Faster output-sensitive parallel convex hulls for $d \leq 3$: optimal sublogarithmic algorithms for small outputs

Neelima Gupta and Sandeep Sen
Department of Computer Science and Engineering,
Indian Institute of Technology,
New Delhi 110016, India.
{neelima,ssen}@cse.iitd.ernet.in

Abstract

In this paper we focus on the problem of designing very fast parallel algorithms for the convex hull problem in two and three dimensions in the arbitrary CRCW model whose running times are output-size sensitive. We present a fast randomized algorithm for planar hulls that runs in expected time $O(\log H \cdot \log\log n)$ and does optimal $O(n \log H)$ work where n and H are the input and output sizes respectively. For $\log H = \Omega(\log\log n)$, we can achieve the optimal running time of $O(\log H)$ for planar hulls while simultaneously keeping the work optimal. In three dimensions, our algorithm runs in expected time $O(\log\log^2 n \cdot \log H)$ with optimal $O(n \log H)$ work for all H. Hence, for $O(\log^{O(1)} n)$ size outputs, our algorithms in two and three dimensions achieve poly$(\log\log n)$ running time and optimal $O(n \log\log n)$ work. The previously known output-sensitive work-optimal algorithms for convex hulls have running times $\Omega(\log n)$ (expected) and $\Omega(\log^3 n)$ in two and three dimensions respectively. Our algorithms assume no input distribution and the running times hold with high probability.

We also describe a very simple $O(\log n \log H)$ time optimal deterministic algorithm for planar hulls which is an improvement for small outputs. For larger output-sizes, a running time of $O(\log n \log\log n))$ can be achieved.

Computational Geometry'96, Philadelphia PA, USA
© 1996 ACM 0-89791-804-5/96/05..$3.50

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) of the points of S defining the boundary of the convex hull of S.

The problem of constructing convex hull has attracted a great deal of attention from the inception of computational geometry. Several sequential algorithms have been proposed for planar hull so far, with 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]. This is true if the output size, i.e. the number of vertices on the hull, is ignored. More specifically, the time-bound of $\theta(n \log n)$ is tight when the ordered output size is $\Omega(n)$. However for cases, 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] gave an algorithm with worst case time complexity $O(n \log H)$. They also showed that this is asymptotically worst case optimal, even if the complexity is measured in terms of both the input and the output size (also see [27]). The algorithm works on modified Divide-and-Conquer technique called "Marriage-before-Conquest". For three dimensions, Preparata and Hong [30], described the first $O(n \log n)$ time algorithm for convex hulls. Clarkson and Shor [15] presented the first randomized $O(n \log H)$ time output-sensitive algorithm which was subsequently derandomized optimally by Chazelle and Matousek [13]. Very recently Chan [10] presented a very elegant approach for output sensitive construction of convex hulls using ray-shooting that achieve optimal $\Theta(n \log H)$ running times for dimensions two and three. In higher dimensions, the quest is still on to design optimal output-sensitive algorithms.

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 so as to minimize $T(n)$ subject to keeping the product $P(n) \cdot T(n)$ close to $O(S(n))$. A parallel algorithm that actually does total work $O(S(n))$ is called a work optimal algorithm. Simultaneously, if one can also match the time lower bound, then the algorithm is the best possible (theoretically).

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). For CRCW, 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, it is shown that

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, $k \geq 1$ 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 significant progress towards achieving this end. It must also be noted that because of the power of the CRCW model, the parallel complexity of a problem is perhaps best understood in this framework where communication is not a serious bottleneck.

1.2 Previous results for two and three dimensional hulls

For planar hulls (2-D 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, 4, 5, 34]. These are known to be worst case optimal in the CREW model. Akl ([3]) described an output-sensitive algorithm for this problem which is optimal for number of processors bounded by $O(n^z)$, $0 < z < 1$. Deng ([16]) described 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 can achieve running times of $\Theta(\log^2 n)$ and $\Theta(\log n)$ in deterministic and randomized CRCW models respectively (Ghouse and Goodrich [19]).

For convex hulls in three-dimensions (3-D hulls) Chow([9]) described an $O(\log^3 n)$ time algorithm using n CREW processors. An $O(\log^2 n \log^* n)$ time algorithm using n processors was obtained by Dadoun and Kirkpatrick([17]) and an $O(\log^2 n)$ time algorithm was designed by Amato and Preparata([6]). Reif and Sen [33] presented an $O(\log n)$ time and $O(n \log n)$ work randomized algorithm for 3-D convex hulls which was the first known worst-case optimal algorithm for three dimensional hulls in the CREW model. This algorithm was recently derandomized by Goodrich [20] who obtained an $O(\log^2 n)$ time $O(n \log n)$ work method for the EREW PRAM. Goodrich and Ghouse [19] described an $O(\log^2 n)$ expected time, $O(min\{n \log^2 H, n \log n\})$ work method, which is output-sensitive but not work-optimal. More recently, Amato et.al([2]) gave a deterministic $O(\log^3 n)$ time, $O(n \log H)$ work algorithm for convex hulls in R^3 on the EREW PRAM.

1.3 Our results and methods

We present algorithms whose running times are output-sensitive in a faster time-range (sublogarithmic) while keeping the work optimal. It must be emphasized that designing optimal output-sensitive parallel algorithms that have to execute in such time-bounds is a challenging task, even more than their sequential counterparts. Not only is the output-size an unknown parameter, 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 $o(\log n)$ time.

Our randomized algorithms are based on an approach of Clarkson and Shor [15] - this gives us a work-optimal algorithm as a starting point. However, its efficient adaptation into the parallel context necessitates a number of sophisticated techniques like bootstrapping and super-linear processors parallel algorithms for convex hulls ([36]) and very fine-tuned analysis. The basic method of [15] 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 first present fast randomized algorithms for convex hulls in two and three dimensions. The expected running times hold with high probability [1]. The fastest algorithm for 2-D hulls runs in $\tilde{O}(\log H)$ time using n processors for $H > \log n$. For smaller output sizes, we present an algorithm that has an expected running time of $O(\log H \cdot \log\log n)$ keeping the number of operations optimal. For 3-D hulls, our algorithm achieves $O(\log\log^2 n \cdot \log H)$ time and does optimal $O(n \log H)$ work. Therefore, for small output-sizes, our results achieve significant improvement over the previously known algorithms both in two and three dimensions. We are not aware of any previous work where the parallel algorithms speed-up optimally with output size in the sublogarithmic time domain.

Next we describe a very simple deterministic $O(\log n \log H)$ time CRCW algorithm that does optimal work. Although this is based on the optimal sequential algorithm of Kirkpatrick and Seidel, a straightforward parallelization yields an $\Omega(\log n poly(\log\log n)$ algorithm, which will be slower than our algorithm for very small output-sizes, namely, for $\log H \in o(\log\log n)$.

2 Randomized algorithms

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 problem of finding the intersection of half-planes (for details see, [18, 32, 34]).

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.

[1]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 $\tilde{O}$ instead of O to denote that the bound holds with high probability.

For 3-dimensional halfspaces, the regions are defined as follows: Take an arbitrary (fixed) plane T, and partition each face of $P(R)$ into trapezoids using the translates of T that pass through the vertices of the face. A region consists of the convex closure of O with a trapezoid from the cutting of the faces.

We delete half-planes that do not intersect with any region containing at least one output point. Consider a region that doesn't contain an output point. Clearly only one half-plane is useful in this region, which is either the boundary 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.

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-spaces intersecting a region $\triangle \in H(R)$ by $L(\triangle)$ and its cardinality $|L(\triangle)|$ by $l(\triangle)$. $L(\triangle)$ will also be referred to as the *conflict list* of $\triangle$ and $l(\triangle)$, its *conflict size*. We will use the following results related to bounding the the size of the reduced problem.

Lemma 2.1 ([15, 34]) *For some suitable constant k and large n,*

$$\Pr[\sum_{\triangle \in H(R)}]l(\triangle) \geq kn] \leq 1/4$$

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

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

Lemma 2.2 ([15, 25]) *For some suitable constant k_1 and large n,*

$$\Pr[\max_{\triangle \in H(R)} l(\triangle) \geq k_1 n/r \cdot \log r] \leq 1/4$$

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 Lemma 2.1 and Lemma 2.2 simultaneously. From Lemma 2.1 and Lemma 2.2 a sample is good with probability at least 1/2. We can actually do better as the following lemma shows.

Lemma 2.3 *We can find a sample R that satisfies both Lemma 2.1 and Lemma 2.2 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 $H^*(R) \leq H$, a good sample clearly satisfies the following property also

Lemma 2.4 ([15]) *For a good sample R,*

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

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 $H \leq r/\log r$.

2.1 Algorithm

Our algorithm works iteratively. Let n_i (respectively r_i) denote the size of the problem (respectively sample size) at the i^{th} iteration with $n_1 = n$. Repeat the following procedure until $r_i > n^\epsilon$ (this condition guarantees that the sample size is never too big) or $n_i < n^\epsilon$ for some fixed ϵ between 0 and 1. If $n_i < n^\epsilon$ then find out the intersection of n_i half-planes directly using Lemma 2.5 else do one more iteration and find out the intersection of n_{i+1} half-planes directly using Lemma 2.5.

We first describe our algorithm for two-dimensional hulls and in section 2.3, we will provide the modifications necessary for three dimensions. The following is a typical iteration for the two-dimensional case

Hull (i)

1. Use the procedure of Lemma 2.3 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. We say that a half-plane intersects a region if its bounding line intersects the region.

   (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) Denote the set of regions containing an output point by $H^*(R)$. We find out this set as follows For every region F do the following:

       i. Find out the half-planes intersecting F and assign as many processors to it(See Lemma 2.1 and step 3.a.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 $F \notin H^*(R)$ else $F \in H^*(R)$.

   (c) For every region $F \in H^*(R)$, Let F_{IR} denote the set of half-planes that intersect with F. Delete a half-plane if it does not belong to $\cup_{F \in H^*(R)} F_{IR}$.

The set of half-planes for the next iteration is $\cup_{F \in H^*(R)} F_{IR}$

Size of the reduced problem for the next iteration is $n_{i+1} = |\cup_{F \in H^*(R)} F_{IR}|$

2.2 Analysis

Lemma 2.5 ([36]) *With $p = nk (k > 1)$, the convex hull of* **n** *points in a three dimensions can be constructed in $\tilde{O}(\log n/\log k)$ steps.*

Remarks: 1. The result in Sen [36] was proved for $k \geq \log^3 n$ - this can be extended to all $k \geq 1$ using an observation of [2].
2. By duality, the above Lemma holds for the problem of finding the intersection of half-planes.

Lemma 2.6 ([34]) *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 $\lambda \geq 1$, 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 nonnegative 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) $\max\{y_j : 1 \leq j \leq n\} = O(\lambda n)$.

Definition For all $n \in N$ *interval allocation* problem is the following: Given n non-negative integers $x_1, \ldots, x_n$, compute n nonnegative 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) $\max\{y_j : 1 \leq j \leq n\} = O(\sum_i x_i)$.

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 2.7 ([7]) *There is a constant $\epsilon > 0$ such that for all given $n, k \in N$, n-color semisorting problems 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 $1 - 2^{-n^\epsilon}$. Alternatively, it can be done in $\tilde{O}(t)$ steps, $t \geq \log^* n$ using n/t processors.*
The problems of **interval allocation** *and* **approximate compaction** *can also be solved in the same bounds.*

Let's analyze Hull (i) first in the context of two-dimensional hulls. Subsequently, we will extend it to the three dimensional hulls.

Our analysis relies heavily on the result of Lemma 2.4. The idea is to reduce the size of the problem to n^ϵ, for some $\epsilon, 0 < \epsilon < 1$. Lemma 2.4 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 analysis, 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 can't 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 big its intersection can be computed in constant time using Lemma 2.5 in each iteration.

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 2.7).

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.

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

After this overview, we will now proceed to the detailed analysis. Let l be the iteration in which the sample size of $\Omega(H^2)$ is achieved for the first time. We'll break-up the analysis in three parts:

1. $i \leq l \quad \Rightarrow \quad n_i = O(n)$.
2. $l < i \leq l + \log\log\log n$.
3. $i > l + \log\log\log n \quad \Rightarrow \quad n_i < n/\log n$ (by Lemma 2.4). Since $r_l = \Omega(H^2)$ therefore $r_i = \Omega(H^{\log\log n}) \Rightarrow n_i = O(\frac{nH \log r_i}{r_i}) = O(n/\log n)$.

By Lemma 2.5 second step and by Lemma 2.6 step 3.b.ii can be done in constant time. The remaining steps except 3.a.i can be done in $O(\log^* n)$ time for $n_i > n/\log n$ and in constant time for $n_i < n/\log n$. These are implemented using procedures for Interval allocation and semisorting. The problem of deleting the half-planes is reduced to the problem of compaction which can be approximated within a constant factor using *approximate compaction*. From Lemma 2.7, Steps 3.a.ii, 3.b.i and 3.c can be done in $\tilde{O}(t)$ steps ($t \geq \log^* n$) 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 us pairs (p_i, s_j) (half-plane p_i intersects region s_j) whose number is bounded by $O(n)$ from Lemma 2.1. 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 appropriate constant k then the condition in step 3.a.ii is violated, discontinue interval allocation and repeat the procedure. After assigning $C[i]$ processors to the i^{th} 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 2.7 steps 3.a.ii and 3.b.i can be done in $\tilde{O}(t)$ steps, $t \geq \log^* n$, using n/t processors or in constant time using $n \log n$ processors.

Assume we have an array P of half-planes, of size $O(n)$. With each region we've 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 i^{th} cell of P. 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 $\tilde{O}(t)$ steps ($t \geq \log^* n$) using n/t processors or constant time using $n \log\log n$ processors (Lemma 2.7).

Therefore all steps except 3.a.i (finding intersections) take time:

$$O(\log^* n) \text{for } i \leq l. \tag{1}$$

$$O(\log^* n) \text{for } l < i \leq l + \log\log\log n. \tag{2}$$

$$\text{and } O(1) \text{for } i > l + \log\log\log n. \tag{3}$$

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

$$O(\log^* n(\log\log H + \log\log\log n) + \log\log n))$$

$$= O(\log^* n \log\log H + \log\log n).$$

The regions that a half-plane intersects can be found out using a locus based searching scheme in $O(\log r / \log k)$ time using k processors by Lemma 4.1 of [36]. Thus step 3.a.i can be done in $O(\log r_i)$ time for $i < l$ and in $O(\log r_i / \log \frac{n}{n_i})$ time for $i > l$. By Lemma 2.4, $n_i < c\frac{nH \log r_{i-1}}{r_{i-1}}$ for some constant c

$$\Rightarrow n/n_i > \frac{r_{i-1}}{cH \log r_{i-1}} \geq \frac{r_{i-2}^2}{cH \log r_{i-1}}$$

$$> r_{i-2}$$

for $i > l+5$ and $H > 2$. The last inequality follows from $r_{i-2} \geq cH \log r_{i-1}$ for the previous values of i and H. As $r_i = (r_{i-2})^4$, $\log r_i / \log \frac{n}{n_i}$ is constant for $i > l+5$. Thus step 3.a.i can be done in constant time for $i > l+5$ [2]. Hence total time for step 3.a.i is

[2]Setting $i > l+5$ is done to simplify calculations - it is not critical for the bounds

$$\sum_{i \leq l+5} O(\log r_i) + \sum_{i > l+5} O(1) \tag{4}$$

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

Let the terminating condition be satisfied in the t^{th} iteration. If $n_t < n^\epsilon$, then computing the intersection of n_t half-planes takes constant time. Otherwise, if $n^\epsilon < r_t < n_t$ or $n^\epsilon < n_t < r_t$ then we've the following:

Let ϵ be $< 1/2$. Since $r_{t-1} < n^\epsilon$, $r_t < n^{2\epsilon}$ so we can afford to do one more iteration within the same bounds. Now $n_{t+1} = O(n_t H \log r_t / r_t) = O(n_t H (\epsilon \log n)/n^\epsilon)$. If $H < n^\delta$ for $\delta < 1$ then n_{t+1} is less than $n^{1-\epsilon+\delta}$. By choosing $\delta < \epsilon$, computing the intersection of n_{t+1} half-planes will take constant time. Otherwise, if $H > n^\delta$, then nothing can be said about n_t or n_{t+1} except that $n_{t+1} = O(n)$. Hence computing the intersection of n_{t+1} half-planes takes $O(\log n)$ time which is $O(\log H)$ (since $H > n^\delta$).

Hence we've the following

Lemma 2.8 *The convex hull of* **n** *points in two dimensions can be constructed in* $O(\max\{\log H, \log\log n\})$ *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. That is, we use $p = n/\rho$ processors where $\rho = \log\log n$ instead of n processors. Use approximate compaction to distribute processors evenly, in step 3c of the algorithm.

In the analysis, $\log^* n$ is replaced by ρ in (1). Processor allocation (respectively step 3.b.ii) in (2) is no longer $\log^* n$ (respectively constant). By using slow-down, $\log^* n$ in (2) is replaced with $\rho / H^{2^j - j} + \log^* n$ for $0 < j \leq \log\log\log n$ ($j = i - l$). The processor allocation and 3.b.ii (in 3) remain constant as $n_i < n/\log n$ implies $p > n_i \log n / \log\log n > n_i \log\log n > n_i \log\log n_i$.

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

$$\sum_{i \leq l} \log\log n + \sum_{i = l+j : 0 < j \leq \log\log\log n} [\rho / H^{2^j - j} + \log^* n]$$

$$\sum_{i > l + \log\log\log n} O(1)$$

$$= O(\log\log n \cdot \log\log H)$$

For step 3.a.i, first term of (4) gets multiplied by ρ. For $i > l$, we split up the analysis in two parts,

$$n_i > p \ : \quad \frac{n_i}{p} \log r_i \ \text{ time}$$

$$n_i \leq p \ : \quad O(1) \ \text{ time}$$

Therefore, the total time, for step 3.a.i is

$$\sum_{i<l+5} \log\log n \log r_i + \sum_{i>l+5, n_i>p} \frac{n_i}{p} \log r_i + \sum_{i>l+5, n_i \leq p} O(1) = O(\log\log n \log H)$$

as the first and second terms are geometric series with $O(\log\log n \log H)$ as the leading term. Remaining terms get multiplied by at most ρ.

Hence, we've the following

Theorem 2.1 *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.*

2.3 Three dimensional hulls

We make the following modifications to the algorithm of section 2.1 for the 3-D hulls.

In Step 1, the samples are chosen according to $r_{i+1} = \max\{r_i^2, h_i^*\}$, where $h_0^* = 0$ and h_i^* for $i > 0$ is defined in the next paragraph. Note that according to this scheme for $i > l$, $r_{i+1} = r_i^2$.

In Step 3.b.ii, we also have to identify those regions $F \notin H^*(R)$, which may contain edges of the final hull but no vertices. This can be detected by constructing the *contours* and subsequent sorting of the vertices of contours. *Contours* are convex polygonal chains on each face of F that are induced by the planes intersecting F. See [33] for details of detecting such regions. Let h_i^* denote the maximum size of any contour over all regions during iteration i.

For analyzing three-dimensional case, we will require the results of the previous section in Step 3.b.(ii) of the algorithm. To detect $H^*(R)$, we construct the *contours* which are two-dimensional hulls of the projected planes on the faces of a region. Recall that h_i^* is the size of the largest contour in iteration i. Clearly $H \geq h_i^*$, so we choose r_{i+1} (the size of random sample in iteration $i+1$) as $\max(r_i^2, h_i^*)$. Using Theorem 2.1 the contours can be constructed in $\tilde{O}(\log\log n \cdot \log h_i^*)$ time with optimal work. The time for this step over the entire algorithm can be expressed as

$$\sum_{i<l} \tilde{O}(\log\log n \cdot \log h_i^*) + \sum_{i \geq l} T2(n_i, H)$$

where $T2(n_i, H)$ denotes the parallel time for constructing two dimensional hulls with output-size H. By our choice of r_{i+1}, $h_i^* \leq r_{i+1}$ and also $r_{i+1} \geq r_i^2$. Hence the first term of the summation can be bounded by $\tilde{O}(\log\log n \sum_{i<l} \log r_i)$ which is $\tilde{O}(\log\log n \cdot \log H)$. The second term can be bounded by $\tilde{O}(\log\log^2 n \log H)$ as the summand can be bounded by $O(\log\log n \log H)$. The work for this step over the entire algorithm is

$$\sum_{h_i \leq H} O(n \log h_i) + \sum_{i>l} O(n_i \log H)$$

where $h_i \geq h_{i-1}^2$ and, $n_i = \tilde{O}(n/H^{2^{i-1}})$. Thus the total work for this step is $\tilde{O}(n \log H)$. Hence, by using $n/\log\log^2 n$ processors, and evenly distributing work using the scheme discussed in the case of two-dimensional hulls, we obtain the following result.

Theorem 2.2 *The convex hull of* **n** *points in three dimensions can be constructed in* $O(\log\log^2 n \cdot \log H)$ *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.*

3 Deterministic algorithm for planar hulls

Our algorithm uses the basic approach of of [28]; however, we would like to minimize the recursion depth for a fast parallel algorithm. Recursion is carried out until the total size of the problem reduces sufficiently after which the problem is solved directly.

We assume for simplicity that no two vertices collide on x- or y- coordinate. We construct the $CH(S)$ in two parts, the upper hull $UH(S)$ and the lower hull $LH(S)$. We'll describe the procedure for upper hull only; the procedure for lower hull is identical and merging is trivial.

3.1 Algorithm

- 1. Find p and q with smallest and largest x-coordinate respectively. if $p = q$ then print p and stop.
- 2. CONNECT(p, q, S)
- where CONNECT(p_k, p_m, S) is begin(CONNECT)
- if the total size of the problem($\frac{n}{2^d}\cdot$ number of subproblems, where d is the depth of recursion) $\leq n/\log n$ then EXIT.
  else

- (a) Find an approximate median $\mathbf{a}$ of x-coordinate of points.

  Let $S_1 = \{p_i : x(p_i) < \mathbf{a}\}$

  $S_2 = \{p_i : x(p_i) > \mathbf{a}\}$

- (b) Find the "bridge" over the vertical line L: x = **a**, i.e.

  $(p_i, p_j) = \text{BRIDGE}(S, \mathbf{a})$

  Bridge or UPPER COMMON TANGENT(UCT) between $UH(S_1)$ and $UH(S_2)$ is the common tangent such that $UH(S_1)$ and $UH(S_2)$ are below it.

- (c) Compute $S_{left} = \{p_i\} \cup \{p \in S_1 \mid x(p) < x(p_i)\}$

  $S_{right} = \{p_j\} \cup \{p \in S_2 \mid x(p) > x(p_j)\}$

  Here we delete all the points lying below the bridge, before calling the recursion - "marriage-before-conquest".

- (d) If ($i = k$) then (this subproblem has only one output point)

  print(p_i)

  else CONNECT(p_k, p_i, S_{left}).

- (e) If ($j = m$) then (this subproblem has only one output point)

  print(p_j)

  else CONNECT(p_j, p_m, S_{right}).

  end(CONNECT)

Solve each subproblem directly using any of the $(n, \log n)$ algorithm mentioned earlier.

Correctness follows from [28].

3.2 Analysis

Lemma 3.1 *Maximum(Minimum) of n elements can be found in $O(\log\log n)$ time using $n/\log\log n$ processors.*

Lemma 3.2 ([26]) *There is a CRCW algorithm that finds an element with rank k such that $\frac{n}{3} \geq k \geq \frac{2n}{3}$ with processor-time complexity $(n/\log\log n, \log\log n)$.*

Lemma 3.3 ([21, 37]) *A two dimensional linear programming problem can be solved in time $O(\log\log^3 n)$ using $n/\log\log^3 n$ processors on a CRCW PRAM.*

Lemma 3.4 ([26]) *Approximate compaction can be done deterministically in $O(\log\log n)$ steps using $n/\log\log n$ CRCW processors.*

Remark: The above implies that processor allocation and approximate prefix sum can be done in the same bounds.

Lemma 3.5 *The size of the problem reduces to $n/\log n$ after $O(max\{\log\log n, \log H\})$ levels of recursion.*

Proof: In order to keep the notations simpler, we assume all the log are taken to base 3/2 in this proof. Let N denote the number of subproblems after $O(max\{\log\log n, \log H\})$ levels of recursion. Since every subproblem has at least one output vertex therefore $N \leq H$.

case 1: $H < \log n$, then total size after $3/2 \log\log n$ levels is $\frac{n}{(3/2)^{2\log\log n}} \cdot N \leq \frac{n}{\log^2 n} \cdot H \leq \frac{n}{\log n}$

case 2: $H > \log n$, then total size after $2\log H$ levels is $\frac{n}{(3/2)^{2\log H}} \cdot N \leq \frac{n}{H^2} \cdot H \leq \frac{n}{\log n}$

Note that $\frac{n}{(3/2)^{2\log\log n}} \cdot N \not\leq \frac{n}{\log n}$ for large H because N may be large.

Using Lemma 3.1, step 1 and using Lemma 3.2, step 2.a 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. The problem of finding the bridge can be mapped to a two dimensional linear programming problem, and therefore can be solved in $O(\log\log^3 n)$ time using $n/\log n$ processors. Hence every level of recursion can be done in $O(\log\log^3 n)$ time.

Remark By using the algorithm of [11], we can avoid 2-D linear programming, thereby taking only $O(\log\log n)$ time for each stage.

3.3 Optimal algorithm for all H

We will now use the Slow-down technique to make this algorithm optimal (work) for all output sizes. Since every level of recursion takes $O(\log\log^3 n)$ steps with $n/\log\log^3 n$ processors, each level will take no more than $O(\log\log^3 N_i + N_i/P)$ steps with P processors, where N_i is the total size of subproblems after i stages. Recall that, from Lemma 3.4, a global processor allocation takes $O(\log\log n)$ steps. From Lemma 3.5, $N_i = O(n/\log n)$ after $i \geq \Omega(\log\log n + \log H)$ levels. Thus with an additional $O(n)$ $(O(n/\log n \cdot \log(n/\log n)))$ work the algorithm can be completed in n/P time for $P \leq n/\log n$. After $O(\log\log n + \log H)$, steps, the total work done is

$$\sum_{i=1}^{O(\log\log n+\log H)} [O(N_i)] for P \leq n/\log\log^3 n.$$

$$= O(n \log H)$$

This follows from the work-bound of Kirkpatrick and Seidel's algorithm. The total time-bound is

$$O(\log\log^4 n + \log\log^3 n \cdot \log H) + 1/P \sum_i O(N_i)$$

$$= O(\log\log^4 n + \log\log^3 n \cdot \log H + O(n \log H/P) \quad (5)$$

Using $P = n/\log n$, in equation 5, the time-complexity is $O(\log n \log H)$. Thus we can formalize our result as

Theorem 3.1 *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 $P = n \log H/(\log n \log\log^3 n)$, the time complexity can be be improved to $O(\log n \cdot \log\log^3 n)$ for large H. By using the approach of [11], the running time can be further decreased to $O(\log n \log\log n)$. Since we do not know H in advance, these bounds will be hard to achieve.

4 Remarks and open problems

We presented a class of output-sensitive parallel algorithms for convex hulls in two and three dimensions that are work optimal and run in polylog time. For small output sizes, we presented an algorithm that improves upon the worst-case optimal algprithms in time bound. The fastest (randomized) algorithm is work-optimal using a linear number of processors for a large range of output size, namely $H \geq \log n$. 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. Obtaining the ideal time bound of $O(\log H)$ even in a parallel decision tree (with optimal work) is a challenging open problem. The other issue is that of speeding up the algorithm 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 $k > 1$.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap. Parallel computational geometry. *Proc. of 25th Annual Symposium on Foundations of Computer Science*, pages 468 – 477, 1985. also appears in full version in Algorithmica, Vol. 3, No. 3, 1988, pp. 293-327.

[2] N.M. Amato, M.T. Goodrich and E.A. Ramos. Parallel Algorithms for Higher-Dimensional Convex Hulls. *Proc. of the 35th Annual FOCS*, pages 683–694, 1994.

[3] S. Akl. Optimal algorithms for computing convex hulls and sorting. *Computing*, 33:1, 1–11, 1984.

[4] M.J. Atallah and M.T. Goodrich. Efficient parallel solutions to some geometric problems. *Journal of Parallel and Distributed Computing*, 3(4):492 – 507, 1986.

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

[6] N.M. Amato and F.P. Preparata. The Parallel 3D convex hull problem revisited. *Internat. Jl. Comput. Geom. Appl.*, 2(2):163-173, 1992.

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

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

[9] A.L. Chow. Parallel Algorithms for Geometric Problems. PhD Thesis, Deptt. of Comp.Sc., Univ. of Illnois, Urbana, IL, 1980.

[10] T.M. Chan. Output-Sensitive Results on convex hulls, extreme points and related problems. ACM Symp. on Comput. Geom., 1995.

[11] Timothy.M.Y. Chan, Jack Snoeyink, Chee-Keng Yap. Output-Sensitive Construction of Polytopes in Four Dimensions and Clipped Voronoi Diagrams in Three. *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms* 1995, pp 282-291.

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

[13] B. Chazelle and J. Matousek. Derandomizing an output-sensitive convex-hull algorithm in three dimensions. Tech. Rept., Princeton University, 1992.

[14] R Cole. An optimal efficient selection algorithm. *Inform. Proc. Lett.*, 26:295–299, 1987/88.

[15] Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry ii. *Discrete Comp. Geom.*, 4:387–421, 1989.

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

[17] N. Dadoun and D.G. Kirkpatrick. Parallel Construction of subdivision hierarchies. *Jl. Comput. Syst. Sc.*, 39:153-165,1989.

[18] H Edelsbrunner . *Algorithms in combinatorial geometry.* Springer-verlag, New York, 1987.

[19] M.Ghouse and M.T. Goodrich. In-place techniques for parallel convex-hull algorithm. *Proc. 3rd ACM Sympos. Parallel Algo. Architect.*, 192-203,1991

[20] M. Goodrich. Geometric partitioning made easier, even in parallel. *Proc. of the 9th ACM Symp. on Computational Geometry*, 73–82, 1993.

[21] M. Goodrich. Fixed-dimensional parallel linear programming via relative ϵ-approximations, *to appear, SODA 1996.*

[22] R L Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Proc. Lett.*, 1:132–133, 1972.

[23] T. Hagerup. Fast deterministic processor allocation. *Proc. of the 4th ACM Symposium on Discrete Algorithms* 1993, pp. 1–10.

[24] T. Hagerup and R. Raman. Waste makes haste: Tight bounds for loose, parallel sorting. *Proc. of the 33rd Annual FOCS*, pages 628- 637, 1992.

[25] D. Haussler and E. Welzl. ϵ-Nets and simplex range queries. *Discrete and Computational Geometry*, 2, 1987, pp. 127–152.

[26] T. Goldberg and U. Zwick. Optimal Deterministic Approximate Parallel Prefix Sums and Their Applications. In *Proc. Israel Symp. on Theory and Computing Systems (ISTCS'95)*, (1995), pp. 220-228.

[27] S. Kapoor and P. Ramanan. Lower bounds for maximal and convex layer problems. *Algorithmica*, pages 447–459, 1989.

[28] D G Kirkpatrick and R Seidel. The ultimate planar convex hull algorithm. *SIAM Jl. of Comput.*, 15(1):287–299, Feb. 1986.

[29] Y. Matias and U. Vishkin. Converting high probability into nearly constant time - with applications to parallel hashing. *Proc 23rd ACM Symp. on Theory of Computing*, 307–316, 1991.

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

[31] F P Preparata. An optimal real time algorithm for planar convex hulls. *Comm. ACM*, 22:402–405, 1979.

[32] F P Preparata and M I Shamos. *Computational Geometry : An Introduction.* Springer-verlag, New York, 1985.

[33] J.H. Reif and S. Sen. Optimal Parallel Randomized Algorithms for three-dimensional convex hulls and related problems. *Siam Jl. of Comput.*, 21(3),466-485, June 92.

[34] S. Rajasekaran and S. Sen. *Random sampling Techniques and parallel algorithm design.* J.H. Reif editor. Morgan, Kaufman Publishers, 1993.

[35] J.H. Reif and S. Sen. *Randomized algorithms for binary search and Load Balancing on fixed connection networks with Geometric Applications.* Siam Jl. of Comput., Vol.23, No. 3, pp 633-651, June 1994.

[36] S. Sen. Lower bounds for algebraic decision trees, complexity of convex hulls and related problems, *Proc. of the 14th FST&TCS, Madras, India* , 1994.

[37] S. Sen. Parallel multidimensional search using approximation algorithms: with applications to linear-programming and related problems. *unpublished manuscript*, 1995.

[38] M I Shamos. *Computational geometry.* PhD thesis, Yale Univ.,New Haven, 1978.

Appendix A

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 2.1 and 2.2. Checking condition of Lemma 2.1 is as given in [34]. We explain below how to check for condition of Lemma 2.2.

For every sector $\triangle$ defined by Q do in parallel.

Let $A(\triangle)$ be the number of half-planes of the $n/\log n$ sampled half-planes intersecting with $\triangle$ and let $X(\triangle)$ be the total number of half-planes intersecting with $\triangle$. Assuming $X(\triangle) > c' \log^2 n$ for some constant c' (the condition holds for the other case) and using Chernoff's bound , $L(\triangle) = k_1(A(\triangle)\log n)$ and $U(\triangle) = k_2(A(\triangle)\log n)$ are lower and upper bounds respectively for $X(\triangle)$ with high probability, for some constants k_1 and k_2. Each sector reports whether to "accept" or to "reject" a sample as follo ws:

For some constant k.

Reject a sample if $L(\triangle) > k(n/r)\log r$ $(X(\triangle) \geq L(\triangle) > k(n/r)\log r)$.

Accept a sample if $U(\triangle) \leq k(n/r)\log r$ $(X(\triangle) \leq U(\triangle) \leq k(n/r)\log r)$.

If $L(\triangle) \leq k(n/r)\log r \leq U(\triangle)$ then accept a sample. ($X(\triangle)/k(n/r)\log r \leq U(\triangle)/L(\triangle)$, which is a constant)

If any sector 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 [33] for details of the construction). Ensuring that $r^c \leq n$, gives us the required bounds. This completes the proof of Lemma 2.3.