CERIAS Tech Report 2003-16

PARALLEL ALGORITHMS FOR MAXIMUM MATCHING IN COMPLEMENTS OF INTERVAL GRAPHS AND RELATED PROBLEMS

by M. G. Andrews, M. J. Atallah,
D. Z. Chen, and D. T. Lee

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

Algorithmica (2000) 26: 263–289
DOI: 10.1007/s004539910013

Algorithmica

Parallel Algorithms for Maximum Matching in Complements of Interval Graphs and Related Problems[1]

M. G. Andrews,[2] M. J. Atallah,[3] D. Z. Chen,[4] and D. T. Lee[5]

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. In this paper 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 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 the proper interval 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. In addition, we present an improved parallel algorithm for maximum matching between overlapping intervals in proper interval graphs.

Key Words. Parallel algorithms, Maximum matching problems, Interval graphs, Complement graphs, EREW PRAM, Hypercubes.

1. Introduction. Consider a set of intervals, $I = \{I_1, I_2, \ldots, I_n\}$, on the x-axis, where interval $I_i = [le(i), re(i)]$ is specified by its two endpoints: the left endpoint, $le(i)$, and the right endpoint, $re(i)$, with $le(i) < re(i)$. Two intervals $I_i = [le(i), re(i)]$ and $I_j = [le(j), re(j)]$ are *disjoint* (to each other) if $re(i) < le(j)$ or $re(j) < le(i)$; 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 such 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

[1] Part of this research was done while the authors were visiting the Leonardo Fibonacci Institute in Trento, Italy, in the summer of 1992. The second author's research was supported in part by the National Science Foundation under Grant CCR-9202807 and by the sponsors of the COAST Laboratory. The third author's research was supported in part by the National Science Foundation under Grant CCR-9623585. The fourth author's research was supported in part by the National Science Foundation under Grants CCR-8901815, CCR-9309743, and CDA-9703228, and by the Office of Naval Research under Grants No. N00014-95-1-1007 and No. N00014-97-1-0514. The fourth author is on leave from the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, USA.
[2] AT&T Bell Laboratories, 2000 N. Naperville Road, Naperville, IL 60566, USA. m.g.andrews@att.com.
[3] Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA. mja@cs.purdue.edu.
[4] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. chen@cse.nd.edu.
[5] Institute of Information Science, Academia Sinica, Nankang, Taiwan 11529. dtlee@iis.sinica.edu.tw.

Received November 20, 1995; revised September 3, 1998. Communicated by M. Y. Kao.

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 [17]. 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 $[le(i), re(i)]$, and vertex i (corresponding to interval i) interchangeably.

A *matching* in a graph G is a subset M of the 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 [12]. However, no deterministic parallel algorithm for computing maximum matchings in general graphs is known that takes polylogarithmic time using a polynomial number of processors [19].

In this paper we consider the following matching problem on a set I of n intervals: Find a maximum cardinality matching M in 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. An $O(n \log n)$ time sequential algorithm for this matching problem was given by Andrews and Lee [3]. A related problem on matching in interval graphs was considered by Moitra and Johnson [26], who gave 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 previously known efficient parallel algorithm (i.e., in polylogarithmic time using a polynomial number of processors) for finding maximum matchings in the complement graphs of interval graphs. Here we study the graph-theoretical problems of computing in parallel maximum cardinality matchings in the complement graphs of interval graphs and interval graphs, provided that their interval models are already available.

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 algorithms compute a maximum matching in $O(\log^3 n)$ time using $O(n/\log^2 n)$ processors on the EREW PRAM and using 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 the proper interval case takes $O(\log n \log\log n)$ time for unsorted input and $O(\log n)$ time for sorted input. The approaches of 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 for arbitrary intervals. Furthermore, if the endpoints of the input intervals are given sorted, we can make our sequential algorithm for computing maximum matchings among arbitrary disjoint intervals run in linear time, as follows: A key subproblem in our parallel algorithms for computing maximum matchings among disjoint intervals is that of finding maximum matchings in convex bipartite graphs, which can be solved sequentially in linear time [15], [25]; by using the optimal sequential algorithm for computing maximum matchings in convex bipartite graphs [15], [25] and by simulating sequentially the rest of our

EREW PRAM algorithm, a maximum matching among arbitrary 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 takes $O(n \log n)$ 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 CREW PRAM algorithm for this problem [26] by a factor of $n/\log n$.

The computational models we use are the EREW PRAM and hypercubes. The PRAM is a synchronous parallel model in which all processors share a common memory and each processor can access any memory location in constant time [19]. The EREW PRAM does not allow more than one processor to access the same memory address simultaneously. We also refer to the CREW PRAM model, which allows simultaneous accesses to the same memory location by multiple processors only if all such concurrent accesses are for reading data only. The CREW PRAM is obviously more powerful than the EREW PRAM. Our hypercube model is the standard one: It has n processors, each with $O(1)$ local memory, and with one-port communication. For a detailed discussion of the hypercube model, the reader is referred to the book by Leighton [24].

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 matching problem among arbitrary disjoint intervals. In Section 4 we present parallel algorithms for the matching problem among disjoint proper intervals. Our improved parallel algorithm for the matching problem on overlapping proper intervals is given in Section 5.

2. Preliminaries. The input consists of a set of n intervals $I = \{I_1, I_2, \ldots, I_n\}$. To avoid cluttering the exposition, we assume without loss of generality 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 of 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 [10] and in $O(\log n \log\log n)$ time on n-processor hypercubes [13]. From now on, we assume that the $2n$ endpoints of 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 two endpoints, with the sorted order of the endpoints corresponding to the increasing order of the processor indices. We also assume without loss of generality that the intervals in I have been relabeled such that $i < j$ implies $le(i)$ occurs before $le(j)$ in the sorted array of endpoints. In the case of proper intervals, $i < j$ also implies $re(i)$ occurs before $re(j)$ in the sorted array of endpoints. This relabeling can be easily carried out 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 [22], [23] and in $O(\log n)$ time on n-processor hypercubes [24].

Given a matching M in I, we say that an interval i is *in* M if i is matched by M. We say that an interval *matches left* (resp., *right*), denoted by a left (resp., right) arrow, if it is matched in M with an interval to its left (resp., right). An interval is *free*, denoted by

a circle, if it is unmatched with respect to M. Interval i matching 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 $re(i) < x(L)$ (resp., $le(i) > x(L)$), 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 intersected 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. The fact that the numbers of intervals in I not cut by V lying on each side of V are the same follows immediately from the definition of V. Since a matching can be obtained in such a way that each interval to the left of V is matched with a distinct interval to the right of V, $b \leq |M|$ follows. □

The following problem, defined by Kim [20], 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, suppose that 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 [20] presented 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 an n-processor hypercube, the red–blue matching problem with sorted input can be solved in $O(\log n)$ time, as follows: First apply Kim's reduction [20] to reduce the problem to the *all nearest smaller values problem* [5] (this reduction takes $O(\log n)$ time on the hypercube since it mainly performs parallel prefix); then use the optimal hypercube algorithms by Chen [9] and Kravets and Plaxton [21] to solve the all nearest smaller values problem in $O(\log n)$ time.

Our interval matching algorithms also make use of convex bipartite graphs which are reviewed 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 $(a_1, a_2, \ldots, a_m)$ and $(b_1, b_2, \ldots, b_n)$, and E is the set of edges. An edge $(a, b) \in E$ implies that $a \in A$ and $b \in B$, and furthermore, $(a_i, b_j) \in E$ if and only if $f_i \leq j \leq l_i$, where f_i (resp., l_i) is the index of the first (resp., last) vertex in B to which a_i is connected.

Maximum cardinality matchings in convex bipartite graphs can be computed sequentially either by combining the linear time algorithm of Gabow and Tarjan [15] for a

special union-find problem with the algorithm of Lipski and Preparata [25] for computing maximum matchings in convex bipartite graphs, or by using the $O(n \log n)$ time algorithm of Gallo [16]. Dekel and Sahni [14] presented a parallel algorithm for computing maximum cardinality matchings in convex bipartite graphs in $O(\log^2 n)$ time using $O(n)$ EREW PRAM processors, and Atallah et al. [4] showed that the problem can be solved in $O(\log n)$ time by using $O(n^3)$ EREW PRAM processors.

The next lemma is needed by our PRAM algorithms. It shows that the processor bound of Dekel and Sahni's parallel algorithm [14] can be improved by a factor of $\log n$.

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 [7] to simulate the parallel algorithm of Dekel and Sahni [14] for computing maximum cardinality matchings in convex bipartite graphs. Note that Dekel and Sahni's algorithm [14] computes a maximum cardinality matching in a convex bipartite graph in $O(\log^2 n)$ time using $O(n)$ EREW PRAM processors. The algorithm in [14] consists of two passes, each pass traversing a complete binary tree of n leaves level by level. At each level, the algorithm in [14] essentially performs a constant number of parallel merges, and the number of operations performed by the algorithm at each level is $O(n)$. The total number of operations performed by the algorithm of Dekel and Sahni [14], therefore, is $O(n \log n)$. By applying Brent's theorem [7] and by using an optimal EREW PRAM merge algorithm [6], [8], [18], we can easily simulate the algorithm [14] in $O(\log^2 n)$ time using $O(n/\log n)$ EREW PRAM processors. □

3. Maximum Matching among Arbitrary Disjoint Intervals. Section 3.1 gives the key observations for our matching algorithms. Section 3.2 presents the basic idea and main algorithmic steps for finding a maximum matching between disjoint intervals in general complement interval graphs. Sections 3.3 and 3.4 show the details of our EREW PRAM and hypercube algorithms, respectively.

3.1. *Useful Observations*. Our algorithms for the general case of disjoint interval 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., match right*), *or the matched intervals in M^* that are to the right of V all have left arrows* (*i.e., match left*).

PROOF. Let M be an optimal matching. Suppose that there are k_l (resp., k_r) intervals in M that are to the left (resp., right) of V and have left (resp., right) arrows. Without loss of generality, assume $k_l \leq k_r$. If $k_l = 0$, then we are done ($M^* = M$). So assume $k_l > 0$. Now swap, in their matching pairs, the k_l intervals that are to the left of V and have left arrows with any k_l intervals that are to the right of V and have right arrows. For example,

let i_l (resp., i_r) be an interval that is to the left (resp., right) of V and has a left (resp., right) arrow in M; swapping i_l with i_r means obtaining, from the two pairs $(i_l, mate_M(i_l))$ and $(i_r, mate_M(i_r))$, the two new pairs $(i_r, mate_M(i_l))$ and $(i_l, mate_M(i_r))$. Clearly, such a swapping is always possible. The matching M^* thus obtained has the same size as M, and it is easy to see that no interval to the left of V matches left in M^*. □

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

PROOF. Let M be an optimal matching that has the property stated in Lemma 3.1. Clearly, there cannot be free intervals to both the left and right of V (otherwise, M would have not been an optimal matching). So without loss of generality, assume that there are $k > 0$ free intervals to the left of V. We shall obtain M^* from M.

We first do the following: Remove from our consideration all the intervals to the left of V that match intervals to the right of V in M, and do the same thing on the right side of V. Note that V is still a middle line in the set I' of the remaining intervals thus obtained. Furthermore, there are at least k intervals of I' to the right of V that are all matched in M.

There are two cases to consider.

Case 1: *The matched intervals to the right of V all have left arrows.* There must be at least k such intervals to the right of V, and since they all have left arrows but do not match in M with intervals to the left of V, they must match with intervals cut by V. Swap any k matched intervals cut by V with the k free intervals to the left of V. The resulting matching is such an optimal matching M^*.

Case 2: *Not all the matched intervals to the right of V have left arrows.* Let i' be an interval to the right of V that has a right arrow and matches with i'' in M (obviously, i'' is also to the right of V). Then we claim that k must be 1. Proof of the claim: If $k > 1$, then M would have not been an optimal matching because we could have swapped i' and i'' with two free intervals to the left of V to increase the size of the matching, a contradiction. Since V is the middle line and there are at least two intervals (i.e., i' and i'') to the right of V, there must be an interval i of I' to the left of V such that i matches with an interval j of I'. Since the optimal matching M satisfies Lemma 3.1 and since i' (to the right of V) matches right, all matched intervals of I' to the left of V must match right. Because i does not match in M with any interval to the right of V, j must be an interval cut by V. Let the only free interval to the left of V be f. Perform a swap among $\{f, i, i', i''\}$ so that (f, i') and (i, i'') are paired and j becomes free. The resulting optimal matching is M^*. □

COROLLARY 3.1. *There exists an optimal matching M^* such that either all noncut intervals of I to the left of V match right or vice versa, and such that all noncut intervals are matched.*

PROOF. Follows from Lemmas 3.1 and 3.2. □

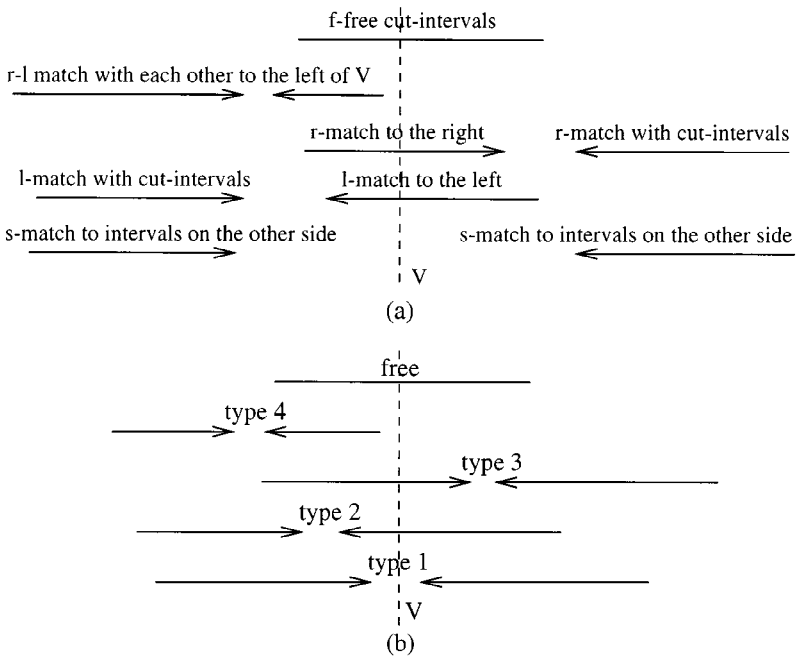

Fig. 1. The form and pair types of an optimal matching as defined in Corollary 3.1.

Consider an optimal matching M^* as defined in Corollary 3.1. For all the cut-intervals, some of them match left (let the number of these be l), some of them match right (let the number of these be r), and maybe some of them are free (let the number of these be f). Without loss of generality, assume that all the noncut 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 in M^*. 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(a), and M^* consists of four different types of matched pairs as illustrated in Figure 1(b).

1. Type 1 matches an interval to the left of V with one to the right of V.
2. Type 2 matches an interval to the left of V with an interval cut by V.
3. Type 3 matches an interval to the right of V with an interval cut by V.
4. Type 4 matches two intervals on the same side of V.

From Lemma 2.1, we know that the number b of intervals on each side of V is the same. We number the intervals to the left (resp., right) of V from 1 to b (resp., from $b+1$ to $2b$) by the *decreasing* x-coordinates of the *right* (resp., *left*) endpoints and store them in an array X (resp., Y). See Figure 2 for an example. Let U denote the set of intervals cut by V. Consider an interval $u \in U$. The set of possible candidates for u to "match left" is the subset of intervals $k, k+1, \ldots, b$ of X, where k is the interval such that $re(k) < le(u)$ and $re(k)$ is closest to $le(u)$ from the left (if $le(u) \leq re(b)$, then this set is empty). Similarly, the

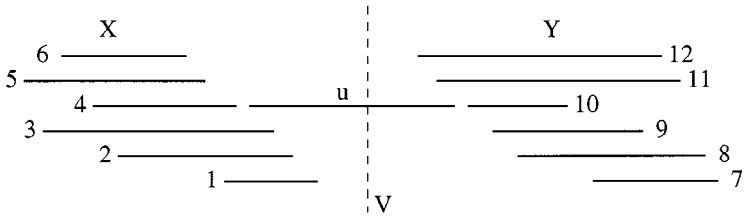

Fig. 2. The candidates for matching with an interval $u \in U$, with $(f_u, l_u) = (4, 10)$.

set of possible candidates for u to "match right" is a subset of intervals $b+1, b+2, \ldots, l$ of Y, where l is the interval such that $re(u) < le(l)$ and $le(l)$ is closest to $re(u)$ from the right (if $re(u) \geq le(b+1)$, then this set is empty). We can combine these two sets for u into a single range of intervals $(k, k+1, \ldots, b, b+1, \ldots, l)$, and represent this range as (f_u, l_u), with $f_u = k$ and $l_u = l$. Note that this representation of all possible matching pairs between U and $X \cup Y = I - U$ is that of a convex bipartite graph.

To determine the intervals in U which are used by an optimal matching M^* as defined in Corollary 3.1, we construct a convex bipartite graph $G = (A, B, E)$ such that $A = U$, $B = X \cup Y$, and for any $u \in A = U$, the edge $(u, b_u) \in E$ if and only if $f_u \leq b_u \leq l_u$. As shown by the next lemma, a maximum cardinality matching M^*_{cb} of G is useful to finding M^*.

LEMMA 3.3. *Let M^*_{cb} be any given maximum cardinality matching of the convex bipartite graph $G = (U, X \cup Y, E)$. Then there is an optimal matching M^* in I as defined in Corollary* 3.1 *that uses only the intervals in M^*_{cb} and $X \cup Y$. That is, for any interval $u \in U$ not matched in M^*_{cb}, there is an optimal matching M^* as defined in Corollary* 3.1 *such that u is not matched in M^*.*

PROOF. Let M' be an optimal matching in I as defined in Corollary 3.1. Consider any interval $u \in U$ such that u is not matched in M^*_{cb} but is matched in M'. We want to show that there is another optimal matching M^* in I as defined in Corollary 3.1 such that u is not matched in M^* (and hence u can be ignored in computing such an optimal matching M^*).

Let M'' be a collection of edges containing the interval pairs (a, b) such that either $(a, b) \in M^*_{cb}$ or $(a, b) \in M'$ but $(a, b) \notin M^*_{cb} \cap M'$. Consider the subgraph $G_{M''}(I)$ on I that is induced by M'' (i.e., the vertices of $G_{M''}(I)$ are the intervals of I in M'' and the edges of $G_{M''}(I)$ are those in M''). Then it is easy to see that $G_{M''}(I)$ consists of a set of connected components each of which is either a path or an even-length cycle whose edges are alternating with respect to M^*_{cb} and M' (the *length* of a path is the number of edges on it). In particular, u is a vertex of $G_{M''}(I)$, and further, u is an end-vertex of such a path in $G_{M''}(I)$ (since u is not matched in M^*_{cb} and is matched in M'). Denote that path in $G_{M''}(I)$ starting at u by P_u. Figure 3 gives an example of P_u, where the edges of P_u in M^*_{cb} (resp., M') are dotted (resp., solid), the arrow on each edge indicates the left or right matching of a noncut interval in M^*_{cb} or M', and the cut intervals (resp., noncut intervals) are denoted by solid (resp., open) circles.

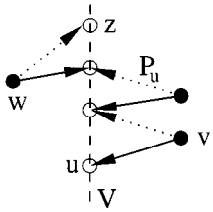

Fig. 3. Illustrating the proof of Lemma 3.3.

Note that, like u, the other end-vertex z of the path P_u is matched in either M^*_{cb} or M' but not both. We now show first that the length of P_u must be even. Suppose this is not the case (i.e., the length of P_u is odd). Then we can exchange in M^*_{cb} the edges of $M^*_{cb} \cap P_u$ with the edges of $M' \cap P_u$, thus increasing the size of M^*_{cb} by one, a contradiction to the optimality of M^*_{cb} in the convex bipartite graph $G = (U, X \cup Y, E)$. In Figure 3 this would happen if w (instead of z) were the other end-vertex of P_u. Now that the length of P_u is even (as in Figure 3), we can then exchange in M' the edges of $M' \cap P_u$ with the edges of $M^*_{cb} \cap P_u$, obtaining another optimal matching M^* in I. Note that in M^*, all noncut intervals are matched and the matching direction of each noncut interval on P_u remains the same as in M'. Hence the form of M^* is as defined in Corollary 3.1. Further, u is clearly not matched in M^*. □

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 matching M_{RB} of the red points to the blue points. We then include more red points (S'_R) and run 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 3.4. *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. Let b_r be the blue point which is matched to r in M_{RB}. By definition, $x(r) < x(b_r)$. Based on Kim's parallel algorithm for red–blue matching [20], we can define a *canonical form* of the optimal matching M_{RB} obtained by that algorithm, as follows.

- For any red–blue pair (r, b) in M_{RB}, there is no unassigned (unmatched) point $q \in S_R \cup S_B$ such that $x(r) < x(q) < x(b)$.

This is because the parallel algorithm [20] simulates the sequential algorithm which scans the points by the increasing x-coordinates. When a red point is encountered, it is pushed onto a stack. When a blue point is encountered, a red point (if any) is removed from the stack [20].

Let r' be the red point matched to b_r in M'_{RB}. There are two cases to consider.

1. $r' \in S_{\mathrm{R}}$. Because of the canonical form of M'_{RB}, we know $x(r') > x(r)$. Thus, $x(b_{r'}) > x(r)$, where $b_{r'}$ is the mate of r' in M_{RB}. Now, $b_{r'}$ must be in M'_{RB} (otherwise, the matching M'_{RB} would not be optimal since we could then add one more matched pair $(r, b_{r'})$ to M'_{RB}). There are two subcases to consider here.
   (a) $b_{r'}$ is matched to $r'' \in S'_{\mathrm{R}}$. In this case, $x(r'') > x(r)$, else the canonical form of M'_{RB} is violated.
   (b) $b_{r'}$ is matched to $r'' \in S_{\mathrm{R}}$. Then we repeat the above argument (for finitely many times) until we get a blue point which is matched to $j \in S'_{\mathrm{R}}$ such that $x(j) > x(r)$.
   Let $p(r) \in S'_{\mathrm{R}}$ be the red point determined by the above two subcases.
2. $r' \in S'_{\mathrm{R}}$. Because of the canonical form of M'_{RB}, we know $x(r') > x(r)$. Let $p(r) = r'$.

Applying repeatedly the argument given above to every point $r \in S_R$ which is matched in M_{RB} but not in M'_{RB}, to find the distinct point $p(r) \in S'_{\mathrm{R}}$, the lemma is then proved. □

Therefore, to convert M'_{RB} to M''_{RB} such that every red point $r \in S_R$ which is matched in M_{RB} is also matched in M''_{RB}, we need to find, for every r which is matched in M_{RB} but unmatched in M'_{RB}, a corresponding point $p(r) \in S'_{\mathrm{R}}$ that is matched in M'_{RB} and is to the right of r. To perform this conversion in parallel for all $r \in (S_{\mathrm{R}} \cap M_{\mathrm{RB}}) - M''_{\mathrm{RB}}$, we use red–blue matching. There are two sets of points, (i) $r \in S_{\mathrm{R}}$ such that $r \in M_{\mathrm{RB}}$ but $r \notin M'_{\mathrm{RB}}$, and (ii) $r' \in S'_{\mathrm{R}}$ such that $r' \in M'_{\mathrm{RB}}$. 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. *Matching Algorithm among Arbitrary Disjoint Intervals.* We begin with a general discussion of the basic idea of our algorithms, and then describe the main algorithmic steps.

3.2.1. *Basic Idea.* Let M^* be an optimal matching in I in the form defined by Corollary 3.1. Then all intervals not cut by V are in M^*. Hence we can start with an initial matching M which consists of b matched pairs formed by matching each interval to the left of V with a distinct interval to the right of V. (Of course, 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 (i.e., all cut intervals) with respect to M.

From Lemma 3.2, we know that all free intervals with respect to M^* are cut by V. Thus, it is not hard to see that it will yield an optimal matching in the form defined by Corollary 3.1 if we achieve the following optimality criterion:

OPTIMALITY CRITERION. Matching the maximum number of cut-intervals (i.e., in the set U) while simultaneously retaining all noncut intervals in the matching.

Based on the optimality criterion, we can increase the cardinality of the matching (from the size of M) by matching the intervals of U in one of the following two ways.

1. For an interval $j \in U$ which can match left (resp., right), find another $j' \in U$ which can match right (resp., left).
2. For two intervals $j, j' \in U$ both of which can match left (resp., right), find a type-4 interval pair (see Figure 1(b)) 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 would belong to M^* if we are able to find two intervals to 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 with any interval to the right of V).

Observe that the above two ways correspond to the two basic types of augmenting paths with respect to M that begin and end with free intervals all of which are in U.

To determine from the intervals of U the candidate intervals for M^*, we compute a maximum matching between the intervals of U and the intervals of M (i.e., a maximum convex bipartite matching), and let U' denote the subset of intervals of U involved in this maximum matching. Note that U' is a maximum subset of the intervals in U that can be matched with those in M, and by Lemma 3.3, all intervals in $U - U'$ need not be considered for M^*. In this matching between U and M, every interval in U' is paired with a distinct interval in M. We partition U' into two subsets: L (resp., R), the subset of intervals of U' which are paired with some intervals of M to the left (resp., right) of V. We refer to the intervals in M not paired with any interval in $L \cup R$ as *unassigned*. Observe that if L and R are equal in size, then we are done. This is because the matching formed by the intervals of $L \cup R$ with their mates in M and by the pairs of unassigned intervals in M from both sides of V is optimal based on the optimality criterion. If L and R are not equal in size, say $|L| > |R|$, then there are two cases to consider:

(1) The sizes of L and R can be made equal (by doing some adjustment).
(2) The sizes of L and R cannot be made equal.

If (say) $|L| > |R|$, then we first attempt to make the sizes of the two sets equal. This is done by finding from the larger set L the intervals which can be paired with unassigned intervals of M to the right of V (i.e., for such intervals of L, change the direction of their matching). If the sizes of L and R can be made equal, then clearly we have an optimal matching by the optimality criterion.

If $|L| > |R|$ and L and R cannot be made equal in size, then we need to match the maximum number of intervals of L (this in consequence will match the maximum number of intervals of U' and hence will yield an optimal matching M^*). Note that if a matching in the form defined in Corollary 3.1 were obtained straightforwardly from M and the current versions of L and R, then $|L| - |R|$ intervals of L would have to be excluded from the matching. We include as many intervals of L in M^* as possible, by

doing the following:

(i) Move as many intervals as possible from L to R (to minimize $|L|-|R|$), by changing the matching direction of these intervals. Note that after this step, the resulted convex bipartite matching between U and M is still optimal and hence Lemma 3.3 applies to this setting.
(ii) Find type-4 pairs from the intervals of M to the right of V such that these type-4 pairs do not take away any matches with R.

Note that finding type-4 pairs on the left side of V will not help match more intervals of L. Hence, it is safe to assume that in M^*, all the intervals to the left of V match right (Corollary 3.1).

We need to find at most $\lfloor(|L|-|R|)/2\rfloor$ type-4 pairs from the intervals to the right of V. Thus when $|L|-|R|$ is not a multiple of 2, one interval of L will necessarily be excluded from M^*. We form a set of intervals which help find such type-4 pairs to the right of V. Let $C = R \cup Y$ be this interval set, where Y is the set of intervals to the right of V. For the set C, we compute (recursively) its optimal matching M_c^*, and then retain in M_c^* all the intervals of R (because the type-4 pairs of Y should not be made at the price of reducing the matches with R). As to be shown by Lemmas 3.6 and 3.7, the optimal matching M_c^* in C thus obtained contains the maximum number of type-4 pairs to the right of V for helping match the intervals of $L \cup R$.

Let m be the number of type-4 pairs in the resulted matching 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 of M_c^* to M^*, and delete any $(|L|-|R|)-2m$ intervals from L. For each remaining $l \in L$, there is a distinct 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 distinct 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 (Figure 1(a)) 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 as defined in Corollary 3.1.

3.2.2. *Main Algorithmic Steps*. We now discuss in detail the various algorithmic steps for computing an optimal matching M^* in I in the form defined in Corollary 3.1.

To determine the initial matching M, we sort the $2n$ endpoints of I and find the location of a middle line V. Then by Lemma 2.1, the number b of intervals to the left of V is the same as the number of intervals to the right of V. Identify these intervals and store them in an array of size $2b$ for M. The remaining intervals are all cut by V and are stored in the array U.

Next, we determine, for all 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 (resp., right) of V in an array X (resp., Y) in decreasing order of their right (resp., left) endpoints and assign a new index to each interval from 1 to b (resp., from $b+1$ to $2b$). For each $u \in U$, find from the intervals in the array X (resp., Y) the right (resp., left) endpoint, f_u (resp., l_u), closest to $le(u)$ (resp., $re(u)$) from the left (resp., right). Then by using the new indices $1, 2, \ldots, 2b$, express the intervals in $X \cup Y$ with which u is disjoint as a range (f_u, l_u) of intervals. Construct a convex bipartite graph $G = (A, B, E)$, with $A = U$, $B = M = X \cup Y = I - U$, and

the edge set $E = \{(a_i, b_j) \mid a_i \in U, b_j \in M,\ f_{a_i} \leq b_j \leq l_{a_i},\ f_{a_i}, l_{a_i} \in M\}$. Compute a maximum matching M^*_{cb} in G. As a result, each $u \in U$ which is a candidate for M^* is assigned (according to M^*_{cb}) to a distinct $i \in M$, and all unmatched intervals of U with respect to M^*_{cb} need not be considered any further for M^* (Lemma 3.3). Depending on the side of V on which i lies, 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). The intervals of M which are paired (resp., not paired) in M^*_{cb} with intervals in $L \cup R$ are said to be *assigned* (resp., *unassigned*).

There are three possible cases depending on the values of $|L|$ and $|R|$: (1) $|L| > |R|$ (the most difficult case), (2) $|L| < |R|$ (the second difficult case), and (3) $|L| = |R|$ (the easy case). If $|L| = |R|$, then by the optimality criterion, we know both L and R are in M^*, and can easily create a representation for M^*. Thus, we only need to further discuss the first two cases.

Note that the parallel convex bipartite matching algorithm we use (the algorithm in [14] plus Lemma 2.2) assigns matches using the lowest indexed vertices of B first. Therefore, if $|L| > |R|$, we must determine whether any $l \in L$ can instead be made match right in a maximum matching in G (so as to make $|L| = |R|$). If $|L| < |R|$, knowing that no interval in R can be "moved" to L because of the way matches are made by the algorithm for Lemma 2.2, we proceed to computing the set C that contains the intervals to the left of V which are candidates for type-4 pairs.

Clearly, when $|L|+|R|$ is an odd integer, it is impossible to make $|L| = |R|$. Hence in this situation, at least one interval of $L \cup R$ must be excluded from the optimal matching M^* (as defined in Corollary 3.1). Furthermore, when $|L|-|R| = 1$ or $|R|-|L| = 1$, the size of M^* is known (since one interval from the larger set does not belong to M^*). Thus we can easily create a representation for M^* (by ignoring an arbitrary interval in the larger set). Henceforth, we, without loss of generality, assume $|L| - |R| \notin \{-1, 0, 1\}$, even after the attempt of balancing the sizes of L and R.

If $|L| > |R|$, then we first try to "balance" the sizes of L and R. To determine the maximum subset of intervals of L which can be made match right, we compute a maximum 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 and only if $re(j) < le(i)$. Thus, we can transform this into an instance of 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 matching between $L \cup R$ and Y thus obtained. If $|M^*_{\mathrm{RB}}| > |R|$, then some intervals of L which formerly matched left in M^*_{cb} are now matched right. However, M^*_{RB} need not include all the intervals of R. Fortunately, as shown in Lemma 3.4, we can convert M^*_{RB} into another optimal matching $M^{*'}_{\mathrm{RB}}$ between $L \cup R$ and Y by performing another red–blue matching (with the red points being the right endpoints of the intervals of $R - M^*_{\mathrm{RB}}$ and the blue points being the right endpoints of the intervals of $L \cap M^*_{\mathrm{RB}}$). If $|M^{*'}_{\mathrm{RB}}| - |R| \geq \lfloor(|L| - |R|)/2\rfloor$, then L and R can be made equal in size (provided that $|L| + |R|$ is an even integer), by moving from L $\lfloor(|L| - |R|)/2\rfloor$ intervals that match in $M^{*'}_{\mathrm{RB}}$ to R; the optimal matching M^* can then be easily obtained from such "balanced" L and R. Otherwise (i.e., $|M^{*'}_{\mathrm{RB}}| - |R| < \lfloor(|L| - |R|)/2\rfloor$), we move all $l \in L$ in $M^{*'}_{\mathrm{RB}}$ from L to R, but after this is done, we still have $|L| > |R|$. Hence, the only way for including more intervals of L in the optimal matching M^* is to look for type-4 pairs from intervals in Y (i.e., those to the right of V).

Assume $|L| > |R|$ still holds after the size-balancing step on L and R. We then form the interval set C for obtaining type-4 pairs from intervals to the right of V, such that these type-4 pairs enable us to match the maximum number of intervals of L in M^*. Let $C = R \cup Y$. Next, we compute recursively a maximum matching M_c^* among disjoint intervals in C. Clearly, $|M_c^*| \geq |R|$. However, possibly not every interval $r \in R$ is included in M_c^*, and it is necessary to retain in an optimal matching $M_c^{*'}$ of C (which we obtain from M_c^*) all the intervals of R.

Before we proceed further, we shall point out that such a maximum matching $M_c^{*'}$ in C contains the needed type-4 pairs in Y that help match the maximum number of intervals of $L \cup R$ in M^* (this will be proved later by Lemmas 3.6 and 3.7). Also, because $|L| > |R|$ and $|X| = |Y|$, $|C| = |R \cup Y| < n/2 = |I|/2$. This is important since the recursive computation of M_c^* in C is then on an interval set of significantly smaller size.

We must be careful not to create type-4 pairs from the intervals in C at the expense of excluding some intervals of R, since this will not increase the matching size in I (based on the optimality criterion). We need to show the following: For every interval $r \in R$ which is not in M_c^*, it is always possible to put r into a matching in C through a swapping operation; furthermore, the swapping can be done in such a way that the size of the resulted matching in C is the same as $|M_c^*|$. In particular, we claim that for every $r \in R - M_c^*$, there exists a distinct interval in M_c^* of one of the following three types, with which r can be swapped:

1. A formerly unassigned interval of Y which is now the left mate of a type-4 pair in M_c^*.
2. An interval which in $M_{RB}^{*'}$ was the right mate of an $r' \in R$ (r' is not necessarily equal to r) and which is now the left mate of a type-4 pair in M_c^*.
3. An interval $r' \in R \cap M_c^*$ (hence, the necessity of swapping is reduced to r').

Figure 4 illustrates the various cases of swapping r. For simplicity, in Figure 4, the notation $m(r)$ is used to denote $mate_{M_{\mathrm{RB}}^{*'}}(r)$. In discussing these cases, we let u denote

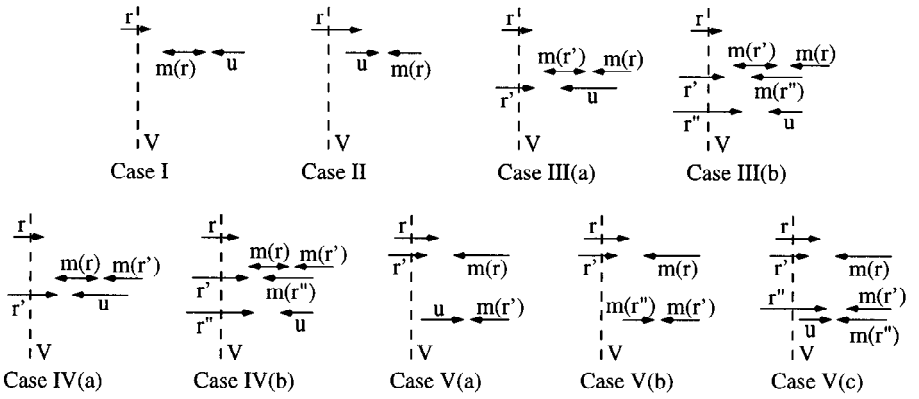

Fig. 4. Illustrating the cases of swapping r with matched pairs in M_c^*.

any formerly unassigned interval of Y; also, we assume that $r, r', r'', r''', \ldots$, are all in R, but $r \notin M_c^*$.

Case I: $mate_{M_{RB}^{*'}}(r)$ *is the left mate of* u *in* M_c^*. Clearly we can swap r and $mate_{M_{RB}^{*'}}(r)$.

Case II: $mate_{M_{RB}^{*'}}(r)$ *is the right mate of* u *in* M_c^*. Clearly we can swap r and u.

Case III: $mate_{M_{RB}^{*'}}(r)$ *is the right mate of* $mate_{M_{RB}^{*'}}(r')$ *in* M_c^*. Then r' must be in M_c^* (otherwise M_c^* is not maximum in C). There are two subcases:

(a) If r' is matched with u in M_c^*, then we can swap r and $mate_{M_{RB}^{*'}}(r')$.
(b) If r' is matched with $mate_{M_{RB}^{*'}}(r'')$ in M_c^*, then r'' must be in M_c^* (otherwise M_c^* is not maximum in C). If r'' is matched with u in M_c^*, then we can swap r and $mate_{M_{RB}^{*'}}(r')$. If r'' is matched with $mate_{M_{RB}^{*'}}(r''')$ in M_c^*, then we use the same argument as before. In any case, r can be swapped with $mate_{M_{RB}^{*'}}(r')$.

Case IV: $mate_{M_{RB}^{*'}}(r)$ *is the left mate of* $mate_{M_{RB}^{*'}}(r')$ *in* M_c^*. Then r' must be in M_c^* (otherwise M_c^* is not maximum in C). There are two subcases:

(a) If r' is matched with u in M_c^*, then we can swap r and $mate_{M_{RB}^{*'}}(r)$.
(b) If r' is matched with $mate_{M_{RB}^{*'}}(r'')$ in M_c^*, then by using the same argument as in Case III(b), r can be swapped with $mate_{M_{RB}^{*'}}(r)$.

Case V: $mate_{M_{RB}^{*'}}(r)$ *is the right mate of* r' *in* M_c^*. Then $mate_{M_{RB}^{*'}}(r')$ must be in M_c^* (otherwise M_c^* is not maximum in C). There are three subcases:

(a) If $mate_{M_{RB}^{*'}}(r')$ is matched with u in M_c^*, then we first swap r and r' to obtain the pair $(r, mate_{M_{RB}^{*'}}(r))$; we then obtain another pair by matching r' with either u or $mate_{M_{RB}^{*'}}(r')$, as in Case I or II.
(b) If $mate_{M_{RB}^{*'}}(r')$ is matched with $mate_{M_{RB}^{*'}}(r'')$ in M_c^* for some $r'' \neq r'$, then r'' must be in M_c^* (otherwise M_c^* is not maximum in C). We first swap r and r' to obtain the pair $(r, mate_{M_{RB}^{*'}}(r))$; we then obtain another pair by matching r' with either $mate_{M_{RB}^{*'}}(r')$ or $mate_{M_{RB}^{*'}}(r'')$, as in Case III or IV.
(c) If $mate_{M_{RB}^{*'}}(r')$ is the right mate of r'' in M_c^* for some $r'' \neq r'$, then $mate_{M_{RB}^{*'}}(r'')$ must be in M_c^* (otherwise M_c^* is not maximum in C). We swap r and r' to obtain the pair $(r, mate_{M_{RB}^{*'}}(r))$. Case V is then applicable to r'.

Thus given M_c^*, we can convert it into another optimal matching $M_c^{*'}$ in C, such that all $r \in R$ are matched in $M_c^{*'}$. Note that the above case analysis immediately gives a sequential procedure for converting M_c^* into $M_c^{*'}$. To obtain $M_c^{*'}$ in parallel straightforwardly based on the above case analysis, one could use a procedure relying on parallel list-ranking [1], [11]. Such a list-ranking-based procedure, although would give an efficient EREW PRAM algorithm, would not lead to a hypercube algorithm as efficient as the one we claimed. Our parallel algorithms avoid using list-ranking and instead are based on the following lemma.

LEMMA 3.5. *It is possible to convert the optimal matching M_c^* in C into 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. We use the following two-step procedure to perform the conversion of M_c^* into $M_c^{*'}$.

(i) For every $r \in R$, color its right endpoint red. Let $R' = R \cap M_c^*$ (i.e., the intervals of R that are matched in M_c^*). For every $r' \in R'$, color the left endpoint of $mate_{M_c^*}(r')$ blue. Note that $R' \subseteq R$ and hence $|R'| \leq |R|$. Perform a red–blue matching on these two sets of colored points, to obtain a matching M_{RB}^t. It is obvious that $|M_{\mathrm{RB}}^t| = |R'|$. Let M_c^t be the matching in C that consists of all type-4 pairs in M_c^* and all pairs in M_{RB}^t. Then M_c^t is an optimal matching in C and there are $|R| - |R'|$ intervals in R still unmatched in M_c^t.

(ii) For all the intervals of R that are unmatched in M_c^t, color their right endpoints as red. For all the right mates of the type-4 pairs in M_c^t, color their left endpoints as blue. Perform a red–blue matching on these two sets of colored points, to obtain a matching M_{RB}^s. Let M_c^s be the matching in C obtained by replacing the type-4 pairs in M_c^t whose right mates occur in M_{RB}^s by the pairs in M_{RB}^s. We claim that the resulting matching M_c^s is the desired optimal matching $M_c^{*'}$ in C.

It is clear that we only need to show the correctness of step (ii). First, note that the case analysis preceding this lemma is applicable to the optimal matching M_c^t in C obtained in step (i). Also, observe that the matching M_{RB}^t, obtained by a red–blue matching [20] among the right endpoints of R (red) and the left endpoints of the right mates of the intervals of R' in M_c^* (blue), has the canonical form as defined in the proof of Lemma 3.4.

We now claim that every interval r of R that is not matched in M_c^t can be matched with the right mate of a distinct type-4 pair in M_c^t.

PROOF OF THE CLAIM. Let $r \in R$ be an unmatched interval in M_c^t. Then in M_c^t, when Case I, II, III, or IV holds for r, clearly the claim is true. When Case V holds for r, $mate_{M_{\mathrm{RB}}^{*'}}(r)$ is the right mate of r' in M_c^t for some $r' \in R$. By the canonical form of the red–blue matching for M_{RB}^t [20], we have $re(r) < re(r')$, and this implies that $re(r) < le(mate_{M_{\mathrm{RB}}^{*'}}(r'))$, as shown in Figure 5(a). (The notation of Figure 5 is the same as Figure 4.) Hence, if Case V(a) or V(b) holds for r, then r certainly can be swapped with the left mate of $mate_{M_{\mathrm{RB}}^{*'}}(r')$ in M_c^t (e.g., Figure 5(b)). If Case V(c) holds for r, then $mate_{M_{\mathrm{RB}}^{*'}}(r')$ is the right mate of r'' in M_c^t for some $r'' \in R$. From $re(r) < le(mate_{M_{\mathrm{RB}}^{*'}}(r'))$ and the canonical form of the red–blue matching for M_{RB}^t, we have $re(r) < re(r'')$ and $re(r) < le(mate_{M_{\mathrm{RB}}^{*'}}(r''))$ (see Figure 5(c)). By inductively using this argument, the claim is proved. □

The claim proved above implies that $M_c^{*'}$ $(= M_c^s)$ thus obtained includes R and is optimal in C. Since the endpoints of all the intervals are sorted, the parallel complexity bounds of the above conversion procedure are the same as those of the parallel algorithms for red–blue matching that we discuss in Section 2. □

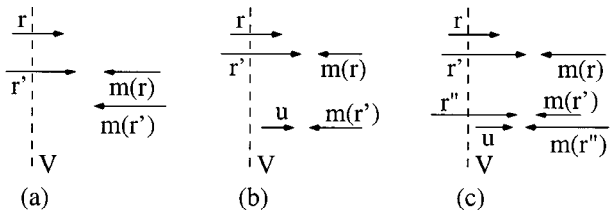

Fig. 5. Illustrating the claim in the proof of Lemma 3.5.

Thus, after computing M_c^*, we perform the red–blue matching twice to obtain $M_c^{*'}$, such that all $r \in R$ are matched in $M_c^{*'}$. We show below that $M_c^{*'}$ indeed contains type-4 interval pairs of Y that help match the maximum number of intervals of $L \cup R$.

LEMMA 3.6. *Let M' be a maximum matching in the interval set $Y \cup L \cup R$. Then $|M'| = |M_c^{*'}|$.*

PROOF. We first prove, by contradiction, that $|M'| \leq |M_c^{*'}|$. Assume that $|M'| > |M_c^{*'}|$ (i.e., the maximum matching M' in $Y \cup L \cup R$ is of a bigger size than the maximum matching $M_c^{*'}$ in $C = R \cup Y$). Recall that $M_c^{*'}$ contains all the intervals of R and R is a maximum subset of intervals of $U' = L \cup R$ that can match with the intervals of Y.

Consider the graph G' defined by the disjoint intervals in $Y \cup L \cup R$. Clearly, $M_c^{*'}$ is a matching in G' (since $C = R \cup Y$ is a subset of $Y \cup L \cup R$). Because $|M'| > |M_c^{*'}|$, by Theorem 9.1 in [27], there is at least one augmenting path in G' with respect to $M_c^{*'}$. Let P be such an augmenting path in G'. Then P begins and ends with free intervals (say, i and j) of G' with respect to $M_c^{*'}$. Note that all interior vertices of P (i.e., the intervals of P that are not at the beginning and end of P) are in $M_c^{*'}$. Also, both i and j are not in R. There are two cases.

(i) *Both i and j are in Y.* Then by replacing those matches of $M_c^{*'}$ in $M_c^{*'} \cap P$ by the matches defined by $P - M_c^{*'}$, the resulted matching $M_c^{*''}$ involves only the intervals in $R \cup Y$ but the size of $M_c^{*''}$ is bigger than $|M_c^{*'}|$ by one, a contradiction to the optimality of $M_c^{*'}$ in $R \cup Y$.

(ii) *At least one of i and j is in L.* Then by replacing those matches of $M_c^{*'}$ in $M_c^{*'} \cap P$ by the matches defined by $P - M_c^{*'}$, the resulted matching $M_c^{*'''}$ in $Y \cup L \cup R$ still contains all the intervals of R (since all interior vertices of P remain in $M_c^{*'''}$). Further, $M_c^{*'''}$ contains at least one more interval (i or j) from L, a contradiction to that R is a maximum subset of intervals of $U' = L \cup R$ that can match with the intervals of Y.

The fact that $|M'| \geq |M_c^{*'}|$ simply follows from that $R \cup Y \subset Y \cup L \cup R$. Hence the lemma is proved. □

LEMMA 3.7. *For any partition of the set U' into subsets L' and R' such that the intervals in L' (resp., R') match with the intervals of X (resp., Y), let $C' = R' \cup Y$ and $M^{*'}_{c'}$ be a maximum matching in C' such that $M^{*'}_{c'}$ contains all the intervals of R'. Then $|M^{*'}_{c'}| \leq |M^{*'}_{c}|$.*

PROOF. Since $C' = R' \cup Y \subset Y \cup U' = Y \cup L \cup R$, we have $|M^{*'}_{c'}| \leq |M'|$, where M' is a maximum matching in $Y \cup U'$. Thus the lemma follows from Lemma 3.6. □

For a partition of U' into L' and R' such that the intervals in L' (resp., R') match with the intervals of X (resp., Y) and such that $|R'| \leq |R|$ (and hence $|L'| \geq |L|$), let $C' = R' \cup Y$ and $M^{*'}_{c'}$ be a maximum matching in C' such that $M^{*'}_{c'}$ contains all the intervals of R'. Further, assume that $M^{*'}_{c'}$ provides the maximum number of type-4 pairs in Y for helping match the intervals of U'. Let $h = |R| - |R'| = |L'| - |L|$. Then there are $h = |L'| - |L|$ more intervals in L' than L that need to match with intervals of X. On the other hand, by Lemma 3.7, $M^{*'}_{c'}$ is of a size $\leq |M^{*'}_{c}|$, and hence can provide at most $h = |R| - |R'|$ more type-4 pairs in Y than $M^{*'}_{c}$. Therefore, any optimal matching in I based on the partition L' and R' of U' can have a size at most as big as the one produced by L, R, and $M^{*'}_{c}$. In fact, for some partitions of U' into L' and R' such that $|L'| - |R'|$ is not minimized (i.e., $|L'| - |R'| > |L| - |R|$), it is possible to have $|M^{*'}_{c'}| < |M^{*'}_{c}|$ (i.e., $M^{*'}_{c'}$ does not provide $h = |R| - |R'|$ more type-4 pairs of Y than $M^{*'}_{c}$ for matching the intervals of L'). Figure 6 gives such an example of L', R', and $M^{*'}_{c'}$. Therefore, $M^{*'}_{c}$ indeed contains type-4 interval pairs of Y that help match the maximum number of intervals of $L \cup R$.

We should also note that Lemmas 3.6 and 3.7 play a key role not only in proving the correctness of our algorithms, but also in achieving the efficiency of our algorithms. They allow our algorithms to compute recursively on the set $C = R \cup Y$ with $|C| < |I|/2 = n/2$. One could have recursively computed a maximum matching in $Y \cup L \cup R$ (instead of a maximum matching in C) and used such a matching to find the needed type-4 interval pairs of Y, but there is no guarantee on $|Y \cup L \cup R| \leq cn$ for any positive constant $c < 1$.

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 of $M^{*'}_{c}$ 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 of $M^{*'}_{c}$ to M^{*} and delete any $(|L| - |R|) - 2m$ intervals from L. For each $r \in R$, there is a distinct

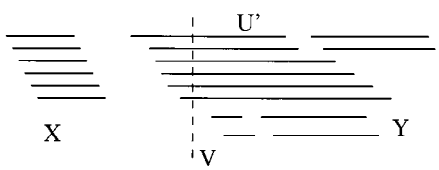

Fig. 6. For $|L'| = 6$ and $|R'| = 0$, $M^{*'}_{c'}$ has two type-4 pairs, but, for $|L| = 4$ and $|R| = 2$, $M^{*'}_{c}$ also has two type-4 pairs.

interval to the right of V with which r is paired; we add these $|R|$ pairs to M^*. For each remaining $l \in L$, there is a distinct interval to the left of V with which l is paired; we add these $|L|$ pairs to M^*. Finally, there are s unassigned intervals remaining on each of the left and right sides of V which form s pairs, and we add them to M^*. The matching M^* thus obtained is a desired optimal matching in the input interval set I.

If $|L| < |R|$, then due to the nature of the parallel convex bipartite matching algorithm (Lemma 2.2), we cannot perform any further "balancing", and hence must proceed directly to computing the set $C = L \cup X$, such that C contains the candidates for type-4 pairs to the left of V (which help match the maximum number of intervals of R). The computation for this case is similar to the case of $|L| > |R|$.

The formal description of our parallel algorithms is now given. The implementation details on the EREW PRAM and on hypercubes are described in the following subsections.

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

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

Output: A set, A, of the left elements of the pairs of an optimal matching M^* in I and a set, B, of the right elements of the pairs of M^*, listed in a 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 $le(i) < le(j)$, if the endpoints are not given already sorted.
2. Compute the location of a middle line V and the set U of intervals cut by V. Let b be the number of intervals to the left (resp., right) of 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 $re(i) > re(j)$.
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 $le(i) > le(j)$.
5. For each $u \in U$, compute the range of intervals in $I - U$ (i.e., $X \cup Y$) which are possible candidates for matching with u, using the labels assigned in Steps 3 and 4. Express this range for u as (*first, last*) (i.e., (f_u, l_u)).
6. Construct a representation for a convex bipartite graph $G = (U, I-U, E)$ with the edge set $E = \{(a_i, b_j) \mid a_i \in U,\ f_{a_i} \leq b_j \leq l_{a_i},\ b_j,\ f_{a_i}, l_{a_i} \in I - U\}$, by using the values of f_u and l_u computed in Step 5. Compute a maximum matching M^*_{cb} in G. Each interval $u \in U \cap M^*_{cb}$ is assigned to an interval of $I - U$ numbered from 1 to $2b$. If the assigned interval is in the range $[1, \ldots, b]$, then u 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 by the remaining intervals in $I - U$. Else, continue.

7. If $|L| > |R|$, let $Red = \{re(j) \mid j \in (L \cup R)\}$, $Blue = \{le(j) \mid j \in Y\}$. Perform a red–blue matching to find out 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}, there is a blue point $b \in Y$ assigned. If not every $r \in R$ is included in M^*_{RB}, then perform another red–blue matching with $Red = \{re(j) \mid j \in R - M^*_{\mathrm{RB}}\}$ and $Blue = \{re(j) \mid j \in L \cap M^*_{\mathrm{RB}}\}$ to transform it (Lemma 3.4) to $M^{*'}_{\mathrm{RB}}$. If $|M^{*'}_{\mathrm{RB}}| - |R| \geq \lfloor(|L| - |R|)/2\rfloor$, then add $\lfloor(|L| - |R|)/2\rfloor$ intervals of L that are in $M^{*'}_{\mathrm{RB}}$ to R and remove them from L. If $|L| - |R| = 1$, then delete one interval from the larger set. At this point, $|L| = |R|$. Return M^* which is obtained as in Step 6. If $|M^{*'}_{\mathrm{RB}}| - |R| < \lfloor(|L| - |R|)/2\rfloor$, then continue (with $|L| \neq |R|$).
8. If $|M^{*'}_{\mathrm{RB}}| - |R| < \lfloor(|L| - |R|)/2\rfloor$, then add any $|M^{*'}_{\mathrm{RB}}| - |R|$ intervals of L that are in $M^{*'}_{\mathrm{RB}}$ to R and delete them from L.
9. At this point, either $|L| > |R|$ (from Step 8) or $|R| > |L|$ (Steps 7 and 8 not executed). If $|L| > |R|$, then let $C = R \cup Y$. If $|R| > |L|$, then let $C = L \cup X$.
10. If $|C| > 2$, then let $M^*_c =$ **G-Disj-Match**(C, A', B'); else, compute $M^*_c = (A', B')$ in a straightforward manner.
11. If $|L| > |R|$, then examine the intervals in A' to see whether all intervals $r \in R$ have been retained in M^*_c. 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 are matched in $M^{*'}_c$ (by using Lemma 3.5). Update A' according to the pairs in $M^{*'}_c$.
12. If $|R| > |L|$, then examine the intervals in B' to see whether all intervals $l \in L$ have been retained in M^*_c. 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 are matched in $M^{*'}_c$ (by using Lemma 3.5). 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 L or R (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^* contains the remaining pairs in (A', B'), the matched pairs from R and L, and the pairs from the remaining intervals in X and in Y. 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 of the algorithms follows from the discussions in Sections 3.2.1 and 3.2.2. The time and processor bounds follow from the descriptions given in the next two subsections. □

3.3. *EREW PRAM Implementation and Analysis.* This subsection gives the implementation and analysis details of the EREW PRAM algorithm.

Step 1 takes $O(\log n)$ time and $O(n)$ processors for sorting [10]. Relabeling can be done by parallel prefix [22], [23] in $O(\log n)$ time and $O(n/\log n)$ processors. In Step 2 the middle line can be found easily in $O(\log n)$ time and $O(n/\log n)$ processors. The set U can be determined by parallel prefix. Identifying the sets X and Y in Steps 3 and 4 and relabeling them can all be done with parallel prefix in $O(\log n)$ time and $O(n/\log n)$ processors.

In Step 5, for every interval $u \in U$, we use the original list of sorted endpoints and find the first right endpoint to the left of $le(u)$ and the first left endpoint to the right of $re(u)$. Each of these two endpoints for u, for all $u \in U$, can be computed by parallel prefix (either along the left-to-right order or the right-to-left order of the sorted endpoint list).

In Step 6 a maximum matching in a convex bipartite graph can be obtained in $O(\log^2 n)$ time and $O(n/\log n)$ processors (Lemma 2.2).

In Step 7 red–blue matching takes $O(\log n)$ time and $O(n/\log n)$ processors [20]. In the original sorted list of endpoints, we keep the right endpoints of the intervals in $L \cup R$ and the left endpoints of the intervals in Y, and relabel them in increasing order of the x-coordinates using parallel prefix. We then label all the intervals in $L \cup R$ red and all the intervals in Y blue and compute the red–blue matching. If we need to do a second red–blue matching, in the original sorted list we keep the right endpoints of L in M^*_{RB} and of R not in M^*_{RB}. We relabel in increasing order of the x-coordinates, then color those of L in M^*_{RB} blue, those of R not in M^*_{RB} red, and compute the red–blue matching.

Step 8 can be easily done in $O(\log n)$ time and $O(n/\log n)$ processors.

In Step 9 the set C can be easily formed by parallel prefix. Note that $|C| < |I|/2 = n/2$.

In Step 10 we make a recursive call on the input set C. In each recursive call, the number of input intervals is less than half that of the previous one.

Steps 11 and 12 can be done in $O(\log n)$ time using $O(n/\log n)$ processors (by Lemma 3.5).

In Step 13 we can identify all non-type-4 pairs in (A', B') in constant time, and compute a compressed list of the type-4 pairs using parallel prefix in $O(\log n)$ time.

Step 14 can be done by parallel prefix to build the compressed lists of (A', B'), R, L, and the pairs of unassigned intervals of X and Y. This step takes $O(\log n)$ time and $O(n/\log n)$ processors.

The time and processor bounds for all nonrecursive steps discussed above are $O(\log^2 n)$ and $O(n/\log n)$, respectively. Let $T(n)$ and $W(n)$ denote the complexity bounds of the time and total number of operations taken by the algorithm, respectively. Then the recurrence relations for $T(n)$ and $W(n)$ are as follows:

$$T(n) \leq \begin{cases} T(n/2) + c_1 \log^2 n & \text{if } n > 2, \\ c_2 & \text{if } n \leq 2, \end{cases}$$

$$W(n) \leq \begin{cases} W(n/2) + c_3 n \log n & \text{if } n > 2, \\ c_4 & \text{if } n \leq 2, \end{cases}$$

where c_1, c_2, c_3, and c_4 are all positive constants. It is very easy to show that $T(n) = O(\log^3 n)$ and $W(n) = O(n \log n)$. Hence the EREW PRAM algorithm takes $O(\log^3 n)$

time and performs $O(n \log n)$ operations. By Brent's theorem [7], the algorithm can be made run in $O(\log^3 n)$ time using $O(n/\log^2 n)$ EREW PRAM processors.

3.4. *Hypercube Implementation and Analysis.* This subsection gives the implementation and analysis details of the algorithm on an n-processor hypercube.

Step 1 takes $O(\log n \log\log n)$ time for sorting [13]. Relabeling can be done by parallel prefix [24] in $O(\log n)$ time. In Step 2 the middle line can be found easily by ranking the endpoints in $O(\log n)$ time. In Steps 3 and 4 the sets U, X, and Y can all be obtained as on the EREW PRAM by using translation and parallel prefix in $O(\log n)$ time [24].

In Step 5 for every interval $u \in U$, we use parallel prefix to find the first right endpoint to the left of $le(u)$ and the first left endpoint to the right of $re(u)$ (either along the left-to-right order or the right-to-left order of the sorted endpoint list).

In Step 6 a maximum matching in a convex bipartite graph can be obtained in $O(\log^2 n)$ time by using Andrews' algorithm [2].

In Step 7 red–blue matching takes $O(\log n)$ time (by using the hypercube algorithm discussed in Section 2). In the original sorted list of endpoints, we use parallel prefix and concentration [24] to extract the right endpoints of the intervals in $L \cup R$ and the left endpoints of the intervals in Y, and relabel them in increasing order of the x-coordinates using parallel prefix. We then label all the intervals in $L \cup R$ red and all the intervals in Y blue. If we need to do a second red–blue matching, in the original sorted list we use parallel prefix and concentration to extract the right endpoints of L in M^*_{RB} and of R not in M^*_{RB}. We relabel in increasing order of the x-coordinates by using parallel prefix. Then color those of L in M^*_{RB} blue, those of R not in M^*_{RB} red, and perform the red-blue matching, in $O(\log n)$ time.

Step 8 can be done by parallel prefix and broadcasting in $O(\log n)$ time.

In Step 9 the set C can be easily formed by parallel prefix. Note that $|C| < |I|/2 = n/2$.

In Step 10 we make a recursive call on the input set C. In each recursive call, the number of input intervals is less than half that of the previous one. Hence, there are altogether $O(\log n)$ recursive calls for the algorithm.

Steps 11 and 12 can be done in $O(\log n)$ time (by Lemma 3.5).

In Step 13 we can identify all non-type-4 pairs in (A', B') in constant time, and compute a compressed list of the type-4 pairs using parallel prefix and concentration, in $O(\log n)$ time.

Step 14 can be done by using parallel prefix and concentration to build the compressed lists of (A', B'), R, L, and the pairs of unassigned intervals of X and Y. The time of this step is $O(\log n)$.

The dominating time complexity among all the nonrecursive steps is $O(\log^2 n)$. Since the algorithm make $O(\log n)$ recursive calls, the worst case time bound of the algorithm on an n-processor hypercube is $O(\log^3 n)$.

4. Matching among Disjoint Proper Intervals. We first give some useful observations for in the next subsection. Section 4.2 then presents the algorithms for maximum matching among disjoint proper intervals.

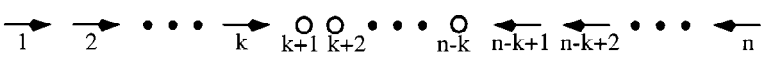

Fig. 7. The form of an optimal matching among proper intervals.

4.1. *Useful Observations*. Our algorithms for maximum matching among disjoint proper intervals are based on the following observations.

LEMMA 4.1. *Given the model of a proper interval graph in sorted order, there exists an optimal matching M^*, of cardinality k, for the disjoint proper intervals such that with respect to M^*, the first k intervals $I_1, I_2, \ldots, I_k$ match right, the last k intervals $I_{n-k+1}, I_{n-k+2}, \ldots, I_n$ match left, and all the intervals $I_{k+1}, I_{k+1}, \ldots, I_{n-k}$ are free* (*i.e., M^* has the form as illustrated in Figure* 7, *in which free intervals are denoted by unfilled circles*).

PROOF. Suppose that we are given an optimal matching M with k matches such that the leftmost left arrow (matching left) is at $i < n - k + 1$. Then, there must be some interval $j > i$ such that either j is unmatched or j is a right arrow (matching right). Let j be the smallest such interval.

Case 1: *j is unmatched.* $j > i$ means $le(j) > le(i)$ and $re(j) > re(i)$. Then either j overlaps with i or j lies to the right of i. In either case, we can perform a swap to obtain a new optimal matching M', so that j becomes matched in M' with $mate_M(i)$ and i is free in M'. Since, $re(mate_M(i)) < le(i)$ and $le(j) > le(i)$, clearly j can be matched with $mate_M(i)$.

Case 2: *j matches right.* By definition, $re(j) < le(mate_M(j))$. Either j overlaps with i or j lies to the right of i. In either case, since $re(i) < re(j)$, we can pair $(i, mate_M(j))$. Also, since $le(i) < le(j)$, we can pair $(mate_M(i), j)$. Now j matches left and i matches right.

By repeatedly performing this swapping operation, one can make the k rightmost intervals match left and the k leftmost intervals match right (this is done by first moving the k left arrows to the k rightmost intervals, and then moving the k right arrows to the k leftmost intervals). □

LEMMA 4.2. *Suppose it is known that the size of an optimal matching is k. Then the following pairs of intervals form an actual maximum matching M^*: $\{(i, n-k+i) \mid i = 1, 2, \ldots, k\}$.*

PROOF. Let M^* be an optimal matching in the form defined in Lemma 4.1. Because of the proper interval graph condition, for any $i \leq k$, if i cannot be matched with $n-k+i$, (i.e., i and $n-k+i$ are not disjoint), then there is no i' with $i < i' < n-k+i$ that does. Hence, i would have to match with some interval $j > n-k+i$. This would give a maximum matching whose size is less than k, a contradiction. □

Without loss of generality, we assume that n is a multiple of 2. Given the form of the optimal matching M^* as specified in Lemmas 4.1 and 4.2, we know that there exists an optimal matching in which all intervals which match right are in the range $[I_1, I_2, \ldots, I_{n/2}]$ and all intervals which match left are in the range $[I_{n/2+1}, I_{n/2+2}, \ldots, I_n]$.

4.2. *Matching Algorithm for Disjoint Proper Intervals.* The outline of our algorithms and the implementation details are described in this subsection.

Algorithm P-Disj-Match(I)

Input: A set $I = \{I_1, I_2, \ldots, I_n\}$ of n proper intervals, each specified by its two endpoints.
Output: A list of pairs of intervals which form a maximum matching M^* in I.

1. Check the input list of endpoints of the intervals in I to see whether it is already in sorted order. If the endpoint list is not in sorted order, then sort the endpoints of I.
2. Compute the size of a maximum matching M between the intervals in $\{I_1, I_2, \ldots, I_{n/2}\}$ which are candidates to match right and the intervals in $\{I_{n/2+1}, I_{n/2+2}, \ldots, I_n\}$ which are candidates to match left.
3. Compute the maximum matching M^* in I, which is of size $|M|$.

THEOREM 4.1. *Given a set I of n proper intervals, algorithm* **P-Disj-Match** *solves the maximum matching problem in $O(\log n)$ time on the EREW PRAM using $O(n/\log n)$ processors if the set of endpoints of I is given sorted and using $O(n)$ processors otherwise.*

PROOF. The correctness and the time and processor bounds follow from the discussion below. □

Checking whether the input endpoint list of I is already in sorted order is done by comparing every two consecutive values of the input list. For example, if there is one pair of consecutive values in the list "out of order" (e.g., the ith value is larger than the $(i+1)$th value), then the list is not sorted in increasing order. If no two consecutive values in the list are "out of order" for the increasing (resp., decreasing) order, then the list is already sorted in increasing (resp., decreasing) order. The checking can be easily done by parallel prefix in $O(\log n)$ time and $O(n/\log n)$ processors.

To compute $|M|$, we have two sets of intervals, and the intervals in the first set can only match right and the intervals in the second set can only match left. An interval j can match left with an interval i if and only if $re(i) < le(j)$. This implies that we only need to use the right (resp., left) endpoints of the intervals in the first (resp., second) set. Therefore, we represent each interval of I by a single point (either its right or left endpoint, depending on whether it is in the first or second set). Then a match can occur between an interval i of the first set $\{I_1, I_2, \ldots, I_{n/2}\}$ and

an interval j of the second set $\{I_{n/2+1}, I_{n/2+2}, \ldots, I_n\}$ if and only if $re(i) < le(j)$. To solve this matching problem, we transform it into a red–blue matching. We color the right endpoints of intervals $I_1, I_2, \ldots, I_{n/2}$ red and the left endpoints of intervals $I_{n/2+1}, I_{n/2+2}, \ldots, I_n$ blue, and compute the red–blue matching M on these colored points.

Let $d = |M|$. Then the maximum matching, M^*, is of size d. By Lemma 4.2, M^* consists of the following d interval pairs of I: $\{(i, n - d + i) \mid i = 1, 2, \ldots, d\}$.

Red–blue matching on sorted points can be done in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM by using Kim's algorithm [20]. Thus, the total time for computing a maximum matching between disjoint proper intervals is $O(\log n)$ time with $O(n/\log n)$ processors with sorted input, and $O(n)$ processors otherwise, on the EREW PRAM.

THEOREM 4.2. *Given a set I of n proper intervals in sorted order, algorithm* **P-Disj-Match** *solves the maximum matching problem in $O(\log n)$ time on n-processor hypercubes.*

PROOF. The main operations of the hypercube algorithm are ranking the endpoints, concentration, parallel prefix, and red–blue matching. The ranking, concentration, and parallel prefix can all be done in $O(\log n)$ time [24]. Given the interval endpoints already sorted, the red–blue matching can be performed also in $O(\log n)$ time by using the hypercube algorithm discussed in Section 2. □

5. Improved Overlapping Matching Algorithm on Proper Intervals. Moitra and Johnson [26] presented parallel algorithms for the related problem of finding a maximum cardinality matching between pairs of overlapping intervals. Let G be a proper interval graph with n vertices and I_G be its interval model. For the proper interval case, the algorithm in [26] is based on the CREW PRAM and takes $O(\log n)$ time and $O(n^2/\log n)$ processors. It computes a depth-first search tree, $T = (V, E')$, where $E' = \{(i, parent(i)) \mid parent(i) \neq 0\}$. Here, $parent(i)$ is the interval overlapping with i which ends first after the end of i, if such an interval exists. Assume G is connected and I_G is sorted. Then for each interval i, $parent(i) = i + 1$ (except the last interval). Each internal node in the tree has only one child and the root of the tree is $last(I_G) = n$ (i.e., $re(n) = \max\{re(k) \mid k \in I\}$). If there is more than one connected component, then it is still true that $parent(i) = i + 1$. However, there will be one interval i with $parent(i) = 0$ for each connected component. $last(I_G)$ will be the root of only one of the unary spanning trees. Moitra and Johnson show that the selection of all the odd labeled edges in each spanning tree constitutes a maximum matching. Computing the function $parent(i)$ is the dominating step in terms of the time and processor costs in [26].

Kim [20] has shown that computing $parent(i)$ can be done in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM, if the endpoints of the intervals are already sorted. Since each internal node in the spanning trees has only one child, we can treat the trees as lists. The initial node of the first list is I_1. For the other lists, the initial nodes are $\{j \mid parent(j - 1) = 0\}$. We can then apply the extended parallel

list ranking of Cole and Vishkin [11]. In the extended list ranking problem, there is more than one list and the problem is to compute the rank of each entry in its own list. This can be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. Therefore, with sorted input, a maximum matching between overlapping proper intervals can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.

6. Conclusion. We have given parallel algorithms in the EREW PRAM and hypercube models for the problem of computing a maximum cardinality matching between pairs of disjoint intervals in an interval model. Previously there was no efficient parallel algorithm known for this problem. 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 n processors on the hypercubes. For the case of proper interval graphs, our algorithm runs in $O(\log n)$ time on the EREW PRAM using $O(n)$ processors if the intervals are not given sorted and using $O(n/\log n)$ processors otherwise. On n-processor hypercubes, our algorithm for the proper interval case takes $O(\log n \log\log n)$ time for unsorted input and $O(\log n)$ time for sorted input.

We have also improved the parallel algorithm for maximum matching between overlapping intervals in a proper interval graph. Our algorithm runs in $O(\log n)$ time on the EREW PRAM using $O(n)$ processors if the input intervals are not given sorted and using $O(n/\log n)$ processors otherwise. The best previously known parallel algorithm for this problem [26] takes $O(\log n)$ time using $O(n^2/\log n)$ processors in the (stronger) CREW PRAM model. Hence our algorithm improves the processor bound while using a less powerful computational model.

Acknowledgment. The authors would like to thank the anonymous referees for carefully reading the manuscript and making helpful suggestions that considerably improved the presentation of this paper.

References

[1] R. Anderson and G. L. Miller, Deterministic Parallel List Ranking, *Algorithmica*, 6 (1991), 859–868.

[2] M. G. Andrews, Efficient Parallel Graph Algorithms on the Hypercube Network Model, Ph. D. Thesis, Northwestern University, December 1992.

[3] M. G. Andrews and D. T. Lee, An Optimal Algorithm for Matching in Interval Graphs, manuscript (1992).

[4] M. J. Atallah, M. T. Goodrich, and S. R. Kosaraju, On the Parallel Complexity of Evaluating Some Sequences of Set Manipulation Operations, *J. Assoc. Comput. Mach.*, 41 (1994), 1049–1088.

[5] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, Highly Parallelizable Problems, *Proc.* 21*st Annual ACM Symp. on Theory of Computing*, 1989, pp. 309–319.

[6] G. Bilardi and A. Nicolau, Adaptive Bitonic Sorting: An Optimal Parallel Algorithm for Shared-Memory Machines, *SIAM J. Comput.*, 18 (1989), 216–228.

[7] R. P. Brent, The Parallel Evaluation of General Arithmetic Expressions, *J. Assoc. Comput. Mach.*, 21(2) (1974), 201–206.

[8] D. Z. Chen, Efficient Parallel Binary Search on Sorted Arrays, with Applications, *IEEE Trans. Parallel Distrib. Systems*, 6(4) (1995), 440–445.
[9] D. Z. Chen, Optimal Hypercube Algorithms for Triangulating Classes of Polygons and Related Problems, *Proc. 7th Internat. Conf. on Parallel and Distributed Computing Systems*, Las Vegas, NV, 1994, pp. 174–179.
[10] R. Cole, Parallel Merge Sort, *SIAM J. Computing*, 17 (1988), 770–785.
[11] R. Cole and U. Vishkin, Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking, *Inform. and Control*, 70 (1986), 32–53.
[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, 1990.
[13] R. Cypher and C. G. Plaxton, Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers, *J. Comput. System Sci.*, 47 (1993), 501-548.
[14] E. Dekel and S. Sahni, A Parallel Matching Algorithm for Convex Bipartite Graphs, *Proc. International Conf. on Parallel Processing*, 1982, pp. 178–184.
[15] H. N. Gabow and R. E. Tarjan, A Linear-Time Algorithm for a Special Case of Disjoint Set Union, *J. Comput. System Sci.*, 30 (1985), 209–221.
[16] G. Gallo, An $O(n \log n)$ Algorithm for the Convex Bipartite Matching Problem, *Oper. Res. Lett*, 3(1) (1984), 31–34.
[17] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[18] T. Hagerup and C. Rub, Optimal Merging and Sorting on the EREW PRAM, *Inform. Process. Lett.*, 33 (1989), 181–185.
[19] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
[20] S. K. Kim, Optimal Parallel Algorithms on Sorted Intervals, *Proc. 27th Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, 1989, pp. 766–775.
[21] D. Kravets and C. G. Plaxton, Optimal Hypercube Algorithm for the All-Nearest Smaller Values Problem, *Proc. 6th IEEE Symp. on Parallel and Distributed Processing*, Dallas, TX, 1994.
[22] C. P. Kruskal, L. Rudolph, and M. Snir, The Power of Parallel Prefix, *IEEE Trans. Comput.*, 34(10) (1985), 965–968.
[23] R. E. Ladner and M. J. Fischer, Parallel Prefix Computation, *J. Assoc. Comput. Mach.*, 27 (1980), 831–838.
[24] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*: *Arrays–Trees–Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.
[25] W. Lipski, Jr., and F. P. Preparata, Efficient Algorithms for Finding Maximum Matchings in Convex Bipartite Graphs and Related Problems, *Acta Inform.*, 15 (1981), 329–346.
[26] A. Moitra and R. C. Johnson, A Parallel Algorithm for Maximum Matching in Interval Graphs, *Proc. Int. Conf. on Parallel Processing*, 1989, Vol. III, pp. 114–120.
[27] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.