Two theorems on the parallel construction of convex hulls*

Jyrki Katajainen
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen East, Denmark
email: jyrki@diku.dk

Abstract

The parallel complexity of the problem of constructing the convex hull of a sorted planar point set is studied. For any point p in the plane, let $x(p)$ and $y(p)$ denote the x- and y-coordinate of p. A planar point set $S = \{p_1, p_2, \ldots, p_N\}$ is said to be *x-sorted* if the points of S are given by increasing x-coordinate, *i.e.*, $x(p_i) \leq x(p_{i+1})$ for all $i \in \{1, 2, \ldots, N-1\}$. The following two results are proved:

(1) Given an x-sorted set S of N points, the convex hull of S can be found in $O(\log N)$ time and $O(N)$ space with $\lceil N/\log N \rceil$ processors on an EREW PRAM.

(2) Given an x-sorted set S of N points, a padded representation of the convex hull of S can be computed in $O(\log\log N)$ time and $O(N)$ space with $\lceil N/\log\log N \rceil$ processors on a WEAK CRCW PRAM. (If the number of points in the convex hull is h, then the full representation of the convex hull is output in an array of size $O(\min\{h^{1+\varepsilon}, N\})$, for any fixed $\varepsilon > 0$.)

It is also shown that these algorithms are asymptotically fastest possible in their respective machine model if we insist on cost-optimality, *i.e.*, that the product of the time complexity and the number of processors used is linear.

1 Introduction

Given a set S of N points in the plane. The *convex hull* of S is the smallest convex region containing all points of S. We use $CH(S)$ to denote the boundary of the convex hull of S. The set E of *extreme points* of S is the smallest subset of S having the property that $CH(E) = CH(S)$. We recognize two versions of the problem of constructing the convex hull of a planar point set.

> EXTREME POINTS: Given a set S of N points in the plane, report for each point $p \in S$ whether it is an extreme point or not.
>
> CONVEX HULL: Given a set S of N points in the plane, produce a complete description of $CH(S)$, *e.g.*, as a circular doubly-linked list.

Kirkpatrick and Seidel [21] showed that the CONVEX HULL problem can be solved in $O(N \log h)$ sequential time, where h denotes the number of *extreme points*. Moreover, they proved that, in the algebraic-computation-tree model of computation, any algorithm solving the EXTREME POINTS problem requires $\Omega(N \log h)$ time.

*After the first writing of this paper, the author was informed that the first result has been proved earlier by Atallah, Chen, and Wagener [5]; and the second result by Berkman, Schieber, and Vishkin [8].

Let $x(p)$ and $y(p)$ denote the x-coordinate and y-coordinate of the point p, respectively. The set $S = \{p_1, p_2, \ldots, p_N\}$ is said to be *x-sorted* if $x(p_1) \leq x(p_2) \leq \cdots \leq x(p_N)$. For x-sorted sets, both of the EXTREME POINTS and CONVEX HULL problems can be solved in $O(N)$ time (see, *e.g.*, [4, 17, 22]).

In this paper we study the parallel compexity of the problem of constructing the convex hull of an x-sorted planar point set. To be more specific, we consider algorithms running in the *real* PRAM model (for the definition of the model, see, for example, the book by JáJá [19]). This special case of the CONVEX HULL problem has previously been studied by various researchers [5, 8, 15, 16].

On a CREW PRAM (and hence also on an EREW PRAM), any algorithm that solves the EXTREME POINTS or CONVEX HULL problem for a set of size N requires $\Omega(\log N)$ time, even if we had arbitrarily many processors with arbitrary power (cf. Section 2). Goodrich [16] gave a CREW-algorithm that constructs the convex hull of an x-sorted set of size N in $O(\log N)$ time and $O(N)$ space with $\lceil N/\log N \rceil$ processors. So his algorithm is asymptotically optimal with respect to time, space, and cost (the product of the time complexity and the number of processors used) in the CREW PRAM model.

In this paper we present a new algorithm for finding the convex hull of an x-sorted set that has the same resource requirements as Goodrich's algorithm, but it runs on an EREW PRAM.[1] A combination of the *fourth-root divide-and-conquer* and *binary divide-and-conquer* techniques is used in the algorithm. Yet another naive — but powerful — technique used is that of *copying*. By creating copies of the data that would have been otherwise accessed concurrently, concurrent reads can be avoided. The forth-root divide-and-conquer helps us in keeping the number of needed copies few.

Observe that Goodrich's algorithm as well as those presented in [2] and [6] are based on the square-root divide-and-conquer strategy. Also the binary divide-and-conquer [7] and cascading divide-and-conquer [11] techniques have been used to solve the CONVEX HULL problem. The potential power of the ith-root divide-and-conquer strategy, for $i > 2$, seems to be neclected in the computing literature, even though it has been used in some earlier works (see, for example, [5, 15]). The fact that the technique is useful when developing EREW-algorithms is, however, a bit surprising.

By allowing concurrent writes, faster solutions to the CONVEX HULL problem are possible. Akl [3] observed that all extreme points can be identified in constant time with N^4 processors.[2] Fjällström *et al.* [15] developed an algorithm that solves the CONVEX HULL problem for x-sorted sets in $O(\log N/\log\log N)$ time with optimal $O(N)$ cost. The problem left open in [15] was whether one can solve the EXTREME POINTS problem by a faster cost-optimal algorithm. In this paper an affirmative answer to this question is given: The extreme points of an x-sorted set of size N can be identidied in $O(\log\log N)$ time and $O(N)$ space with $\lceil N/\log\log N \rceil$ processors on a WEAK CRCW PRAM. [3] (That is, we use the weakest COMMON CRCW PRAM model, in which only concurrent writes of the value one are allowed.) We prove also that under certain conditions the above algorithm is asymptotically fastest possible even if N processors would be available.

Observe that, with a polynomial number of processors, $\Omega(\log N/\log\log N)$ is a lower bound for the problem of counting h, the number of extreme points, or for the problem of reporting the extreme points in an array of size h. We show, however, that a *padded*

[1] The same result has been proved earlier by Atallah, Chen, and Wagener [5], but the implementation details of their algorithm differs from those of the algorithm to be presented in this paper.

[2] In [3] the model of computation was not a PRAM, but the algorithm given there is readily implemented in constant time in any CRCW PRAM variant (*e.g.*, on a WEAK CRCW PRAM) that is able to compute the AND function of N bits in $O(1)$ time with N processors.

[3] A similar result has been proved earlier by Berkman, Schieber, and Vishkin [8].

representation of the convex hull can be computed as efficiently as the extreme points. As an output we get the convex hull in an array of size $O(\min\{h^{1+\varepsilon}, N\})$, for any fixed $\varepsilon > 0$.

The lower bounds for the parallel complexity of the EXTREME POINTS and CONVEX HULL problems are presented in Section 2. The EREW-algorithm is described and analysed in Section 3. The CRCW-algorithms are given in Section 4.

2 Lower bounds

It is clear that the CONVEX HULL problem cannot be easier than the EXTREME POINTS problem. Therefore, a lower bound for the parallel time complexity of the latter problem is only given.

Theorem 2.1. *Let S be an x-sorted set of N points in the plane. Any algorithm that identifies the extreme points of S requires*

(1) $\Omega(\log N)$ time on an CREW PRAM, even if arbitrarily many processors with arbitrary computing power are available;

(2) $\Omega(\log\log N)$ time on a PRIORITY CRCW PRAM, if at most N processors are available.

Proof. The problem of computing the maximum of n bits/integers can be reduced to the problem of identifying the extreme points of a set of size $O(n)$. The exact reductions are omitted here. The basic observation is that even if the points are given in x-sorted order, the order in y-direction can be arbitrary. The claimed lower bounds follow from the corresponding lower bounds known for the maximum problem on a CREW PRAM [12] and a PRIORITY CRCW PRAM [14]. □

The proof of the above theorem leaves open following questions:

(1) We assumed that only the x-sorted order of the points is given as input. On a CRCW PRAM, a faster solution might be possible for the special case where the input sequence is monotonic both in the x- and y-directions.

(2) The lower bound of Fich *et al.* [14] for computing the maximum is based on a Ramsay theoretic argument, and therefore to be valid the input numbers must be of huge size (double exponential). It might be possible to devise a faster CRCW-algorithm for the special case, in which the coordinates of the input points are small integers.

3 A cost-, time-, and space-optimal EREW-algorithm

In this section our purpose is to give an algorithm for finding the convex hull of an x-sorted set which is asymptotically optimal with respect to time, space, and processor utilization. More precisely, we prove the following

Theorem 3.1. *Given an x-sorted set S of N points in the plane, the convex hull of S can be constructed in $O(\log N)$ time and $O(N)$ space with $\lceil N/\log N \rceil$ processors on an EREW PRAM.*

As an immediate consequence we get

Corollary 3.2. *Given a set S of N points in the plane, the convex hull of S can be computed in $O(\log N)$ time and $O(N)$ space with N processors on an EREW PRAM.*

Proof. The points of S can be sorted with respect to their x-coordinates in $O(\log N)$ time and $O(N)$ space with N processors by using the parallel mergesort routine [10]. These resource requirements dominate the costs since, by Theorem 3.1, after x-sorting only $\lceil N/\log N \rceil$ processors are needed for constructing the convex hull of S in logarithmic time. □

Remark 3.3. An interesting open problem is whether one can develop an adaptive algorithm for constructing convex hulls such that the logarithmic running time is guaranteed by using at most $\lceil (N \log h)/\log N \rceil$ processors, where h denotes the number of extreme points. Observe that h is not known beforehand, so a dynamic processor allocation should be possible.

The rest of this section is devoted to the proof of Theorem 3.1.

We start with some preliminary definitions. Let $S = \{p_1, p_2, \ldots, p_N\}$ be an arbitrary x-sorted set of N points in the plane. We assume that the points of S are given in an array. For the sake of simplicity, we do not make any distinction between the set S and the array storing the points of S. As done in many previous convex-hull algorithms, we divide the computation of $CH(S)$ into two parts. First, we compute the *upper hull* of S, denoted $UH(S)$, which consists of those points of $CH(S)$ that lie above the line going through the points p_a and p_b, where p_a is the point with the maximum y-coordinate among the points whose x-coordinate equals to $x(p_1)$, and p_b is the point with the maximum y-coordinate among the points whose x-coordinate equals to $x(p_N)$. Second, we compute the *lower hull* of S, $LH(S)$, which is defined analogously with the upper hull. $CH(S)$ is easily constructed in $O(1)$ by a single processor from the monotone subchains $UH(S)$ and $LH(S)$. For the sake of symmetry, we may restrict our presentation to the computation of $UH(S)$.

We can make the simplifying assumption that no two points of S have the same x-coordinate. If this is not the case, the points having the same x-coordinate lie in consecutive positions in the input array, and it is a simple matter to select the point with the largest y-coordinate within each segment of points with equal x-coordinate. When many points agree in their x-coordinates, only the point with the largest y-coordinate can be part of $UH(S)$. The standard tool (see, *e.g.*, [19, Exercise 2.5]) that we need here is an algorithm for computing the segmented prefix sums (with the assosiative operation max).

Fix $d = \lceil \log N \rceil$. For two subsets S_1 and S_2 of S, we denote $S_1 < S_2$ if the x-coordinates of all points in S_1 are smaller than the x-coordinate of any point in S_2. To represent an upper hull of a subset $S' \subseteq S$, we use the simple pointer-based data structure introduced by Fjällström *et al.* [15]. We call the data structure here a *compact collection of d-hulls*, or simply a *d-collection*. Assume that a subset S' of S is divided into subsets $S_1, S_2, \ldots, S_m$, such that the size of each subset is at most d, $S_1 < S_2 < \ldots < S_m$, and $\bigcup_{i=1}^{m} S_i = S'$. Further, let $S'_1, S'_2, \ldots, S'_{m'}$ denote those of the subsets $S_1, S_2, \ldots, S_m$ that have points on $UH(S')$. A d-collection of S' has two levels:

(1) At the bottom level, for each $i \in \{1, 2, \ldots, m'\}$ we store the points of $UH(S'_i) \cap UH(S')$ in x-sorted order in a subarray of size at most d.

(2) At the top level, we have an array of size m, where only the first m' entries are in use. The jth entry, $j \in \{1, 2, \ldots, m'\}$, contains two pointers to the corresponding subset S'_j in the bottom level specifying the portion of $UH(S'_j)$ that belong to $UH(S')$. If S_i does not have any points on $UH(S')$, there is no entry corresponding to this subset.

As a preprocessing step, the input set S is divided into pieces of size d (except perhaps the last one) and the convex hulls of these pieces are determined (in parallel) by applying

any linear-time sequential algorithm [4, 17, 22]. Hence, we need here at most $\lceil N/\log N \rceil$ processors and use $O(d) \subseteq O(\log N)$ time. The input to our upper hull algorithm is the above collection of d-hulls (an array of pointers to d-hulls) and the algorithm will then compute the upper hull (represented as a d-collection) for the points in the d-hulls.

When combining the upper hulls of the sets S_1 and S_2 that are disjoint, *e.g.*, $S_1 < S_2$, the main task is to compute the *upper common tangent* to the convex chains $UH(S_1)$ and $UH(S_2)$. For this purpose, the following results are useful for us.

Lemma 3.4. *Let $S = \{p_1, p_2, \ldots, p_m\}$ be a set of m points in the plane, ordered by x-coordinate. Given the upper hull of $S_1 = \{p_1, \ldots, p_i\}$ and $S_2 = \{p_{i+1}, \ldots, p_m\}$, for some i, $i \in \{1, 2, \ldots, m\}$, the upper hull of S can be computed in $O(\log m)$ sequential time, provided that both $UH(S_1)$ and $UH(S_2)$ are represented, e.g., as an* augmented 2-3 tree*, in which each node has pointers to its leaves with smallest and largest x-coordinate, respectively. (For the definition of 2-3 trees, see, e.g., [1].)*

Lemma 3.5. *Let $S = \{p_1, p_2, \ldots, p_m\}$ be a set of m points in the plane, ordered by x-coordinate. Given the upper hull of $S_1 = \{p_1, \ldots, p_i\}$ and $S_2 = \{p_{i+1}, \ldots, p_m\}$, for some i, $i \in \{1, 2, \ldots, m\}$, the upper common tangent of $UH(S_1)$ and $UH(S_2)$ can be found in $O(c^2)$ time with $\lceil m^{1/c} \rceil$ processors on a CREW PRAM, provided that $UH(S_1)$ and $UH(S_2)$ are given compactly in arrays.*

The latter lemma has an immediate corollary stated below.

Corollary 3.6. *Let $S = \{p_1, p_2, \ldots, p_m\}$ be a set of m points in the plane, ordered by x-coordinate. Given the upper hulls of $S_1 = \{p_{i_0+1}, \ldots, p_{i_1}\}$, $S_2 = \{p_{i_1+1}, \ldots, p_{i_2}\}$, $\cdots$, $S_k = \{p_{i_{k-1}+1}, \ldots, p_{i_k}\}$, where $i_0 = 0, i_k = m$, and $i_j \leq i_{j+1}$ for all $j \in \{1, 2, \ldots, k-1\}$. For all $a, b \in \{1, 2, \ldots, k\}$, $a \neq b$, the upper common tangents of $UH(S_a)$ and $UH(S_b)$ can be found in $O(c^2)$ time with $k^2 \cdot \lceil m^{1/c} \rceil$ processors on a CREW PRAM, provided that every $UH(S_j)$ is given compactly in an array.*

We cannot apply the results mentioned above directly due to two reasons:

(1) The upper hulls are not represented as d-collections used by us, and

(2) Lemma 3.5 and Corollary 3.6 require the CREW PRAM model.

It is a simple matter to transform a d-collection representation of a convex chain to an augmented 2-3 tree. In our algorithm, we need only a linear-time sequential subroutine for doing this, but a fast parallel routine could also be developed (cf. [19, Exercise 2.32]). The transformation of a 2-3 tree to a d-collection is a bit more complicated. Katajainen [20] showed how the representation of a sorted set of n elements can be efficiently converted from a 2-3 tree to a sorted array of size n. Since, a d-collection is easily constructed from a sorted array, we have the following

Lemma 3.7. *Let C be a convex chain of m vertices. Given the 2-3 tree representation of C, a compact d-collection corresponding to C can be created in $O(\log m)$ time and $O(m)$ space with $\lceil m/\log m \rceil$ processors on an EREW PRAM.*

It is not difficult to see that the results of Lemma 3.5 and Corollary 3.6 are also valid if d-collections are used for representing the convex chains instead of arrays. The verification of this fact is left to the reader.

Let us now concentrate on the question, how the algorithms of Lemma 3.5 and Corollary 3.6 are made to run on an EREW PRAM. Here we use the well-known simulation result saying that a p-processor CREW-algorithm can be no more than $O(\log p)$ times faster than the best p-processor EREW-algorithm for the same problem. For the proof of the next lemma, see, for example, [13, Theorem 30.1].

Lemma 3.8. *A p-processor EREW PRAM can simulate a step of a p-processor CREW PRAM in $O(\log p)$ time by using $O(p)$ extra space.*

By Corollary 3.6 and Lemma 3.8, we have

Lemma 3.9. *Let $S = \{p_1, p_2, \ldots, p_m\}$ be a set of m points in the plane, ordered by x-coordinate. Given the d-collections corresponding to the upper hulls of $S_1 = \{p_{i_0+1}, \ldots, p_{i_1}\}$, $S_2 = \{p_{i_1+1}, \ldots, p_{i_2}\}$, $\cdots$, $S_k = \{p_{i_{k-1}+1}, \ldots, p_{i_k}\}$, where $i_0 = 0, i_k = m$, and $i_j \leq i_{j+1}$ for all $j \in \{1, 2, \ldots, k-1\}$. For all $a, b \in \{1, 2, \ldots, k\}$, $a \neq b$, the upper common tangents of $UH(S_a)$ and $UH(S_b)$ can be found in $O(c^2 \log k + c \log m)$ time with $k^2 \cdot \lceil m^{1/c} \rceil$ processors on an EREW PRAM.*

Now we are ready to present the algorithm for computing the upper hull for a collection of d-hulls. The main part of the algorithm uses the fourth-root divide-and-conquer technique. The problems of size less than d^4 are handled by using the binary divide-and-conquer technique.

Algorithm U-HULL
INPUT: A collection C of d-hulls of the sets $S_1, S_2, \ldots, S_{\lceil m/d \rceil}$, $S_1 < S_2 < \cdots < S_{\lceil m/d \rceil}$. Processors $P_1, P_2, \ldots, P_{\lceil m/d \rceil}$ are allocated for this task.
OUTPUT: An upper hull of the points in the d-hulls of C.
If $m < d^4$ **then**
Step 1.1 Construct an augmented 2-3 tree corresponding to each d-hull in C. Allocate one processor to each of these tasks.
Step 1.2 Combine the hulls paiwise level-by-level until only one tree T is left (cf. Lemma 3.4). Each combining is done by a single processor, but at every level the combinings are done in parallel.
Step 1.3 Transform T corresponding to $UH(\bigcup_{i=1}^{\lceil m/d \rceil} S_i)$ to a d-collection (cf. Lemma 3.7). Allocate all the available $\lceil m/d \rceil$ processors to this task.
elseif $m \geq d^4$ **then**
Step 2 Calculate $\lceil m^{1/4} \rceil$ and copy the answer to all processors.
Step 3 Divide the collection C into $\lceil m^{1/4} \rceil$ subcollections C_i and compute the upper hulls of the subcollections in parallel by calling the algorithm U-HULL recursively. Allocate $m^{3/4}/d$ processors to each subtask, *i.e.*, one processor per a d-hull in C_i. Let $UH(C_i)$ denote the outcomes of these computations.
Step 4 Compute the upper common tangents for each pair $UH(C_i), UH(C_j)$, $i, j \in \{1, 2, \ldots, \lceil m^{1/4} \rceil\}$, $i \neq j$ (cf. Lemma 3.9). Fix $k = \lceil m^{1/4} \rceil$ and $c = 4$, so that only the first $m^{3/4}$ of the $\lceil m/d \rceil$ processors are used here. (Recall that $m \geq d^4$.)
Step 5 Let $t_{i,j}$ denote the upper common tangent between $UH(C_i)$ and $UH(C_j)$. For each i, $i \in \{1, 2, \ldots, \lceil m^{1/4} \rceil\}$, compute the tangent ℓ_i with the smallest slope in $\{t_{i,1}, \ldots, t_{i,i-1}\}$ and the tangent r_i with the largest slope in $\{t_{i,i+1}, \ldots, t_{i,\lceil m^{1/4} \rceil}\}$.
Step 6 Take $m^{3/4}/d$ copies of every ℓ_i and r_i. Use these copies to update the top level pointers (if necessary) for every d-hull of $UH(C_i)$, $i \in \{1, 2, \ldots, \lceil m^{1/4} \rceil\}$.
Step 7 Remove those top level entries in the d-collection $UH(C_i)$ that point to an empty set. Create a new d-collection containing all non-empty d-hulls in $\bigcup_{i=1}^{\lceil m^{1/4} \rceil} UH(C_i)$, *e.g.*, by using a parallel prefix subroutine.
end if
end Algorithm U-HULL

As a post-processing step, the points in the obtained d-collection are moved to an array. Here a prefix sum routine is again needed. This step requires $O(\log N)$ time and $O(N)$ space with $\lceil N/\log N \rceil$ processors.

The overall structure of the algorithm U-HULL is similar to that of the algorithm by Atallah and Goodrich [6]. They used square-root divide-and-conquer instead of fourth-root divide-and-conquer. Therefore, the correctness of our algorithm follows immediately from the correctness of this earlier algorithm.

We used fourth-root divide-and-conquer to avoid concurrent reads. The critical point is Step 4 but Lemma 3.9 already showed how concurrent reads can be avoided there. In all other steps standard tools (prefix sums computation, copying, minimum/maximum finding) are used and these are known to run in the EREW PRAM model.

Let us now analyse the running time of the algorithm U-HULL. In Steps 1.1–3 we have $\lceil m/d \rceil$ d-hulls and $\lceil m/d \rceil$ processors. In Step 1.1, an augmented 2-3 tree corresponding to a d-hull is constructed in $O(d)$ time, and with $\lceil m/d \rceil$ processors all d-hulls can be handled in parallel. According to Lemma 3.4, Step 1.2 requires $O(\log d)$ time and this is done $O(\log d)$ times, which totals $O(\log^2 d)$. By Lemma 3.7 and the self-simulating property of EREW PRAMs, Step 1.3 is executed in $O(d)$ time with $\lceil m/d \rceil$ processors. Hence, the total time required by Steps 1.1–3 is $O(d)$ with $\lceil m/d \rceil$ processors.

Step 2 takes $O(\log m)$ time with $\lceil m/d \rceil$ processors. Observe that $\lceil m^{1/4} \rceil$ can be computed without the square-root operation if, after distributing m to all processors, each processor P_i computes i^4 and then the processor P_j who observes of being the first, for which $j^4 \geq m$, reports its number to other processors. Here copying is again needed but it will take $O(\log m)$ time as required.

The processor allocation in Step 3 can be done statically since each processor knows m. Hence, Step 3 takes $T(\lceil m^{3/4} \rceil) + O(1)$ time with $\lceil m/d \rceil$ processors, where $T(m)$ denotes the time needed for the whole algorithm.

According to Lemma 3.9, Step 4 uses $O(\log m)$ time with $\lceil m^{3/4} \rceil$ processors. The minimum and maximum finding in Step 5, copying in Step 6, and compaction in Step 7 can be solved by standard routines that all require $O(\log m)$ time with $\lceil m/d \rceil$ processors. Since the total number of items involved in these computations is only $\lceil m/d \rceil$, a fewer number of processors would be enough to execute these steps.

To sum up, we have the recurrence

$$T(m) = \begin{cases} T(\lceil m^{3/4} \rceil) + O(\log m) & \text{if } m \geq d^4 \\ O(d) & \text{if } m < d^4 \end{cases}$$

for the execution time of the algorithm U-HULL. Therefore, the computation of the upper hull for $\lceil N/d \rceil$ d-hulls ($d = \lceil \log N \rceil$) takes time

$$T(N) \leq O(\sum_{i=0}^{\infty} (\frac{3}{4})^i \log N) + O(\log N) \subseteq O(\log N).$$

Hence, the upper hull for a set of size N can be computed in $O(\log N)$ time, since all pre-processing, the algorithm U-HULL, and all post-processing can be executed in that time. No more than $\lceil N/\log N \rceil$ processors are employed in any step. We will not give a detailed analysis of the space requirements, but it is not difficult to see that $O(N)$ space is used. This completes the proof of Theorem 3.1.

4 A faster cost- and space-optimal CRCW-algorithm

The algorithm of Section 3 can be made faster on a WEAK CRCW PRAM. However, the compaction will be a hidden bottleneck in the algorithm. To avoid this, we show first how the extreme points of an x-sorted planar point set are identified fast. When all extreme

points are available in x-sorted order, a padded representation of the convex hull can be computed efficiently.

We utilize in our algorithm more or less standard CRCW-subroutines, whose efficiency is summarized in the next two lemmas. In both lemmas the model of computation is a WEAK CRCW PRAM.

Lemma 4.1. [24] *The maximum (minimum) of m elements can be computed in*

(1) $O(1)$ time and $O(m^2)$ space with m^2 processors;

(2) $O(\log\log m)$ time and $O(m)$ space with $\lceil m/\log\log m\rceil$ processors.

Lemma 4.2. [9] *Given a bit-array B of size m, for each position of B the index to the nearest previous 1-bit can be computed in*

(1) $O(1)$ time and $O(m\log\log m)$ space with $\lceil m\log\log m\rceil$ processors;

(2) $O(\log\log m)$ time and $O(m)$ space with $\lceil m/\log\log m\rceil$ processors.

Remark that all results stated in the above lemmas are not the best possible, but we gave these in the form we shall use them.

Let S be any x-sorted set of N points given as input. Fix $e = \lceil\log\log N\rceil^2$. To be able to utilize the algorithm of Lemma 3.5, when computing extreme points, we use a variant of the d-collection data structure, called a *chained collection of e-hulls*, or simply an *e-collection*. As in compact d-collections, this new data structure has two levels. The bottom level is as before but now the size of the subsets is bounded by e. If some of the subsets does not contain any extreme point, at the top level we simply mark the subset *non-active* and add a new pointer to the nearest previous subset that is still *active*. To update these pointers we use Lemma 4.2.

It is not difficult to show that the results similar to those given in Lemma 3.5 and Corollary 3.6 are also valid for e-collections. The verification of this fact is left to the interested reader.

Now the basic algorithm for identifying the extreme points of S is similar to the algorithm U-HULL. Again the computation is divided into two parts. First, we identify the extreme points on the upper hull of S and then those on the lower hull of S. For the sake of symmetry, only the algorithm for identifying upper hull points is given here.

We can further assume that the x-coordinates of the input points are distinct. If this is not the case, the point with the maximum y-coordinate, within every segment of points having equal x-coordinates, is identified and those points that are dominated by some other point with a larger y-coordinate are simply discarded from consideration (but the input array is not compacted). The segment maximums are found by using the part (2) of Lemma 4.1. The processors are allocated to each subtask by using the part (2) of Lemma 4.2.

In the preprocessing step, the input points are divided into disjoint pieces of size e (except perhaps the last one) and the convex hull of every piece is computed. We allocate $\lceil\log\log N\rceil$ processors for each piece and use the parallel algorithm developed in Section 3 to construct its convex hull. Since a WEAK CRCW PRAM is self-simulating, the computation can be slowed down such that this preprocessing uses $O(\log\log N)$ time (and $O(N)$ space) with $\lceil N/\log\log N\rceil$ processors.

Since concurrent reads are allowed on a CRCW PRAM, the description of the algorithm can be simplified considerably. Now all copying is no more necessary. In the CRCW-algorithm given below, we describe only those steps that are different from the corresponding steps of the algorithm U-HULL.

Algorithm U-POINTS
INPUT: A collection C of e-hulls of the sets $S_1, S_2, \ldots, S_{\lceil m/e \rceil}$, $S_1 < S_2 < \cdots < S_{\lceil m/e \rceil}$. Processors $P_1, P_2, \ldots, P_{\lceil m/e^{1/2} \rceil}$ are allocated for this task.
OUTPUT: An upper hull of the points in the e-hulls of C.
If $m < e^4$ **then**
Steps 1.1–3 Otherwise as in the algorithm U-HULL, but the outcome is an e-collection.
elseif $m \geq e^4$ **then**
Step 2 Calculate $\lceil m^{1/4} \rceil$.
Step 3 Divide the collection C into $\lceil m^{1/4} \rceil$ subcollections C_i and compute the extreme points of the subcollections in parallel by calling the algorithm U-POINTS recursively. Allocate $m^{3/4}/e^{1/2}$ processors to each task. Let $EP(C_i)$ denote the outcomes of these computations.
Step 4 Compute the upper common tangents for each pair $EP(C_i), EP(C_j)$, $i, j \in \{1, 2, \ldots, \lceil m^{1/4} \rceil\}, i \neq j$ (cf. Corollary 3.6). Fix $k = \lceil m^{1/4} \rceil$ and $c = 4$, so that only the first $m^{3/4}$ of the $\lceil m/e^{1/2} \rceil$ processors are used here. (Recall that $m \geq e^4$.)
Step 5 As in the algorithm U-HULL.
Step 6 Update the top level pointers (if necessary) for each e-hull of $EP(C_i)$, $i \in \{1, 2, \ldots, \lceil m^{1/4} \rceil\}$.
Step 7 Create a new e-collection containing all e-hulls in $\bigcup_{i=1}^{\lceil m^{1/4} \rceil} EP(C_i)$. For each non-active e-hull, compute its nearest previous active e-hull.
end if
end Algorithm U-POINTS

As a post-processing step, each input point is marked to be an extreme point if is active in its e-hull in the computed e-collection. This is easily checked in $O(1)$ time per point, or if the computation is slowed down in $O(\log \log N)$ time with $\lceil N/\log \log N \rceil$ processors.

Finally, let us analyse the performance of the algorithm U-POINTS. As in the algorithm U-HULL, Steps 1.1–3 require $O(e)$ time in total. Since concurrent reads are allowed, Step 2 uses only $O(1)$ time. As earlier Step 3 needs $T(\lceil m^{3/4} \rceil) + O(1)$ time. According to Corollary 3.6, Step 4 takes now $O(1)$ time with $m^{3/4}$ processors. In Step 5 we can allocate $m^{1/2}$ processors for each minimum/maximum finding task. By the part (1) of Lemma 4.1, Step 5 uses $O(1)$ time with $m^{3/4}$ processors. Step 6 takes $O(1)$ time with m/e processors. In Step 7 the new e-collection can be created in $O(1)$ time with m/e processors. The most critical part of the algorithm is the point where the nearest previous active e-hulls are computed. Here we really need all the processors available. The new e-collection contain $\lceil m/e \rceil$ entries. By the part (2) of Lemma 4.2, with $(m/e) \log \log(m/e) \leq m/e^{1/2}$ processors the computation of the previous nearest active e-hulls can be carried out in $O(1)$ time. Hence, we obtain the recurrence

$$T(m) = \begin{cases} T(\lceil m^{3/4} \rceil) + O(1) & \text{if } m \geq e^4 \\ O(e) & \text{if } m < e^4 \end{cases}$$

for the running time of the algorithm U-POINTS. It is easy to see that $T(N) \in O(\log \log N)$.

The discussion above is now summarized in the following

Theorem 4.3. *Given an x-sorted set of N points, the extreme points of S can be identified in $O(\log \log N)$ time and $O(N)$ space with $\lceil N/\log \log N \rceil$ processors on a WEAK CRCW PRAM.*

By using the part (2) of Lemma 4.2, the extreme points can be chained together. Now by applying Ragde's compaction theorem (see [18, 23]) the extreme points can be compacted in $O(1)$ time with N processors to an array of size $O(\min\{h^{1+\varepsilon}, N\})$, where h denotes the number of extreme points and ε is a fixed positive constant. Thus, we have

Theorem 4.4. *Given an x-sorted set of N points, a padded representation of the convex hull of S can be computed in $O(\log\log N)$ time and $O(N)$ space with $\lceil N/\log\log N\rceil$ processors on a WEAK CRCW PRAM.*

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman: *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[2] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, C. Yap: Parallel computational geometry. *Algorithmica* **3** (1988) 293–327.

[3] S.G. Akl: A constant-time parallel algorithm for computing convex hulls. *BIT* **22** (1982) 130–134.

[4] A.M. Andrew: Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters* **9** (1979) 216–219.

[5] M.J. Atallah, D.Z. Chen, H. Wagener: An optimal parallel algorithm for the visibility of a simple polygon from a point. *Journal of the ACM* **38** (1991) 516–533.

[6] M.J. Atallah, M.T. Goodrich: Efficient parallel solutions to some geometric problems. *Journal of Parallel and Distributed Computing* **3** (1986) 492–507.

[7] M.J. Atallah, M.T. Goodrich: Parallel algorithms for some functions of two convex polygons. *Algorithmica* **3** (1988) 535–548.

[8] O. Berkman, B. Schieber, U. Vishkin: The parallel complexity of finding the convex hull of a monotone polygon. Unpublished manuscript.

[9] O. Berkman, U. Vishkin: *Recursive star-tree parallel data-structure.* Technical report **UMIACS-TR-90-40**, Institute for Advanced Computer Studies, University of Maryland, College Park, Md., 1990. A preliminary version appears in *Proc. of the 30th Annual Symposium on Foundations of Computer Science*, IEEE, 1989, pp. 196–202.

[10] R. Cole: Parallel merge sort. *SIAM Journal on Computing* **17** (1988) 770–785.

[11] R. Cole, M.T. Goodrich: Optimal parallel algorithms for point-set and polygon problems. *Algorithmica* **7** (1992) 3–23.

[12] S. Cook, C. Dwork, R. Reischuk: Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing* **15** (1986) 87–97.

[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms.* The MIT Press, 1990.

[14] F.E. Fich, F. Meyer auf der Heide, P. Ragde, A. Wigderson: One, two,... infinity: lower bounds for parallel computations. In *Proc. of the 17th Annual ACM Symposium on Theory of Computing*, ACM, 1985, pp. 48–58.

[15] P.-O. Fjällström, J. Katajainen, C. Levcopoulos, O. Petersson: A sublogarithmic convex hull algorithm. *BIT* **30** (1990) 378–384.

[16] M.T. Goodrich: Finding the convex hull of a sorted point set in parallel. *Information Processing Letters* **26** (1987) 173–179.

[17] R.L. Graham: An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters* **1** (1972) 132–133.

[18] T. Hagerup: On a compaction theorem of Ragde. *Information Processing Letters* **43** (1992) 335–340.

[19] J. JáJá: *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, 1992.

[20] J. Katajainen: Efficient parallel algorithms for manipulating sorted sets. Manuscript. A preliminary version will appear in *Proc. of the 17th Annual Computer Science Conference.*

[21] D.G. Kirkpatrick, R. Seidel: The ultimate planar convex hull algorithm? *SIAM Journal on Computing* **15** (1986) 287–299.

[22] M.H. Overmars, J. van Leeuwen: Maintenance of configurations in the plane. *Journal of Computer and System Sciences* **23** (1981) 166–204.

[23] P. Ragde: The parallel simplicity of compaction and chaining. *Journal of Algorithms* **14** (1993) 371–380.

[24] Y. Shiloach, U. Vishkin: Finding the maximum, merging, and sorting in a parallel computation model. *Journal of Algorithms* (1981) **2**, 88–102.