Information Processing Letters 26 (1987/88) 173–179
North-Holland

4 December 1987

FINDING THE CONVEX HULL OF A SORTED POINT SET IN PARALLEL *

Michael T. GOODRICH

Department of Computer Science, Purdue University, West Lafayette, IN 47907, U.S.A.

Communicated by G.R. Andrews
Received 20 January 1987
Revised 11 May 1987

We present a parallel algorithm for finding the convex hull of a sorted planar point set. Our algorithm runs in O(log n) time using O(n/log n) processors in the CREW PRAM computational model, which is optimal. One of the techniques we use to achieve these optimal bounds is the use of a parallel data structure which we call the *hull tree*.

Keywords: Parallel algorithms, computational geometry, divide-and-conquer, convex hull problem

1. Introduction

Given n points in the plane, the convex hull problem is that of finding which of these points belong to the perimeter of the smallest convex region (a polygon) containing all n points. We are interested in solving this problem efficiently in parallel in the CREW PRAM computational model (i.e., the synchronous parallel model where processors share a common memory in which concurrent reads are allowed, but no two processors can simultaneously write to the same memory location). More formally, we are interested in finding the fastest algorithm which minimizes the product tp, where t is the time complexity of the algorithm and p is the number of processors used by it.

The convex hull problem is well known in computational geometry, and has been well studied in sequential computational models (see [14]). Yao [20] has shown that this problem has an $\Omega(n \log n)$ sequential lower bound (in the quadratic decision-tree model) if the points are input in arbitrary order, and there are a number of algorithms which achieve this lower bound [10,11,17,18]. If we are given the points in sorted order (e.g., by increasing x-coordinate), however, it is well known that we can solve the convex hull problem sequentially in O(n) time only [10].

Also, a considerable amount of work has been done on finding convex hulls in parallel. For example, Chazelle [5] shows how to solve the problem systolically on an n-node linear array of processors in O(n) time, and Miller and Stout [15] present an O($\sqrt{n}$) time solution on an n-node square-mesh of processors. Although both of these algorithms are optimal for their respective computational models, they are sub-optimal if implemented on a CREW PRAM. The first convex hull algorithm for the CREW PRAM model is due to Chow [6], and runs in O($\log^2 n$) time using O(n) processors. Since then, Aggarwal et al. [1] and Atallah and Goodrich [2,3] have been able to improve this to O(log n) time still using only O(n) processors. By a simple simulation argument it is easy to see that these latter algorithms are optimal, since they have a tp product which is O(n log n). Of course, this assumes that the input points are given in arbitrary order. As it turns out, each of these optimal CREW PRAM algorithms [1,2,3] share a common structure in that they consist of a sorting step followed by a parallel

* This research was supported by the Office of Naval Research under Grants Nos. N00014-84-K-0502 and N00014-86-K-0689, and the National Science Foundation under Grant No. DCR-84-51393, with matching funds from AT&T.

divide-and-conquer step, both of which require O(log n) time using O(n) processors. Thus, these algorithms are not optimal if the input points are given in sorted order, for even though we can skip the sorting step in this case, the previous algorithms' second phase will still have a tp product of O(n log n).

In this paper we give a CREW PRAM algorithm which finds the convex hull in O(log n) time using only O(n/log n) processors if we are given the points in sorted order (e.g., by increasing x-coordinates). This algorithm is clearly optimal, since it has a tp product which is O(n). One technique which is often utilized to reduce processor bounds is the method of 'cutting-off' a parallel recursion early and finishing the remaining subproblems sequentially [7,8]. It is not clear how to apply this technique here, however, to reduce the processor bound to O(n/log n), because this would not provide enough processors to quickly perform the merge procedures for higher levels of the recursion. Instead, the method we use involves introducing a parameter d into the recursion to govern how the subproblems are divided, and using a parallel data structure, which we call the *hull tree*, to aid in quickly merging subproblem solutions. In the next section we make some preliminary definitions and observations. In Section 3 we describe the hull tree data structure, studying some of its properties, and in Section 4 we give our algorithm for constructing the convex hull of a point set in which input points are given in sorted order.

2. Preliminaries

We first present some definitions and observations. For any point p in the plane we let $x(p)$ and $y(p)$ respectively denote the x- and y-coordinate of p. We say a list $S = (p_1, p_2, \ldots, p_n)$ of points in the plane is x-sorted if the points of S are listed by increasing x-coordinate (i.e., $x(p_i) \leqslant x(p_{i+1})$). We generalize this to a collection of point sets $\Pi = (S_1, S_2, \ldots, S_m)$, saying that Π is *x-sorted* if each S_i is x-sorted and all the points in each S_i are no greater than any point in S_{i+1}.

Let an x-sorted point set S be given. We denote a clockwise listing of the points which belong to the convex hull of S by $CH(S)$. Let p_1 and p_n be the points of S with the smallest and largest x-coordinate, respectively. Clearly, p_1 and p_n are both in $CH(S)$. They divide $CH(S)$ into two sets: an upper hull, $UH(S)$, consisting of points from p_1 to p_n, inclusive, in the clockwise listing of $CH(S)$, and a lower hull, $LH(S)$, consisting of points in $CH(S)$ from p_n to p_1, inclusive. Without loss of generality, for the remainder of this paper we shall concentrate on the problem of computing $UH(S)$, as the method for computing $LH(S)$ is symmetric. Given two disjoint upper hulls $UH(R)$ and $UH(S)$, we refer to the common tangent T such that both $UH(R)$ and $UH(S)$ are below T as the *upper common tangent* between $UH(R)$ and $UH(S)$. Also, when we say that a point p is 'to the left' of another point q, we mean that $x(p) < x(q)$. For simplicity of expression, we also assume that the input points have distinct x-coordinates and no three points are collinear (our results can easily be modified for the general case).

We make use of the fact that the parallel prefix of a sequence of n integers can be computed in O(log n) time using O(n/log n) processors [12,13]. Recall that, in the parallel prefix problem, we are given an array of integers $(a_1, a_2, \ldots, a_n)$ and wish to compute all the partial sums $s_k = \sum_{i=1}^{k} a_i$.

As mentioned above, our method for constructing upper hulls involves the use of a data structure which we call the *hull tree*. Since the skeleton of this data structure is a binary tree, we need the following definitions. Let B be a binary tree. We define the height of B, denoted *height*(B), to be the length of the longest leaf-to-root path. Let π be a leaf-to-root path. We say that a node v belongs to the *left fringe* (respectively *right fringe*) of π if v is not on π and is the left child (respectively right child) of a node on π. We describe the hull tree data structure in the next section.

3. The hull tree data structure

Let R be an x-sorted set of n points in the plane. We define the hull tree data structure as

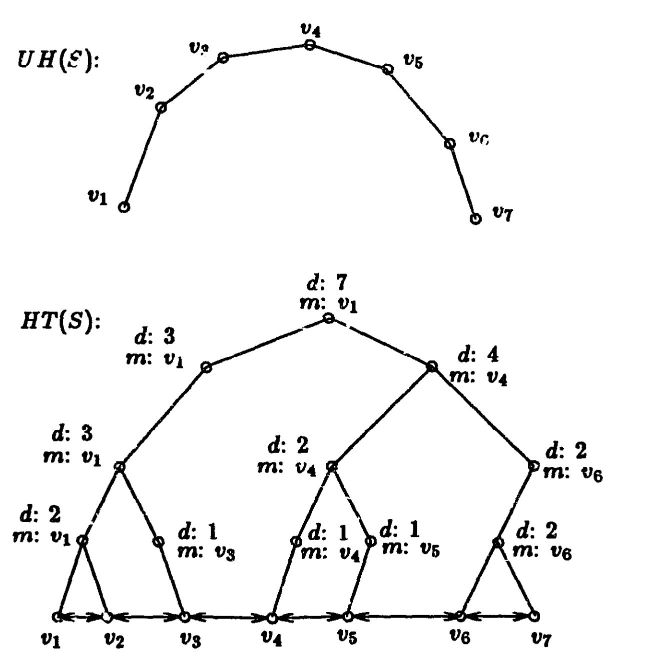

Fig. 1. An example hull tree $HT(S)$ for $UH(S)$. The d and m labels are given for each internal node, and the *succ* and *prev* pointers are denoted by arrows at the leaves.

follows. It is a binary search tree $HT(R)$ which stores the points of $UH(R)$ in its leaf nodes, sorted from left to right by increasing x-coordinates. For simplicity of expression, for each leaf node v we also let v denote the point in $UH(R)$ associated with this node. With each leaf v we store two labels $prev(v)$ and $succ(v)$ which are, respectively, the predecessor and successor points of v in $UH(R)$. For each internal node $v \in T$ we let $Desc(v)$ denote the set of descendent leaves of v. With each internal node $v \in T$ we store two labels $d(v)$ and $m(v)$ which are, respectively, the number of points in $Desc(v)$ and the point in $Desc(v)$ with minimum x-coordinate (see Fig. 1). In the following lemmas we study some of the properties of hull trees.

3.1. Lemma. *Let (R_1, R_2) be an x-sorted collection of two planar point sets. Given hull trees $HT(R_1)$ and $HT(R_2)$, we can find the common upper tangent of $UH(R_1)$ and $UH(R_2)$ in $O(h)$ time using a single processor where*

$$h = height(HT(R_1)) + height(HT(R_2)).$$

Proof. The method is based on the binary search procedure of Overmars and Van Leeuwen [16] for finding the common upper tangent between two convex polygons. The proof follows from the fact that the binary tree structure and the labels *pred, succ, d,* and *m* can be used to mimic the binary search method (see [9] for details). □

Thus, we can quickly find the common tangent of the hulls represented in two different hull trees. In the next lemma we show how to perform a ***split*** operation quickly on a hull tree.

3.2. Lemma. *Let R be an x-sorted planar point set, and let $HT(R)$ be a hull tree for $UH(R)$. Given any x-coordinate x_0 we can* ***split*** *$HT(R)$ into two hull trees $HT(R_1)$ and $HT(R_2)$ such that each point in R_1 has x-coordinate at most x_0 and each point in R_2 has x-coordinate greater than x_0, and this construction can be done in $O(h)$ time using a single processor, where*

$$h = height(HT(R)).$$

Proof. The method is to trace a root-to-leaf path searching for x_0 between the m label values, copying the nodes on this path as we go. In the original path we delete any children on the right fringe and in the copied path we delete children on the left fringe. Once we have reached the location in the leaf level where x_0 belongs, we update the label $pred(succ(v))$ to nil, and then update $succ(v)$ to nil, where v is the leaf node with greatest x-coordinate less than or equal to x_0. We then retrace our steps in each path, updating the d and m labels as we go so their new values are correct. It is clear that this method takes at most $O(height(HT(R)))$ time using a single processor. □

Note that the previous two lemmas both involve the use of a single processor. In the following two lemmas we explore some of the ways hull trees can be utilized in parallel. Both of these lemmas involve doing various computations on a collection of hull trees.

3.3. Lemma. *Let $\Pi = (R_1, R_2, \ldots, R_m)$ be an x-sorted collection of planar point sets, and let $S = R_1 \cup R_2 \cup \cdots \cup R_m$. If we have a hull tree $HT(R_i)$ constructed for each $UH(R_i)$, then for any $i =$*

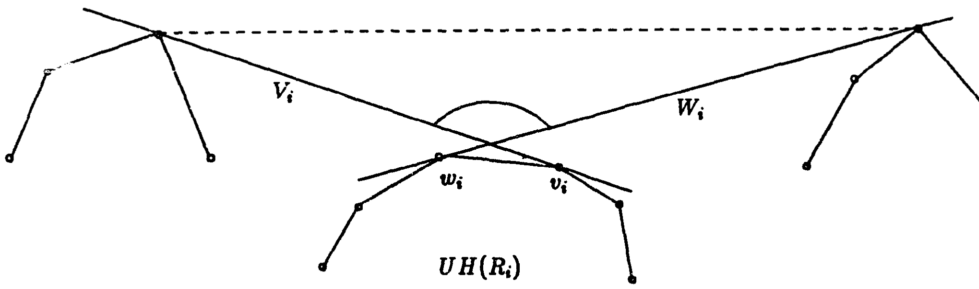

Fig. 2. An illustration of the case when none of $UH(R_i)$'s points are in $UH(S)$, because V_i and W_i form an angle which is less than 180°.

1, 2,..., m we can construct a hull tree for $UH(R_i) \cap UH(S)$ *in* $O(h + \log m)$ *time using* $O(m)$ *processors in the CREW PRAM model, where*

$$h = \max_{1 \leq i \leq m} \{height(HT(R_i))\}.$$

Proof. Let $i \in \{1, 2, \ldots, m\}$ be given. Our method for constructing a hull tree H_i' containing the points in $UH(R_i) \cap UH(S)$ is as follows. Assign a single processor to each pair (i, j), $j = 1, 2, \ldots, i-1, i+1, \ldots, m$, and, using the method of Lemma 3.1, find the common upper tangent $T_{i,j}$ between $UH(R_i)$ and $UH(R_j)$. This will take at most $O(h + \log m)$ time (it takes $O(\log m)$ time to compute the value of h). Let V_i be the tangent with smallest slope in $\{T_{i,1}, \ldots, T_{i,i-1}\}$ (i.e., V_i is the smallest-slope tangent which 'comes from the left' of $UH(R_i)$), and let W_i be the tangent with largest slope in $\{T_{i,i+1}, \ldots, T_{i,m}\}$ (i.e., W_i is the largest-slope tangent which 'comes from the right' of $UH(R_i)$). Both V_i and W_i can clearly be found in $O(\log m)$ time by the m processors assigned to R_i. Let v_i be the point of contact of V_i with $UH(R_i)$, and let w_i be the point of contact of W_i with $UH(R_i)$. Since neither V_i nor W_i can be vertical, they intersect and form an angle (with interior pointing upward). If this angle is less than 180° (as in Fig. 2), then none of the points of $UH(R_i)$ belong to $UH(S)$. This is because in this case the straight-line segment joining the other endpoints of V_i and W_i (which are contained in $CH(S)$) is entirely above $UH(R_i)$; hence, no vertex of $UH(R_i)$ can belong to $UH(S)$. In this case, the hull tree H_i' is empty. Otherwise (as in Fig. 3), if this angle is greater than 180°, then all the points from v_i to w_i, inclusive, belong to $UH(S)$. For if the angle between V_i and W_i is greater than 180°, then the points from all the other $UH(R_j)$'s must be below V_i and W_i. In this case, we can construct H_i' by performing two ***split*** operations on $HT(R_i)$, one to remove points with x-coordinates less than $x(v_i)$ and one to remove points with x-coordinate

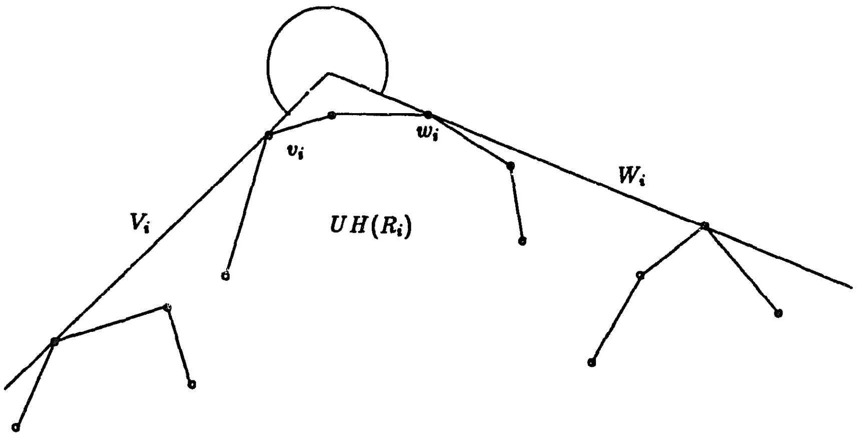

Fig. 3. The points between v_i and w_i, inclusive, are in $UH(S)$, because V_i and W_i form an angle which is at least 180°.

greater than $x(w_i)$. These ***split*** operations can be done in $O(h)$ time using a single processor by Lemma 3.2. Thus, the entire computation requires $O(h + \log m)$ time using $O(m)$ processors. □

In the next lemma we show that we can use the method of the previous lemma to construct a hull tree for the upper hull of the union of the sets in Π from hull trees for each set in Π.

3.4. Lemma. *Let $\Pi = (R_1, R_2, \ldots, R_m)$ be an x-sorted collection of planar point sets, and let $S = R_1 \cup R_2 \cup \cdots \cup R_m$. If we have a hull tree $HT(R_i)$ constructed for each $UH(R_i)$, then we can construct a hull tree $HT(S)$ for $UH(S)$ in $O(h + \log m)$ time using $O(m^2)$ processors, where*

$$h = \max_{1 \leqslant i \leqslant m} \{ height(HT(R_i)) \}.$$

Also, the resulting tree will have height at most $h + \lceil \log m \rceil$.

Proof. By Lemma 3.3 we can assign $O(m)$ processors to each R_i and construct a hull tree H_i' for $UH(R_i) \cap UH(S)$ in $O(h + \log m)$ time. We can then perform a parallel prefix computation to remove any empty trees from the list $H_1', H_2', \ldots, H_m'$. This takes $O(\log m)$ time using $O(m/\log m)$ processors. Let $H_{i_1}', H_{i_2}', \ldots, H_{i_k}'$ be the resulting list of nonempty hull trees. We then construct a hull tree $HT(S)$ by building a complete binary tree 'on top' of the H_{i_j}''s (that is, each leaf of this tree is the root of an H_{i_j}'). This new hull tree clearly has maximum height at most $\lceil \log m \rceil + h$. The total time is clearly $O(h + \log m)$ and the number of processors is $O(m^2)$. □

In the next section we show how the hull tree data structure can be used to find the upper hull of an x-sorted point set.

4. The upper hull algorithm

Our method for constructing the upper hull $UH(S)$ of an x-sorted planar point set S is the following. We call the procedure ***MakeHullTree***, defined below, passing it the set S and the integer $\lceil \log n \rceil$, where $n = |S|$. This procedure constructs a hull tree $HT(S)$ for $UH(S)$ with height at most $O(\log n)$. We can then construct the array $UH(S)$ from the hull tree $HT(S)$ by a parallel prefix procedure which we describe at the end of this section. The algorithm, which is given below, will construct a hull tree $HT(S)$ for $UH(S)$ in $O(d + \log n + \log d \log\log n)$ time using $O(n/d)$ processors, where d is any positive integer.

Algorithm ***MakeHullTree***(S, d)

Input: A set S of n points in the plane, sorted by increasing x-coordinate, and an integer d.

Output: A hull tree $HT(S)$ for $UH(S)$.

Method: Our algorithm is based on the lemmas of Section 2, and uses the $\sqrt{n}$-divide-and-conquer technique [1,2]. The divide-and-conquer method we use differs from that of [1,2], however, in that we divide based on the value of the integer parameter d. In addition, we stop the recursion when the problem size is less than d and solve the remaining subproblems sequentially. This allows us to get by with only $O(n/d)$ processors. The details are given below.

***Step* 1.** If the number of points in S is at most d, then find the upper hull $UH(S)$ of each S and construct a hull tree $HT(S)$ for $UH(S)$ sequentially. Constructing $UH(S)$ in this case can be done in $O(d)$ time [10], after which we can clearly construct a hull tree for $UH(S)$ of height $\lceil \log d \rceil$ in $O(d)$ additional time. This completes the computation for this case, so for the remainder of this algorithm we assume that S contains more than d points.

Step* 2.** For simplicity of notation we let $N = \lceil n/d \rceil$. Partition S into an x-sorted collection of $\lceil \sqrt{N} \rceil$ subsets $R_1, R_2, \ldots, R_{\lceil \sqrt{N} \rceil}$, each of which has size $O(\sqrt{nd})$. Recursively call procedure ***Make-HullTree(R_i, d) for each R_i in parallel. After this parallel recursive call returned, we have a hull tree representing each $UH(R_i)$.

***Step* 3.** Construct a hull tree representing $UH(S)$ from the hull trees $HT(R_1), \ldots, HT(R_{\lceil \sqrt{N} \rceil})$. This is done using Lemma 3.4 with $m = \sqrt{n/d}$, and takes $O(h + \log n)$ time using $O(n/d)$ processors, where

$$h = \max_{1 \leqslant i \leqslant N} \{ height(HT(R_i)) \}.$$

end

We analyze Algorithm ***MakeHullTree*** in the following lemma.

4.1. Lemma. *Given a set S of n points in the plane sorted by increasing x-coordinate, Algorithm* ***MakeHullTree*** *constructs a hull tree representing $UH(S)$ in $O(d+\log n+\log d \log\log n)$ time using $O(\lceil n/d\rceil)$ processors in the CREW PRAM model. The hull tree it produces has maximum height of $2\log n$.*

Proof. The maximum height of the produced hull tree, $h(n)$, the running time, $T(n)$, of the algorithm, and the number of processors, $P(n)$, can be expressed in the following recurrence relations:

$$h(n)=\begin{cases}\lceil \log n\rceil & \text{if } n\leqslant d,\\ h(\sqrt{nd})+\lceil \log\sqrt{n/d}\,\rceil & \text{otherwise,}\end{cases}$$

$$T(n)=\begin{cases}b_1 d \quad \text{if } n\leqslant d,\\ T(\sqrt{nd})+b_2\big(\log n+h(\sqrt{nd})\big)\\ \qquad \text{otherwise,}\end{cases}$$

$$P(n)=\begin{cases}1 \quad \text{if } n\leqslant d,\\ \max\{\lceil n/d\rceil, \sqrt{n/d}\,P(\sqrt{nd})\}\\ \qquad \text{otherwise,}\end{cases}$$

where b_1 and b_2 are constants. These equations imply that $h(n)\leqslant 2\log n$, they imply that $T(n)$ is $O(d+\log n+\log d\log\log n)$, and that $P(n)$ is $O(\lceil n/d\rceil)$ [9]. This completes the proof. □

Thus, by assigning $d=\lceil\log n\rceil$ we have that we can construct a hull tree for $UH(S)$ with height $O(\log n)$ in $O(\log n)$ time using $O(n/\log n)$ processors in the CREW PRAM model. Now we only have to show how to construct the array $UH(S)$ from $HT(S)$ in $O(\log n)$ additional time using $O(n/\log n)$ processors.

After the hull tree for $UH(S)$ is constructed we can build $UH(S)$ by the following method. For each processor $i\in\{0,1,\dots,\lceil m/\log n\rceil\}$ we locate the leaf of H which has rank $i\lceil\log n\rceil$, using the d label stored at each node in the tree to direct the search. This takes $O(\log n)$ time. Now, for each processor i, we can follow *succ* pointers from this point to find the next $\lceil\log n\rceil$ entries in the hemispherical chain (in parallel for each processor i). Thus, we can compute for each leaf of H how many vertices precede it. Thus, we can convert the HQ-tree representation to an array representation by writing each vertex to its position in the array. This can all clearly be done in $O(\log n)$ time using $O(m/\log n)$ processors.

4.2. Theorem. *The convex hull of an x-sorted point set can be constructed in $O(\log n)$ time using $O(n/\log n)$ processors in the CREW PRAM computational model.*

5. Conclusion

We have shown how to solve the planar convex hull problem in $O(\log n)$ time using $O(n/\log n)$ processors for the case when the input points are given in sorted order, which is optimal. This, of course, immediately implies that the convex hull of a monotone polygon can be found in these same bounds. Recall that a polygon P is *monotone* with respect to a line L if every perpendicular to L intersects the boundary of P in at most two points. Another corollary of our result is that the common intersection of n half-planes sorted by their slopes can be constructed in $O(\log n)$ time using $O(n/\log n)$ processors, by using the duality transformation of [4,19]. We achieved these optimal bounds by using a parallel data structure which we call the hull tree. Constructing the convex hull of a point set has many applications, and we suspect that hull trees can be used to find efficient parallel algorithms for many other problems involving sorted point sets.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlang and C. Yap, Parallel computational geometry, Proc. 25th IEEE Symp. on Foundations of Computer Science (1985) 468–477.

[2] M.J. Atallah and M.T. Goodrich, Efficient parallel solutions to some geometric problems, J. Parallel & Distributed Comput. 3 (1986) 492–507.

[3] M.J. Atallah and M.T. Goodrich, Parallel algorithms for some functions of two convex polygons, Proc. 24th Allerton Conf. on Communication, Control and Computing (1986) 758–767.

[4] K.Q. Brown, Geometric Transformations for Fast Geometric Algorithms, Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, December 1979 (cited in [14]).

[5] B. Chazelle, Computational geometry on a systolic chip, IEEE Trans. Comput. C-33 (9) (1984) 774–785.

[6] A. Chow, Parallel Algorithms for Geometric Problems, Ph.D. Thesis, Univ. of Illinois at Urbana-Champain, December 1980.

[7] 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 Computation (1986) 206–219.

[8] H. El Gindy, A parallel algorithm for triangulating simplical point sets in space with optimal speed-up, Proc. 24th Allerton Conf. on Communication, Control, and Computing, 1986.

[9] M.T. Goodrich, Efficient Parallel Techniques for Computational Geometry, Ph.D. Thesis, Dept. of Computer Science, Purdue Univ., 1987.

[10] R.L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Inform. Process. Lett. 1 (1972) 132–133.

[11] D.G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm?, SIAM J. Comput. 15 (1) (1986) 287–299.

[12] C.P. Kruskal, L. Rudolph and M. Snir, The power of parallel prefix, Proc. 1985 Internat. Conf. on Parallel Processing, St. Charles, IL (1985) 180–185.

[13] R.E. Ladner and M.J. Fischer, Parallel prefix computation, J. ACM 24 (4) (1980) 831–838.

[14] D.T. Lee and F.P. Preparata, Computational geometry—A survey, IEEE Trans. Comput. C-33 (12) (1984) 1072–1101.

[15] R. Miller and Q.F. Stout, Computational geometry on a mesh-connected computer, Proc. 1984 IEEE Internat. Conf. on Parallel Processing (1984) 66–73.

[16] M.H. Overmars and J. Van Leeuwen, Maintenance of configurations in the plane, J. Comput. System Sci. 23 (1981) 166–204.

[17] F.P. Preparata, An optimal real-time algorithm for planar convex hulls, Comm. ACM 22 (7) (1979) 402–405.

[18] F.P. Preparata and S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, Comm. ACM 20 (2) (1977) 87–93.

[19] F.P. Preparata and D.E. Muller, Finding the intersection of n half-spaces in time $O(n \log n)$, Theoret. Comput. Sci. 8 (1979) 45–55.

[20] A.C. Yao, A lower bound to finding convex hulls, J. ACM 28 (1981) 780–787.