A Randomized Parallel 3D Convex Hull Algorithm For Coarse Grained Multicomputers*

Frank Dehne† Xiaotie Deng‡ Patrick Dymond‡ Andreas Fabri§
Ashfaq A. Khokhar¶

Abstract

We present a randomized parallel algorithm for constructing the 3D convex hull on a generic p-processor coarse grained multicomputer with arbitrary interconnection network and n/p local memory per processor, where $\frac{n}{p} \geq p^{2+\epsilon}$ (for some arbitrarily small $\epsilon > 0$). For any given set of n points in 3-space, the algorithm computes the 3D convex hull, with high probability, in $O(\frac{n \log n}{p})$ local computation time and $O(1)$ communication phases with at most $O(\frac{n}{p})$ data sent/received by each processor. That is, with high probability, the algorithm computes the 3D convex hull of an arbitrary point set in time $O(\frac{n \log n}{p} + \Gamma_{n,p})$, where $\Gamma_{n,p}$ denotes the time complexity of one communication phase.

In the terminology of the BSP model, our algorithm requires, with high probability, $O(1)$ supersteps and a synchronization period $\Theta(\frac{n \log n}{p})$. In the LogP model, the execution time of our algorithm is asymptotically optimal for several architectures.

*This work was partially supported by the Natural Sciences and Engineering Research Council of Canada and the ESPRIT Basic Research Actions Nr. 7141 (ALCOM II).

†School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. Email: dehne@scs.carleton.ca

‡Dept. of Computer Science, York University, North York, Canada M3J 1P3. Email: {deng,dymond}@cs.yorku.ca

§Dept. of Computer Science, Utrecht University, 3508 TB Utrecht, The Netherlands. Email: andreas@cs.ruu.nl

¶School of EE and Dept. of Computer Sci., Purdue University, West Lafayette, IN 47907, USA. Email: ashfaq@cs.purdue.edu

SPAA'95 Santa Barbara CA USA© 1995 ACM 0-89791-717-0/95/07.$3.50

1 Introduction

The Model

Since the memory access cost in a parallel machine is different from that of a uniprocessor, speedup results for theoretical PRAM algorithms do not necessarily match the speedups observed on real machines [8] [38]. Valiant attempts to solve this problem by introducing slackness in the number of processors, mapping memory through hash functions, and routing messages in a pipelined fashion [41]. Given sufficient slackness, this allows PRAM algorithms to be executed optimally on coarse grained parallel systems. Valiant points out, however, that one may want to design algorithms that utilize local computations and minimize global operations [40] [41]. Gerbessiotis and Valiant [26] describe circumstances where PRAM simulations can not be performed efficiently, among others if the factor g is high. The algorithm presented in this paper considers this case for the 3D convex hull problem.

Several other different models point into the same direction: Papadimitriou and Yannakakis formulate a communication delay model and discuss the issue of using redundant local computation to avoid large communication cost [36]. Aggarwal et.al. combine communication latency with the PRAM model [2]. Gibbons suggests a phase PRAM model [23]. The LogP model introduced by Culler, et.al. and the C^3 model by Hambrusch and Khokhar [28] use the BSP model as a starting point, focus on the technological trend from fine grained parallel computers toward coarse-grained parallel computers, and advocate portable parallel algorithm designs for coarse-grained multiprocessor systems [14]. Applying Valiant's method for analysing router efficiency for the implementation time of global operations, it can be shown that a parallel algorithm for coarse grained parallel computers can be asymptotically optimally implemented across different parallel architectures when the number of global op-

erations is sufficiently small [40] [41] [20]. These observation have created some interest in designing coarse grained parallel algorithms with a small number of communication phases [10], [17], [16], [18], [19], [30]. The algorithm presented in this paper considers this case for the 3D convex hull problem and presents an optimal solution on coarse-grained machines.

The 3D Convex Hull Problem

For the 3-dimensional convex hull problem studied in this paper, Amato and Preparata [4] give an almost work optimal deterministic NC^1 algorithm for the CREW PRAM. Chow [13] presented algorithms for the EREW PRAM and for the cube connected cycles interconnection network with distributed memory. The algorithms need time $O(\log^3 n)$ and $O(\log^X n)$, respectively. Reif and Sen [37] were the first to give a time and work optimal randomized algorithm for the CREW PRAM. Goodrich *et. al.* [25] adapted this algorithm for an external memory model with an array of disks and parallel processors. For higher-dimensional convex hulls Amato *et. al.* [3] present $O(\log n)$ time algorithms that use $O(n \log n + n^{\lfloor d/2 \rfloor})$ work. Dehne *et. al.* [16] proposed an algorithm for the coarse grained model. The time complexity is $O(\frac{n \log n}{p} + \Gamma_{n,p})$, with high probability, assuming a uniform point distribution. Under this assumption, the algorithm performs only a constant number of communication phase. $\Gamma_{n,p}$ denotes the time complexity of one communication phase with at most $O(\frac{n}{p})$ data sent/received by each processor.

The Results

In this paper we improve considerably on our previous results in [16] and [19]. We present a randomized parallel algorithm for a p-processor coarse grained multicomputers with local memories of size $\frac{n}{p} \geq p^{2+\epsilon}$ ($\epsilon > 0$ arbitrarily small) and arbitrary interconnection network which computes the convex hull of n arbitrary points in 3-space in time $\tilde{O}(\frac{n \log_2 n}{p} + \Gamma_{n,p})$ [1].

Every processor spends a total local computation time of $\tilde{O}(\frac{n \log n}{p})$. The algorithm uses only $\tilde{O}(1)$ global communication phases with at most $\tilde{O}(\frac{n}{p})$ data sent/received by each processor.

With respect to [16], the algorithm presented here allows for an arbitrary input distribution. In particular, it allows for inputs created by mapping a 2D Voronoi diagram problem to a 3D convex hull problem. The techniques used in this paper are very different from the ones presented in [16] and [19]. The randomization methods presented are very different from the ones previously reported, e.g., in [37] and related papers.

Using Valiant's terminology for the BSP model [40], our algorithm requires, with high probability, only $O(1)$ supersteps and a synchronization period $L = \Theta(\frac{n \log n}{p})$. This cannot (at least not apparently) be achieved by techniques described in [37, 25] or divide-and-conquer methods. In the LogP model, the execution time of our algorithm is $O(L + (o+g)\frac{n}{p} + \frac{n \log n}{p})$ [14], which is asymptotically optimal for several different architectures [40] [41] [20].

The algorithm presented in this paper is currently being implemented in MPI, and we expect to include timing results for different architectures in the journal version of this paper.

Finally, we note that our algorithm can also be transformed into an optimal $O(\log n)$ time PRAM algorithm. We include a sketch at the end of this paper.

2 Outline of the Algorithm

Input: *Each processor stores n/p points of S.*
Output: *Each processor p_i stores a set E_i of $O(\frac{n}{p})$ edges of* $\mathrm{CH}(S)$.

(1) Computing a sample convex hull

All processors compute globally a random sample $R \subset S$ of size $O(\frac{n}{p})$. R is broadcast to all processors.

Each processor computes the convex hull $\mathrm{CH}(R)$. *We can assume w.l.o.g. that the origin lies in the interior of* $\mathrm{CH}(R)$ *and that each face is a triangle. The points inside* $\mathrm{CH}(R)$ *are removed.*

(2) Computing the generating sets

Compute p sets $S_i \subset S$ of size $O(\frac{n}{p})$ each, referred to as the *generating sets*, such that for each edge e of the convex hull $\mathrm{CH}(S)$ there exists a set S_i which contains both endpoints of e:

With each face t of $\mathrm{CH}(R)$ *associate a cone and a closed halfspace. The cone C_t is defined by the origin as apex and rays starting at the origin and passing through the three vertices of t. The closed halfspace H_t is defined by the plane through t and does not contain the origin. Let K_t and T_t denote the subsets of points of S that are not inside* $\mathrm{CH}(R)$ *but contained in C_t and H_t, respectively. (Note that K_t and T_t contain the vertices of t).*

Each processor p_i computes data sets (R_i, S_i) of size $O(\frac{n}{p})$. The sets R_i form a partition of

[1] $X = \tilde{O}(f(n))$, if and only if $(\forall c > c_0 > 1)$ $\mathrm{prob}\{X \geq cf(n)\} \leq \frac{1}{n^{g(c)}}$ where c_0 is a fixed constant and $g(c)$ is a polynomial in c with $g(c) \to \infty$ for $c \to \infty$ [33]

the set of faces of $\mathrm{CH}(R)$. *Details on how to obtain the partition will be discussed in Section 3.2. The sets* S_i *are subsets of* S, *where* $s \in S_i$ *if and only if either* $s \in K_t$ *for some face* $t \in R_i$ *or* $s \in T_t$ *for some face* t *of* $\mathrm{CH}(R)$ *not in* R_i *that is edge-adjacent to some face of* R_i.

Each processor p_i *computes the convex hull* $\mathrm{CH}(S_i)$. *The points inside* $\mathrm{CH}(S_i)$ *are removed.*

(3) Disconnecting internal edges

Remove all edges connecting a point $x \in CH(S)$ with a point $y \in S - CH(S)$:

Each processor p_i *creates two copies of each edge* (v, w) *of* $\mathrm{CH}(S_i)$ *one with key* v *and one with key* w. *All these edges are sorted globally with respect to their endpoints.*

For each endpoint v *and incident edge* (v, w) *consider the induced ray starting at* v *and passing through* w. *The convex hull* $\mathrm{CH}(\mathcal{R}_v)$ *of the set* $\mathcal{R}_v$ *of rays induced by the incident edges of* v *is computed. The incident edges inside* $\mathrm{CH}(\mathcal{R}_v)$ *are removed, and* v *is removed if* v *is inside* $\mathrm{CH}(\mathcal{R}_v)$.

Each remaining edge e *is sent back to processor* p_i, *if* e *was originally part of* $\mathrm{CH}(S_i)$.

(4) Selecting the global convex hull edges in each generating set

Select from the edges remaining after Step 3 those edges that are part of $CH(S)$:

(a) *each processor* p_i *computes for each triangle* t *of* R_i *the point of* S_i *with maximum distance to the plane defined by* t. *We refer to such a point as a* furthest point *in* S_i.

(b) *On each processor* p_i *let* G_i *be the subgraph of* $\mathrm{CH}(S_i)$ *induced by those edges that remain after Step 3. Each processor* p_i *computes the set* E_i *of edges of* G_i *reachable in* G_i *from a furthest point in* S_i.

3 Algorithm Details and Analysis

We will now discuss in more detail the steps outlined in the above algorithm and give a probablistic analysis. Let $k \in I\!N$ be a parameter to be determined later.

3.1 Computing a sample convex hull

We need the following

Lemma 1 [33] *Consider a random variable* X *with binomial distribution. Let* n *be the number of trials, each of which is successful with probability* q. *The expectation of* X *is* $E(X) = nq$.
(a) $\mathrm{prob}\{X > cnq\} \leq e^{-\frac{1}{2}(c-1)^2 nq}$, *for* $c > 1$
(b) Let $\beta \geq 4$ *(not necessarily fixed). If* $\beta^2 nq \geq \alpha \ln(n)$ *for some constant* $\alpha > 0$, *then* $X = \tilde{O}(\beta nq)$.

We now outline how to select a random sample $R \subset S$ of size $\tilde{O}(\frac{n}{p})$. Each processor p_i performs a biased random coin flip for each point of S stored at p_i, such that each point is selected with probability $\frac{pk}{n}$. From the above lemma follows that $|R| = \tilde{O}(pk)$ and that if $k = \Omega(\ln(n))$ then the number of points selected by a processor p_i is $\tilde{O}(k)$.

Since we store R at each processor, it is necessary that $|R| \leq \tilde{O}(\frac{n}{p})$. Hence, we obtain

Requirement 1 $\quad \Omega(\ln(n)) \leq k \leq \frac{n}{p^2}$

In order to compute the convex hull $\mathrm{CH}(R)$ of R we apply any optimal sequential ($O(\frac{n}{p}\log_2 \frac{n}{p})$ time) 3D convex hull algorithm [35, 15].

3.2 Computing the generating sets

In Step 2 of the algorithm we compute p subsets S_i of S, such that each edge of the convex hull $\mathrm{CH}(S)$ is generated in the computation of a convex hull $\mathrm{CH}(S_i)$. Before we give an algorithm for computing the generating sets S_i, we show that in Step 2 (at least) all edges of the convex hull $\mathrm{CH}(S)$ are generated.

Lemma 2 *Every edge of* $\mathrm{CH}(S)$ *is an edge of at least one* $\mathrm{CH}(S_i)$, $1 \leq i \leq p$.

Proof. Let (v, w) be an edge in $\mathrm{CH}(S)$; see Figure 1. Consider the line $l(v, w)$ parallel to (v, w) and contained in the plane defined by v, w and the origin such that $l(v, w)$ is tangent to the sample convex hull $\mathrm{CH}(R)$. Recall that the origin lies inside $\mathrm{CH}(R)$. The line $l(v, w)$ intersects an edge e of $\mathrm{CH}(R)$ (if it intersects at a vertex we consider any edge incident to this vertex). Let t' and $t"$ be the faces of $\mathrm{CH}(R)$ adjacent to e. By construction there exists at least one R_i such that $\{t', t"\} \cap R_i \neq \emptyset$. We recall that $s \in S$ is in S_i if and only if either $s \in K_t$ for some face $t \in R_i$ or $s \in T_t$ for some face t of $\mathrm{CH}(R)$ not in R_i that is edge-adjacent to some face of R_i. Thus, v and w are both in S_i and (v, w) is an edge of $\mathrm{CH}(S_i)$. □

We next discuss how we can construct p generating sets S_i, each containing no more than $O(\frac{n}{p})$ points of S.

For each face t of $\mathrm{CH}(R)$ we can easily compute $|K_t|$, the number of points in a cone C_t, by constructing a subdivision hierarchy [21] for the convex polyhedron $\mathrm{CH}(R)$. On this subdivision hierarchy we perform a ray shooting query for each point $s \in S$, using a ray that starts at the origin and

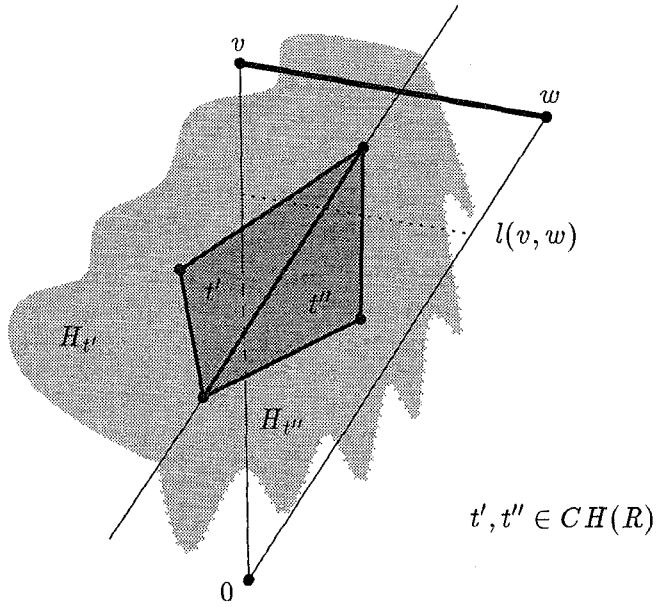

Figure 1: Illustration to the proof of Lemma 2.

passes through s. Each query can be answered in time $O(\log |\mathrm{CH}(R)|)$. The search structure is of size $O(|\mathrm{CH}(R)|)$ and can be computed sequentially in time $O(|\mathrm{CH}(R)|)$. Each point is contained in exactly one cone. Each processor performs the ray shooting queries for its point set, and the p results for each of the $\frac{n}{p}$ faces are added after a global sort of the faces.

Lemma 3 *(a) For any fixed $c \geq 5$, with probability at least $1 - \frac{1}{n^{c-4}}$ there exists no triangle t such that $|T_t| \geq c\frac{n\ln(n)}{pk}$.*
(b) For any fixed $c \geq 5$, with probability at least $1 - \frac{1}{n^{c-4}}$ there exists no triangle t such that $|K_t| \geq c\frac{n\ln(n)}{pk}$.

Proof. *(a)* Consider a fixed $c \geq 5$. For some fixed triangle t, $\mathrm{prob}\{|T_i| = j\} \leq (1 - \frac{pk}{n})^j$ and $\mathrm{prob}\{|T_t| \geq c\frac{n\ln(n)}{pk}\} \leq \sum_{j=\lceil c\frac{n\ln(n)}{pk}\rceil}^{n} \mathrm{prob}\{|T_t| = j\} \leq \left(n - c\frac{n\ln(n)}{pk}\right)\left(1 - \frac{pk}{n}\right)^{c\frac{n\ln(n)}{pk}} = \left(n - c\frac{n\ln(n)}{pk}\right)\left(\left(1 - \frac{1}{\frac{n}{pk}}\right)^{\frac{n}{pk}}\right)^{c\ln(n)} \leq \left(n - c\frac{n\ln(n)}{pk}\right)e^{-c\ln(n)}$ As there are less than n^3 possible triangles, $\mathrm{prob}\{$ there exists a t with $|T_t| \geq c\frac{n\ln(n)}{pk}\} \leq n^3\left(n - c\frac{n\ln(n)}{pk}\right)e^{-c\ln(n)}$. The claim follows from $n^3(n - c\frac{n\ln(n)}{pk})e^{-c\ln(n)} \leq \frac{1}{n^{c-4}} \Leftrightarrow n^{c-4}\, n^3\left(n - c\frac{n\ln(n)}{pk}\right) \leq e^{c\ln(n)} \Leftrightarrow (c-4)\ln(n) + 3\ln(n) + \ln\left(n - c\frac{n\ln(n)}{pk}\right) \leq c\ln(n) \Leftrightarrow c - 4 \leq \frac{1}{\ln(n)}(c\ln(n) - 4\ln(n)) = c - 4$

(b) Follows immediately, since $K_t \subset T_t$ for each face t of $\mathrm{CH}(R)$.

□

It is easy to partition the faces of $\mathrm{CH}(R)$ into p subsets R_i such that for each R_i, $\sum_{t\in R_i} |K_t| = O(\frac{n}{p})$. However, a set R_i from such a partitioning can have $O(pk)$ edge adjacent faces not in R_i and can induce a set S_i containing $O((\frac{n}{p})\log_2 n)$ points. Our construction of the p sets R_i is based on the following separator theorem:

Lemma 4 *[31] Let G be any m-vertex planar graph. Then the vertices of G can be partitioned in three sets V_1, U and V_2, such that no edge joins a vertex in V_1 with a vertex in V_2, neither V_1 nor V_2 have total size exceeding $\frac{m}{2}$, and separator U contains no more than $2\sqrt{2}\sqrt{m}/(1 - \sqrt{2/3})$ vertices. Furthermore V_1, U and V_2 can be determined sequentially in time $O(m)$.*

Consider the m-vertex dual graph G of $\mathrm{CH}(R)$and define the cost of a node of G corresponding to a face t of $\mathrm{CH}(R)$ as $|K_t|$. Note that $m = |R| = \tilde{O}(pk)$. In order to partition the vertices of G into p subsets $W_1, \ldots, W_p$, we apply the above separator theorem. We do not use a weighted separator theorem. Instead, we recursively apply Lemma 4 until the total node cost of every subgraph is at most $O(\frac{n}{p})$. Note that, the cost of each node is $O(\frac{n}{p})$. We obtain less than $2p$ subproblems and we pair them arbitrarily yielding the sets $W_1, \ldots, W_p$. Let R_i, $1 \leq i \leq p$, be the set of faces of $\mathrm{CH}(R)$corresponding to the vertices in W_i. Since $|K_t| \leq \frac{cn\ln(n)}{pk}$ for some constant c, the above recursion scheme continues for at most $l = \log_2 p + \log_2 \ln(n) + \log_2 c$ levels. The number of nodes in separators at the i^{th} level is bounded by $2^{i-1}d\sqrt{\frac{m}{2^{i-1}}}$, where $d = 2\sqrt{2}/(1 - \sqrt{2/3})$. The total number of nodes in all separators obtained in this procedure is bounded by $\sum_{i=1}^{l} 2^{i-1}d\sqrt{\frac{m}{2^{i-1}}} = \sqrt{m}d\frac{(\sqrt{2})^l - 1}{\sqrt{2}-1} = O(p\sqrt{k\ln(n)})$. Let B be the multiset of points contained in sets T_t for faces t corresponding to some separator, counting the multiple presence of points.

The total number $|B|$ of such points is bounded by $|B| = O(\frac{n\ln(n)\sqrt{\ln(n)}}{\sqrt{k}})$.

Requirement 2 $k \geq (l\ln(n)\sqrt{\ln(n)})^2$

Choosing $k \geq (l\ln(n)\sqrt{\ln(n)})^2$, we obtain $|B| = O(\frac{n}{l})$. In time $O(\log(|R|))$ we determine for each point whether it belongs to B or not. Then, we determine those points of B which correspond to the first level separator. These points are assigned to S_i, $1 \leq 1 \leq p$. The remaining points of B are

partitioned into two parts: Those in sets T_t where t corresponds to a vertex in V_1, and those in sets T_t where t corresponds to a vertex in V_2. This partition can be obtained in $O(\log(|R|))$ time for each point in B. Therefore, for each point in B, the number of operations on each level is $O(\log n)$ and the total number of operations on l levels is $O(l|B|\log n)$, which is $O(n\log n)$. Since each point of B requires the same number of operations, they can be distributed over the p processors such that each processor needs $O(\frac{n\log n}{p})$ operations.

As the level of recursion increases from 1 to l, the number of adjacent faces decreases geometrically. The number of faces which are edge-adjacent to a face of $\mathrm{CH}(R)$ in R_i is bounded by $\sum_{j=1}^{l}\sqrt{(1/2)^j}\ \sqrt{|\mathrm{CH}(R)|} = \sqrt{|\mathrm{CH}(R)|}\sum_{j=0}^{l}\sqrt{(1/2)^j}$ $= O(\sqrt{|\mathrm{CH}(R)|}) = \tilde{O}(\sqrt{pk})$. Hence, it follows from Lemma 3 that

$$|S_i| = \tilde{O}\left(\frac{n}{p} + \sqrt{pk}\,\frac{n\ln(n)}{pk}\right).$$

Since it is necessary that S_i is of size $\tilde{O}(\frac{n}{p})$ we obtain

Requirement 3 $\quad k \geq p\ln^2(n)$.

Requirements 1 - 3 hold if $\frac{n}{p} \geq p^{2+\epsilon}$ for any fixed $\epsilon > 0$.

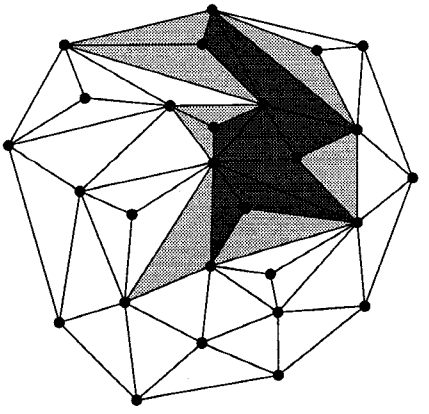

Figure 2: A set R_i of faces, and the faces which are edge-adjacent to faces of R_i.

3.3 Disconnecting internal edges

In Step 3 of the algorithm we start removing from generating sets S_i those edges that are not edges of $\mathrm{CH}(S)$.

Amato and Preparata [4] observed that, given a set of rays $\mathcal{R}_v$ in $\mathbb{R}^3$ which all start at vertex v, the problem of computing their convex hull $\mathrm{CH}(\mathcal{R}_v)$ can be reduced to a two dimensional convex hull computation. The algorithm first determines a plane H_v that does not pass through v and intersects all rays of $\mathcal{R}_v$. If there is no such plane H_v, then v is inside the convex hull $\mathrm{CH}(\mathcal{R}_v) = \mathbb{R}^3$. Hence, v is inside $\mathrm{CH}(S)$. If such a plane H_v exists, we can determine the convex hull $\mathrm{CH}(\mathcal{R}_v)$ by computing the 2D convex hull of the intersection points of the rays with H_v.

We perform a sequential algorithm if $|\mathcal{R}_v| \leq O(\frac{n}{p})$. Otherwise, we reduce the problem to solving, in parallel, the following linear programming problem:

Let the plane H_v be defined by equation $ax + by + cz = d$. It passes through v and all vertices w_i of edges (v, w_i) lie on one side of H_v. Let (x_i, y_i, z_i) be the coordinates of point w_i. This defines the constraints $ax_i + by_i + cz_i \geq d$ for the vertices w_i and the function $f(a,b,c) = ax_v + by_v + cz_v - d$, with (x_v, y_v, z_v) as coordinates of vertex v. The parallel time complexity is $\tilde{O}(\frac{n\log_2 n}{p} + \Gamma_{n,p})$ with $O(1)$ communication rounds [19].

Lemma 5 *After execution of Step 3 the following holds*
(a) No edge of $\mathrm{CH}(S)$ *is removed.*
(b) The edges that are contained in $\mathrm{CH}(S)$ *form a single connected component which does not contain any vertex or edge that is not in* $\mathrm{CH}(S)$.

Proof. *(a)* Step 2 removes only edges and vertices which lie inside the convex hull $\mathrm{CH}(\mathcal{R}_v)$ of rays incident to a vertex v. Clearly an edge inside $\mathrm{CH}(\mathcal{R}_v)$ also lies inside $\mathrm{CH}(S)$.

(b) Assume that a vertex not in $\mathrm{CH}(S)$ is connected to a vertex v in $\mathrm{CH}(S)$ by some path of edges remaining after Step 3. Consider the edge of this path incident to v. This edge is deleted in Step 3 as it lies inside $\mathrm{CH}(\mathcal{R}_v)$; a contradiction. □

3.4 Selecting the global convex hull edges in each generating set

Step 4(a) computes for each triangle $t \in R_i$ the point of S_i that has maximal distance to the plane through t. Such a furthest point is clearly in $\mathrm{CH}(S)$.

To determine the furthest points for R_i we use a subdivision hierarchy [21] for the convex polyhedron $\mathrm{CH}(S_i)$. We perform for each plane through a triangle in R_i a search to determine its tangents to $\mathrm{CH}(S_i)$. The size of the search structure is $O(|\mathrm{CH}(S_i)|)$ and it can be constructed sequentially in time $O(|\mathrm{CH}(S_i)|)$. The sequential search needs logarithmic time for each triangle. Hence, the time complexity of Step 4(a) is $O(|R_i|\log_2|S_i|)$.

In Step 4(b) each processor p_i computes the union of those connected components of G_i that

contain a furthest point in S_i. The sequential time complexity is $O(|G_i|) = O(|S_i|)$.

Lemma 6 *(a) Every edge of* $\mathrm{CH}(S)$ *is contained in at least one set* E_i, $1 \leq i \leq p$.
(b) No set E_i, $1 \leq i \leq p$, *contains an edge that is not in* $\mathrm{CH}(S)$.

Proof. *(a)* Consider an edge (v, w) of $\mathrm{CH}(S)$. By Lemma 2, there exists a set S_i such that (v, w) in $\mathrm{CH}(S_i)$. From Lemma 5 it follows that (v, w) is not deleted in Step 3. Due to the definition of R_i, v is contained in H_t for some $t \in R_i$, and $\mathrm{CH}(S) \cap H_t \subset S_i$. By Lemma 5 and the fact that H_t is a halfspace it follows that the edges incident to $\mathrm{CH}(S) \cap H_t$ form a connected component. Furthermore, a point in S_i with maximum distance to the plane defined by t is in $\mathrm{CH}(S) \cap H_t$. Hence, $(v, w) \in E_i$.

(b) Follows immediately from Lemma 5. □

4 Summary

Lemmas 2, 5 and 6 imply the following

Theorem 1 *The convex hull of n points in 3-space can be computed on a p-processor coarse grained multicomputer with* $\frac{n}{p} \geq p^{2+\epsilon}$, $\epsilon > 0$, *in time* $\tilde{O}(\frac{n \log_2 n}{p} + \Gamma_{n,p})$.

By standard duality transform [12] we obtain:

Corollary 1 *The Voronoi diagram of a set of n points in the Euclidean plane can be computed on a p-processor coarse grained multicomputer with* $\frac{n}{p} \geq p^{2+\epsilon}$, $\epsilon > 0$, *in time* $\tilde{O}(\frac{n \log_2 n}{p} + \Gamma_{n,p})$.

Note that in order to compute the Voronoi diagram the algorithm can be simplified. We do not need Step 4 of the algorithm, as the Voronoi diagram problem is equivalent to a convex hull problem where all points are vertices of the convex hull. All internal edges which are produced in Step 2 of the algorithm are removed in Step 3.

5 A Final Remark

We note that our algorithm can also be transformed into an optimal $O(\log n)$ time PRAM algorithm which is considerably simpler than those previously reported. The following is a sketch of how the 4 steps of our algorithm could also be implemented on a PRAM with n processors.

1. For a sample size sufficiently small, e.g., $pk = n^{\frac{1}{100}}$, Step 1 can be performed using any $O(\log n)$ time algorithm, e.g. [4].
2. Step 2 can be implemented by choosing the parameter p sufficiently small, e.g. $p = n^c$ and $k = p^c$ for some constant $c > 1$ (e.g. $c = 6$). Note that p is now only a parameter for the sample convex hull, not the number of processors. We apply the planar separator algorithm of [24]
3. In Step 3, we solve n 2D convex hull problems using the method by Amato and Preparata. Note that, the total size of the n 2D convex hull problems is $O(n)$ and, thus, the total time is $O(\log n)$ using n processors.
4. For Step 4, we apply any optimal NC^1 connected component algorithm [27][39].

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry, *Algorithmica*, Vol. 3, pages 293–327, 1988.

[2] A. Aggarwal, A.K. Chandra, and M. Snir. On Communication Latency in PRAM Computations. *SPAA89*, pages 11–21.

[3] N.M. Amato, M. Goodrich, E. Ramos. Parallel Algorithms for high-dimensional convex hulls. *Proc. 35th Annual IEEE Symposium on Foundations in Computer Science*, pp. 683–694, 1994.

[4] N.M. Amato, F.P. Preparata. An NC^1 Parallel 3D Convex Hull Algorithm. *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 289–297, 1993.

[5] M.J. Atallah and M.T. Goodrich. Efficient parallel solutions to some geometric problems *Journal Parallel and Distributed Computing*, Vol. 3, pages 492–507, 1986.

[6] M.J. Atallah and M.T. Goodrich. Parallel algorithms for some functions of two convex polygons. *Algorithmica*, Vol. 3, pages 535–548, 1988.

[7] N. Alon, and N. Meggido. Parallel Linear Programming in Fixed Dimension Almost Surely in Constant Time. *FOCS90*, pages 574–582.

[8] R.J. Anderson, and L. Snyder. A Comparison of Shared and Nonshared Memory Models of Computation. *Proc. of IEEE*, vol 79(4),pages 480–487.

[9] M.J. Atallah and J.-J. Tsay. On the parallel-decomposability of geometric problems. *Proc. 5th Annual ACM Symposium on Computational Geometry*, pages 104–113, 1989.

[10] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith, and M. Zagha. A Comparison of Sorting Algorithms for the Connection Machine CM-2. *SPAA91*, pages 3–16.

[11] K.E. Batcher. Sorting networks and their applications. *Proc. AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[12] K.Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters,* Vol. 9, 1979, pages 223–228.

[13] A. Chow. Parallel Algorithms for Geometric Problems. PhD thesis, Univ. of Illinois atUrbana-Champaign, 1980.

[14] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *Presented in the 4th ACM SIGPLAN Sym. on Principles and Practice of Parallel Programming*, pages 235–261, 1993.

[15] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry Theory and Application*, Vol. 3, Nr. 4, pages 185–212, 1993.

[16] F. Dehne, A. Fabri, and C. Kenyon. Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time. *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, pages 586–593, 1994.

[17] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers, *Proc. ACM 9th Annual Computational Geometry*, pages 298–307, 1993

[18] X. Deng and N. Gu. Good Programming Style on Multiprocessors. *Proceedings of IEEE Symposium on Parallel and Distributed Processing*, Dallas, October, 1994, pp.538-543,

[19] X. Deng. A Convex Hull Algorithm for Coarse Grained Multiprocessors. *Proc. 5th International Symposium on Algorithms and Computation*, 1994.

[20] X. Deng and P. Dymond, Efficient Routing and Message Bounds for Optimal Parallel Algorithms. *to appear in IPPS 1995*, Santa Barbara, April, 1995.

[21] D.P. Dobkin and D.G Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, Vol. 27, pages 241–253, 1983.

[22] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS Monographs on Theoretical Computer Science, Vol. 10, Springer, 1987.

[23] P. Gibbons. A More Practical PRAM Model. *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 158-168, 1989.

[24] M.T. Goodrich, Planar Separators and Parallel Polygon Triangulation *To appear in J. CSSS.*

[25] M.T. Goodrich, J.J. Tsay, D.E. Vengroff, and J.S. Vitter. External-memory computational geometry. *Proc. 34th IEEE Symposium on Foundations of Computer Science*, pages 714–723, 1993.

[26] A.V. Gerbessiotis and L.G. Valiant. Direct Bulk-Synchronous Parallel Algorithms. *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, Vol. 621, pages 1–18, Springer Verlag, 1992.

[27] S. Halperin, and U. Zwick. *An optimal randomized logrithmic time connectivity algorithm for the EREW PRAM.* SPAA94, pages 1–10.

[28] S.E. Hambrusch and A.A. Khokhar. C^3: An Architecture-independent Model for Coarse-Grained Parallel Machines. *Proceedings of the 6-th IEEE Symposium on Parallel and Distributed Processing*, 1994.

[29] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[30] Hui Li, and K. C. Sevcik. Parallel Sorting by Overpartitioning. *SPAA94*, pages 46–56, 1994.

[31] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, Vol. 9, No. 3, pages 615–627, 1980.

[32] K. Mehlhorn. Graph Algorithms and NP-Completeness. Springer Verlag New York, NY, 1984.

[33] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, NY, 1993.

[34] R. Miller and Q.F. Stout. Efficient parallel convex hull algorithms. *IEEE Trans. Comput.*, C-37, Vol. 12, pages 1605–1618, 1988.

[35] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[36] C.H. Papadimitriou and M. Yannakakis, Towards An Architecture Independent Analysis of Parallel Algorithms. *STOC 20* (1988), pages 510-513.

[37] J.H. Reif and S. Sen. Optimal parallel algorithms for 3D convex hulls and related problems. *SIAM Journal of Computing*, Vol. 21, pages 446–485, 1992 , pages 394–404, 1989, with corrigendum in *SIAM Journal of Computing*, Vol. 23, pages 447–448, 1994.

[38] L. Snyder. Type architectures, shared memory and the corollary of modest potential. *Annu. Rev. Comput. Sci. 1*, pages 289–317, 1986.

[39] Y. Shiloach and U. Vishkin. An $o(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3(1), pages 57–67, 1983.

[40] L.G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, Vol. 33, pages 103–111, 1990.

[41] L.G. Valiant, General Purpose Parallel Architectures. *Handbook of Theoretical Computer Science.* Edited by J. van Leeuwen, the MIT Press/Elsevier, 1990, pages 943-972.