Discrete Comput Geom 9:371–385 (1993)

Discrete & Computational
Geometry

Constructing Arrangements Optimally in Parallel*

Michael T. Goodrich

Department of Computer Science, The Johns Hopkins University,
Baltimore, MD 21218, USA
goodrich@cs.jhu.edu

Abstract. We give two optimal parallel algorithms for constructing the arrangement of n lines in the plane. The first nethod is quite simple and runs in $O(\log^2 n)$ time using $O(n^2)$ work, and the second method, which is more sophisticated, runs in $O(\log n)$ time using $O(n^2)$ work. This second result solves a well-known open problem in parallel computational geometry, and involves the use of a new algorithmic technique, the construction of an ε-pseudocutting. Our results immediately imply that the arrangement of n hyperplanes in $\mathbb{R}^d$ in $O(\log n)$ time using $O(n^d)$ work, for fixed d, can be optimally constructed. Our algorithms are for the CREW PRAM.

1. Introduction

A geometric structure of recognized importance in computational geometry is the arrangement defined by n hyperplanes in $\mathbb{R}^d$, i.e., the combinatorial structure describing the cells of $\mathbb{R}^d$ determined by the hyperplanes, as well as the adjacency information for these cells [15], [25], [38], [40] (see Fig. 1 for a two-dimensional example). Indeed, in his highly regarded book on algorithms in combinatorial geometry, Edelsbrunner argues that "arrangements of hyperplanes are at the very heart of computational geometry" [25]. Even in the plane, where there is an arrangement of lines (which is a planar graph), there are many applications for this structure (see [25], [38], and [40]). Moreover, by a well-known duality between hyperplanes and points, the arrangement can also be used to solve a number of problems dealing with points in $\mathbb{R}^d$ [16], [25], [40], [27]. We are interested in the parallel complexity of constructing arrangements.

* This research was supported by the National Science Foundation under Grants CCR-8810568 and CCR-9003299, and by the NSF and DARPA under Grant CCR-8908092.

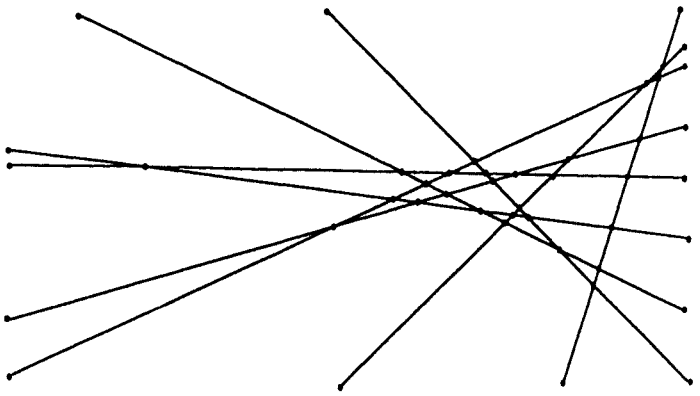

Fig. 1. A line arrangement.

The first optimal sequential algorithms for constructing line arrangements were developed independently by Chazelle *et al.* [16] and by Edelsbrunner *et al.* [27]. The main idea behind these methods is to construct the arrangement incrementally one line at a time. By an interesting "zone lemma" [16], [27], it can be shown that only $O(n)$ time is needed to insert each line; hence, the entire line arrangement can be constructed in $O(n^2)$ time. Moreover, this approach can be generalized to constructing the arrangement of hyperplanes in $\mathbb{R}^d$ in $O(n^d)$ time. This is of course optimal, since the arrangement has $\Omega(n^d)$ size. Incidentally, a line arrangement can also be constructed in $O(n^2)$ time by sweeping the plane with a vertical *pseudoline* [26], i.e., a line l that is "topologically" equivalent to a vertical line in the sense that the intersections of l and the lines of the arrangement are in order by y-coordinates.

A parallel algorithm is said to be *optimal* if the product of its time and number of processors matches the sequential lower bound for the problem it solves. Thus, an optimal parallel algorithm for line-arrangement construction would have to have a time-processor product of $\Theta(n^2)$. The main obstacle to designing such an optimal algorithm is that the paradigms that led to efficient sequential algorithms seem inherently sequential. There is, of course, a trivial suboptimal parallel algorithm analogous to a brute-force sequential method, where the intersections determined by each line are computed and then the intersections along each line are sorted. If this algorithm is implemented using an optimal sorting algorithm [20], [30], then it runs in $O(\log n)$ time using $O(n^2)$ processors.[1] Unfortunately, it is not at all clear how any of the known parallel techniques for deriving improved processor bounds may be applied to this algorithm, including Brent's theorem [12], the "sequential subsets" method [28], and the "accelerating cascades" paradigm [21]. This has prompted a number of researchers to pose as an open problem the existence of an $O(\log n)$ time CREW PRAM line-arrangement

[1] Unless stated otherwise all processor bounds mentioned in this paper are for the CREW PRAM, the synchronous shared memory parallel model where several processor may access the same memory location only if they are all reading from that location [24], [42], [44].

algorithm that uses $O(n^2/\log n)$ processors [4], [5], [29], [31]. In this paper we show that, in fact, a line arrangement can be optimally constructed in $O(\log n)$ time using $O(n^2/\log n)$ processors, which is optimal.

The previous best deterministic parallel algorithm for this problem is due to Anderson *et al.* [5], and runs in $O(\log n \log^* n)$ time using $O(n^2/\log n)$ processors. There is also a randomized parallel algorithm, due to Hagerup *et al.*, that runs in $O(\log n)$ expected time using $O(n^2/\log n)$ processors [31].

There has also been some previous work on solving the related problem of constructing the arrangement of n line segments [8], [9] in parallel, as well. For example, Chow [17] shows how to determine all the pairwise intersections of n axis-parallel segments in $O((1/\varepsilon)\log n + k_{\max})$ time using $O(n^{1+\varepsilon})$ processors [17], where $\varepsilon > 0$ is a small constant and $k_{\max}$ is the maximum, taken over all input segments s, of the number of intersections on s. In [29] the author shows how to construct the arrangement of such segments in $O(\log n)$ time using $O(n + k/\log n)$ processors, and how to construct a general segment arrangement in $O(\log n)$ time using $O(n \log n + k)$ processors. Rüb [41] shows how to construct such an arrangement in $O(\log n \log\log n)$ time using $O(n + k)$ processors. Of course, when applied to the line-arrangement problem these methods do not beat the trivial brute-force method. Recently, Clarkson *et al.* [19] have shown how to construct a segment arrangement in $O(\log n)$ expected time using $O(n + k/\log n)$ processors on a probabilistic CRCW PRAM, which, when applied to the line-arrangement problem, matches the bounds of Hagerup *et al.* (albeit for the more-powerful CRCW PRAM model, where concurrent-writes are allowed and are resolved arbitrarily).

In this paper we present two optimal parallel algorithms for line-arrangement construction. The first is quite simple and runs in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors. The main idea of this algorithm is to apply the parallel divide-and-conquer paradigm using a data structure of Anderson *et al.* [5] and a "truncated" zone lemma due to the author [29] to perform the "marry" step efficiently. The second method is more sophisticated and runs in $O(\log n)$ time using $O(n^2/\log n)$ processors. This algorithm is based on the efficient construction of a structure we call an *ε-pseudocutting*, which is a decomposition of the plane into pseudotrapezoids (i.e., trapezoids whose top and bottom edges are defined by pseudolines) such that each pseudotrapezoid intersects only a "few" lines.

2. A Simple Parallel Method

We begin with some definitions. Suppose we are given a set S of n lines in the plane. For simplicity of expression we assume that there are no vertical lines in S (it is easy to modify our algorithm for the more general case). The *arrangement* for S, denoted $A(S)$, is the planar graph $G = (V, E)$ such that V is the set of intersections formed by the lines in S and E is the set of edges defined by consecutive intersections along lines in S. The *depth* of an edge e in $A(S)$ is the number of lines of S that are directly above e, i.e., the number intersected by a vertical ray emanating upward from a point on e (other than one of e's endpoints).

The *k-level* in $A(S)$ is the set of edges at depth k [25]. Clearly, the k-level is a monotone chain of edges in $A(S)$, i.e., a chain that is intersected only once by any vertical line. In general, a k-level can have as many as $\Omega(n \log k)$ edges, but will always have no more than $O(n\sqrt{k})$ edges [25]. Of course, the total size of all k-levels in $A(S)$ is $\Theta(n^2)$.

Our algorithm description, which follows, makes considerable use of k-levels, and builds upon the approach of Anderson *et al.* [5]. We begin by dividing the set S into two equal-sized sets S_1 and S_2, such that the lines in S_1 all have slope smaller than the lines in S_2. This can easily be done in $O(\log n)$ time using $O(n/\log n)$ processors, assuming the lines in S are presorted by slope [20], [30]. We then recursively construct the arrangements $A(S_1)$ and $A(S_2)$ in parallel.

As might be expected, the most difficult step in our construction is to merge $A(S_1)$ and $A(S_2)$ into $A(S)$. To facilitate this we construct arrays that store each of the levels in $A(S_1)$ and $A(S_2)$ ordered left to right. This can be done in $O(\log n)$ time using $O(n^2)$ work by the list-ranking algorithm[2] of Anderson and Miller [6] or Cole and Vishkin [22]. We distinguish the k-levels such that k is a multiple of $\lceil \log n \rceil$ as *starter levels* in the arrangement. Anderson *et al.* make an interesting observation about sets of lines that are separated by their slopes, namely,

Observation 2.1 [5]. *If l is a line in S_i, then l intersects each level of $A(S_j)$, $i \neq j$, exactly once.*

As they show, this observation can be used to build a parallel data structure for efficiently finding the ordered intersections of a line l in S_1 with all the lines in $A(S_2)$. Simply, each starter level of $A(S_2)$ is stored in an array (as above). To intersect $l \in S_1$ with $A(S_2)$ $O(n/\log n)$ processors are assigned and l's intersection with each starter level is found by a binary search. This cuts l into $n/\log n$ segments. Then, for each such segment s, the sequential search method of Chazelle [13] is performed to crawl iteratively around the faces s intersects[3] to discover all the intersections of s with $A(S_2)$. In [29] the author shows, via a "truncated" zone lemma, that, given the locations of s's endpoints in the arrangement, such a sequential search can be performed in time proportional to the number of lines s intersects, which in this case is $O(\log n)$. Thus, the entire computation can be performed in $O(\log n)$ time using $O(n/\log n)$ processors, giving us the following lemma.

Lemma 2.2. *Given an arrangement $A(S_i)$, a data structure can be constructed that allows the computation, for any line l with slope outside the range of slopes in S_i, of the ordered list of intersections of l with $A(S_i)$ in $O(\log n)$ time using $O(n/\log n)$*

[2] Recall that in the list-ranking problem a linked list L is given and it is required to determine for each element $x \in L$ the rank of x in L.

[3] The iterative crawling method of Chazelle requires that each edge on a face f stores a pointer to the rightmost point on f. This can be computed using a list-ranking procedure [6], [22] for each face in parallel, which requires $O(\log n)$ time using $O(n^2/\log n)$ processors.

processors. Moreover, this data structure can be constructed in $O(\log n)$ *time using* $O(n^2/\log n)$ *processors.*

We complete the merging of $A(S_i)$ and $A(S_2)$, then, for each line l in S_1, by computing the ordered list of its intersections with $A(S_2)$ using Lemma 2.2. In parallel we also perform a similar computation for each line in S_2. We complete the construction of $A(S)$ by merging, for each line l, l's sorted lists of intersections in $A(S_1)$ and $A(S_2)$, respectively. This can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors [10], [11], [35], [43] for each line l, or $O(n^2/\log n)$ overall. This completes the construction.

The time complexity, $T(n)$, of this method is characterized by the recurrence

$$T(n) = T(n/2) + b \log n$$

for some constant b. Thus, this method runs in $O(\log^2 n)$ time. The work complexity, $W(n)$, is characterized by the recurrence

$$W(n) = 2W(n/2) + cn^2$$

for some constant c. This implies that this method uses $O(n^2)$ work. By an easy application of Brent's theorem [12] (also see [33] and [34]), then, we can therefore derive the following theorem:

Theorem 2.3. *The arrangement of* n *lines in the plane can be constructed in* $O(\log^2 n)$ *time using* $O(n^2/\log^2 n)$ *processors in the CREW PRAM model.*

In the next section we show how to extend this approach to construct an ε-pseudocutting for S, the main structure employed by our optimal $O(\log n)$-time method.

3. Constructing an ε-Pseudocutting

Recent developments on the theory of ε-cuttings have proven useful for solving a number of problems in computational geometry (e.g., see [1], [2], [14], [32], and [39]). The general paradigm is that a set X of n hyperplanes in $\mathbb{R}^d$ (lines in $\mathbb{R}^2$) is given and a parameter $\varepsilon > 0$, and it is desired to decompose the space into $O(r^d)$ constant-size polytopes (e.g., trapezoids in $\mathbb{R}^2$) such that each polytope intersects at most $\lceil n/r \rceil$ hyperplanes, where $r = 1/\varepsilon$. Such a set is called an *ε-cutting* for X. Building on results of Matoušek [39], Chazelle [14] shows that such an ε-cutting can be constructed in $O(nr^{d-1})$ time. This is closely related to the random sampling techniques of Clarkson [18] and Haussler and Welzl [32], which state that if the arrangement of r randomly chosen hyperplanes from X is triangulated, a decomposition of the space into $O(r^d)$ constant-size polytope is produced such that the expected number of hyperplanes intersecting any polytope is $O((n/r)\log r)$. These approaches have turned out to be very powerful, having applications to a number

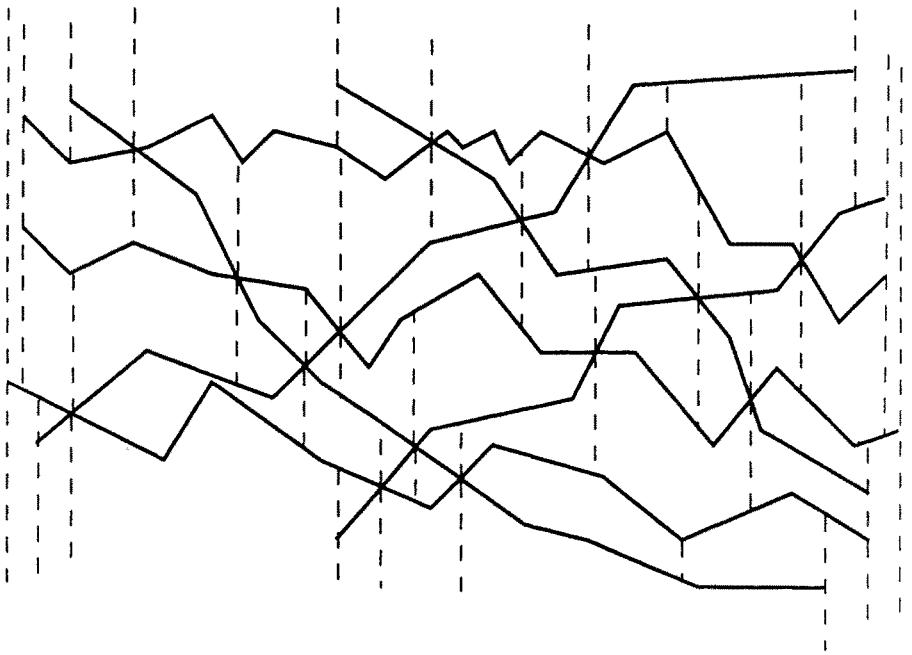

Fig. 2. An ε-pseudocutting.

of different problems in discrete and computational geometry (e.g., see [2] and [14]).

Our method for optimally constructing the arrangement of a set S of n lines in the plane is based on a similar ε-cutting approach. Unfortunately, none of these previous algorithms translate into an efficient deterministic parallel algorithm running in $O(\log n)$ time (which is what we would require for our arrangement construction procedure). Thus, we do not attempt to construct a true ε-cutting for S. Instead, we construct a decomposition of the plane that is "almost" an ε-cutting. In particular, we show in this section that if a set S of n lines in the plane is given, a collection of r^2 pseudotrapezoids (i.e., trapezoids with top and bottom edges that are pseudo line segments) can be constructed such that each pseudotrapezoid intersects at most $O(nt/r)$ lines in S, where $t \geq 1$ and $r \geq 1$ are integer parameters. By *pseudo line segments* we mean piecewise linear curve segments that are x-monotone and such that any two intersect each other at most once. (See Fig. 2.) Our method extends the approach of the previous section, and results in an efficient parallel algorithm.

We begin by sorting the lines in S by slope, and dividing this sorted list into t groups $S_1, S_2, \ldots, S_t$ of size $O(n/t)$ each, so that each line in S_i has smaller slope than every line in S_j if $i < j$. By using a slightly different implementation of the method of Anderson *et al.* [5] we can construct each $A(S_i)$ in $O(\log n)$ time using $O((n/t)^2 \log\log n/t)$ work (where their iterative process is stopped after two iterations[4]). Also, for each $A(S_i)$, we construct an array representation of each level of

[4] In general, $O((n/t)^2 \log^{(i)} n/t)$ work can be achieved by stopping their method after i iterations, but $i = 2$ is sufficient for our purposes.

$A(S_i)$. Using the list-ranking algorithm of Anderson and Miller [6] or that of Cole and Vishkin [22] this can be done in $O(\log n)$ time for all S_i's using a total of $O(n^2/t)$ work. Thus, all the $A(S_i)$'s and their levels can be constructed in $O(\log n)$ time using $O((n^2/t) \log \log n/t)$ work.

We define each k-level of $A(S_i)$ such that k is a multiple of $\lceil n/r \rceil$ (including the first and last levels) to be a *superlevel* in $A(S_i)$. Note that there are $O(r/t)$ superlevels per arrangement $A(S_i)$, and the total size of all the superlevels in any $A(S_i)$ is $O(n^2/t^2)$ (since $|S_i|$ is $O(n/t)$).

For each line $l \in S$, we compute the intersection of l with the superlevels in $A(S_j)$ for $j \neq i$ if $l \in S_i$. This gives us t sorted lists of intersections for each line l. Our method for doing this is to intersect l with $A(S_j)$ and then compress out the intersection points not on superlevels by a parallel prefix computation.[5] Since we can apply Lemma 2.2 to compute the intersection of l with $A(S_j)$ and we can apply an optimal parallel prefix method to perform the compression step [36], [37], this computation can be implemented in $O(\log n)$ time using $O(n^2)$ work in total (n lines $*$ t groups $*$ $O(n/t)$ work per group).

We next construct A', the arrangement of superlevels. The following lemma expresses a property concerning levels that is crucial to our construction.

Lemma 3.1. *Let $\mathscr{L}$ and $\mathscr{M}$ be two levels with $\mathscr{L} \subset A(S_i)$ and $\mathscr{M} \subset A(S_j)$, $i \neq j$. Then $\mathscr{L}$ and $\mathscr{M}$ intersect in a single point.*

Proof. This lemma follows immediately from a similar lemma given by Edelsbrunner [25, Lemma 14.4 on p. 338]. □

Therefore, by the above lemma, the intersections between $A(S_i)$ and $A(S_j)$ can be viewed as the intersections defined by two sets of parallel pseudolines, i.e., they define a "pseudogrid." Note that, by Lemma 3.1, A' is an arrangement of pseduolines. Moreover, for any superlevel $\mathscr{L}$, say from $A(S_i)$, $\mathscr{L}$ has $O(r/t)$ intersections with the superlevels in $A(S_j)$, $j \neq i$; hence, each superlevel $\mathscr{L}$ has $O(r)$ intersections with all the other superlevels. Also note that the total number of line segments making up A' is $O(n^2/t + r^2)$.

In our method for constructing A' we distinguish two kinds of intersections for a line $l \in S_i$: *proper intersections*, which are formed between l and a superlevel in some $A(S_j)$, $j \neq i$, and *improper intersections*, which are formed between l and the superlevels in $A(S_i)$. Note that proper intersections are points, while improper intersections are line segments. There are $O(n^2/t)$ improper intersections and $O(nr)$ proper intersections determined by A' and the lines in S. (See Fig. 3.)

Our method for constructing A', then, is as follows. First, for each line l, we sort the list of proper intersections along l. This amounts to merging t sorted lists, each of size $O(r/t)$; hence, can be implemented in $O(\log t \log \log r)$ time using $O(r \log t)$ work [11], [35] per line (for $O(nr \log t)$ work overall). We then merge the list of improper intersections along l with this list of proper intersections along l, and "throw away" each proper intersection that does not fall on an

[5] Recall that in the parallel prefix problem an array A of numbers is given and it is desired to compute each prefix sum $s_k = \sum_{i=1}^{k} A[i]$.

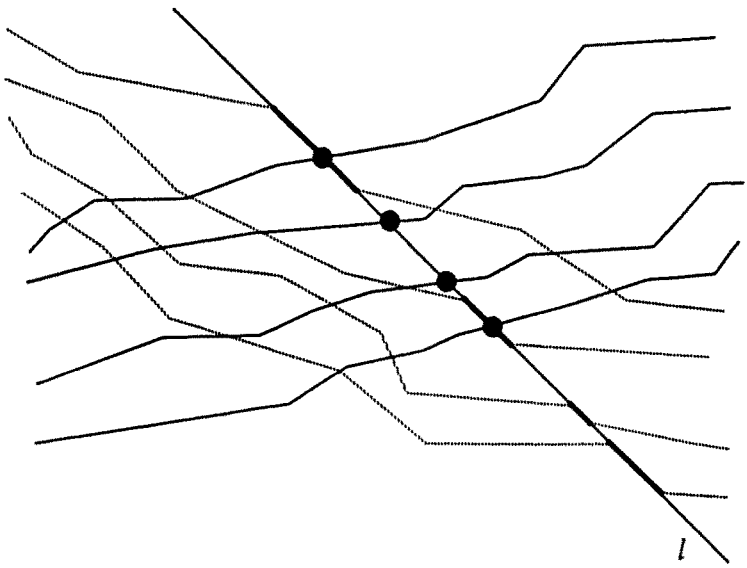

Fig. 3. An example superlevel arrangement A' for the case $t = 2$. The superlevels in $A(S_1)$ are shown by dotted lines and the superlevels in $A(S_2)$ are shown by solid lines. Also shown is a line l in S_1, together with its proper intersections (shown as bold dots) and its improper intersections (shown with bold line segments).

improper intersection of l. Finally, we construct the proper intersections along each superlevel by performing a list-ranking procedure, to link up the proper intersections that fall along each edge in the superlevel. This requires $O(\log n)$ time and $O(n^2/t + nr)$ work. Thus, given all the $A(S_i)$'s we can construct A' in $O(\log n + \log t \log\log r)$ time using a total of $O(nr \log t + n^2/t + nr)$ work.

So, how close is A' to being an ε-pseudocutting? It is quite close as it turns out. Note that there are $O(r^2)$ faces in A', which are defined by the $O(r^2)$ intersection points. Moreover, each face has at most $2t$ pseudoedges on its boundary. Each such face is not quite a pseudotrapezoid, but we do know that, since each pseudoedge on f's boundary is x-monotone, f's boundary can be decomposed into two chains of pseudoedges: an upper chain and a lower chain. We also have the following property for A':

Lemma 3.2. *The number of proper intersections along the boundary of any face of A' is $O(nt/r)$.*

Proof. Let f be a face of A'. By construction, f lies between two superlevels in each $A(S_i)$. Index each edge e on f by the index of the group S_i such that e is in a superlevel of $A(S_i)$. As a simple corollary of Lemma 3.1, it is easy to show that the edges in a left-to-right listing of the lower chain (resp. upper chain) have strictly increasing (resp. decreasing) indices. Suppose, for the sake of contradiction, that a level $\mathscr{L}$ of $A(S_i)$ has more than two proper intersections with f. Let a, b, and c be the indices of the first three proper intersections $\mathscr{L}$ has with f. That is, $\mathscr{L}$

enters f at a, exits at b, and enters again at c. Since $\mathscr{L}$ is x-monotone, b and c must both be on the lower chain of f or both on the upper chain. Without loss of generality, suppose they are both on the lower chain. Then $i < b$ and $c < i$. However, this contradicts the fact that the indices on the lower chain of f have increasing indices. Therefore, each $A(S_i)$ can contribute at most $O(n/r)$ proper intersections to the boundary of f. The lemma follows, then, since there are t groups. □

Thus, the only types of intersections that could cause a cell to intersect more than $O(nt/r)$ lines are improper intersections. Thus, we must further partition the cells in A' to limit the number of improper intersections per cell boundary (and also to enforce the property that each face in the resulting subdivision is a pseudotrapezoid). We do this by extending two vertical rays (one up and one down) from each of $O(r^2)$ distinguished points on the pseudoedges of A' to the first points of A' that the rays intersect, respectively. For a distinguished point p we call these points p's vertical shadows. The points we distinguish in this way include each point defined by the intersection of two superlevels and each point on a superlevel and whose rank in that level is a multiple of $\lceil n/r \rceil$ (when listed left to right). Adding edges from each distinguished point p to its vertical shadows forms a decomposition of A' into pseudotrapezoids with the desired "ε-pseudo-cutting" property for S. That is, this would imply a total of $O(r^2)$ pseudotrapezoids, each of which intersects at most $O(nt/r)$ lines.

To implement this strategy we need to locate, for each distinguished point p, the position of p and p's "vertical shadows" in $A(S_1), A(S_2), \ldots, A(S_t)$. This would then allow us to "walk" efficiently through each of the arrangements from p to its vertical shadows using Chazelle's [13] sequential search strategy. Unfortunately, the total work we wish to allow for this collection of point locations is $O(nrt^{O(1)})$, not $O(r^2 \log n)$, which would be the work of doing an independent point location for each point involved. Thus, we must use the adjacency information for the arrangement A' to aid us in performing all the necessary point locations in a batched fashion.

We proceed as follows. For each face f in A', we sort the proper intersections around f, by the counterclockwise ordering. By Lemma 3.2 and a simple application of Brent's theorem [12] this can be implemented in $O(\log n)$ time using $O((nt/r) \log nt/r)$ work per face f [20], [30], for a total work bound that is $O(nrt \log nt/r)$. Let $P(f)$ denote the resulting list. We merge $P(f)$ with the list of endpoints of the line segments that are improper intersections around f (i.e., the boundary edges of f), which can be done for all the faces of A' in $O(\log n)$ additional time using a total of $O(n^2/t + nrt)$ work [10], [11], [35], [43]. Let $I(f)$ denote the resulting list of intersection points around f, let $IU(f)$ denote the sublist of those intersections on f's upper chain, and let $IL(f)$ denote the sublist of those intersections on f's lower chain. We merge $IU(f)$ and $IL(f)$, which can also be done in $O(\log n)$ time using a total of $O(n^2/t + nrt)$ work for all faces in A'. For each distinguished point p on f this immediately gives us the position of p in $I(f)$ and the position in $I(f)$ of p's "vertical shadow," q, on the other side of f. To give us the position of p and q in an arrangement $A(S_i)$ we perform a parallel prefix

computation on the list $I(f) \cup D(f)$, where $D(f)$ is the set of all distinguished points and shadow points on the boundary of f. The goal of parallel prefix is to compress out all the points of $I(f) \cup D(f)$ that are not on an edge of $A(S_i)$ and are not in $D(f)$. We perform such a parallel prefix computation for each $i = 1, \ldots, t$ in parallel. By another simple application of Brent's theorem, this can be implemented in $O(\log n)$ time using $O(n^2 + nrt^2)$ work. Since this gives us the position of p and its vertical shadow in each arrangement $A(S_i)$, we may then assign a single processor to each distinguished point p and "walk" from p to its vertical shadow while computing the intersections of this vertical segment with the edges in $A(S_i)$, a computation that requires at most $O(n/r)$ time [29]. Thus, since there are $O(r^2)$ distinguished points, the total work for all of these traversals is $O(nrt)$. This gives us all the pseudotrapezoids we desire as well as giving us the set of lines cutting each pseudotrapezoid. The total time for completing the construction of the ε-pseudocutting, given A', then, is $O(\log n + n/r)$ using $O(n^2 + nrt \log nt/r + nrt^2)$ work. We summarize:

Theorem 3.3. *Given a set S of n lines in the plane, and parameters $r \geq 1$ and $t \geq 1$, a decomposition of the plane into $O(r^2)$ pseudotrapezoids can be constructed such that each pseudotrapezoid intersects at most $O(nt/r)$ pseudotrapezoids of S. This construction can be implemented in $O(\log n + \log t \log\log r + n/r)$ time using $O(n^2 + (n^2/t) \log\log n/t + nrt \log nt/r + nrt^2)$ work in the CREW PRAM model.*

Before we can present our method for optimal arrangement construction, there is one more algorithmic tool we need to present.

4. Counting Intersections in a Disk

In our arrangement construction algorithm we wish to determine, without actually computing them, the number of intersection points contained in the interior of a pseudotrapezoid of our ε-pseudocutting decomposition. This problem is topologically equivalent to the problem of simply counting intersections in a disk. That is, suppose we are given a collection S of n chords in a disk D and wish to determine the number of chord intersections in D. The method we describe actually does more than simply determine the total number of intersections; it also determines, for each chord c, the number, $k(c)$, of intersections determined by c. Our method is by divide and conquer.

1. Imagine cutting the boundary of D at some specified point so as to define a curve, C. This defines a total ordering of chord endpoints based on their rank in a counterclockwise listing along C. Sort the left endpoints by this ordering (for any chord c, we define c's *left* endpoint to be c's first endpoint in this ordering). This can be done in $O(\log n)$ time using $O(n)$ processors [20], [30], and provides the preprocessing for our divide-and-conquer scheme.

2. Divide the curve C into two curves A and B by cutting C at x, the median left endpoint of the curves in S. Divide S into three groups: S_{AA}, S_{AB}, and S_{BB},

where $S_{\alpha\beta}$ denotes the set of all chords whose "left" endpoint is an α and whose right endpoint is on β. Note that $|S_{AA}| + |S_{AB}| + |S_{BB}| = |S|$, and $|S_{\alpha\beta}| \leq |S|/2$ for $\alpha\beta \in \{AA, AB, BB\}$. Recursively solve the problem for S_{AA}, S_{AB}, and S_{BB} in parallel (we associate the curve A with S_{AA}, the curve B with S_{BB}, and the curve C with S_{AB}).

Comment. Having recursively computed all the intersections between chords in S_{AA}, S_{AB}, and S_{BB}, respectively, we have only to compute the intersections determined by chords in different sets. Note, however, that no chord in S_{AA} can intersect a chord in S_{BB}. Thus, we need only consider intersections determined by chords in S_{AA} and S_{AB} (resp. S_{AB} and S_{BB}). Since these two cases are symmetric, let us restrict our attention to those intersections determined by chords in S_{AA} and S_{AB}. The following observation establishes the easy, but important, property we exploit to compute the number of such intersections efficiently.

Observation 4.1. *Let s and t be two chords with $s \in S_{AA}$ and $t \in S_{AB}$. The chords s and t intersect if and only if the left endpoint of t occurs between the endpoints of s in a counterclockwise listing.*

3. Let T be a sorted listing of the left endpoints of chords in S_{AB}. Let U be a sorted listing of the left and right endpoints of chords in S_{AA}. Merge T and U. Given this merge we can immediately compute the number of chords in S_{AB} intersecting a particular chord c in S_{AA}. In particular let $a(c)$ (resp. $b(c)$) be the rank in T of c's left (resp. right) endpoint. Then the number of chords c intersects in S_{AB} is $b(c) - a(c)$, by the above observation. The true value for $k(c)$ can therefore be calculated by summing this value with the recursively computed value for $k(c)$. A similar computation can be performed for each c in S_{AB}. This step requires $O(\log n)$ time and $O(n/\log n)$ processors [10], [11], [35], [43].

4. Repeat step 3 to determine the intersections between S_{AB} and S_{BB} and update the $k(c)$ values accordingly. Summing all the $k(c)$ values over all chords in C gives us the value of k.

Thus, in $O(\log^2 n)$ time and $O(n \log n)$ work we can determine the number of intersection points in the circle. Since the essential computation in each step involves merging recursively constructed lists of endpoints, with a little more effort, the cascading divide-and-conquer paradigm of Atallah *et al.* [7] can be applied to implement this algorithm in $O(\log n)$ time using $O(n)$ processors. Nevertheless, $O(\log^2 n)$ time is sufficient for our purposes, so we will not elaborate on how $O(\log n)$ time can be achieved.

Note that our intersection-counting method did not depend on any geometric properties of the disk; it simply depended on the property that the boundary of the enclosing curve was simple and the endpoints were sorted around the boundary. Thus, it can also be applied to pseudotrapezoids. We summarize:

Lemma 4.2. *Given a collection of chords C in a pseudotrapezoid, the number of intersections $k(c)$ along each chord c in C can be computed in $O(\log n)$ time using $O(n)$ processors in the CREW PRAM model.*

In the next section we show how to combine the methods of the previous two sections to design a fast arrangement-construction algorithm.

5. Fast Arrangement Construction

We show in this section how to construct the arrangement of n lines in $O(\log n)$ time using $O(n^2/\log n)$ processors. The method is to construct, in $O(\log n)$ time, an ε-pseudocutting for $r = n/(\log\log n)^2$ and $t = \log\log n$ by Theorem 3.3. This implies that our construction uses $O(n^2)$ work, and results in $O(n^2/(\log\log n)^4)$ pseudotrapezoids, each of which intersects at most $O((\log\log n)^3)$ lines. That is, if we "cut" each line at its proper intersections, this ε-pseudocutting determines $O(n^2/\log\log n)$ line *segments*, with at most $O((\log\log n)^3)$ segments per pseudotrapezoid. We then use our method for counting intersections in a "disk" to determine the number of intersections in each pseudotrapezoid. We then group the pseudotrapezoids into $O(n^2/\log n)$ "buckets," such that each bucket contains at most $O(\log n/\log\log n)$ segments and determines at most $O(\log n)$ intersections (in the interiors of the pseudotrapezoids in this bucket). This can easily be done in $O(\log n)$ time using $O(n^2)$ work, say, by parallel prefix computations. Finally, we assign a single processor to each bucket, and let that processor apply the sequential segment-arrangement method of Chazelle and Edelsbrunner to construct the arrangement in each pseudotrapezoid in this bucket. The method of Chazelle and Edelsbrunner runs in $O(n_b \log n_b + k_b)$ time, where n_b is the number of segments in bucket b and k_b is the number of intersections these segments determine. In our case, n_b is always $O(\log n/\log\log n)$ and k_b is always $O(\log n)$. Therefore, no processor will take more than $O(\log n)$ time to complete the construction of the arrangements in its assigned pseudotrapezoids. This completes the construction and gives us the following theorem:

Theorem 5.1. *The arrangement of n lines in the plane can be constructed in $O(\log n)$ time using $O(n^2/\log n)$ processors in the CREW PRAM model, which is optimal.*

6. Discussion

We have shown how to construct optimally the arrangement of n lines in the plane in $O(\log n)$ time, solving a well-known open problem in parallel computational geometry [4], [29], [5]. Using the "induction" argument of Anderson *et al.* [5], this immediately implies that the arrangement of n hyperplanes in $\mathbb{R}^d$, for fixed d, can be optimally constructed in $O(\log n)$ time using $O(n^d/\log n)$ processors.

In the sequential setting the construction of a line arrangement is the bottleneck computation for a number of problems, including hidden-line elimination [23] and the problem of finding the minimum-area triangle determined by three points taken from a set of n points in the plane [16]. Our methods immediately imply $O(\log n)$-time, optimal-work parallel methods for these problems.

Besides solving an important subproblem in many applications, the previous sequential line-arrangement algorithms [16], [26], [27] have also contributed important ideas that eventually led to an optimal method for constructing the arrangement of n line segments [15]. An interesting open question, then, is the following: Can the arrangement of n line segments be deterministically constructed in $O(\log n)$ time using $O(n + k/\log n)$ processors, where k is the number of intersections? Currently, the only optimal output-sensitive methods are either randomized [19] or for special cases, such as axis-parallel segments [29].

Acknowledgments

We would like to thank Richard Anderson, Richard Cole, and S. Rao Kosaraju for several helpful discussions related to topics discussed in this paper. We would also like to thank an anonymous referee for a number of helpful comments that significantly improved the level of rigor in our algorithm descriptions.

References

1. P. K. Agarwal, Partitioning Arrangements of Lines, II: Applications, *Discrete Comput. Geom.* **5** (1990), 533–573.
2. P. K. Agarwal, Geometric Partitioning and Its Applications, Technical Report CS-1991-27, Dept. of Computer Science, Duke University, 1991.
3. A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, Parallel Computational Geometry, *Algorithmica* **3** (1988), 293–328.
4. A. Aggarwal and J. Wein, *Computational Geometry*, M.I.T. Report MIT/LCS/RSS 3, 1988.
5. R. Anderson, P. Beame, and E. Brisson, Parallel Algorithms for Arrangements, *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 298–306. (An expanded version of this paper is available as Technical Report 89-12-08, Dept. of Computer Science and Engineering, University of Washington, 1989.)
6. R. J. Anderson and G. L. Miller, Deterministic Parallel List Ranking, *Proc. 3rd Aegean Workshop on Computing, AWOC '88*, Lecture Notes in Computer Science, Vol. 319, Springer-Verlag, Berlin, 1988, pp. 81–90.
7. M. J. Atallah, R. Cole, and M. T. Goodrich, Cascading Divide-and-Conquer: A Technique for Designing Parallel Algorithms, *SIAM J. Comput.* **18** (1989), 499–532.
8. J. L. Bentley and T. Ottmann, Algorithms for Reporting and Counting Geometric Intersections, *IEEE Trans. Comput.* **28** (1979), 643–647.
9. J. L. Bentley and D. Wood, An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles, *IEEE Trans. Comput.* **29** (1980), 571–576.
10. G. Bilardi and A. Nicolau, Adaptive Bitonic Sorting: An Optimal Parallel Algorithm for Shared Memory Machines, Technical Report 86-769, Dept. of Computer Science, Cornell University, August 1986.
11. A. Borodin and J. E. Hopcroft, Routing, Merging, and Sorting on Parallel Models of Computation, *J. Comput. System Sci.* **30** (1985), 130–145.
12. R. P. Brent, The Parallel Evaluation of General Arithmetic Expressions, *J. Assoc. Comput. Mach.* **21** (1974), 201–206.
13. B. Chazelle, Reporting and Counting Segment Intersections, *J. Comput. System Sci.* **32** (1986), 156–182.
14. B. Chazelle, An Optimal Convex Hull Algorithm and New Results on Cuttings, *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, 1991, pp. 29–38.

15. B. Chazelle and H. Edelsbrunner, An Optimal Algorithm for Intersecting Line Segments in the Plane, *J. Assoc. Comput. Mach.* **39** (1992), 1–54.
16. B. Chazelle, L. J. Guibas, and D. T. Lee, The Power of Geometric Duality, *BIT* **25** (1985), 76–90.
17. A. Chow, Parallel Algorithms for Geometric Problems, Ph.D. thesis, Computer Science Dept., University of Illinois, 1980.
18. K. L. Clarkson, New Applications of Random Sampling in Computational Geometry, *Discrete Comput. Geom.* **2** (1987), 195–222.
19. K. L. Clarkson, R. Cole, and R. E. Tarjan, Randomized Parallel Algorithms for Trapezoidal Diagrams, *Internat. J. Comput. Geom. Appl.* **2** (1992), 117–134.
20. R. Cole, Parallel Merge Sort, *SIAM J. Comput.* **17** (1988), 770–785.
21. R. Cole and U. Vishkin, Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Designing Parallel Algorithms, *Proc. 18th ACM Symp. on Theory of Computing*, 1986, pp. 206–219.
22. R. Cole and U. Vishkin, Approximate and Exact Parallel Scheduling with Applications to List, Tree and Graph Problems, *Proc. 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 478–491.
23. F. Dévai, Quadratic Bounds for Hidden-Line Elimination, *Proc. 2nd ACM Symp. on Computational Geometry*, 1986, pp. 269–275.
24. P. W. Dymon and S. A. Cook, Hardware Complexity and Parallel Computation, *Proc. 21st IEEE Symp. on Foundations of Computer Science*, 1980, pp. 360–372.
25. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.
26. H. Edelsbrunner and L. J. Guibas, Topologically Sweeping an Arrangement, *J. Comput. System Sci.* **38** (1989), 165–194.
27. H. Edelsbrunner, J. O'Rourke, and R. Seidel, Constructing Arrangements of Lines and Hyperplanes with Applications, *SIAM J. Comput.* **15** (1986), 341–363.
28. M. T. Goodrich, Efficient Parallel Techniques for Computational Geometry, Ph.D. thesis, Dept. Computer Sciences, Purdue University, 1987.
29. M. T. Goodrich, Intersecting Line Segments in Parallel with an Output-Sensitive Number of Processors, *SIAM J. Comput.* **20** (1991), 737–755.
30. M. T. Goodrich and S. R. Kosaraju, Sorting on a Parallel Pointer Machine with Applications to Set Expression Evaluation, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 190–195.
31. T. Hagerup, H. Jung, and E. Welzl, Efficient Parallel Computation of Arrangement of Hyperplanes in d Dimensions, *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 290–297.
32. D. Haussler and E. Welzl, ε-Nets and Simplex Range Queries, *Discrete Comput. Geom.* **2** (1987), 127–151.
33. J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
34. R. M. Karp and V. Ramachandran, Parallel Algorithms for Shared-Memory Machines, in *Handbook of Theoretical Computer Science*, Vol. A, J. Van Leeuwen, ed., MIT Press, Cambridge, MA, 1990, pp. 869–942.
35. C. P. Kruskal, Searching, Merging, and Sorting in Parallel Computation, *IEEE Trans. Comput.* **32** (1983), 942–946.
36. C. P. Kruskal, L. Rudolph, and M. Snir, The Power of Parallel Prefix, *Proc. 1985 Internat. Conf. on Parallel Processing*, 1985, pp. 180–185.
37. R. E. Ladner and M. J. Fischer, Parallel Prefix Computation, *J. Assoc. Comput. Mach.* **27** (1980), 831–838.
38. D. T. Lee and F. P. Preparata, Computational Geometry—A Survey, *IEEE Trans. Comput.* **33** (1984), 872–1101.
39. J. Matoušek, Approximations and Optimal Geometric Divide-and-Conquer, *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, pp. 505–511.
40. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
41. C. Rüb, Line Segment Intersection Reporting in Parallel, *Algorithmica* **8** (1992), 119–144.

42. W. L. Ruzzo, On Uniform Circuit Complexity, *J. Comput. System Sci.* **22** (1981), 365–383.
43. Y. Shiloach and U. Vishkin, Finding the Maximum, Merging, and Sorting in a Parallel Computation Model, *J. Algorithms* **2** (1981), 88–102.
44. J. C. Wyllie, The Complexity of Parallel Computation, Ph.D. thesis, Technical Report 79-387, Dept. of Computer Science, Cornell University, 1979.

Received August 29, 1991, and in revised form September 4, 1992.