



ACADEMIC
PRESS

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT

J. Parallel Distrib. Comput. 63 (2003) 488–500

Journal of
Parallel and
Distributed
Computing

<http://www.elsevier.com/locate/jpdc>

Faster output-sensitive parallel algorithms for 3D convex hulls and vector maxima[☆]

Neelima Gupta and Sandeep Sen*

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

Received 25 August 2000; accepted 28 February 2003

Abstract

In this paper we focus on the problem of designing very fast parallel algorithms for the convex hull and the vector maxima problems in three dimensions that are output-size sensitive. Our algorithms achieve $O(\log \log^2 n \log h)$ parallel time and optimal $O(n \log h)$ work with high probability in the CRCW PRAM where n and h are the input and output size, respectively. These bounds are independent of the input distribution and are faster than the previously known algorithms. We also present an optimal speed-up (with respect to the input size only) sublogarithmic time algorithm that uses superlinear number of processors for vector maxima in three dimensions.

© 2003 Elsevier Science (USA). All rights reserved.

1. Introduction

The study of algorithms is mainly concerned with developing algorithms for well-defined problems that are close to the best-possible (for the problem at hand). The most common measure of the performance of an algorithm is related to the running time of the algorithm. For any non-trivial problem, the running time of an algorithm increases monotonically with the size of the input. Hence the efficiency of an algorithm is usually measured as a function of the input-size. However, for many geometric problems, additional parameters like the size of the output capture the complexity of the problem more accurately enabling us to design superior algorithms. Algorithms whose running times are sensitive to the output-size have been actively pursued for problems like convex hulls [Cha95, CM92, CS89, Mat92, Sei86].

1.1. Parallel output-size sensitive algorithms

The primary objective of designing parallel algorithms is to obtain very fast running time while keeping the total work (the processor-time product) close to the best sequential algorithms. Problems for which output-size sensitive algorithms are known, it becomes especially important to have similar parallel algorithms. Unless we design parallel algorithms that are output-sensitive, the corresponding (output-sensitive) sequential may be faster for many instances. (We will often use the shorter term *output-sensitive* instead of output-size sensitive.)

The task of designing output-size sensitive algorithms on parallel machines is even more challenging 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 divide-and-conquer method is not directly effective as it cannot divide the output evenly—in fact herein lies the crux of the difficulty in designing ‘fast’ output-sensitive algorithms. By ‘fast’ we imply $O(\log h)$ or something very close, where h is the output size. The sequential output-size sensitive algorithms often use techniques like gift wrapping or incremental construction which are inherently sequential.

[☆] Some of the results of this paper appeared in a preliminary version [GS96] in the Twelfth Annual Symposium on Computational Geometry, 1996, Philadelphia, USA.

*Corresponding author. Tel.: +91-11-659-1293; Fax: +91-11-686-8765.

E-mail addresses: neelima@cse.iitd.ernet.in (N. Gupta), ssen@cse.iitd.ernet.in (S. Sen).

1.2. Parallel algorithms and model

The parallel random access machine (PRAM) has been the most popular model for designing algorithms because of its close relationship with the sequential models. For example if $S(n)$ is the best-known sequential time complexity for the 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 concurrent read exclusive write (CREW) PRAM model, the convex hull cannot be constructed faster than $\Omega(\log n)$ time, because of an $\Omega(\log n)$ bound for computing the maximum (minimum). Note that this bound is independent of the output-size. However, this bound is not applicable to the CRCW model. For parallel algebraic decision tree model, Sen [Sen97] has obtained a trade-off between the number of processors and possible speed-up for a wide range of problems in computational geometry. For convex hulls, the following is known.

Lemma 1.1 (Sen [Sen97]). *Any randomized algorithm in the parallel bounded degree 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, $k > 1$, 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.

Our algorithms are designed for the CRCW PRAM model. Although we always work in a Euclidean space \mathbb{E}^d we are mainly interested in the combinatorial complexity of the algorithm. We pretend that a single memory location can hold a real number and a processor can perform an arithmetic operation involving real numbers in a single step. In reality there may be numerical errors due to finite word lengths that we ignore. This is a non-trivial issue and deserves a separate treatment. This model is consistent with the model that is used for sequential computational geometry, called ‘Real’-RAM. For our randomized algorithms, we further assume that processors have access to $O(\log n)$ random bits in constant time.

1.3. Convex hulls

Convex polygons are very important objects in computational geometry, and in a large number of cases give rise to very efficient algorithms because of their nice property, namely convexity.

Definition 1.1. Given a set $S = \{p_1, p_2, \dots, p_n\}$ of n points in \mathbb{E}^d (the Euclidean d -dimensional space), the convex hull of S is the smallest convex polytope 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 .

In two dimensions, the ‘ordering’ in the output is simply the clockwise (or counter-clockwise) ordering of the vertices of the hull. In three dimensions it can be the cyclic ordering of the hull edges around each hull vertex.

The problem of constructing convex hulls has attracted a great deal of attention from the inception of computational geometry. In three dimensions, Preparata and Hong [PH77], described the first $O(n \log n)$ time algorithm. Clarkson and Shor [CS89] presented the first (randomized) optimal output-sensitive algorithm which ran in $O(n \log h)$ expected time, where h is the number of hull vertices. Their algorithm was subsequently derandomized optimally by Chazelle and Matoušek [CM92]. Chan [Cha95] 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. Recently, Chan [Cha95] designed an output-sensitive algorithm for convex hulls in d dimensions which runs in $O(n \log^{O(1)} h + h^{\lfloor d/2 \rfloor})$ time. Seidel [Sei86] computes F faces of the convex hull of n points in a fixed dimension d in $O(n^2 + F \log n)$ time. This can be slightly improved to $O(n^{2 - (2/(\lfloor d/2 \rfloor + 1)) + \epsilon} + F \log n)$, for any fixed $\epsilon > 0$, using a technique of Matoušek [Mat92]. In a recent paper, Amato and Ramos [AR96] have shown that the convex hull of n points in \mathbb{R}^{d+1} , $2 \leq d \leq 4$, can be computed in $O((n + F) \log^{d-1} F)$ time.

In the context of parallel algorithms, for convex hulls in three dimensions (3-D hulls) Chow [Cho80] 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 [DK89] and an $O(\log^2 n)$ time algorithm was designed by Amato and Preparata [AP92]. Reif and Sen [RS92] presented an $O(\log n)$ time and $O(n \log n)$ work randomized algorithm which was the first-known worst-case optimal algorithm for three-dimensional hulls in the CREW model. Their algorithm deals with the dual equivalent of the convex hull problem namely the intersection of half-spaces in three dimensions. The algorithm works on a

divide-and-conquer approach. It works in $O(\log n)$ phases pruning away the redundant half-spaces and keeping the work to linear in each phase. This algorithm was derandomized by Goodrich [Goo93] who obtained an $O(\log^2 n)$ time, $O(n \log n)$ work method for the EREW PRAM. In the context of output-size sensitive algorithm, one faces the problem of dividing the output points evenly so that one can finish in $O(\log h)$ phases, keeping the work linear in each phase, especially when the output-size is not known. In [GS97], the authors have shown that the convex hulls in two dimensions can be constructed in $O(\log \log n \log h)$ time with optimal work with high probability. In three dimensions, Goodrich and Ghouse [GG91] 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. [AGR94] gave a deterministic $O(\log^3 n)$ time, $O(n \log h)$ work algorithm for convex hulls in \mathbb{R}^3 on the EREW PRAM.

In higher dimensions, Amato et al. [AGR94] have shown that the convex hull of n points in \mathbb{R}^d can be constructed in $O(\log n)$ time with $O(n \log n + n^{\lfloor d/2 \rfloor})$ work with high probability and in $O(\log n)$ time with $O(n^{\lfloor d/2 \rfloor} \log^{c(\lceil d/2 \rceil - \lfloor d/2 \rfloor)} n)$ work deterministically, where $c > 0$ is a constant.

In this paper, we present a randomized algorithm for the convex hulls in three dimensions that is faster for all output-sizes while maintaining the optimal output-sensitive work bound. The algorithm exploits an observation of Clarkson and Shor [CS89], namely, that of iterative pruning of non-extreme points. We also make use of a number of sophisticated techniques like bootstrapping and super-linear processors parallel algorithms for convex hulls [Sen97] combined with a very fine-tuned analysis.

1.4. Vector maxima

Let T be a set of vectors in \mathbb{R}^d . The partial order \leq_M is defined for two vectors $x = (x_1, x_2, \dots, x_d)$ and $y = (y_1, y_2, \dots, y_d)$ as follows: $x \leq_M y$ (x is dominated by y) if and only if $x_i \leq y_i$ for all $1 \leq i \leq d$.

Definition 1.2. A vector $v \in T$ is said to be maximal if it is not dominated by any other vector $w \in T$. The problem of maximal vectors is to determine all maximal vectors in a given set of input vectors.

Relatively little work has been done in the context of parallel algorithms for the vector maxima problem. Efficient sequential algorithms are known for two and three dimensions (the $O(n \log n)$ —time algorithm is worst case optimal). However, for $h \in o(\log n)$ (h is the number of vertices on the hull) a better ($O(nh)$ time) algorithm can be easily obtained

in two dimensions by finding the point with maximum y coordinate, deleting all the points dominated by it and repeating on the reduced problem. This algorithm takes linear time for constant h . Kirkpatrick and Seidel presented an $O(n \log h)$ time algorithm and showed that the bound is tight with respect to the input and the output sizes [KS86]. In the context of parallel algorithms for two dimensions, an $O(\log n)$ time optimal algorithm can be obtained easily by using any of the $(n, \log n)$ algorithm for sorting followed by a straight forward divide-and-conquer. The best-known result for three-dimensional maxima is $O(\log n)$ time and $O(n \log n)$ operations due to Atallah et al. [ACG89] in which they have used parallel merging techniques. Using the techniques of [GS97] an $O(\log \log n \log h)$ time, optimal work algorithm can be obtained for vector maxima in two dimensions also. We present an algorithm for vector maxima in three dimensions.

1.5. Our results

Our algorithms are randomized and always provide correct output and the bounds (running time) hold with high probability independent of the input distribution. Such algorithms are often referred to as Las Vegas algorithms. The term high probability implies probability exceeding $1 - 1/n^c$ for any predetermined constant c where n is the input-size. We will use the notation \tilde{O} instead of O to denote that the bound holds with high probability.

Our algorithms achieve $O(\log \log^2 n \log h)$ time with optimal $O(n \log h)$ work with high probability. 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 work where the parallel algorithms speed-up optimally with the output size in the sublogarithmic time domain.

We also present an optimal speed-up (with respect to the input size only) sublogarithmic time algorithm that uses super-linear number of processors for vector maxima in three dimensions.

2. Some known useful results

Definition 2.1. For all $n, m \in \mathbb{N}$ and $\lambda \geq 1$, the m -color semi-sorting problem of size n and with slack λ is defined as follows: Given n integers (colors) x_1, \dots, x_n in the

range $0, \dots, m$, compute n non-negative integers y_1, \dots, y_n (the placement of x_i) such that:

- (1) All the x_i of the same color are placed in contiguous locations (not necessarily consecutive).
- (2) $\max\{y_j : 1 \leq j \leq n\} = O(\lambda n)$.

Definition 2.2. For all $n \in \mathbb{N}$ interval allocation problem is defined as follows: Given n non-negative integers x_1, \dots, x_n , compute n non-negative integers y_1, \dots, y_n (the starting addresses of intervals of sizes 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 2.3. 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 2.1 (Bast and Hagerup [BH93]). *There is a constant $\varepsilon > 0$ such that for all given $n, k \in \mathbb{N}$, n -color semi-sorting problem of size n and with slack $O(\log^{(k)} n)$ can be solved on a CRCW PRAM in $O(k)$ time, using $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\varepsilon}$. Alternatively, it can be done in $\tilde{O}(t)$ 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.

Definition 2.4. Given n input keys in an array, the problem of *padded-sorting* is to place them in a sorted order in an array of size $(1 + \lambda)n$, where $\lambda > 0$ is called the padding factor.

Lemma 2.2 (Hagerup and Raman [HR92]). *The problem of padded-sorting can be solved in expected $O(\log n / \log k)$ time with nk processors in a CRCW PRAM with high probability.*

Lemma 2.3 (Brent [Bre74]). *Let A be a parallel algorithm that executes in time T and performs a total number of W operations (each operation takes unit time). Then the algorithm can be executed in time $O(\lfloor W/p \rfloor + T)$ using p processors.*

The above lemma is referred to as *Brent's slow down lemma* or simply the *slow down lemma*.

Lemma 2.4 (Gupta and Sen [GS97]). *The convex hull of n points in two dimensions can be constructed in $O(\log h \log \log n)$ expected time and $O(n \log h)$ opera-*

tions with high probability in a CRCW PRAM model where h is the number of points on the hull.

Lemma 2.5. *The vector maxima of n vectors in two dimensions can be computed in $O(\log h \log \log n)$ expected time and $O(n \log h)$ operations with high probability in a CRCW PRAM model where h is the number of maximal vectors.*

Proof. The result can be obtained using the techniques used for the 2d-convex hulls in [GS97]. \square

Lemma 2.6 (Sen [Sen97]). *The vector maxima of n vectors in two dimensions can be computed in $\tilde{O}(\log n / \log k)$ time using nk CRCW processors.*

3. Brief overview of the algorithm for convex hulls

The problem of constructing the convex hull of points in three dimensions is well known to be equivalent to the problem of finding the *intersection of half-spaces*. Here we give an algorithm for the latter which implies a solution for the former.

Let us denote the input set of half-spaces by S and their intersection by $P(S)$. We construct the intersection $P(R)$ of a random sample R of r half-spaces and filter out the redundant half-spaces, i.e. the half-spaces which do not contribute to $P(S)$. Without loss of generality, we can assume that the origin lies inside the intersection. 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. We further partition trapezoids into triangles. The convex closure of the origin O with a triangle from the cutting of the faces defines a region which we call the *cones*. These cones will be intersected by bounding planes of a number of half-spaces that were not chosen in the sample. We say that a half-space *intersects* a cone if its bounding plane intersects the cone. We say that a half-space *conflicts* with a cone if its bounding plane intersects the cone. A cone is said to be *critical* if it contains an output point. We delete the half-spaces which do not intersect with any critical cone. The procedure is repeated on the reduced problem.

To prove any interesting result we must determine how quickly the problem size decreases. The random sampling lemma in the next section show that for a large sample ($> \Omega(h^2)$) the size of the problem decreases very quickly.

4. Random sampling lemma

Let $H(R)$ denote the set of cones induced by a sample R and let $H^*(R)$ denote the set of critical cones. We will

denote the set of half-spaces intersecting a cone $\Delta \in H(R)$ 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.

Lemma 4.1 (Clarkson and Shor [CS89], Rajasekaran and Sen [RS93]). *For some suitable constant k and large n ,*

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

where the probability is taken over all possible choices of the 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 4.2 (Clarkson and Shor [CS89], Haussler and Welzl [HW87]). *For some suitable constant k_1 and large n ,*

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

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

A sample is ‘good’ if it satisfies the properties of Lemmas 4.1 and 4.2 simultaneously. Clearly, a sample is ‘good’ with probability at least $\frac{1}{16}$. We can actually do better as we show in the following lemma.

Lemma 4.3. *We can find a sample R which satisfies both Lemmas 4.1 and 4.2 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 technique of Reif and Sen [RS92]. For details see Appendix A. \square

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

Lemma 4.4. *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 cones that contain at least one output point.

This will be used repeatedly in the analysis to estimate the non-redundant half-spaces whenever $h \leq r/\log r$.

The above Lemma can be generalized to any configuration space with bounded valence. Let N be a set of objects. A configuration space defined by N is denoted by $\Pi(N)$. A configuration $\sigma \in \Pi(N)$ is defined by the pair $(D(\sigma), L(\sigma))$ where $D(\sigma)$ is the set of objects in N defining σ and $L(\sigma)$ is the set of objects in N conflicting with σ . $L(\sigma)$ is called the *conflict list* of σ . We denote the cardinality $|L(\sigma)|$ by $l(\sigma)$ and call it the *conflict size*. Let $\Pi^i(N)$ denote the set of configurations in $\Pi(N)$ with $l(\sigma) = i$. Let R be a random subset of N . Define a subspace $\Pi(R)$ as the set of configurations $\sigma \in \Pi(N)$ with $D(\sigma) \subseteq R$ and $L(\sigma) \cap R$ as the conflict list. Then, $\Pi^0(R)$ is the set of configurations in $\Pi(R)$ with $L(\sigma) \cap R = \emptyset$. Then we have the following lemma.

Lemma 4.5 (Ketan [Ket94]). *For a configuration space $\Pi(N)$, for some suitable constant k and large n ,*

$$\Pr \left[\sum_{\sigma \in \Pi^0(R)} l(\sigma) \geq kn \right] \leq 1/c$$

for some constant $c > 1$, where the probability is taken over all possible choices of the random subset R .

Definition 4.1. A configuration space is said to have bounded valence if the number of configurations having the same set of defining objects is bounded (constant).

Lemma 4.6 (Ketan [Ket94]). *For a configuration space $\Pi(N)$ with bounded valence, for some suitable constant k_1 and large n ,*

$$\Pr \left[\max_{\sigma \in \Pi^0(R)} l(\sigma) \geq k_1(n/r) \log r \right] \leq 1/c$$

for some constant $c > 1$, where the probability is taken over all possible choices of the random subset R such that $|R| = r$.

5. Algorithm

We give below a general algorithm for both the convex hull and the vector maxima problems. In the later sections we describe how each step is carried out in the context of a specific problem and then give a combined analysis.

Let S be the input set of n objects. Objects are half-spaces for the intersection of half-spaces or the convex hull and they are vectors or points for the vector maxima problem. The algorithm is iterative. Let n_i (respectively r_i) denote the size of the problem (respectively the sample size) at the i th iteration with $n_1 = n$. A typical iteration of our algorithm is shown in Fig. 1. Repeat the procedure until $r_i > n^e$ (this condition

Procedure Rand(*i*)

1. Use Resampling and Polling to choose a ‘good’ sample R of size $r_i = \text{constant}$ for $i = 1$ and $\max\{r_{i-1}^2; h_{i-1}^*\}$ for $i > 1$ where h_i^* is the maximum output size of the corresponding 2d problem (dened later in Sections 6.2 and 7.3. The reason for including h_{i-1}^* is also explained there).
2. Solve the problem on R .
3. Dene regions on the basis of the solution obtained in Step 2 — explained in Sections 3 and 7.1. Let $H(R)$ denote the set of regions induced by R . We say that an input object is *redundant* if it does not contribute to the output. Filter out the redundant input as follows.
 - (a) i. For every input object find out the regions it conflicts with - the term *conflict* is defined in Section 3 for convex hulls and in Section 7.1 for vector maxima.
 - ii. If the sum, taken over all the input objects, of the number of regions it conflicts with is $O(n)$ then continue else go to 1.
 - (b) Call a region *critical* if it contains an output. Denote the set of critical regions by $H^*(R)$. Find out the set $H^*(R)$ — explained in Sections 6.2 and 7.3.
 - (c) Delete an input object if it does not belong to $\cup_{\Delta \in H^*(R)} L(\Delta)$.
4. The input for the next iteration is $\cup_{\Delta \in H^*(R)} L(\Delta)$. Size of the reduced problem for the next iteration is $n_{i+1} = |\cup_{\Delta \in H^*(R)} L(\Delta)|$. Increment i .

end.

Fig. 1.

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 solve the problem directly else do one more iteration and solve the problem directly.

5.1. Overview of analysis

Since $r_i < n^\epsilon$, Step 2 can be done in constant time using sublogarithmic time, super-linear number of processors algorithm for the problem or by a brute force method. Also, Step 3(a)i can be reduced to a point location problem and hence can be done in $O(\log r_i / \log \frac{p}{m})$ time with p processors. Steps 3(a)ii and 3(c) can be done by applications of interval allocation and compaction which can be done very fast by Lemma 2.1. The hard part of the algorithm is Step 3(b). We shall discuss the algorithm for the convex hull problem and the vector maxima problem in three dimensions giving the details of Step 3(b).

6. Convex hulls

In order to get a fast algorithm we must be able to determine the intersections of the half-spaces with the cones and determine the critical cones quickly.

6.1. Finding intersections quickly

To find the intersections of the bounding planes of the half-spaces with the cones quickly we use the locus-

based scheme as described by Reif and Sen [RS92]. Reif and Sen extended the preprocessing scheme for point location due to Dobkin and Lipton. For our purpose, their result can be restated as the following lemma:

Lemma 6.1. *Given a set of m planes in \mathbb{E}^3 , it can be preprocessed in $O(\log m / \log k')$ time using $O(m^7 k')$ processors, such that given an arbitrary query point with k processors, the unique cell containing the point can be reported in $O(\log m / \log k)$ time. The space required is $O(m^7)$.*

For each of the cells in 3-space, we can precompute the cones that the corresponding plane intersects by choosing a representative point in each cell and testing it against all the cones. We also store the number of intersecting cones for each of the subdivisions.

We have the following result.

Lemma 6.2. *Suppose we have a set R of r half-spaces and n processors. For $r = O(n^\epsilon)$, $\epsilon < 1/8$, $P(R)$ can be preprocessed in constant time so that given any plane with k processors we can report the list of cones it intersects in $O(\log r / \log k)$ time.*

6.2. Finding the critical cones

Consider a cone C . Consider the intersection of $P(S)$ with $\text{bd}(C)$, the boundary of C . This intersection consists of three two-dimensional convex polygonal chains called *contours*, one on each of the three faces of

C that contain the origin O . We can construct the contours using algorithm for two-dimensional convex hulls. Note that all half-spaces contributing an edge to a contour contribute a face to $P(S)$. Hence if h_i^* denote the maximum size of any contour over all the regions during iteration i then $h_i^* \leq h \forall i$. Unfortunately there may be

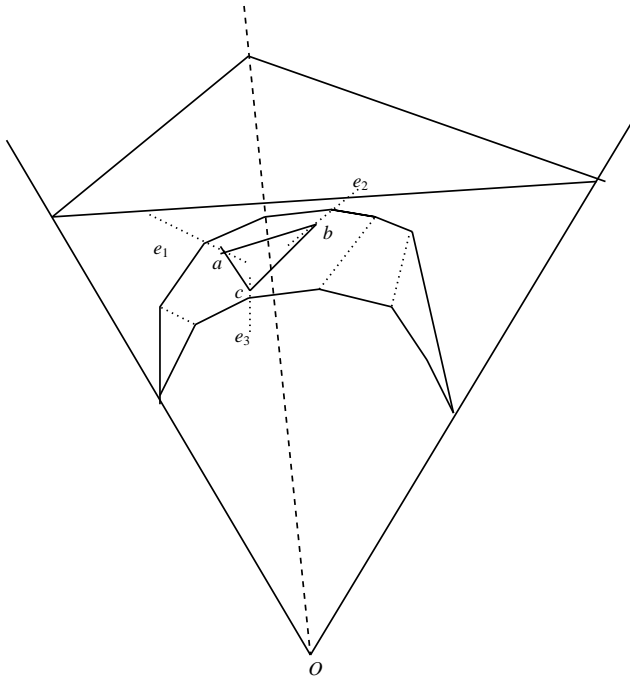


Fig. 2. The half-space defined by the vertices a, b, c contributes the face abc to $P(S)$ but does not show up in any of the contours.

half-spaces contributing a face to $P(S)$ that do not show up in the contours. Consider a half-space whose bounding plane chops off a portion of the polytope within a cone (see Fig. 2).

Now, if a cone C contains an output vertex then clearly all the output vertices in C cannot be contributed by half-spaces which do not contribute an edge to a contour, i.e. at least one of the output vertices in C is contributed by a half-space that contributes an edge to a contour (in fact at least two half-spaces that contribute a vertex to a contour) (see Fig. 3). Thus it suffices to concentrate only on those half-spaces each of which contributes a vertex to a contour. Let p be a vertex on a contour of C . Let e be the output edge defined by the two half-spaces contributing the two edges defining p . Then e contributes an output vertex in C if and only if it does not contribute a vertex to any other contour of C . Let the vertices of the contours be labelled by the two half-spaces contributing the edges defining them. Sort the vertices by their labels. A half-space contributes an output vertex in C if and only if it defines a vertex (on a contour) whose label appears exactly once (if a label appears twice then the edge between the corresponding two vertices does not contribute a vertex to C , for example, the edge E in Fig. 3 does not contribute an output vertex to the cone).

Lemma 6.3. *In the i th iteration of the algorithm critical cones can be determined in $\tilde{O}(\log \log n \log h_i^*)$ time with optimal work.*

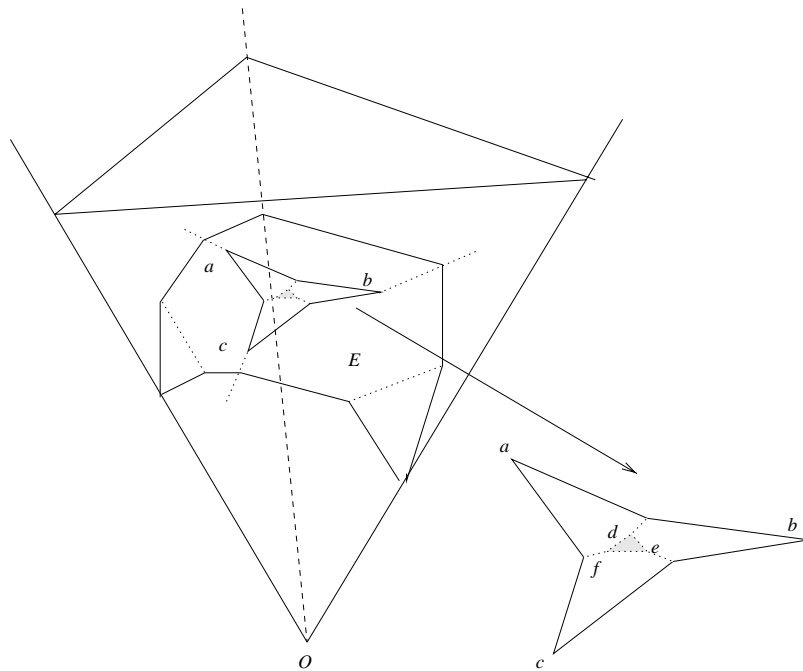


Fig. 3. a, b, c, d, e and f are all output vertices. a, b, c are contributed by the half-spaces each of which contributes a vertex to the contours. The edge E does not contribute an output vertex to the cone.

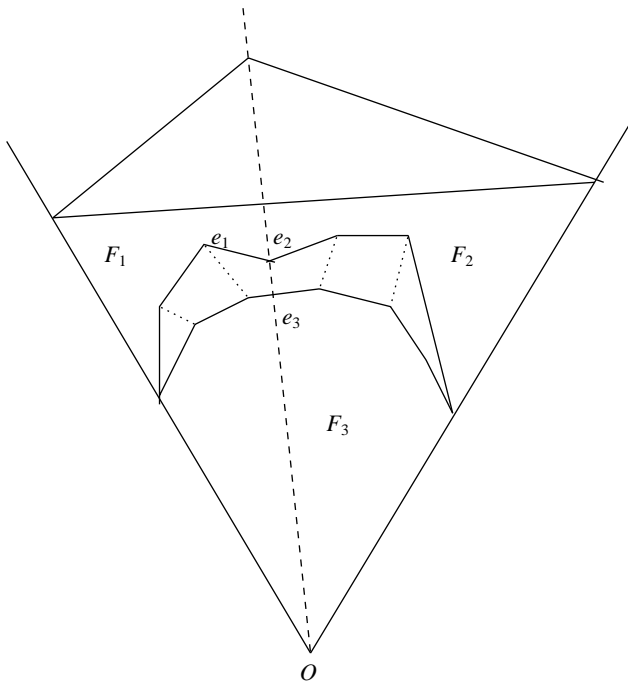


Fig. 4. A cone not containing any output vertex. The edges e_1, e_2 and e_3 are coplanar, say they lie in a plane P . Then, e_i is the intersection of P with the face F_i of the cone for $i = 1, 2, 3$.

Proof. The time and work bounds for determining the critical cones (Fig. 4) are dominated by those for computing the 2d convex hulls. The result follows from Lemma 2.4. \square

7. Vector maxima

Let S be the set of input vectors. We pick up a random sample R of the input vectors and compute its maxima and define regions. We say that a vector *conflicts* with a region if it can potentially dominate the vectors in that region. Determine the critical regions, i.e. the regions containing an output vector, delete the vectors not conflicting with any critical region and iterate on the reduced problem. As in the case of convex hulls, in order to get a fast algorithm we must be able to detect the vectors conflicting with a region and the critical regions quickly. At first, we will define the regions for the problem and then explain how to find out the conflicting vectors and determine the critical regions.

7.1. Define regions

In this section we describe how to define regions for this problem and in the next section we explain how to determine the critical regions. We will use the terms *vectors* and *points* interchangeably. Let us first define a configuration space over S and denote it by $\Pi(S)$.

Consider the projection of all the points of S in the yz plane. Also include the points at infinity on y and z axes. Consider a triplet of points p_1, p_2 and p_3 . Consider all possible rectangles formed by their y and z coordinates ($3 \times 3 = 9$), for example rectangle formed by the lines $y = y(p_1), y = y(p_2), z = z(p_1)$ and $z = z(p_3)$ where $y(p)$ and $z(p)$ are the y and z coordinates of a point p . Consider a rectangle bounded by the lines $y = y_1, y = y_2, z = z_1$ and $z = z_2$ with $y_1 < y_2$ and $z_1 < z_2$ where $y_1, y_2 \in \{y(p_1), y(p_2), y(p_3)\}$ and $z_1, z_2 \in \{z(p_1), z(p_2), z(p_3)\}$. Let C be a rectangular vertical strip defined by the rectangle above. That is, C is a rectangular vertical strip bounded by the planes $y = y_1, y = y_2, z = z_1$ and $z = z_2$ with $y_1 < y_2$ and $z_1 < z_2$.

We define a configuration σ as a rectangular cylinder $\{(x, y, z) : x \geq x(p), y_1 \leq y \leq y_2, z_1 \leq z \leq z_2\}$, where p is a point of S in C and $x(p)$ denote the x coordinate of p . Thus each rectangular vertical strip defines a number of configurations (as many as the number of points of S in C), each differs from the other in at least one point i.e. p . Thus $D(\sigma)$ consists of points defining the bounding planes of σ , i.e. the triplet p_1, p_2, p_3 and p (also see Fig. 5).

We say that a point in S *conflicts* with σ if and only if it can potentially dominate a point in σ . Clearly a constant (at most 9) number of configurations have the same set of defining points (a point p may belong to all the nine regions obtained by the rectangles defined by the y and z coordinates of p_1, p_2, p_3). Hence our configuration space has bounded valence. Thus for a randomly chosen sample R of S the sampling Lemmas 4.5 and 4.6 hold in this case also.

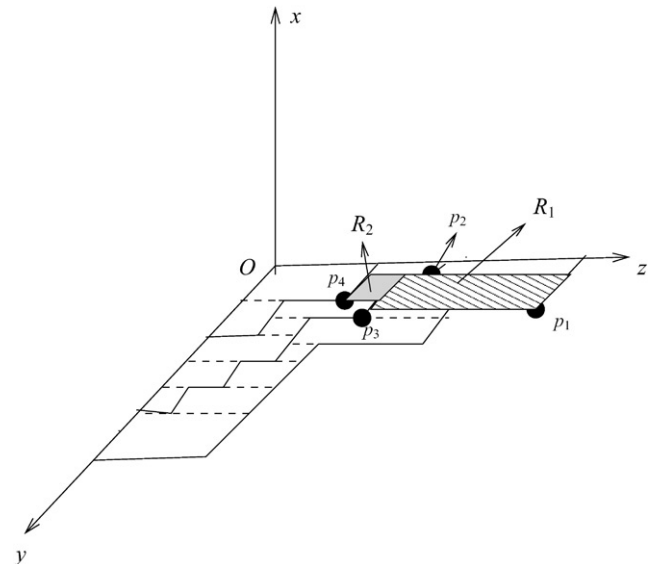


Fig. 5. The configuration σ_1 and σ_2 defined by R_1 and R_2 , respectively, have the defining sets $D(\sigma_1) = \{p_1, p_2, p_3, p^*\}$ and $D(\sigma_2) = \{p_2, p_3, p_4, q^*\}$ where p^* and q^* are the points of R with maximum x -coordinate in the rectangular vertical strip defined by the rectangles R_1 and R_2 , respectively.

Next we define a subspace of $\Pi(R)$, denote it by $\Pi^*(R)$ and show that $\Pi^*(R) \subseteq \Pi^0(R)$. Then, the results of Lemmas 4.5 and 4.6 hold for $\Pi^*(R)$ also. We define the configurations in $\Pi^*(R)$ as follows: denote the vector maxima of the random sample R by $P(R)$. Take the projection of the vertices of $P(R)$ on the yz plane. This consists of layers of 2d-maxima on the yz plane as shown in Fig. 5. Form rectangles by drawing perpendiculars, parallel to z -axis from a vertex in a layer to adjacent layers or y -axis whichever is closer. Clearly a point can contribute an edge to at most four rectangles. Hence the number of rectangles is $O(r)$, where r is the size of R . Notice that a rectangle in the yz plane is defined by exactly three points (one on one layer of the 2d-maxima and two on the other). For rectangles adjacent to y - or z -axis the third point may be a point at infinity on y - or z -axis. Each rectangle so formed defines a number of configurations in $\Pi(R)$ (as many as the number of points of R in the rectangular vertical strip defined by the rectangle).

Consider a rectangular vertical strip C defined by the rectangle bounded by the lines $y = y_1, y = y_2, z = z_1$ and $z = z_2$ with $y_1 < y_2$ and $z_1 < z_2$. That is, C is a rectangular vertical strip bounded by the planes $y = y_1, y = y_2, z = z_1$ and $z = z_2$ with $y_1 < y_2$ and $z_1 < z_2$. Let p^* be a point of R in C with maximum x -coordinate. We define a configuration $\sigma \in \Pi^*(R)$ as a rectangular cylinder $\{(x, y, z): x \geq x(p^*), y_1 \leq y \leq y_2, z_1 \leq z \leq z_2\}$. Thus $D(\sigma)$ is the set of points defining the bounding planes of σ (see Fig. 5). Thus $\Pi^*(R) \subseteq \Pi(R)$. Clearly, the set of points of R , conflicting with $\sigma \in \Pi^*(R)$ is empty, i.e. $L(\sigma) \cap R$ is empty. That is $\Pi^*(R) \subseteq \Pi^0(R)$.

Each configuration $\sigma \in \Pi^*(R)$ defines a region which we shall denote by F . There are $O(r)$ regions.

7.2. Finding intersections

The regions that a point conflicts with can be determined by using the locus-based scheme. That is, the vertices of $P(R)$ are sorted on x, y and z coordinates. They define r^3 cubes. Take a representative point in each cube and move it around in the cube. The set of regions it conflicts with can only change when it crosses the boundary of the cube. Fig. 6 illustrates this in dimension two. Hence for each cube one can precompute the regions it conflicts with by taking a representative point in each cube and checking it against each region. This can be done using r processors for each cube. The preprocessing can be done in constant time with r^4 processors. Thus the regions that a point conflicts with can then be determined by a binary search. Hence we have the following lemma:

Lemma 7.1. Suppose we have a set R of r vectors and n processors. For $r = O(n^\epsilon), \epsilon < 1/4, P(R)$ can be preprocessed in constant time so that given any vector with k

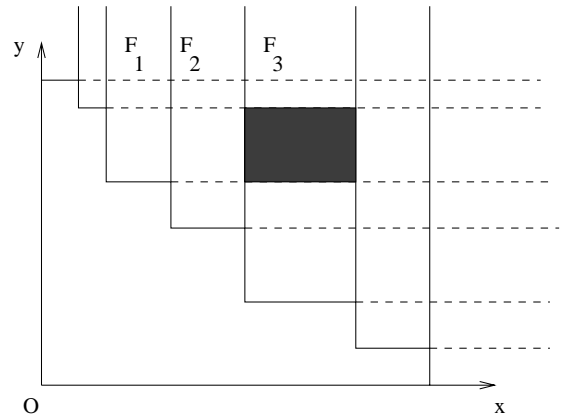


Fig. 6. The shaded rectangle (a cube in three dimensions) is the locus of points conflicting with the regions F_1, F_2, F_3 .

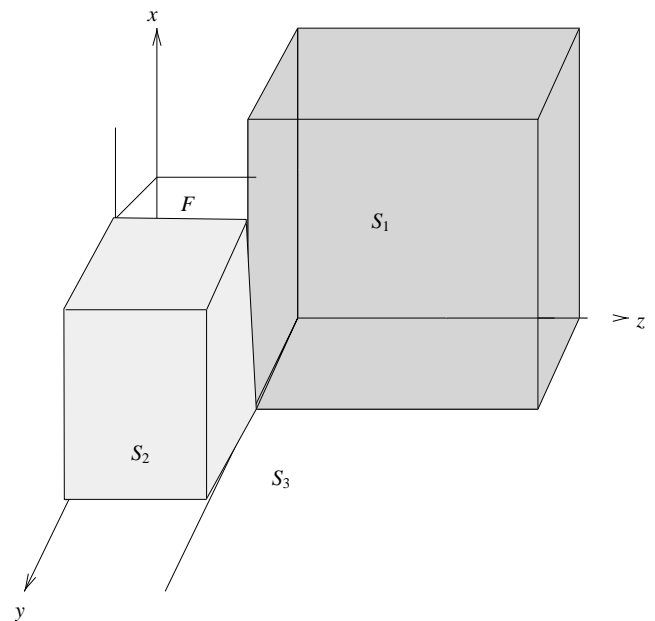


Fig. 7.

processors we can report the list of regions it conflicts with in $O(\log r / \log k)$ time.

7.3. Finding the critical regions

For each region F as defined in Section 7.1, define $S_1 = \{(x, y, z): x \geq x(p^*), y_1 \leq y \leq y_2, z \geq z_2\}$, $S_2 = \{(x, y, z): x \geq x(p^*), y \geq y_2, z_1 \leq z \leq z_2\}$, $S_3 = \{(x, y, z): x \geq x(p^*), y \geq y_2, z \geq z_2\}$.

See Fig. 7. Clearly, a point in S conflicts with F if and only if it is in $S_1 \cup S_2 \cup S_3 \cup F$.

Claim 7.1. Consider the projection of points in $S_1 \cup S_3$ on xy plane and compute their two-dimensional maxima. Denote it by M_1 . A point p in a region F is dominated by a

point in $S_1 \cup S_3$ if and only if it is dominated on xy coordinates by some vertex on M_1 .

Proof. Let $p \in F$ is dominated on x, y coordinates by a vertex p' of M_1 . Then, since $z(p) \leq z_2 \leq z(p')$, p is dominated by p' in $S_1 \cup S_3$.

Conversely, let $p \in F$ is dominated by a point p^* in $S_1 \cup S_3$. If projection of p^* is on M_1 then we are through else there exists $p' \in S_1 \cup S_3$, whose projection is a vertex on M_1 , that dominates p^* on x, y coordinates, which implies p' dominates p on x, y coordinates. \square

Claim 7.2. A vertex on M_1 is an output vertex.

Proof. For this notice that a point in S_3 can be dominated only by points in S_3 and a point in S_1 can be dominated only by points in $S_1 \cup S_3$. A vertex on M_1 is not dominated on x, y coordinates by any point in $S_1 \cup S_3$ and hence is an output vertex. \square

Hence if h_i^* denotes the maximum size of the 2d maxima over all the regions during iteration i then $h_i^* \leq h \forall i$.

Lemma 7.2. Let M be a vector maxima of some points in the xy plane. Let m be the number of points on M . Given a point p in xy plane, whether it is dominated by a point on M can be determined in $O(\log m / \log k)$ time with k processors.

Proof. Let $p_1, p_2 \dots p_m$ be the points on M in the order of decreasing x coordinate. If $x(p) > x(p_1)$ then p is not dominated by any point on M . Otherwise, find out p_i, p_{i+1} such that $x(p_{i+1}) < x(p) \leq x(p_i)$ by a binary search on the x coordinates of the points on M . p is clearly dominated by a point on M if and only if $y(p) \leq y(p_i)$. Hence the result. \square

Thus the points in F which are dominated by points in $S_1 \cup S_3$ can be determined by a binary search on M_1 . Similarly, the points in F which are dominated by points in $S_2 \cup S_3$ can be determined. If there is a point in F which is not dominated by any point in $S_1 \cup S_2 \cup S_3$ (it is not necessarily an output point because it may be dominated by some point in F) then F contains an output point.

Lemma 7.3. In the i th iteration of the algorithm the critical regions can be determined in $\tilde{O}(\log \log n \log h_i^*)$ time with $O(n \log h_i^*)$ work with high probability.

Proof. The time and work bound for determining the critical regions are dominated by those for computing the 2d vector maxima. The required bounds follow from Lemma 2.5. \square

8. Analysis

Assume that $h = O(n^\delta)$ for some δ between 0 and 1, for otherwise the problem can be solved in $O(\sum \log r_i) = O(\log n) = O(\log h)$ time with $O(n \log h)$ work.

Since $r_i = O(n^\varepsilon)$, $\varepsilon < 1$ therefore Step 2 can be done in constant time by a brute-force method followed by sorting with r_i^2 processors. Step 3(a)i (finding conflicts) can be done in $O(\log r_i / \log \frac{r_i}{h_i^*})$ time using p processors by Lemma 6.2 for convex hulls and by Lemma 7.1 for vector maxima. An application of semi-sorting then gives us the set of input objects conflicting with a region which can be done in $O(\log \log n)$ time with linear work by Lemma 2.1. Critical regions can be determined in $\tilde{O}(\log \log n \log h_i^*)$ time and $O(n \log h_i^*)$ work by Lemma 6.3 for convex hulls and by Lemma 7.3 for vector maxima. Redundant objects are deleted by an application of compaction which can also be done in $O(\log \log n)$ time with linear work by Lemma 2.1. The processor reallocation can also be done in the same bounds by Lemma 2.1.

The time and work bounds for all the steps except 3(a)i are thus dominated by those for Step 3(b), i.e. determining the critical regions. By Lemmas 2.4 and 2.5 the time for Step 3(b) over the entire algorithm can be expressed as

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

where l is the iteration in which the sample size $> \Omega(h^2)$ for the first time and $T2(n_i, h)$ denotes the parallel time for solving the two-dimensional problem with the 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 $\tilde{O}(\log \log n \sum_{i < l} \log r_{i+2})$ which is $\tilde{O}(\log \log n \log h)$ (since the summation is a geometric series with $\log h$ as the leading term). The second term can be bounded by $\tilde{O}(\log \log^2 n \log h)$ as the summand can be bounded by $\tilde{O}(\log \log n \log h)$ and the maximum number of iterations is $O(\log \log n)$. The work for this step over the entire algorithm is

$$\begin{aligned} & \sum_{r_i \leq h} \tilde{O}(n \log h_i^*) + \sum_{i > l} \tilde{O}(n_i \log h) \\ & \leq \sum_{r_i \leq h^2} \tilde{O}(n \log r_i) + \log h \sum_{i > l} \tilde{O}(n_i), \end{aligned}$$

$$\begin{aligned} & \text{where } n_i = \tilde{O}(n_{i-1}/2) \text{ for } i > l, \\ & = \tilde{O}(n \log h). \end{aligned}$$

By Brent's slow down lemma all steps except 3(a)i can be in $O(\log \log^2 n \log h)$ time with $n / \log \log^2 n$ processors. Strictly speaking, we must also add to this the time for processor allocation but it has already been accounted for in the previous calculations.

Using Lemmas 6.2 and 7.1, the time for Step 3(a) with $p = n/\log \log^2 n$ processors is

$$\leq \log \log^2 n \sum_{i \leq l} \log r_i + \sum_{p \leq n_i, i > l} \frac{n_i}{p} \log r_i + \sum_{p > n_i, i > l} \frac{\log r_i}{\log \frac{p}{n_i}}$$

The first term of the summation is bounded by $O(\log \log^2 n \log h)$. The second term is bounded by $O(\log \log^2 n)$ since by Lemma 4.4, for $i > l$, we have $n_i \log r_i = O(n_{i-1} \log^2 r_i/r_{i-1}) = O(n_{i-1} \log^2 r_{i-1}/r_{i-1}) = O(n_{i-1}/\sqrt{r_{i-1}}) = O(n_{i-1}/h) = O(n_{i-1}/2)$.

When $n_i < p$, then

$$\begin{aligned} n_i &= O(n_{i-1} h (\log r_{i-1}) / r_{i-1}) = O(p h (\log r_{i-1}) / r_{i-1}) \\ \Rightarrow p/n_i &= \Omega(r_{i-1} / h \log r_{i-1}) \\ &= \Omega(\sqrt{r_{i-1}} / \log r_{i-1}) \quad \text{for } i > l. \end{aligned}$$

So $O(\log r_i / \log \frac{p}{n_i})$ above is constant. Thus the last term of the summation is bounded by the maximum number of iterations, i.e. by $O(\log \log n)$. The total time over all the iterations is thus $O(\log \log^2 n \log h)$.

Time to solve the problem directly: Let the terminating condition be satisfied in the t th iteration. If $n_t < n^\epsilon$, then the problems can be solved in constant time using a brute force method. Otherwise, if $n^\epsilon < r_t < n_t$ or $n^\epsilon < n_t < r_t$ then we have the following: clearly, $r_{t-1} < n^\epsilon$ and $r_t < n^{2\epsilon}$. Let ϵ be $< 1/2$. Then, we can afford to do one more iteration within the same bounds. Now, $n_{t+1} = O(n_t h \log r_t / r_t) = O(n_t h (2\epsilon \log n) / n^\epsilon) = O(n^{1-\epsilon/2} h)$. So, for $\delta \leq \epsilon/4$, $h = O(n^{\epsilon/4}) \Rightarrow n_{t+1}$ is $O(n^{1-\epsilon/4})$ and hence the problems can be solved directly in constant time.

We summarize our results in the following theorems.

Theorem 8.1. *The convex hull of n points in three dimensions can be constructed in $O(\log h \log \log^2 n)$ 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.*

Theorem 8.2. *The vector maxima of n points in three dimensions can be constructed in $O(\log h \log \log^2 n)$ 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 maxima.

9. Sublogarithmic algorithm for 3d maxima

In this section we present a sublogarithmic time algorithm that achieves optimal speed-up (with respect to the input-size only) and uses super-linear number of processors for the vector maxima problem in three dimensions.

9.1. Algorithm

Let S be the input set of n vectors. A sublogarithmic time randomized algorithm to compute their maxima is shown in Fig. 8.

9.2. Analysis

Assume the availability of nk processors. For $c = 3$, the sample size r is $(nk)^{\frac{1}{3}} \log n$ and the maxima of these points can be computed in constant time using r^2 processors by a brute-force method. Critical regions are determined in $O(\log n / \log k)$ time for point location plus $O(\log n / \log k)$ time for semi-sorting (using Lemma 2.2) plus $O(\log n / \log k)$ time by Lemma 2.6 to compute the 2d maxima and a binary (multiway) search on it. Compaction (deletion of redundant points) can be done in the required bounds using sorting. Thus if $T(n, m)$ represents the parallel running time for the input size n with m processors then

$$T(n, nk) = T(n / (nk)^{\frac{1}{c}}, nk / (nk)^{\frac{1}{c}}) + a \log n / \log k$$

constants c and a are greater than 1. The solution of this recurrence with appropriate stopping criterion is $O(\log n / \log k)$ by induction. Hence we have the following.

Theorem 9.1. *The three-dimensional vector maxima problem can be solved in $O(\log n / \log k)$ with nk processors in a CRCW model, with high probability.*

Algorithm 3D_Max1(S)

1. Pick up a good sample R of size $(nk)^{\frac{1}{c}} \log n$, for some constant $c > 1$ which will be decided later.
2. Compute $P(R)$.
3. Define regions as explained in Section 7.1. Denote the set of regions by $H(R)$.
4. Determine the set $H^*(R)$ of critical regions and delete the redundant (dominated) points as explained in Section 7.3.
5. If the size of a subproblem is $O(\log^{\Omega(1)} n)$ then solve directly else solve recursively.

end.

Fig. 8.

Appendix A

Since the events of Lemmas 4.1 and 4.2 would fail only with constant probability, the probability that the conditions would fail in $O(\log n)$ independent trials is less than $1/n^\alpha$ for some $\alpha > 0$. So we select $O(\log^2 n)$ independent samples. One of them is ‘good’ with high probability. However, to determine if a sample is ‘good’ we will have to do Step 3(a) $\log^2 n$ times each of which will take $O(\log r)$ time with linear number of processors. We cannot afford to do this. We use the technique called *polling* given by Reif and Sen [RS92]. That is, we try to estimate the number of half-planes intersecting a region using only a fraction of the input half-planes.

Consider a sample Q . Check it against a randomly chosen sample of size $n/\log^3 n$ of the input half-planes for the conditions of Lemmas 4.1 and 4.2. For every region Δ defined by Q , let $A(\Delta)$ be the number of half-planes of the $n/\log^3 n$ sampled half-planes intersecting with Δ and let $X(\Delta)$ be the total number of half-planes intersecting with Δ . Let $X(\Delta) > c' \log^4 n$ for some constant c' —the conditions of Lemmas 4.1 and 4.2 hold trivially for the other case (for $r < cn/\log^4 n$ for some constant c , $(n \log r)/r > n/r > (1/c) \log^4 n > (1/cc')X(\Delta)$ and $\sum_{\Delta} X(\Delta) \leq c'r \log^4 n < c'cn$). By using Chernoff's bounds for binomial random variables, $L = k_1(\sum_{\Delta} A(\Delta) \log^3 n)$ and $U = k_2(\sum_{\Delta} A(\Delta) \log^3 n)$ are the lower and the upper bounds, respectively, for $X = \sum_{\Delta} X(\Delta)$ with high probability, for some constants k_1 and k_2 . For some constant k , accept or reject a sample as shown in Step 2 of the Procedure Polling. With high probability about $\log n$ samples will be accepted. Clearly, $\sum_{\Delta} A(\Delta) \leq (k/k_1)n/\log^3 n$ for the accepted samples. Also, again by using Chernoff's bounds for binomial random variables, $L(\Delta) = k'_1(A(\Delta) \log^3 n)$ and $U(\Delta) = k'_2(A(\Delta) \log^3 n)$ are the lower and the upper bounds, respectively, for $X(\Delta)$ with high probability, for some constants k'_1 and k'_2 . For some constant k' , accept or reject a sample as shown in Step 3 of the Procedure Polling. If any region reports ‘reject’ the sample then reject the sample. The Procedure Polling works for $r = o(n/\log^4 n)$.

Procedure Polling

1. Select $O(\log^2 n)$ random samples independently. Denote them by Q_1, Q_2, \dots, Q_m , $m = O(\log^2 n)$.
2. For Q_i , $i = 1$ to m do in parallel
 - (a) Reject the sample if $L > kn$ ($X \geq L > kn$) and stop.
 - (b) Accept the sample if $U \leq kn$ ($X \leq U \leq kn$).
 - (c) If $L \leq kn \leq U$ then accept the sample ($X/kn \leq U/L$, which is a constant).
3. Let Q'_1, Q'_2, \dots, Q'_l denote the set of samples accepted above. Then, $l = \Omega(\log n)$ with high probability.

For Q'_i , $i = 1$ to l do in parallel

For each $\Delta \in H(Q'_i)$ do in parallel

(a) Reject the sample if $L(\Delta) > k'(n/r) \times \log r$ ($X(\Delta) \geq L(\Delta) > k'(n/r) \log r$).

(b) Accept the sample if $U(\Delta) \leq k'(n/r) \times \log r$ ($X(\Delta) \leq U(\Delta) \leq k'(n/r) \log r$).

(c) If $L(\Delta) \leq k'(n/r) \log r \leq U(\Delta)$ then accept the sample.

$(X(\Delta)/k'(n/r) \log r \leq U(\Delta)/L(\Delta))$, which is a constant)

end

For sample size r , the entire procedure runs in $O(\log r)$ steps using n processors after building a data-structure for point location [RS92] of size r^c in $O(\log r)$ time, where c is a fixed constant (for a sample of size $O(n^\epsilon)$, intersection of half-planes can be computed in constant time, the region that a half-plane intersects can be computed in $O(\log r)$ time by Lemma 6.1 and an application of semi-sorting gives the half-planes that a region intersects in constant time by Lemma 2.1). Ensuring that $r^c \leq n$, gives us the required bounds. This completes the proof.

References

- [AGR94] N.M. Amato, M.T. Goodrich, E.A. Ramos, Parallel algorithms for higher-dimensional convex hulls, Proceedings of the 35th Annual FOCS, Santa Fe, NM, 1994, pp. 683–694.
- [AP92] N.M. Amato, F.P. Preparata, The parallel 3d convex hull problem revisited, Internat. J. Comput. Geom. Appl. 2 (2) (1992) 163–173.
- [AR96] N.M. Amato, E.M. Ramos, On computing voronoi diagrams by divide-prune-and-conquer, ACM Symposium on Computational Geometry, Vol. 12, Philadelphia, PA, 1996, pp. 166–175.
- [ACG89] M.J. Atallah, R. Cole, M.T. Goodrich, Cascading divide-and-conquer: a technique for designing parallel algorithms, SIAM J. Comput. 18 (1989) 499–532.
- [BH93] H. Bast, T. Hagerup, Fast parallel space allocation, estimation and integer sorting, Technical Report, MPI-I-93-123, 1993.
- [Bre74] R. Brent, Parallel evaluation of general arithmetic expressions, J. ACM 21 (1974) 201–206.
- [Cha95] T.M. Chan, Output-sensitive results on convex hulls, extreme points and related problems, ACM Symposium on Computational Geometry, 1995.
- [CM92] B. Chazelle, J. Matoušek, Derandomizing an output-sensitive convex-hull algorithm in three dimensions, Tech. Report, Princeton University, 1992.
- [Cho80] A. Chow, Parallel algorithms for geometric problems, Ph.D. Thesis, University of Illinois, Urbana-Champaign, 1980.
- [CS89] K.L. Clarkson, P.W. Shor, Applications of random sampling in computational geometry ii, Discrete Comput. Geom. 4 (1989) 387–421.
- [DK89] N. Dadoun, D.G. Kirkpatrick, Parallel construction of subdivision hierarchies, J. Comput. System Sci. 39 (1989) 153–165.

- [GG91] M. Ghouse, M.T. Goodrich, In-place techniques for parallel convex-hull algorithm, Proceedings of the Third ACM Symposium on Parallel Algorithms Architecture, Hilton Head, SC, 1991, pp. 192–203.
- [Goo93] M.T. Goodrich, Geometric partitioning made easier, even in parallel, Proceedings of the Ninth ACM Symposium on Computational Geometry, San Diego, CA, 1993, pp. 73–82.
- [GS96] N. Gupta, S. Sen, Faster output-sensitive parallel convex hulls for $d \leq 3$: optimal sublogarithmic algorithms for small outputs, ACM Symposium on Computational Geometry, 1996, pp. 176–185.
- [GS97] N. Gupta, S. Sen, Optimal, output-sensitive algorithms for constructing planar hulls in parallel, Comput. Geom.: Theory Appl. 8 (1997) 151–166.
- [HR92] T. Hagerup, R. Raman, Waste makes haste: tight bounds for loose parallel sorting, FOCS 33 (1992) 628–637.
- [HW87] D. Haussler, E. Welzl, ϵ -nets and simplex range queries, Discrete Comput. Geom. 2 (2) (1987) 127–152.
- [Ket94] M. Ketan, Computational Geometry: An Introduction through Randomized Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1994.
- [KS86] D.G. Kirkpatrick, R. Seidel, The ultimate planar convex hull algorithm, SIAM J. Comput. 15 (1) (1986) 287–299.
- [Mat92] J. Matoušek, Linear optimization queries, ACM Symposium on Computational Geometry, Vol. 8, Berlin, Germany, 1992, pp. 16–25.
- [PH77] F.P. Preparata, S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, Comm. ACM 20 (1977) 87–93.
- [RS93] S. Rajasekaran, S. Sen, in: J.H. Reif (Ed.), Random Sampling Techniques and Parallel Algorithm Design, Morgan, Kaufman, Los Altos, CA, 1993.
- [RS92] J.H. Reif, S. Sen, Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems, Siam J. Comput. 21 (3) (1992) 466–485.
- [Sei86] R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, Proceedings of the ACM Annual Symposium on Theory of Computing, Vol. 18, Berkeley, CA, 1986, pp. 404–413.
- [Sen97] S. Sen, Lower bounds for algebraic decision trees, parallel complexity of convex hulls and related problems, Theoret. Comput. Sci. 188 (1997) 59–78.