ELSEVIER

Computational Geometry 8 (1997) 151–166

Computational Geometry
Theory and Applications

Optimal, output-sensitive algorithms for constructing planar hulls in parallel

Neelima Gupta, Sandeep Sen *,[1]

Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi 110016, India

Communicated by M. Atallah; submitted 8 February 1995; accepted 1 June 1995

Abstract

In this paper we focus on the problem of designing very fast parallel algorithms for the planar convex hull problem that achieve the optimal $O(n \log H)$ work-bound for input size n and output size H. Our algorithms are designed for the arbitrary CRCW PRAM model.

We first describe a very simple $O(\log n \log H)$ time optimal deterministic algorithm for the planar hulls which is an improvement over the previously known $\Omega(\log^2 n)$ time algorithm for small outputs. For larger values of H, we can achieve a running time of $O(\log n \log\log n)$ steps with optimal work.

We also present a fast randomized algorithm that runs in expected time $O(\log H \cdot \log\log n)$ and does optimal $O(n \log H)$ work. For $\log H = \Omega(\log\log n)$, we can achieve the optimal running time of $O(\log H)$ while simultaneously keeping the work optimal. When $\log H$ is $o(\log n)$, our results improve upon the previously best known $\Theta(\log n)$ expected time randomized algorithm of Ghouse and Goodrich. The randomized algorithms do not assume any input distribution and the running times hold with high probability. © 1997 Elsevier Science B.V.

Keywords: Parallel algorithm; Computational geometry; Convex hull; Randomized algorithm

1. Introduction

Given a set $S = \{p_1, p_2, \ldots, p_n\}$ of n points, the convex hull of S is the smallest convex polygon containing all the points of S. The convex hull problem is to determine the ordered list $CH(S) \subseteq S$ defining the boundary of the convex hull of S.

The problem of constructing the convex hull has attracted a great deal of attention from the inception of computational geometry. Several sequential algorithms have been proposed for planar hull with the worst case time bound $O(n \log n)$ [22,30,31]. As the problem of sorting can be reduced to the convex hull problem, this is worst case optimal [38]. However, this is true only if the output size, i.e., the

* Corresponding author. E-mail: ssen@cse.iitd.ernet.in.

[1] Research supported in part by DST project SR/OY/M-06/94.

PII S0925-7721(97)00009-6

number of vertices on the hull, is large. More specifically, the time-bound of $\Theta(n \log n)$ is tight when the ordered output size is $\Omega(n)$. When the output size is much smaller, say constant, it is easy to design an $O(n)$ algorithm like *Jarvis' March*. This algorithm actually solves the problem in $O(nH)$ time, where H is the output size. Kirkpatrick and Seidel [28] designed an algorithm with a worst case time complexity $O(n \log H)$. They also showed that this is asymptotically optimal when the complexity is measured in terms of both the input and the output sizes (also see [27]). However, their algorithm is quite complex and is rarely used in practice. Chan et al. [9] discovered a simplified version of Kirkpatrick and Seidel's algorithm that is more practical to implement. Very recently Chan [8] presented a very elegant approach for output sensitive construction of convex hulls using ray-shooting that achieves $O(n \log H)$ running time.

1.1. Parallel algorithms

The primary objective of designing parallel algorithms is to obtain very fast solutions to problems keeping the total work (the processor-time product) close to the best sequential algorithms. For example if $S(n)$ is the best known sequential time complexity for input size n, then we aim for a parallel algorithm with $P(n)$ processors and $T(n)$ running time that minimizes $T(n)$ while keeping the work $P(n) \cdot T(n)$ close to $O(S(n))$. A parallel algorithm that actually does total work $O(S(n))$ is called a work-optimal or simply an optimal algorithm. Simultaneously, if the algorithm also matches the time lower bound then it is the best possible (asymptotically).

The fastest possible time-bound clearly depends on the parallel machine model. For example, in the case of CREW model, the convex hull cannot be constructed faster than $O(\log n)$ time, irrespective of the output-size because of an $\Omega(\log n)$ bound for computing maximum (minimum). The above bound does not apply to the CRCW model. For a stronger model, Sen [36] has obtained exact trade-off between number of processors and possible speed-up for a wide range of problems in computational geometry. For convex hulls, Lemma 1.1 is shown.

Lemma 1.1 [36]. *Any randomized algorithm in the parallel decision tree model for constructing convex hull of n points and output-size H, has a parallel time-bound of $\Omega(\log H / \log k)$ using kn processors, $k \geqslant 1$ in the worst case.*

In other words, for super-linear number of processors, a proportional speed-up is not achievable and hence these parallel algorithms cannot be considered efficient. The best or the *ultimate* that one can hope for under the circumstances is an algorithm that achieves $O(\log H)$ time using n processors.

The result of this paper makes progress towards achieving this end. It may be noted that, although we use the CRCW model, we do not make use of its full arithmetic instruction set and as such it is weaker than the parallel decision tree model. The parallel complexity of a problem is arguably better understood in this framework where communication is not a serious bottleneck.

1.2. Previous results

For planar hulls, in the context of PRAM (Parallel Random Access Machine) model, there exist a number of algorithms with $O(\log n)$ running time and $O(n \log n)$ operations [1,5,6,33]. These are known to be worst case optimal in the CREW model. Akl [2] describes an output-sensitive algorithm

for this problem which is optimal for number of processors bounded by $O(n^z)$, $0 < z < 1$. Deng [16] describes an algorithm that runs in $O(\log n)$ parallel time using $n/\log n$ processors when H is constant. The fastest $O(n \log H)$ work-optimal parallel algorithms have running times of $\Theta(\log^2 n)$ and $\Theta(\log n)$ in deterministic and randomized CRCW models respectively (Ghouse and Goodrich [18]).

1.3. Our results and methods

We present algorithms whose running times are output-sensitive even in the sublogarithmic time-range while keeping the work optimal. For designing fast output-sensitive algorithms, we have to cope with the problem that the output-size is an unknown parameter. Moreover, we also have to rapidly eliminate input points that do not contribute to the final output without incurring a high cost. The two most successful approaches used in the sequential context, namely gift-wrapping (or ray-shooting approach of Chan) and divide-and-conquer do not translate into fast parallel algorithms. By 'fast' we imply $O(\log H)$ or something very close. The gift-wrapping (or ray-shooting) is inherently sequential taking about $O(H)$ sequential phases. Even the divide-and-conquer method is not particularly effective as it cannot divide the *output* evenly – in fact this aspect is crux of the difficulty of designing fast output-sensitive algorithms that run in $O(\log H)$ time.

We first describe a deterministic $O(\log n \log H)$ time CRCW algorithm that does optimal work. This algorithm is based on an optimal sequential algorithm of Chan et al. [9] – an identical bound can be achieved by using the approach of Kirkpatrick and Seidel instead. A straightforward parallelization would give an $\Omega(\log n \cdot \text{poly}(\log \log n))$ algorithm, which will be slower than our algorithm for $H = o(\log n)$. For larger output sizes, we can make the algorithm run in $O(\log n \log \log n)$ steps. It may be noted here that the $\Theta(\log n)$ expected time algorithm of Ghouse and Goodrich was based on the algorithm of Kirkpatrick and Seidel.

In Section 3, we present fast randomized algorithms for the planar hulls. The fastest algorithm runs in $\widetilde{O}(\log H)$ expected time using n processors for $H > \log^{\varepsilon} n$, $\varepsilon > 0$. The expected running times hold with high probability[2]. For smaller output sizes, we present an algorithm that has an expected running time of $\widetilde{O}(\log H \cdot \log \log n)$ keeping the number of operations optimal. Therefore, for small output-sizes, our results improve the previously known bounds.

Our randomized algorithms are based on an approach of Clarkson and Shor [13] – this gives us a work-optimal algorithm as a starting point. However, its efficient adaptation in the parallel context required a number of sophisticated techniques like bootstrapping and super-linear processors parallel algorithms, and very fine-tuned analysis. The basic method of [13] prunes away the redundant points efficiently to a stage where number of points is small enough to run the worst-case algorithms. This is not true for a parallel algorithm where one cannot obtain commensurate speed-up with processor advantage (Lemma 1.1). This is one of the first non-trivial applications of super-linear processor algorithms in computational geometry to obtain speed-up for a situation where initially there is no processor advantage. Our work establishes a close connection between fast output-sensitive parallel algorithms and super-linear processor algorithms. Consequently, our algorithms become increasingly faster than the previous algorithms as the output size decreases. We are not aware of any previous

[2] In this paper, the term high probability implies probability exceeding $1 - 1/n^c$ for any predetermined constant c where n is the input-size. The notation that will be used is $\widetilde{O}$ instead of O to denote that the bound holds with high probability.

work where the parallel algorithms speed-up optimally with output size in the sublogarithmic time domain.

2. Deterministic algorithm for planar hulls

In this section we present a deterministic algorithm to compute the planar hulls. We assume for simplicity that no two vertices collide on x or y coordinates. We construct the $\mathrm{CH}(S)$ in two parts, the upper hull $\mathrm{UH}(S)$ and the lower hull $\mathrm{LH}(S)$. We shall describe the procedure for upper hull only; the procedure for lower hull is identical and merging is trivial.

Our algorithm uses the basic approach of [28]; however, we would like to minimize the number of stages required to reduce the total size of the problem sufficiently after which the problem is solved directly. Our presentation actually makes use of the simplification given in [9].

2.1. Algorithm

1. Find p and q with smallest and largest x-coordinate respectively.
   If $p = q$ then print(p) and stop.
2. U_Hull(S) where U_Hull(S) is
   begin(U_Hull)
   - If the *total size* of the problem ($(n/2^d)\cdot$ number of subproblems) at the dth stage is $\leqslant n/\log n$ then go to 3 (note that the *total size* refers to a global count of the sizes of all the subproblems at stage d and is not local to the recursive invocation of U_Hull);
     Else
     (a) Pair up the unpaired points in S.
     (b) Find an approximate median m of the slopes of the lines defined by the pairs.
     (c) Find the upper hull vertex p_m with a tangent of slope m;
     Print(p_m).
     (d) Compute $S_{\text{left}} = \{p \in S \mid x(p) < x(p_m)\}$, $S_{\text{right}} = \{p \in S \mid x(p) > x(p_m)\}$.
     (e) Discard points from S_{left} (respectively S_{right}) that are right (respectively left) end points of pairs with slope less (respectively greater) than m.
     (f) If S_{left} is not empty then U_Hull(S_{left}).
     (g) If S_{right} is not empty then U_Hull(S_{right}).
   end(U_Hull).
3. Solve each subproblem directly using any of $(n, \log n)$ algorithms [1,5,6,33].

The correctness of algorithm U_Hull follows from [9]. The processors executing print(p_i) statements write 1 in the respective cells of an array A and attach the respective point with it. This array, together with the output of solving the subproblems directly, is later compressed to give the output vertices in order.

2.2. Analysis

Lemma 2.1. *The maximum (minimum) of n elements can be found in $\mathrm{O}(\log\log n)$ time using $n/\log\log n$ CRCW processors.*

Lemma 2.2 [19]. *There is a CRCW algorithm that finds an element with rank k such that $n/3 \geqslant k \geqslant 2n/3$ with processor-time complexity $(n/\log\log n, \log\log n)$.*

Lemma 2.3 [19]. *Interval Allocation with $o(1)$ padding factor and ε-approximate prefix sum ($\varepsilon = o(1)$) can be done deterministically in $O(\log\log n)$ steps using $n/\log\log n$ CRCW processors.*

Remark. The above implies that processor allocation can be done in the same bounds.

Observation 2.1. At every stage, each subproblem has at least one output vertex.

Lemma 2.4. *The size of the problem reduces to $n/\log n$ after $O(\max\{\log\log n, \log H\})$ stages.*

Remark. In order to keep the notations simpler, we assume all logarithms are taken to base 3/2 in this lemma.

Proof. Let N denote the number of subproblems after $O(\max\{\log\log n, \log H\})$ stages. Since every subproblem has at least one output vertex therefore $N \leqslant H$.
Case 1. $H \leqslant \log n$, then total size after $2\log\log n$ stages is

$$\frac{n}{(3/2)^{2\log\log n}} \cdot N \leqslant \frac{n}{\log^2 n} \cdot H \leqslant \frac{n}{\log n}.$$

Case 2. $H \geqslant \log n$, then total size after $2\log H$ stages is

$$\frac{n}{(3/2)^{2\log H}} \cdot N \leqslant \frac{n}{H^2} \cdot H \leqslant \frac{n}{\log n}. \qquad \square$$

Using Lemma 2.1 Steps 1 and 2(c) and using Lemma 2.2 Step 2(b) can be done in $O(\log\log n)$ time using $n/\log\log n$ processors. The number of surviving subproblems can be found at every step using (approximate) prefix sum. Hence every stage requires $O(\log\log n)$ time with $n/\log\log n$ processors. By Brent's slow-down lemma every stage can be done in $O(\log n)$ time with $n/\log n$ processors. In the end the problem can be solved directly in $O(\log n)$ time with $n/\log n$ processors.

Lemma 2.5. *The convex hull of n points in a plane can be constructed in $O(\log n \cdot (\log H + \log\log n))$ time using $O(n/\log n)$ CRCW processors. For $H \geqslant \log^{\varepsilon} n$, $\varepsilon > 0$, this is optimal.*

2.3. Optimal algorithm for all H

We will now exploit the slow-down technique further to make this algorithm work-optimal for all output sizes. Since every level of recursion takes $O(\log\log n)$ steps with $n/\log\log n$ processors, each level will take no more than $O(\log\log N_i + N_i/P)$ steps with P processors, where N_i is the total size of subproblems after i stages. Recall that, from Lemma 2.3, a global processor allocation takes $O(\log\log n)$ steps. From Lemma 2.4, $N_i = O(n/\log n)$ after $i \geqslant \Omega(\log\log n + \log H)$ levels.

Thus with an additional $O(n)$ work (for computing the hull of $n/\log n$ points), the algorithm finishes in a further n/P steps for $P \leqslant n/\log n$. After $O(\log\log n + \log H)$ steps the total work done is

$$\sum_{i=1}^{O(\log\log n + \log H)} \left[O(N_i)\right] = O(n \log H) \quad \text{for } P \leqslant n/\log\log n.$$

This follows from the work-bound of Chan et al.'s [9] algorithm. The total time-bound is

$$\begin{aligned} &O(\log\log^2 n + \log\log n \cdot \log H) + 1/P \sum_i O(N_i) \\ &= O(\log\log^2 n + \log\log n \cdot \log H) + O(n \log H/P). \end{aligned} \tag{1}$$

Using $P = n/\log n$, in Eq. (1), the time-complexity is $O(\log n \log H)$. Thus we can formalize our result as follows.

Theorem 2.1. *The convex hull of n points in a plane can be constructed in $O(\log n \log H)$ time using $n/\log n$ processors in a deterministic CRCW PRAM.*

Remark. Using $P = n \log H/(\log n \log\log n)$, the time complexity can be be improved to $O(\log n \cdot \log\log n)$ for large H. Since we do not know H in advance, this bound will be hard to achieve in practice.

3. Randomized algorithm

We present a randomized algorithm which solves the dual equivalent of the convex hull problem namely *intersection of half-planes*. The convex hull problem is well known to be equivalent to the problem of finding the intersection of half-planes (for details, see [17,32,33]).

Let us denote the input set of half-planes by S and their intersection by $P(S)$. The idea is to construct the intersection of a random sample R of r half-planes and filter out the redundant half-planes, i.e., the half-planes which do not participate in $P(S)$. Without loss of generality, we can assume that the origin lies inside the intersection. Let $h_1, h_2, \ldots, h_r$ be the vertices of the intersection in a cyclic order. Consider the triangles of the form Oh_1h_2 (O being the origin), which we call regions. These will be intersected by a number of half-planes that were not chosen in the sample. In this paper, we shall say that a half-plane intersects a region if its bounding line intersects the region.

We delete the half-planes that do not intersect any region containing at least one output point (see Figs. 1 and 2). Consider a region that does not contain any output point. Clearly, only one half-plane is useful in this region, which is either the bounding half-plane of the region, which we retain, or some half-plane that intersects the region internally (and hides all other half-planes). Such a half-plane must intersect at least one of the regions containing an output point and is therefore retained. In Fig. 3, the half-plane P hides all other half-planes in the regions Oh_2h_3 and Oh_3h_4 but the region which it intersects in the extreme left, namely Oh_1h_2, and in the extreme right, namely Oh_4h_5, contain output points.

The above procedure is repeated on the reduced problem.

To prove any interesting result we must determine how quickly the problem size decreases. Let $H(R)$ denote the set of regions induced by a sample R and let $H^*(R)$ denote the set of regions that contains at least one output point. We will denote the set of half-planes intersecting a region $\Delta \in H(R)$ by $L(\Delta)$ and its cardinality $|L(\Delta)|$ by $l(\Delta)$. $L(\Delta)$ will also be referred to as the *conflict list* of Δ and $l(\Delta)$, its *conflict size*. We will use the following results related to bounding the size of the reduced problem.

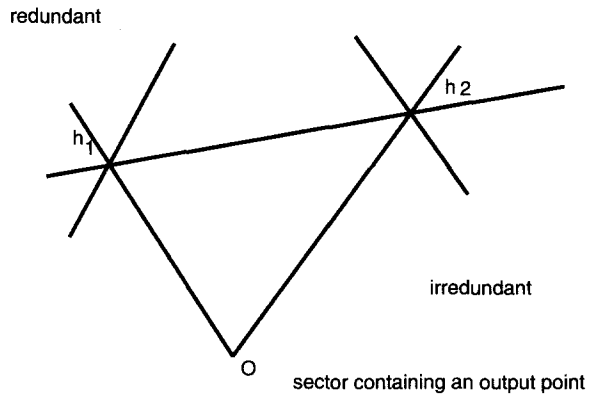

Fig. 1. Region containing at least one output vertex.

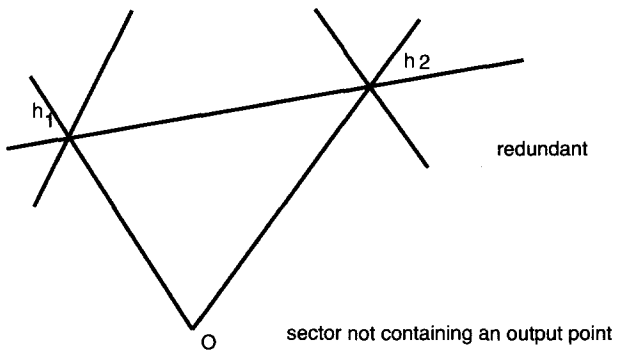

Fig. 2. Region with no output vertices.

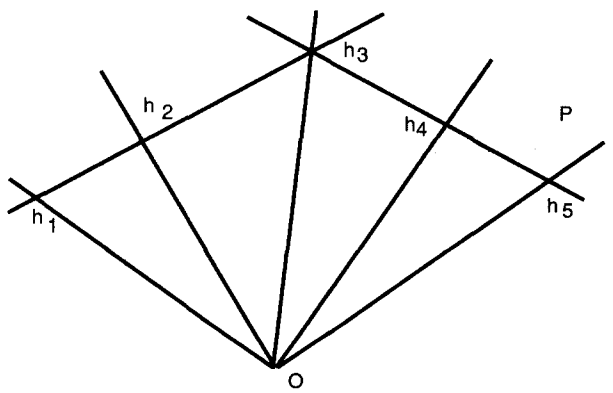

Fig. 3. Half-plane P will be retained because of regions h_1 and h_5 that contain output vertices.

Lemma 3.1 [13,33]. *For some suitable constant k and large n,*

$$\Pr\left[\sum_{\Delta\in H(R)} l(\Delta) \geqslant kn\right] \leqslant 1/4,$$

where probability is taken over all possible choices of random sample R.

The above lemma gives a bound on the size of the union of the conflict lists. The following gives a bound on the maximum conflict size.

Lemma 3.2 [13,26]. *For some suitable constant k_1 and large n,*

$$\Pr\left[\max_{\Delta\in H(R)} l(\Delta) \geqslant k_1(n/r)\log r\right] \leqslant 1/4,$$

where probability is taken over all possible choices of random sample R such that $|R| = r$.

A sample is "good" if it satisfies the properties of Lemmas 3.1 and 3.2 simultaneously. From Lemmas 3.1 and 3.2 a sample is good with probability at least $1/2$. We can actually do better as the following lemma shows.

Lemma 3.3. *We can find a sample R which satisfies both Lemmas 3.1 and 3.2 simultaneously with high probability. Moreover, this can be done in $O(\log r)$ time and $O(n \log r)$ work with high probability.*

Proof. This is done using Resampling and Polling. For details see Appendix A. □

Since $|H^*(R)| \leqslant H$, a good sample clearly satisfies the following property also.

Lemma 3.4. *For a good sample R,*

$$\sum_{\Delta\in H^*(R)} l(\Delta) = O(nH\log r/r),$$

where $|R| = r$ and $H^*(R)$ is the set of all regions that contain at least one output point.

This will be used repeatedly in the analysis to estimate the non-redundant half-planes whenever $H \leqslant r/\log r$.

Remark. We can actually find a sample that satisfies a stronger property than Lemma 3.4, namely the sum can be bounded by $O(nH/r)$. This matches the bound of Clarkson and Shor [13] and it can be done in the same time bounds of Lemma 3.3 by using the two level sampling procedure of Chazelle and Friedman [11].

3.1. Algorithm

Our algorithm works iteratively. Let n_i (respectively r_i) denote the size of the problem (respectively sample size) at the ith iteration with $n_1 = n$. Repeat the following procedure until $r_i > n^\varepsilon$ (this condition guarantees that the sample size is never too big) or $n_i < n^\varepsilon$ for some fixed ε between 0

and 1. If $n_i < n^{\varepsilon}$ then find out the intersection of n_i half-planes directly using Lemma 3.5 else do one more iteration and find out the intersection of n_{i+1} half-planes directly using Lemma 3.5.

The following is a description of the ith iteration of the algorithm.

Rand-Hull

1. Use the procedure of Lemma 3.3 to choose a "good" sample R of size r_i = constant for $i = 1$ and r_{i-1}^2 for $i > 1$.
2. Find out the intersection of half-planes of R.
3. Filter out the redundant half-planes as follows.
   (a) (i) For every half-plane find out the regions that it intersects.
   (ii) If the sum, taken over all the half-planes, of the number of regions intersecting a half-plane is $O(n)$ then continue else go to 1.
   (b) Compute $H^*(R)$ as follows.
   For every region Δ do the following:
   (i) Find out the half-planes intersecting Δ and assign as many processors to it (see Lemma 3.1 and Step 3a(ii) above).
   (ii) Consider the points of intersection of the bounding lines of half-planes with the radial edges of the region. If the points closest to the origin belong to the same line then $\Delta \notin H^*(R)$ else $\Delta \in H^*(R)$.
   (c) Delete a half-plane if it does not belong to $\bigcup_{\Delta \in H^*(R)} L(\Delta)$.
4. The set of half-planes for the next iteration is $\bigcup_{\Delta \in H^*(R)} L(\Delta)$ and its size is

$$n_{i+1} = \left| \bigcup_{\Delta \in H^*(R)} L(\Delta) \right|.$$

Increment i and go to 1.

3.2. Analysis

We begin by stating some of the results that will be used for analyzing the algorithm Rand-Hull in the previous section.

3.2.1. Background

Lemma 3.5 [36]. *With $p = nk$ $(k > 1)$, the convex hull of n points in two dimensions can be constructed in $\tilde{O}(\log n / \log k)$ steps.*

Remark. By duality, the above lemma holds for the problem of finding the intersection of half-planes.

Lemma 3.6 [33]. *The maximum (minimum) of n elements can be determined in constant time with high probability using n CRCW PRAM processors.*

The following three problems arise in the context of processor reallocation and compaction in our parallel algorithm.

Definition. For all $n, m \in N$ and $\lambda \geqslant 1$, the *m-color semisorting* problem of size n and with slack λ is the following. Given n integers $x_1, \ldots, x_n$ in the range $0, \ldots, m$, compute n non-negative integers $y_1, \ldots, y_n$ (the placement of x_i) such that
(1) all the x_i of the same colour are placed in contiguous locations (not necessarily consecutive).
(2) $\max\{y_j\colon\ 1 \leqslant j \leqslant n\} = O(\lambda n)$.

Definition. For all $n \in N$ *interval allocation* problem is the following. Given n non-negative integers $x_1, \ldots, x_n$, compute n non-negative integers $y_1, \ldots, y_n$ (the starting addresses of intervals of size x_i) such that
(1) for all i, j, the block of size x_i starting at y_i does not overlap with the block of size x_j starting at y_j.
(2) $\max\{y_j\colon\ 1 \leqslant j \leqslant n\} = O(\sum_i x_i)$.

Definition. Given n elements, of which only d are *active*, the problem of *approximate compaction* is to find the placement for the *active* elements in an array of size $O(d)$.

Lemma 3.7 [7]. *There is a constant $\varepsilon > 0$ such that for all given $n, k \in N$, n-color semisorting problem of size n and with slack $O(\log^{(k)} n)$ can be solved on a CRCW PRAM using $O(k)$ time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^{\varepsilon}}$. Alternatively, it can be done in $\widetilde{O}(t)$ steps, $t \geqslant \log^* n$ using n/t processors.*

The problems of interval allocation *and* approximate compaction *can also be solved in the same bounds.*

3.2.2. Overview

We assume the availability of linear number of processors. Our result relies heavily on the result of Lemma 3.4. The idea is to reduce the size of the problem to n^{ε}, for some ε, $0 < \varepsilon < 1$. Lemma 3.4 tells us that if $r = \Omega(H^2)$, the problem size can be reduced quickly. Notice that a large sample size reduces the problem size faster but increases the time for each iteration. Hence we must achieve a balance between the number of iterations and the time spent in each iteration. For the purpose of analyzing we divide our algorithm in three phases.

Initial phase. Initially we start with a sample of constant size and keep squaring it until it is $\Omega(H^2)$. Until now we cannot guarantee any reduction in the problem size. However, since the sample sizes are small we do not spend too much time in this phase ($O(\log H)$ time).

Main phase. We keep squaring the sample size in subsequent iterations thereby achieving a good reduction in the problem size until the problem size has reduced to n^{ε}.

Terminating phase. Solve the problem directly.

Since our sample size is never too large ($r_i \leqslant n^{\varepsilon}$), Step 2 can be done in constant time using Lemma 3.5 in each iteration. From Lemma 3.6, Step 3(b)(ii) can be done in constant time. The regions that a half-plane intersects can be obtained in $O(\log r / \log k)$ time using k processors. In the initial phase when no significant reduction in problem size is achieved there is no processor advantage, hence each iteration takes $O(\log r)$ time and hence a total of $O(\log H)$ time (a geometric series with $O(\log H)$ leading term). In the main phase, because of significant processor advantage this step takes constant time in each iteration.

Processor allocation and approximate compaction (last steps) can be done in $O(\log^* n)$ time in each iteration (Lemma 3.7).

As $r_i = r_1^{2^i}$, the initial phase requires $O(\log\log H)$ iterations and the main phase requires $O(\log\log n)$ iterations.

Hence the total time is $O(\log H + \log^* n \cdot \log\log H)$ for the initial phase and $O(\log^* n \cdot \log\log n)$ for the main phase.

Using some additional techniques and more careful analysis we will be able to eliminate the extra $O(\log^* n)$ factor from the main phase. Hence total time for the main phase is $O(\log\log n)$ for sufficiently large n.

The terminating phase can be shown to take $O(\log H)$ time.

3.2.3. Detailed analysis

We will now present the details of the analysis. Let l be the iteration in which the sample size of $\Omega(H^2)$ is achieved for the first time. Then

- *Initial phase*: $i \leqslant l \Rightarrow n_i = O(n)$.
- *Main phase* is analyzed as two sub-phases:

$$l < i \leqslant l + \log\log\log n \quad \Rightarrow \quad n_i = O\big(n \log H / H^{2^{i-l}}\big),$$

$$i > l + \log\log\log n \quad \Rightarrow \quad n_i < O(n/\log n).$$

Steps 3(a)(ii), 3(b)(i) and 3(c) will be implemented using procedures for Interval allocation and Semisorting. The problem of deleting half-planes can be reduced to compaction which can be approximated using the procedure for *approximate compaction*. From Lemma 3.7, Steps 3(a)(ii), 3(b)(i) and 3(c) can be done in $\tilde{O}(t)$ steps ($t \geqslant \log^* n$) using n/t processors or in constant time using $n \log n$ processors. Below we describe procedures to check for the condition in Step 3(a)(ii) and do Step 3(b)(i).

In Step 3(a)(i) each half-plane finds the regions it intersects. This gives pairs (p_i, s_j) (half-plane p_i intersects region s_j) whose number is bounded by $O(n)$ from Lemma 3.1. We call s_j the color of p_i. Notice that the regions that a half-plane intersects are contiguous and therefore we only need to store the left-end region and right-end region (say in clockwise order) with every half-plane, say in an array C. Clearly, we can also store the number of regions that a half-plane intersects. Now think of $C[i]$ as a request for memory cells. Solve the problem of interval allocation for C. If any processor tries to use an index beyond kn (for an appropriately chosen constant k), the condition in Step 3(a)(ii) must have been violated. Then discontinue interval allocation and repeat the procedure. After assigning $C[i]$ processors to the ith half-plane and completing interval allocation, we can put (p_i, s_j) pairs in an array (call it A) of size $O(n)$.

Apply r-color (for sample size r) semisorting algorithm on A. It will put all the half-planes intersecting a given region together, with possible blanks, in another array, say B, of $O(n)$ size. From Lemma 3.7 Steps 3(a)(ii) and 3(b)(i) can be done in $\tilde{O}(t)$ steps, $t \geqslant \log^* n$, using n/t processors or in constant time using $n \log n$ processors.

Assume that we have an array D of half-planes, of size $O(n)$. With each region we have a number of processors associated (assigned in Step 3(b)(i)), one for each intersecting half-plane. Each of these processors knows whether its region contains an output point or not. If a processor is associated with a

region containing an output point and with the half-plane p_i then it writes 1 in the ith cell of D. Now problem of deleting the half-planes is reduced to the problem of compaction which can be approximated within a constant factor using *approximate compaction*. This takes $\widetilde{O}(t)$ steps ($t \geqslant \log^* n$) using n/t processors or constant time using $n \log\log n$ processors (Lemma 3.7).

Therefore, all steps except 3(a)(i) (finding the intersections) take

$$\mathrm{O}(\log^* n) \text{ time} \quad \text{for } i \leqslant l, \tag{2}$$

$$\mathrm{O}(\log^* n) \text{ time} \quad \text{for } l < i \leqslant l + \log\log\log n, \quad \text{and} \tag{3}$$

$$\mathrm{O}(1) \text{ time} \quad \text{for } i > l + \log\log\log n. \tag{4}$$

As $r_i = r_1^{2^i}$, $l \leqslant \log\log H$ and as $r_i < n^\varepsilon$ for $0 < \varepsilon < 1$, the maximum number of iterations is $\mathrm{O}(\log\log n)$. Thus total time over all iterations for all steps except Step 3(a)(i) is

$$\mathrm{O}\big(\log^* n(\log\log H + \log\log\log n) + \log\log n\big) = \mathrm{O}(\log^* n \log\log H + \log\log n). \tag{5}$$

The regions that a half-plane intersects can be found out using a locus based searching scheme in $\mathrm{O}(\log r/\log k)$ time using k processors by Lemma 4.1 of [36]. Thus Step 3(a)(i) can be done in $\mathrm{O}(\log r_i)$ time for $i < l$ and in $\mathrm{O}(\log r_i/\log(n/n_i))$ time for $i > l$. By Lemma 3.4,

$$n_i < c\frac{nH\log r_{i-1}}{r_{i-1}} \quad \text{for some constant } c$$

$$\Rightarrow \quad n/n_i > \frac{r_{i-1}}{cH\log r_{i-1}} > \Omega\big(\sqrt{r_{i-1}}/\log r_{i-1}\big) \quad \text{for } i > l$$

as $r_{i-1} = \Omega(H^2)$ or $H = \mathrm{O}(\sqrt{r_{i-1}})$. By using $r_i = r_{i-1}^2$, the above inequality implies that $\log r_i/\log(n/n_i)$ is constant for $i > l$. Thus Step 3(a)(i) can be done in constant time for $i > l$. Hence total time for Step 3(a)(i) is

$$\sum_{i\leqslant l} \mathrm{O}(\log r_i) + \sum_{i>l} \mathrm{O}(1). \tag{6}$$

The first term, a geometric series with $\mathrm{O}(\log H)$ as the leading term, is $\mathrm{O}(\log H)$ and the second term is clearly $\mathrm{O}(\log\log n)$.

Let the terminating condition be satisfied in the tth iteration. If $n_t < n^\varepsilon$, then computing the intersection of n_t half-planes takes constant time. Otherwise, if $n^\varepsilon < r_t < n_t$ or $n^\varepsilon < n_t < r_t$ then we have the following.

Let ε be $< 1/2$. Clearly, $r_{t-1} < n^\varepsilon$ and $r_t < n^{2\varepsilon}$. Hence we can afford to do one more iteration within the same bounds. Now,

$$n_{t+1} = \mathrm{O}(n_t H\log r_t/r_t) = \mathrm{O}\big(n_t H(2\varepsilon\log n)/n^\varepsilon\big) = \mathrm{O}\big(n^{1-\varepsilon/2}H\big).$$

If $H < n^{\varepsilon/4}$ then n_{t+1} is $\mathrm{O}(n^{1-\varepsilon/4})$ and hence computing the intersection of n_{t+1} half-planes will take constant time. Otherwise, if $H > n^{\varepsilon/4}$ then nothing can be said about n_t or n_{t+1} except that $n_{t+1} = \mathrm{O}(n)$. Hence computing the intersection of n_{t+1} half-planes takes $\mathrm{O}(\log n)$ time which is $\mathrm{O}(\log H)$ (since $H > n^{\varepsilon/4}$).

Hence we have the following lemma.

Lemma 3.8. *The convex hull of n points in two dimensions can be constructed in* $\mathrm{O}(\max\{\log H, \log\log n\})$ *time with high probability using a linear number of CRCW processors.*

Remark. For $\log H = \Omega(\log\log n)$, this attains the ideal $O(\log H)$ running time with n processors.

Next, we will use the standard slow-down method to make the algorithm more efficient for all values of H. That is, we use $p = n/\rho$ processors where $\rho = \log\log n$, instead of n processors. We will use approximate compaction to distribute processors evenly in Step 3(c) of the algorithm.

In the initial phase, the processor allocation and approximate compaction can be done in $O(\log\log n)$ time and remaining steps take $O(\log\log n \cdot \log r_i)$ time (by Brent's slow-down lemma) in each iteration. The total time for this phase is thus $O(\log\log n \cdot \log H)$.

In the main phase second part (i.e., $n_i \leqslant n/\log n$), $p \geqslant n_i \log\log n_i$, so by Lemma 3.7 each step still takes constant time. In the first part (i.e., $n_i > n/\log n$), processor allocation and approximate compaction can be done in $O(\max\{\log^* n, n_i/p\})$ time. So in Eq. (3), $\log^* n$ is replaced with $\rho/H^{2^j-j} + \log^* n$ for $0 < j \leqslant \log\log\log n$ ($j = i - l$). Recall that l is the iteration in which the sample size of $\Omega(H^2)$ is achieved for the first time.

Hence, total time for all the steps except 3(a)(i) is

$$\sum_{i\leqslant l} \log\log n + \sum_{i=l+j:\ 0<j\leqslant \log\log\log n} \left[\rho/H^{2^j-j} + \log^* n\right] + \sum_{i>l+\log\log\log n} O(1)$$
$$= O(\log\log n \cdot \log\log H).$$

To find out the intersecting regions we analyze two cases separately, namely when $n_i > p$ and when $n_i \leqslant p$. Then this step takes

- $O((n_i/p)\log r_i)$ time when $n_i \geqslant p$ by work-load distribution. The running time for this phase is thus bounded by

$$\sum_{i>l} \frac{n_i}{p} \log r_i = O(\log\log n) \quad \text{as } n_i = O\left(n/H^{2^{i-l}-(i-l)}\right).$$

- $O(\log r_i/\log(p/n_i))$ time when $n_i < p$ due to processor advantage. Then

$$n_i = O(n_{i-1} H \log r_{i-1}/r_{i-1}) = O(pH \log r_{i-1}/r_{i-1})$$
$$\Rightarrow \quad p/n_i = \Omega(r_{i-1}/H \log r_{i-1}) = \Omega\left(\sqrt{r_{i-1}}/r_{i-1}\right).$$

So $O(\log r_i/\log(p/n_i))$ above is constant.

Strictly speaking, we must also add to this the time for processor-allocation but it has already been accounted for in the previous calculations.

The time-bound for the terminating phase gets multiplied by at most ρ. Hence, we have the following theorem.

Theorem 3.1. *The convex hull of n points in two dimensions can be constructed in $O(\log H \cdot \log\log n)$ expected time and $O(n \log H)$ operations with high probability in a CRCW PRAM model where H is the number of points on the hull.*

4. Remarks and open problems

We presented a class of output-sensitive parallel algorithms for planar hulls that are work optimal and run in polylog time. For small output sizes, we presented an algorithm that improves upon the

worst-case optimal algorithms in time bound. The fastest (randomized) algorithm is work-optimal using a linear number of processors for a large range of output size, namely $H \geqslant \log^{\epsilon} n$. Recall that, for uniform distribution, the expected output size is about $\log n$. For very small output sizes, our algorithms are work-optimal although the time complexity does not match the "ideal" bound of $O(\log H)$ time using n processors. It may be noted that $\Omega(\log \log n)$ is a lower bound for any deterministic parallel algorithm for convex hulls (using n processors) since it can be used for extremal selection; however the same may not be true for randomized algorithms. The other issue is that of speeding up the algorithms further using a superlinear number of processors. From [36], the lower bound in such cases is $\Omega(\log H/\log k)$ for $k \cdot n$ processors where $k > 1$. This appears to be a very challenging theoretical problem.

Some of the ideas presented in this paper are applicable to the problem in three dimensions. In [23], the authors present a randomized $O(\log \log^2 n \log H)$ expected time optimal algorithm for 3-D hulls. Amato et al. [3] have developed an $O(\log^3 n)$ time, optimal $O(n \log H)$ work algorithm for 3-D hulls in the EREW model. The results of the randomized algorithm can be made to hold with n-exponential probability by taking a larger number of samples, namely n^{α} for a suitable $\alpha < 1$ for the Resampling step. The remaining (randomized) sub-routines used in the algorithm are known to execute with n-exponential probability.

Acknowledgements

The authors wish to thank the reviewers whose detailed comments were very helpful in improving the presentation of the paper. In particular, the bound of the deterministic algorithm in Theorem 2.1 was obtained following a very useful suggestion made by one of the reviewers.

Appendix A

Proof of Lemma 3.3. Select $O(\log n)$ samples. We know that one of them is "good" with high probability. Consider a sample Q. Check it against a randomly chosen sample of size $n/\log n$ of the input half-planes for the condition of Lemmas 3.1 and 3.2. Checking condition of Lemma 3.1 is as given in [33]. We explain below how to check for condition of Lemma 3.2.

For every region Δ defined by Q do in parallel. Our procedure works for $r = o(n/\log^2 n)$.

Let $A(\Delta)$ be the number of half-planes of the $n/\log n$ sampled half-planes intersecting with Δ and let $X(\Delta)$ be the total number of half-planes intersecting with Δ. Let $X(\Delta) > c' \log^2 n$ for some constant c' – the condition of Lemma 3.3 holds trivially for the other case. By using Chernoff's bounds for binomial random variables, $L(\Delta) = k_1(A(\Delta) \log n)$ and $U(\Delta) = k_2(A(\Delta) \log n)$ are lower and upper bounds, respectively, for $X(\Delta)$ with high probability, for some constants k_1 and k_2. Each region reports whether to "accept" or to "reject" a sample as follows for some constant k.

- Reject a sample if $L(\Delta) > k(n/r) \log r$ $(X(\Delta) \geqslant L(\Delta) > k(n/r) \log r)$.
- Accept a sample if $U(\Delta) \leqslant k(n/r) \log r$ $(X(\Delta) \leqslant U(\Delta) \leqslant k(n/r) \log r)$.
- If $L(\Delta) \leqslant k(n/r) \log r \leqslant U(\Delta)$ then accept a sample $(X(\Delta)/k(n/r) \log r \leqslant U(\Delta)/L(\Delta)$, which is a constant).
- If any region reports "reject" a sample then reject the sample.

For sample size r, the entire procedure runs in $O(\log r)$ steps using n processors after building a data-structure of size r^c in $O(\log r)$ time, where c is a fixed constant (see [34] for details of the construction). Ensuring that $r^c \leqslant n$, gives us the required bounds. This completes the proof of Lemma 3.3. □

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C.-K. Yap, Parallel computational geometry. in: Proc. 25th Annual Sympos. Found. Comput. Sci. (1985) 468–477; also: Algorithmica 3 (3) (1988) 293–327.

[2] S. Akl, Optimal algorithms for computing convex hulls and sorting, Computing 33 (1) (1984) 1–11.

[3] N.M. Amato, M.T. Goodrich and E.A. Ramos, Parallel algorithms for higher-dimensional convex hulls, in: Proc. 35th Annual Sympos. Found. Comput. Sci. (1994) 683–694.

[4] N.M. Amato and F.P. Preparata, The parallel 3D convex hull problem revisited, Internat. J. Comput. Geom. Appl. 2 (2) (1992) 163–173.

[5] M.J. Atallah and M.T. Goodrich, Efficient parallel solutions to some geometric problems, J. Parallel Distrib. Comput. 3 (4) (1986) 492–507.

[6] M.J. Atallah and M.T. Goodrich, Parallel algorithm for some functions of two convex polygons, Algorithmica 3 (4) (1988) 535–548.

[7] H. Bast and T. Hagerup, Fast parallel space allocation, estimation and integer sorting, Technical Report MPI-I-93-123 (June 1993).

[8] T.M. Chan, Output-sensitive results on convex hulls, extreme points and related problems, in: Proc. ACM Sympos. Comput. Geom. (1995).

[9] T.M.Y. Chan, J. Snoeyink and C.-K. Yap, Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three, in: Proc. 6th ACM–SIAM Sympos. Discrete Algorithms (1995) 282–291.

[10] B. Chazelle and D. Dobkin, Intersection of convex objects in two and three dimensions, J. ACM 34 (1) (1987) 1–27.

[11] B. Chazelle and J. Fredman, A deterministic view of random sampling and its use in geometry, Combinatorica 10 (3) (1990) 229–249.

[12] A.L. Chow, Parallel algorithms for geometric problems, Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL (1980).

[13] K.L. Clarkson and P.W. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387–421.

[14] R. Cole, An optimal efficient selection algorithm, Inform. Process. Lett. 26 (1987/1988) 295–299.

[15] N. Dadoun and D.G. Kirkpatrick, Parallel construction of subdivision hierarchies, J. Comput. Syst. Sci. 39 (1989) 153–165.

[16] X. Deng, An optimal parallel algorithm for linear programming in the plane, Inform. Process. Lett. 35 (1990) 213–217.

[17] H. Edelsbrunner, Algorithms in Combinatorial Geometry (Springer, New York, 1987).

[18] M. Ghouse and M.T. Goodrich, In-place techniques for parallel convex-hull algorithm, in: Proc. 3rd ACM Sympos. Parallel Algorithms Arch. (1991) 192–203.

[19] T. Goldberg and U. Zwick, Optimal deterministic approximate parallel prefix sums and their applications, in: Proc. Israel Sympos. Theory Comput. Systems (ISTCS '95) (1995) 220–228.

[20] M. Goodrich, Geometric partitioning made easier, even in parallel, in: Proc. 9th ACM Sympos. Comput. Geom. (1993) 73–82.

[21] M. Goodrich, Fixed-dimensional parallel linear programming via relative ε-approximations, in: Proc. Sympos. Discrete Algorithms (1996) 1–10.

[22] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Inform. Process. Lett. 1 (1972) 132–133.
[23] N. Gupta and S. Sen, Faster output-sensitive parallel convex hulls for $d \leqslant 3$: optimal sublogarithmic algorithms for small outputs, in: Proc. ACM Sympos. Comput. Geom. (1996) 176–185.
[24] T. Hagerup, Fast deterministic processor allocation, in: Proc. 4th ACM Sympos. Discrete Algorithms (1993) 1–10.
[25] T. Hagerup and R. Raman, Waste makes haste: Tight bounds for loose, parallel sorting, in: Proc. 33rd Annual Sympos. Found. Comput. Sci. (1992) 628–637.
[26] D. Haussler and E. Welzl, ε-nets and simplex range queries, Discrete Comput. Geom. 2 (1987) 127–152.
[27] S. Kapoor and P. Ramanan, Lower bounds for maximal and convex layer problems, Algorithmica (1989) 447–459.
[28] D.G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm? SIAM J. Comput. 15 (1) (1986) 287–299.
[29] Converting high probability into nearly constant time – with applications to parallel hashing, in: Proc 23rd ACM Sympos. Theory Comput. (1991) 307–316.
[30] F.P. Preparata, An optimal real time algorithm for planar convex hulls, Comm. ACM 22 (1979) 402–405.
[31] F.P. Preparata and S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, Comm. ACM 20 (1977) 87–93.
[32] F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction (Springer, New York, 1985).
[33] S. Rajasekaran and S. Sen, in: J.H. Reif, Ed., Random Sampling Techniques and Parallel Algorithm Design (Morgan Kaufmann Publishers, San Mateo, CA, 1993).
[34] J.H. Reif and S. Sen, Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems, SIAM J. Comput. 21 (3) (1992) 466–485.
[35] J.H. Reif and S. Sen, Randomized algorithms for binary search and Load Balancing on fixed connection networks with geometric applications, SIAM J. Comput. 23 (3) (1994) 633–651.
[36] S. Sen, Lower bounds for algebraic decision trees, complexity of convex hulls and related problems, in: Proc. 14th FST&TCS, Madras, India (1994) 193–204.
[37] S. Sen, Parallel multidimensional search using approximation algorithms: with applications to linear-programming and related problems, in: Proc. ACM Sympos. Parallel Algorithms and Architectures (1996) 251–260.
[38] M.I. Shamos, Computational geometry, Ph.D. Thesis, Yale University, New Haven (1978).