is not, we delete the interval from L with its right endpoint closest to V . Note that since we are using the floor function, if $k_l - k_r = 1$, then that interval in L need not be included in the optimal matching M^* . If $k_l = k_r$, then the size of the optimal matching is known and we proceed with creating the representation of the optimal matching. If it is still the case of $k_l > k_r$, then we next compute, based on Lemma 3.3, the set C of intervals (for obtaining type-4 pairs from intervals to the right of V). We find the unique interval $j \in L$ such that $r(j)$ is closest to V . As shown in Lemma 3.3, there exists an optimal matching in I such that its type-4 pairs consist of only the intervals of Y whose left endpoints are to the left of $r(j)$. However, because some of these intervals may already be assigned to cut intervals in $L \cup R$, we must include more intervals in C when computing M_c^* . The intervals to be included in C are as follows:

1. All unassigned intervals $y \in Y$. Clearly, at this stage, there is no unassigned interval of Y which is to the right of $r(j)$ (otherwise j would have been made to ‘match right’ in the balancing step).
2. All the matched pairs $(r, \text{mate}_{M_{RB}'}(r))$ of M_{RB}' such that $r \in R$ and $l(\text{mate}_{M_{RB}'}(r)) < r(j)$. Let R' denote this set of matched pairs so obtained from M_{RB}' . The pairs R' are required because a swap with previously unassigned intervals may occur, thus creating a type-4 pair. In the example of Figure 3, clearly we can perform a swap to generate new pairs (r, u_2) and $(\text{mate}_{M_{RB}'}(r), u_1)$.

Any assigned interval $y \in Y$ such that $r(j) < l(y)$ need not be included in C . This is because such an interval y can only be included in a type-4 pair at the expense of making its former mate (say, $r \in R$) unpaired. This will not increase the size of the matching.

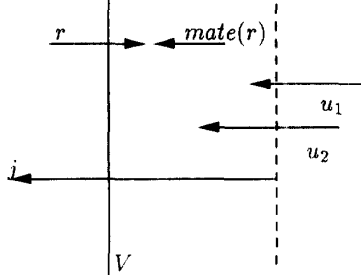


Figure 3: An example for inclusion of $r \in R$.

To show this, consider the following two cases.

1. r does not overlap with some unassigned interval $y' \in Y$. This situation cannot exist because it implies that we can increase the size of M_{RB}^* by pairing (r, y') and (j, y) , violating the optimality of M_{RB}^* .
2. r overlaps with all the unassigned intervals in Y . If y is paired with an unassigned interval, then r is free as there is no other unassigned interval with which it can be paired. Thus, we do not increase the size of the matching by including y in C .

For the interval set C , we compute (recursively) the maximum matching M_c^* between disjoint intervals. Clearly, $|M_c^*| \geq |R'|$. However, not every interval $r \in R \cap R'$ is necessarily included in M_c^* . We will show next that it is necessary to retain in the matching the cut intervals in $R \cap R'$.

We must be careful not to create type-4 pairs at the expense of removing cut intervals from the set R' since this will not increase the size of the matching. This can be shown as follows: For every cut interval $r \in R'$ which is not in M_c^* , it is possible to put r back to the matching through a swapping operation; furthermore, the swapping can be performed such that the size of M_c^* is unchanged. We claim that there exists a distinct interval of one of the following 2 types, with which r can be swapped:

1. A formerly unassigned interval which is now a left mate of a type-4 pair in M_c^* .
2. An interval which was the right mate of an $r' \in R'$ (r' not necessarily equal to r) and which is now the left mate of a type-4 pair in M_c^* .

See the full paper for a discussion of the various cases where a swap can occur.

Thus given M_c^* , we want to convert it, in parallel, to a new optimal matching $M_c^{*'}$ in C , such that all $r \in R \cap R'$ which are not matched in M_c^* are matched in $M_c^{*'}$. Let $R'' = R \cap R'$. Note that the above case

analysis gives a sequential procedure to convert M_c^* to $M_c^{*'}$. To obtain $M_c^{*'}$ in parallel, we need the following lemma.

Lemma 3.6 *It is possible to convert the optimal matching M_c^* in C to another optimal matching $M_c^{*'}$ in C , such that every interval $r \in R''$ is matched in $M_c^{*'}$, in $O(\log n)$ time on an $O(n/\log n)$ -processor EREW PRAM and on an n -processor hypercube.*

Proof: See the full paper. □

So after computing M_c^* , we perform twice the red-blue matching to obtain $M_c^{*'}$, such that all $r \in R''$ are matched in $M_c^{*'}$. Let m denote the number of type-4 pairs in $M_c^{*'}$. If $m \geq \lfloor (|L| - |R|)/2 \rfloor$, then we take $\lfloor (|L| - |R|)/2 \rfloor$ type-4 pairs from $M_c^{*'}$ and add them to M^* . All other type-4 pairs in $M_c^{*'}$ are ignored (these intervals will be paired with intervals on the other side of V). If $m < \lfloor (|L| - |R|)/2 \rfloor$, we add all the type-4 pairs in $M_c^{*'}$ to M^* and delete $(|L| - |R|) - 2m$ intervals from L . For each $r \in R$, there is a unique interval to the right of V with which it is paired, we add these $|R|$ pairs to M^* . For each $l \in L$, there is a unique interval to the left of V with which it is paired, we add these $|L|$ pairs to M^* . Finally, there are s unassigned intervals remaining on both the left and right sides of V which form s pairs, and we add them to M^* . The matching M^* thus obtained is an optimal matching in the input interval set I . This is because there exists no augmenting path that includes an unmatched interval to the right of V such that the size of the matching can be increased from the size of M^* (otherwise, either $M_c^{*'}$ would have not been optimal in C or M_{cb}^* would have not been optimal in the convex bipartite graph $G = (U, I - U, E)$).

If $k_r > k_l$, then due to the nature of the convex bipartite matching algorithm (Lemma 2.2), we cannot perform further 'balancing' and hence proceed directly to computing the set C of intervals to the left of V which are candidates for type-4 pairs. The computation for this case is symmetric to the case of $k_l > k_r$.

The formal description of the main steps of our parallel algorithms is now given. The implementation details on the EREW PRAM and on hypercubes are described in the full paper. The algorithms produce a maximum matching of the set of intervals I .

Algorithm G-Disj-Match(I, A, B)

Input: A set $I = \{I_1, I_2, \dots, I_n\}$ of n intervals, each specified by its two endpoints.

Output: A set, A , of first elements of the pairs of an optimal matching M^* in I and a set, B , of second elements of the pairs of the matching M^* , listed in corresponding order.

1. Sort by the x -coordinates of the left and right endpoints in I and relabel the intervals so that

- $i < j$ if $l(i) < l(j)$, if the endpoints are not given already sorted.
2. Compute the location of the middle line V and the set U of intervals cut by V . Let $|I - U| = 2b$, the number of intervals not cut by V .
 3. Obtain the subset X of the intervals to the left of V and maintain them in the order of decreasing right endpoints. Assign these intervals a label from 1 to b such that $i < j$ if $r(i) > r(j)$.
 4. Obtain the subset Y of the intervals to the right of V and maintain them in the order of decreasing left endpoints. Assign these intervals a label from $b + 1$ to $2b$ such that $i < j$ if $l(i) > l(j)$.
 5. Compute for each $u \in U$, the range of intervals in $I - U$ (i.e., $X \cup Y$) which are possible candidates for matching, using the labels assigned in Steps 3 and 4. Express this range as $(first, last)$ (i.e., (f_u, l_u)).
 6. Construct a convex bipartite graph $G = (U, I - U, E)$ with the edge set $E = \{(a_i, b_j) \mid a_i \in U; b_j \in I - U, f_{a_i} \leq b_j \leq l_{a_i}\}$, by using the values of f_u and l_u computed in Step 5. Compute a maximum matching M_{cb}^* in G . (M_{cb}^* gives the maximum number of intervals of U that can match left or right.) After the matching, each interval of U that is matched has an interval numbered from 1 to $2b$ assigned. If the assigned interval is in the range $1 \dots b$, then the interval matches left, otherwise it matches right. Let L (resp., R) be the subset of intervals of U that match left (resp., right) in M_{cb}^* . If $|L| = |R|$, then return as M^* the matched pairs from L and R and the matched pairs formed from the intervals in $I - U$. ($l \in L$ goes in B , its mate in A . $r \in R$ goes in A , its mate in B . Each interval in X that is unassigned goes in A ; each interval in Y that is unassigned goes in B). Else continue.
 7. If $|L| > |R|$, let $Red = \{r(j) \mid j \in (L \cup R)\}$, $Blue = \{l(j) \mid j \in Y\}$. Perform a red-blue matching to see how many intervals of L can also match right. Let M_{RB}^* denote the resulting maximum red-blue matching. For each red point ($r \in L \cup R$) in M_{RB}^* , we have a blue point ($b \in Y$) assigned. If not every $r \in R$ is included in M_{RB}^* , then we do another red-blue matching with $Red = \{r(j) \mid j \in R \text{ and } j \notin M_{RB}^*\}$ and $Blue = \{r(j) \mid j \in L \text{ and } j \in M_{RB}^*\}$ to transform it (Lemma 3.5) to M_{RB}^* . If $|M_{RB}^*| - |R| \geq \lfloor (|L| - |R|)/2 \rfloor$, then we add $\lfloor (|L| - |R|)/2 \rfloor$ intervals of L that are in M_{RB}^* to R and remove them from L . If $|L| - |R| = 1$, then delete one interval from the larger set (since the two sets L and R must be equal in size). At this point, $|L| = |R|$. M^* is obtained as follows.

Every interval in R goes in A , its mate goes in B . Each interval in L goes in B , its mate goes in A . Each unassigned interval in X goes in A ; each unassigned interval in Y goes in B . Return. Else continue (with $|L| \neq |R|$).
 8. If $|M_{RB}^*| - |R| < \lfloor (|L| - |R|)/2 \rfloor$, then add $|M_{RB}^*| - |R|$ intervals of L that are in M_{RB}^* to R and delete them from L .
 9. At this point, either $|L| > |R|$ or $|R| > |L|$. If $|L| - |R|$ is an odd number, then delete one interval (whose left or right endpoint is closest to V) from the larger set. If $|R| > |L|$, then we find $j \in R$, whose $l(j)$ is closest to V . Let $z = l(j)$. Compute the set, C , of intervals from X whose right endpoints are to the right of z (i.e., $r(c) > z, c \in X$). If $mate_{M_{RB}^*}(c) \in L$, then we also include $mate_{M_{RB}^*}(c)$ in C . Let L' denote the intervals of L in C . Similarly, if $|L| > |R|$, we find $j \in L$ whose $r(j)$ is closest to V . We let $z = r(j)$, and compute the set C such that $l(c) < z, c \in Y$. If $mate_{M_{RB}^*}(c) \in R$, then we also include $mate_{M_{RB}^*}(c)$ in C . Let R' denote the intervals of R in C .
 10. Let $M_c^* = \mathbf{G-Disj-Match}(C, A', B')$.
 11. If $|L| > |R|$, then examine the intervals in A' to see if all the intervals $r \in R \cap R'$ have been retained. If not, convert the optimal matching M_c^* in C to another optimal matching $M_c^{*'} in C , such that all the intervals of $R \cap R'$ are matched in $M_c^{*'}$ (by using Lemma 3.6). Update A' according to the pairs in $M_c^{*'}$.$
 12. If $|R| > |L|$, then examine the intervals in B' to see if all the intervals $l \in L \cap L'$ have been retained. If not, convert the optimal matching M_c^* in C to another optimal matching $M_c^{*'}$ in C , such that all the intervals of $L \cap L'$ are matched in $M_c^{*'}$ (by using Lemma 3.6). Update B' according to the pairs in $M_c^{*'}$.
 13. Identify the type-4 pairs in (A', B') and delete all other pairs (which are pairs involving intervals in R or L).
 14. Let $w = \lfloor (|L| - |R|)/2 \rfloor$. If $|A'| < w$, then remove intervals from R (resp., L) whichever is larger so that $|R| = |L| + 2|A'|$ (resp., $|L| = |R| + 2|A'|$). If $|A'| \geq w$, then delete $(|A'| - w)$ pairs from (A', B') . M^* is obtained as follows. Return the remaining pairs in (A', B') and the matched pairs from R and L . (An interval in L goes in B , its mate in A . An interval in R goes in A , its mate in B . Each interval in A' goes in A ; each interval in B' goes in B .) Each interval in X that is

unassigned goes in A ; each interval in Y that is unassigned goes in B . Return.

Theorem 3.1 *Given a set I of n intervals, algorithm **G-Disj-Match** solves the maximum matching problem between disjoint intervals in $O(\log^3 n)$ time using $O(n/\log^2 n)$ processors on the EREW PRAM and using n processors on the hypercubes.*

Proof: The correctness and time and processor complexities follow from the descriptions presented in the full paper. \square

References

- [1] M. G. Andrews, "Efficient Parallel Graph Algorithms on the Hypercube Network Model," Ph.D. Thesis, Northwestern University, December 1992.
- [2] M. G. Andrews, M. J. Atallah, D. Z. Chen, and D. T. Lee, "Parallel Algorithms for Maximum Matching in Interval Graphs," private manuscript (1995).
- [3] M. G. Andrews and D. T. Lee, "An Optimal Algorithm for Matching in Interval Graphs," private manuscript (1992).
- [4] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, "Highly Parallelizable Problems," *Proc. 21st Annual ACM Symp. on Theory of Computing*, 1989, pp. 309–319.
- [5] G. Bilardi and A. Nicolau, "Adaptive Bitonic Sorting: An Optimal Parallel Algorithm for Shared-Memory Machines," *SIAM J. Comput.*, 18 (1989), pp. 216–228.
- [6] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," *J. of the ACM*, 21 (2) (1974), pp. 201–206.
- [7] D. Z. Chen, "Efficient Parallel Binary Search on Sorted Arrays, with Applications," to appear in *IEEE Trans. on Parallel and Distributed Systems*.
- [8] D. Z. Chen, "Optimal Hypercube Algorithms for Triangulating Classes of Polygons and Related Problems," *Proc. 7th International Conf. on Parallel and Distributed Computing Systems*, Las Vegas, Nevada, 1994, pp. 174–179.
- [9] R. Cole, "Parallel Merge Sort," *SIAM J. Computing*, 17 (1988), pp. 770–785.
- [10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.
- [11] R. Cypher and C. G. Plaxton, "Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers," *Proc. 22nd Annual ACM Symp. on Theory of Computing*, 1990, pp. 193–203.
- [12] E. Dekel and S. Sahni, "A Parallel Matching Algorithm for Convex Bipartite Graphs," in *Proc. International Conference on Parallel Processing*, 1982, pp. 178–184.
- [13] H. N. Gabow and R. E. Tarjan, "A Linear-Time Algorithm for a Special Case of Disjoint Set Union," *Journal of Computer and System Sciences*, 30 (1985), pp. 209–221.
- [14] G. Gallo, "An $O(n \log n)$ Algorithm for the Convex Bipartite Matching Problem," *Operations Research Letters*, 3 (1) (1984), pp. 31–34.
- [15] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
- [16] T. Hagerup and C. Rub, "Optimal Merging and Sorting on the EREW PRAM," *Information Processing Letters*, 33 (1989), pp. 181–185.
- [17] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
- [18] S. K. Kim, "Optimal Parallel Algorithms on Sorted Intervals," *27th Annual Allerton Conference on Communication, Control, and Computing*, 1989, pp. 766–775.
- [19] D. Kravets and C. G. Plaxton, "Optimal Hypercube Algorithm for the All-Nearest Smaller Values Problem," to appear in the *6th IEEE Symp. on Parallel and Distributed Processing*, Dallas, 1994.
- [20] C. P. Kruskal, L. Rudolph, and M. Snir, "The Power of Parallel Prefix," *IEEE Trans. on Computers*, C-34 (10) (1985), pp. 965–968.
- [21] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *J. of the ACM*, 27 (1980), pp. 831–838.
- [22] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.
- [23] W. Lipski, Jr. and F. P. Preparata, "Efficient Algorithms for Finding Maximum Matchings in Convex Bipartite Graphs and Related Problems," *Acta Informatica*, 15 (1981), pp. 329–346.
- [24] A. Moitra and R. C. Johnson, "A Parallel Algorithm for Maximum Matching in Interval Graphs," *Proc. Int. Conf. Parallel Processing*, 1989, Vol. III, pp. 114–120.