INVITED SURVEY PAPER *Special Issue on Algorithm Engineering: Surveys*

Parallel Algorithms for Convex Hull Problems and Their Paradigm

Wei CHEN[†], **Koji NAKANO**[†], *and* **Koichi WADA**[†], *Members*

SUMMARY A convex hull is one of the most fundamental and interesting geometric constructs in computational geometry. Considerable research effort has focused on developing algorithms, both in serial and in parallel, for computing convex hulls. In particular, there are few problems whose parallel algorithms are so thoroughly studied as convex hull problems. In this paper, we review the convex hull parallel algorithms and their paradigm. We provide a summary of results and introduce several interesting topics including typical techniques, output-size sensitive methods, randomized approaches, and robust algorithms for convex hull problems, with which we may see the highlights of the whole research for parallel algorithms. Most of our discussion uses the PRAM (Parallel Random Access Machine) computational model, but still we give a glance at the results of the other parallel computational models such as mesh, mesh-of-trees, hypercube, recofigurable array, and models of coarse grained multicomputers like BSP and LogP.

key words: *convex hulls, parallel algorithms, randomized algorithms, output-size sensitive algorithms, robust computational geometry*

1. Introduction

Given a set G of geometrical objects in $\mathcal{R}^d$, $d \geq 2$, the convex hull of G, denoted as $CH(G)$, is the smallest convex region containing all the elements of G. Convex hulls are one of the most fundamental geometric constructs. Its position in computational geometry is very similar (but not exactly equal) to that of sorting in ordinary computation. In addition to considerable interest in their own right, convex hulls are often useful in solving a lot of problems which are apparently unrelated at a glance. Much research effort has focused on developing algorithms, both in serial and parallel, for computing convex hulls. Especially, convex hull problems are one of few problems whose parallel algorithms are thoroughly studied, therefore a survey of them may help us to know the outline of the whole research of parallel algorithms.

1.1 Sequential Algorithms

Let S be a set of n points in $\mathcal{R}^d$, $d \geq 2$, the size of the convex hull (the number of the vertices which constitutes the boundary of the convex hull) of n points is $\Theta(n^{\lfloor d/2 \rfloor})$ in the worst case, and its construction requires $\Omega(n \log n + n^{\lfloor d/2 \rfloor})$ time [31], [61]. Optimal deterministic sequential algorithms for computing $CH(S)$ have been long known for the cases $d = 2$, 3 [41], [60]. When $d = 2$ and the points of S are sorted, say, sorted by x coordinate, $CH(S)$ can be solved in $O(n)$ time by Graham scan [41]. In higher dimensions, $d \geq 4$, Seidel proposed two deterministic algorithms. His first algorithm [65] runs in $O(n \log n + n^{\lceil d/2 \rceil})$ time, which is optimal for even d, and later he gave an $O(n^{\lfloor d/2 \rfloor} \log n)$ solution [66]. For some time, the only solutions optimal in higher dimensions were the randomized incremental algorithms of Clarkson and Shor [25], and the subsequent randomized method of Seidel [67]. Chazelle [15] gave the first optimal deterministic algorithm in higher dimensions, which was simplified by Brönnimann et al. [13]. The optimality of the above algorithms is measured with respect to the worst-case complexity for the size of the resulting convex hull.

On the other hand, when the size of the output is considered, it may be possible to beat the worst-case lower bounds since the size of the convex hull may range from $O(1)$ to $O(n^{\lfloor d/2 \rfloor})$. Accounting for output size, the lower bound becomes $\Omega(h + n \log h)$, where h is the size of the convex hull of points in $\mathcal{R}^d$. The first output-sensitive algorithm, due to Kirkpatrick and Seidel [47], computed the convex hull in $\mathcal{R}^2$ in $O(n \log h)$ time. For 3-dimension, Edelsbrunner and Shi [32] gave an $O(n \log^2 h)$ time output-sensitive algorithm, and Clarkson and Shor [25] gave an optimal randomized output-size sensitive solution, which was optimally derandomized by Chazellel and Matoušek [16]. In higher dimensions, the only known deterministic output-size sensitive method, due to Seidel [66], runs in time $O(n^2 + h \log n)$, which can be slightly improved to $O(n^{2-(2/(\lfloor d/2 \rfloor + 1))+\epsilon} + h \log n)$, for any fixed $\epsilon > 0$, using a technique of Matoušek [51].

Much work has focused on developing robust algorithms. Given a simple polyhedron P of $\mathcal{R}^d$, the following concepts are used to describe the convexity of P and the degree of approximation of P to $CH(S)$: (1) P is a δ-hull of S ($\delta \geq 0$) if all the vertices of P are taken from S and no point of S lies farther than δ outside P, (2) P is a δ-superhull of S ($\delta \geq 0$) if P contains all the points of S and P has at most $O(n)$ vertices which lie no farther than δ outside $CH(S)$, (3) P is ϵ-weakly convex ($\epsilon \geq 0$) if there exists some way of

Manuscript received June 29, 1999.
Manuscript revised October 20, 1999.

[†]The authors are with the Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan.

perturbing each vertex of P no farther than ϵ so that P becomes convex, and (4) P is ϵ-strongly convex if P is convex and remains convex even after each vertex of P is perturbed as far as ϵ. Using imprecise computations, Fortune [35] gave an $O(n \log n)$ time algorithm for computing an $c\alpha$-weakly $c\alpha$-hull in $\mathcal{R}^2$ for the error unit α of primitive operations and constant $c > 0$, which turns to be the correct convex hull if properly perturbing each vertex at most $c\alpha$. Li and Milenkovic [50] presented the algorithm for computing an ϵ-strongly convex $(12\epsilon + 288\sqrt{2}\mu)$-hull in $\mathcal{R}^2$ with rounded arithmetic in $O(n \log n)$ time, where μ is the rounded unit. Guibas, Salesin and Stolfi [39] showed an $O(n^3 \log n)$ time algorithm for finding an ϵ-strongly convex $(6\epsilon + 7\alpha)$-hull in $\mathcal{R}^2$ with imprecise computations, where α is the error unit of primitive operations. Chen, Deng, Wada and Kawaguchi [18] gave an $O(n \log n)$ time ($O(n)$ time if S is sorted) algorithm for finding an ϵ-convex $(2 + 4\sqrt{2})\epsilon$-superhull in $\mathcal{R}^2$ with correct computation, and they are considering to generalize it into the algorithm using imprecise computations.

The convex hull algorithms for the geometric objects other than points have been also developed. The convex hull of a simple polygon is found in linear time [11], [42], [48], [52], [68], and the same method can be extended to a linear time algorithm for finding the convex hull of a splinegon which consists of piecewise-smooth Jordan curved-segments [30], [64]. To compute the convex hull of a planar curve which consists of algebraic curved-segments, Bajaj and Kim [9] proposed an algorithm whose running time is linear in the number of algebraic curved-segments, and is a small polynomial in the maximal degree d of the algebraic curves. Rappaport [62] gave an $O(n \log n)$ time algorithm for finding the convex hull of n discs in $\mathcal{R}^2$, and later Devillers and Golin [29] presented an $O(n \log n)$ time incremental algorithm for the same problem. Nielsen and Yvinec [57] gave an $O(n \log h)$ output-size sensitive algorithm for computing the convex hull of discs, convex homothets, non-overlapping objects. Boissonnat, Cérézo, Devillers, Duquesne and Yvinec [12] proposed an $O(n^{\lceil d/2 \rceil} + n \log n)$ time algorithm for finding the convex hull of n spheres in dimension $\mathcal{R}^d$, which is optimal when $d < 3$ or d is even. It can be also used to compute the convex hull of n homothetic convex objects of $\mathcal{R}^d$ in $O(k(n^{\lceil d/2 \rceil} + n \log n))$ time, if the combinatorial complexity of each object is k.

1.2 Parallel Algorithms for PRAM

The construction of the convex hull has received great attention in almost all of parallel computational models. PRAM (Parallel Random Access Machine) is a synchronous parallel computational model employing a number of processors which share a common memory. For exclusive-write PRAMs (the CREW and EREW models) it is known that $\Omega(\log n)$ time is required to compute the convex hull of a set S of n points in $\mathcal{R}^d$, $d \geq 2$ [26]. Optimal deterministic 2-dimensional convex hull algorithms running in $O(\log n)$ time using $O(n \log n)$ work (the product of the time and the number of processors) for the CREW PRAM were given by Atallah and Goodrich [7], [8] and Aggarwal et al. [1], and for the EREW PRAM by Miller and Stout [53]. For sorted planar points, optimal convex hull algorithms running in $O(\log n)$ time using $O(n)$ work for the CREW PRAM were proposed by Goodrich [37] and for the EREW PRAM by W. Chen, Nakano, Masuzawa and Tokura [20] and D. Z. Chen [17]; and for the CRCW PRAM an algorithm running in $O(\log n / \log \log n)$ time using $O(n)$ work was given by Fjällström et al. [34], optimal algorithms running in $O(\log \log n)$ time using $O(n)$ work were given by Chen et al. [20] and Berkman et al. [10], and an $O(1)$ time and $O(n^{1+\epsilon})$ work algorithm was proposed by Chen et al. [20]. Chen et al. [19] also showed that the prefix convex hulls of sorted planar points can be found optimally in the CREW PRAM in $O(\log n)$ time using $O(n)$ work. For 3-dimensional convex hulls, using n processors in the CREW PRAM, $O(\log^3 n)$ time was achieved by Chow [24] and Aggarwal et al. [1], $O(\log^2 n \log^* n)$ time was obtained by Dadoun and Kirkpartrick [27], and $O(\log^2 n)$ time was achieved by Amato and Preparata [2]. For some time, the only solution to the 3-dimensional convex hull problem optimal with respect to time or work was the $O(\log n)$ time and $O(n \log n)$ work randomized algorithm for the CREW PRAM by Reif and Sen [63]. By derandomizing Reif and Sen's algorithm Goodrich obtained an $O(\log^2 n)$ time work-optimal method for the EREW PRAM. Amato and Preparata [4] gave a time-optimal method using $O(n^{1+\epsilon})$ work for the CREW PRAM, where $\epsilon > 0$ is any fixed constant. In higher dimensions, Amato, Goodrich and Ramos [3] showed algorithms for the EREW PRAM running in $O(\log n)$ time using $O(n^{\lfloor d/2 \rfloor} \log^{c(\lceil d/2 \rceil - \lfloor d/2 \rfloor)} n)$ work for $d \geq 4$, where $c > 0$ is a constant, which is optimal for even d. They also gave an $O(\log n)$ time and $O(n \log n + n^{\lfloor d/2 \rfloor})$ work randomized algorithm for $d \geq 3$.

With standard parallel techniques, one can obtain output-size sensitive algorithms in the CRCW PRAM by implementing the method of Kirkpatrick and Seidel [47] for finding the convex hull of planar points in $O(\log^2 n)$ time using $O(n \log h)$ work, and that of Edelsbrunner and Shi [32] for finding the convex hull of points in $\mathcal{R}^3$ in $O(\log^3 n)$ time using $O(n \log h)$ work. Using a randomized CRCW PRAM model, Ghouse and Goodrich [36] gave an $O(\log n)$ time and $O(n \log h)$ work method for $\mathcal{R}^2$, and an $O(\log^2 n)$ time and $O(\min\{n \log^2 h, n \log n\})$ work method for $\mathcal{R}^3$. Gupta and Sen [40] improved the results to $O(\log h \log \log n)$ time and $O(n \log h)$ work in the same model.

There are also some parallel robust algorithms. Chen, Wada and Kawaguchi [22] gave a parallel ro-

bust algorithm for the EREW PRAM which finds an ϵ-strongly convex $O(\epsilon + \beta)$-hull of n planar points with imprecise computations in $O(\log^3 n)$ time using $O(n \log^3 n)$ work, where β is the error unit of primitive operations. Recently, Castanho, Chen and Wada [14] give a parallel algorithm for the EREW PRAM which finds a $(2 + 8\sqrt{2})\epsilon$-superhull of n planar points with correct computation in $O(\log n)$ time using $O(n \log n)$ work ($O(n)$ work if the points are sorted), and they are considering to generalize it into the algorithm using imprecise computations. Parallel convex hull algorithms for other geometric objects have been also developed. Chen et al. [20] and Wagener [70] showed that the convex hull of a simple polygon can be found in $O(\log \log n)$ time using $O(n)$ work for the CRCW PRAM. They also showed that the convex hull of a simple polygon can be found in $O(\log n)$ time using $O(n)$ work for the EREW PRAM or in $O(1)$ time using $O(n^{1+\epsilon})$ work for the CRCW PRAM. Their algorithms can be extended easily for computing the convex hull of a splinegon which consists of piecewise-smooth Jordan curved-segments and for computing the convex hull of a curve which consists of algebraic curved-segments. Chen, Wada and Kawaguchi [23] proposed parallel algorithms for finding the convex hull of n discs in $\mathcal{R}^2$, which runs in $O(\log^{1+\epsilon} n)$ time using $O(n/\log^{\epsilon} n)$ processors or in $O(\log n \log \log n)$ time using $O(n \log^{1+\epsilon} n)$ processors for any positive constant ϵ in the EREW PRAM. The algorithms also work for more general curves in $\mathcal{R}^2$.

1.3 Parallel Algorithms for Other Parallel Computational Models

Convex hull algorithms have been also considered in mesh, mesh-of-trees, hypercube, and models of coarse grained multicomputers like BSP and LogP. Jeong and Lee [46] and Miller and Stout [54] gave $O(\sqrt{n})$ time algorithms for computing the convex hull of n points of $\mathcal{R}^2$ or $\mathcal{R}^3$ for a $\sqrt{n} * \sqrt{n}$ mesh. For planar points, Lee and Jou [49] proposed an $O(\log n)$ time algorithm for $n * n$ mesh-of-trees, Stojmenovic [69] and Holey and Ibarra [44] proposed $O(\log^2 n)$ time and n processors algorithms for hypercube. When n planar points are given ordered one per processor on a machine with n processors, Miller and Stout [53] proposed an $O(\log n)$ time algorithm for hypercube, an $O(\log^3 n/(\log \log n)^2)$ time algorithm for pyramid, tree, and mesh-of-trees, and an $O(\log^2 n)$ time algorithm for reconfigurable mesh; Atallah and D. Z. Chen [5] provided an $O(\log n)$ time algorithm for constructing the convex hull of a simple polygon chain in hypercube. In addition, many other randomized and deterministic convex hull algorithms have been given for the models of coarse grained multicomputers like BSP and LogP [6], [28], [33], [38], [71]. Among the deterministic algorithms, Ferreira et al. [33] introduced a convex hull algorithm in $\mathcal{R}^2$ using a constant number of communication phases and local computation time $O(n \log n/P)$ for n points and P processors, where $n \geq P^{1+\epsilon}$. Zhou et al. [71] gave a new parallel convex hull algorithm in $\mathcal{R}^2$ using a constant number of communication phases for all values of $P \leq n$ while maintaining optimal local computation time.

1.4 Configuration of Other Sections

Since so many researches have been done so far for convex hull problems, it is impossible to explain all the results and the methods. In the following sections, we focus only on several interesting topics including typical techniques used for convex hull algorithms, output-size sensitive methods, randomized approaches and robust algorithms. For the reason of space, we neither give exclusive section for output-size sensitive methods nor for randomized approaches. Instead, we describe an output-size sensitive convex hull algorithm based on randomized approaches to show the basic ideas of both in Sect. 3. Readers can find more details from the papers we have referred. Except the last section, our discussion is based on PRAM. We contribute the last section to other parallel computational models.

2. Typical Techniques for Convex Hull Problems

Divide-and-conquer is perhaps the most important technique used in parallel algorithms. A traditional d-way divide-and-conquer divides a given problem into d subproblems, recursively solves the subproblems in parallel, and then combines the solutions of the subproblems into the solution of the given problem. However, the traditional divide-and-conquer fails if the given problem is difficult to be divided or the solutions of the subproblems are difficult to be combined. A generalized divide-and-conquer technique called *multi-level divide-and-conquer* which combines more than one divide-and-conquer processes together is useful in these cases. In this section, we introduce the d-way divide-and-conquer with the convex hull algorithms of points in $\mathcal{R}^2$ and in $\mathcal{R}^3$, and introduce the multi-level divide-and-conquer with the convex hull algorithms of curved-segments in $\mathcal{R}^2$.

2.1 Traditional Divide-and-Conquer

Let S be a set of n points. An $n^{1/2}$-way divide-and-conquer can be used to compute the convex hull of planar points optimally in the CREW PRAM [1]. Using the pair of points of S with the maximum and the minimum x-coordinate $CH(S)$ can be partitioned into two parts: the upper hull $UH(S)$ and the lower hull $LH(S)$. By symmetry, it is sufficient to show how to compute $UH(S)$. By sorting S according to x-coordinate of the points in $O(\log n)$ time using n processors, $UH(S)$ can be found as follows.

Outline of the Convex Hull Algorithm in $\mathcal{R}^2$

1. Divide S into $n^{1/2}$ subsets S_1, S_2, ..., $S_{n^{1/2}}$ each with $n^{1/2}$ points according to x-coordinate, and recursively find $UH(S_i)$ for each i $(1 \leq i \leq n^{1/2})$ in parallel.
2. Find $UH(S)$ by merging $UH(S_i)$ $(1 \leq i \leq n^{1/2})$ as follows.
   a. Find the common tangent t_{ij} of $UH(S_i)$ and $UH(S_j)$ for each pair of i and j $(1 \leq i < j \leq n^{1/2})$ such that $UH(S_i)$ and $UH(S_j)$ lie under it.
   b. For each i $(1 \leq i \leq n^{1/2})$, find the common tangents rt_i and lt_i such that rt_i has the largest slope in $\{t_{ij},\ i < j \leq n^{1/2}\}$, and lt_i has the smallest slope in $\{t_{ji},\ 1 \leq j < i\}$, respectively. Assuming that lt_i and rt_i contact $UH(S_i)$ at vertices u and v, respectively, if $u = v$ or u lies in the left of v, the vertices of $UH(S_i)$ lie between u and v belong to $UH(S)$, otherwise no vertex of $UH(S_i)$ belongs to $UH(S)$.
   c. Concatenate the vertices of $UH(S_i)$ which belong to $UH(S)$ from $i = 1$ to $n^{1/2}$. □

The common tangent of $UH(S_i)$ and $UH(S_j)$ can be find in $O(\log n)$ time with single processor, therefore, the common tangents for all pairs of i and j can be found in $O(\log n)$ time by using n processors. The common tangents rt_i and lt_i can be found by maximum and minimum computation in $O(\log n)$ time using $n/\log n$ processors. The vertices of $UH(S)$ can be concatenated by using prefix computing in $O(\log n)$ time using $n/\log n$ processors. Therefore, $UH(S)$ can be found in $T(n) = T(n^{1/2}) + O(\log n)$ $(= O(\log n))$ time using n processors in the CREW PRAM. Just as we see in the above algorithm, n^{ϵ}-way divide-and-conquer, where $0 < \epsilon < 1$, is widely used for most convex hull problems. In the following, we give another algorithm which time-optimally computes the convex hull of points in $\mathcal{R}^3$ in $O(\log n)$ time using $n^{1+\alpha}$ processors for any constant $\alpha > 0$ in the CREW PRAM [4].

Outline of the Convex Hull Algorithm in $\mathcal{R}^3$

1. Partition S, by z-coordinate, into n^{α} subsets each of size $n^{1-\alpha}$; let $m = n^{\alpha}$ and denote the ith subsets by S_i.
2. Recursively compute $CH(S_i)$ $(1 \leq i \leq m)$ in parallel.
3. Merge $CH(S_i)$, $1 \leq i \leq m$, to form $CH(S) = CH(\cup_{1 \leq i \leq m} CH(S_i))$. □

It is easily seen that if the merging process can be done in $O(\log n)$ time using $n^{1+\alpha}$ processors in the CREW PRAM, $CH(S)$ can be found in $T(n) = O(\log n) + T(n^{1-\alpha})$ $(= O(\log n))$ time using $n^{1+\alpha}$ processors in the CREW PRAM. In the following, we consider the merging process.

Consider two separable convex hulls $CH(P)$ and $CH(Q)$, and the convex hull of their union $CH(P \cup Q)$. A vertex v is *external* if it is a vertex of $CH(P \cup Q)$, and otherwise it is *internal*. In addition, a vertex $v \in CH(P)$ is said to be a *seam* vertex if it is an external vertex and it is incident to some vertex $w \in CH(Q)$ on $CH(P \cup Q)$, i.e., one of its neighbors on $CH(P \cup Q)$ is a vertex of $CH(Q)$. External vertices that are not seam vertices are referred to as *e-external* vertices and external vertices that are seam vertices are referred to as *s-external* vertices.

Outline of the Merging Process

1. For all i, $1 \leq i < j \leq n$, compute $CH(CH(S_i) \cup CH(S_j))$.
2. Let S^* denote the set of all vertices that resulted external in all pairwise merges in Step 1, and s-external in at least one pairwise merge. For each $v \in S^*$, construct the set A_v, which consists of all vertices that were adjacent s-external vertices to v in some pairwise merge, i.e., if $v \in CH(S_i)$ and w *in* A_v, then w belongs to v's neighborhood on $CH(CH(S_i) \cup CH(S_j))$, where $w \in CH(S_j)$ for some $1 \leq j \leq m$ and $j \neq i$.
3. Consider vertex $v \in S^*$; assume that $v \in CH(S_i)$ and let $n_i(v)$ denote v's neighborhood on $CH(S_i)$. For each such vertex v, classify if v is external or internal to $A_v \cup n_i(v)$.
4. Adjoin to the set $\{v|\ v \in S$ and v has been classified external in Step 3 $\}$, the set of points that resulted e-external in all pairwise merges, these are the vertices of $CH(S)$. □

The above merging process can be done in $O(\log n)$ time using $n^{1+\alpha}$ processors in the CREW PRAM. The details can be found in [4].

2.2 Multi-Level Divide-and-Conquer

Traditional divide-and-conquer does not always work. In this section, we consider the problem of finding the convex hull of a set S of n discs in the plane [23]. Since the boundary of the discs are arcs, $CH(S)$ consists of the portions of the arcs. Let $CH(S)$ be represented by the sequence of the arcs listing in counter-clockwise order. It is known that $CH(S)$ consists of at most $2n-1$ arcs. A straight line passing through the leftmost and the rightmost points separates $CH(S)$ into two hulls: the upper one $UH(S)$ and the lower one $LH(S)$. Like the case of points, only $UH(S)$ is constructed in the following. Two sets of arcs are *separated* if the arcs of them lie on different sides of a vertical line. A traditional m-way divide-and-conquer computes $UH(S)$ in the following two methods: (1) dividing the n discs into m equally-sized subsets of discs, recursively finding the upper hull of each subset in parallel, and then merging these m upper hulls into $UH(S)$, and (2) partitioning

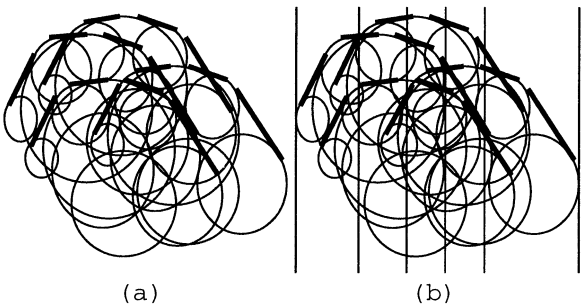

Fig. 1 Finding the convex hull of discs with mulit-level divide-and-conquer.

the n discs into m equally-sized separate sets with $m-1$ vertical lines, recursively computing the upper hull of each subset in parallel, and then merging the m upper hulls into $UH(S)$. To make the presentation simple, in the following, we let that a traditional divide-and-conquer consists of two steps and call the part before the merge step the *recursive step*. What makes the first method inefficient is that the m upper hulls obtained in the recursive step may intersect with each other and they may have altogether $\Theta(mn)$ intersections, and in order to merge the upper hulls, we may have to find these $\Theta(mn)$ intersections. In the second method, the m upper hulls obtained in the recursive step do not intersect with each other, but they may contain altogether $\Theta(mn)$ arcs, hence it is also inefficient.

Multi-level divide-and-conquer (MDC) is similar to a traditional one except it contains more than one recursive step. Without using MDC, the most efficient algorithm finds $UH(S)$ in $O(\log^2 n)$ time using $O(n/\log n)$ processors. With MDC, the problem can be solved by combining the above two methods as follows:

Outline of the convex hull algorithm for discs with MDC

1. **(First recursive step)** Divide the set S of n discs into δ subsets such that each subset contains n/δ discs, and then recursively construct the upper hull of each subset in parallel (Fig. 1 (a)). (Note that these upper hulls may intersect with each other.)
2. **(Second recursive step)** Using $n/\delta - 1$ vertical lines, partition the δ upper hulls which are obtained in the first step and have at most $2n-1$ arcs totally into n/δ separated parts, and then recursively find the upper hull of each part in parallel (Fig. 1 (b)).
3. **(Merge step)** Merge the n/δ upper hulls obtained in the second recursive step into the upper hull of S. □

In the second recursive step, partitioning δ upper hulls by $n/\delta - 1$ lines increases at most $n-\delta$ arcs, i.e., the total size of the subsets does not exceed $3n$; furthermore, the n/δ upper hulls produced in the second division step are separated, therefore they can be merged into $UH(S)$ easily by computing the common tangents of the hulls. When $\delta = n^{1/2}$, the merge step can be done in $O(\log n)$ time using $n/\log n$ processors. Therefore, the upper hull of n discs in the plane can be found in $T(n) = T(n^{1/2}) + T(3n^{1/2}) + O(\log n)$ $(= O(\log n \log\log n))$ time using $P(n) = \max\{n^{1/2}P(n^{1/2}),\ n^{1/2}P(3n^{1/2}),\ O(n/\log n)\}$ $(= O(n\log n))$ processors. By adjusting the parameter δ, the algorithm can be developed into a work optimal one which runs in $O(\log^{1+\epsilon} n)$ time using $O(n/\log^{\epsilon} n)$ processors. The algorithm can be also generalized into one for constructing the convex hull of curved-segments in $\mathcal{R}^2$. Readers can find the details in [23]. It is shown that the MDC technique has many other applications [21].

3. Output-size Sensitive Algorithms Based on Randomized Approaches

In this section, we introduce an output-size sensitive convex hull algorithm. It is a Las Vegas randomized one which finds the convex hull of a set S of n planar points in $O(\log n)$ time using $O(n\log h)$ work with high probability [36], where h is the size of the convex hull. In the following, we show how to compute the upper hull. The algorithm is similar in structure to randomized quicksort, picking a point at random from the input, then splitting the input about that point and recurring. However, the upper hull edge lying above the splitting point called *bridge* is found before recursion. That is, after finding the bridge, S is divided into three subsets S_1, S_2 and S_3 whose points lie in the left, in the right and under the bridge, respectively, then the process is executed for S_1 and S_2 recursively. In this sense, the algorithm uses the "marriage-before-conquest" paradigm. If the subsets S_1 and S_2 had the same size, in general, if the subproblems had the same size at each level of recursion, 2^{i-1} bridges (edges of the convex hull) would be found in the ith recursive step, therefore the algorithm would finish in $O(\log h)$ recursive steps. Usually, the subproblems have different sizes, therefore the algorithm and the analysis of complexity are much more complecated. In the following, the outline of the algorithms for finding the bridge of a splitting point, and furthermore finding the upper hull is given. The further details of the algorithms and analysis should be found in [36].

Outline of Bridge-Finding-Procedure

1. Find a random sample of size $\Theta(k)$ from the given set S of n planar points, in constant time using n processors, where k is sufficiently small and the sample is uniformaly random with probability $\geq 1 - 2(e/2)^{-k}$.
2. Solve the base problem, i.e., finding the edge of the upper hull of the sample which lies above the given splitting point, deterministically, in constant time

using k^3 processors.

3. Check for each point whether it violates the solution just found, i.e., whether it can be the candicate of the endpoins of the requested bridge, in constant time using n processors. If so, it remains as a survivor to be in the next base problem.
4. Repeat Steps 1-3 such that the survivors can be compressed into a problem of size $O(k)$. The solution to this problem is the bridge sought.

Lemma 1: With probability $\geq 1 - e^{-\Omega(k^r)}$, where r $(0 < r < 1)$ is a constant, the upper hull edge through the splitting point can be found in a constant number of iterations, of solving base problems.

Outline of the Convex Hull Algorithm

The algorithm consists of $O(\log n)$ steps with high probability. At the begining of the algorithm, all the points of S are set to be active. After executing the ith phase, $n(i)$ bridges (edges of the upper hull) of S have been found in left to right order, where $n(0)$ is considered as 0, and the problem is reduced to find the upper hulls of S_1, S_2, ..., $S_{n(i)+1}$, respectively, where S_j $(1 \leq j \leq n(i)+1)$ and S_{j+1} are subsets of S and their points lie in the left and the right of jth bridge, respectively. Let $A = \{S_j \mid S_j \neq \emptyset$, and $1 \leq j \leq n(i)+1\} = \{S_{t_1}, S_{t_2}, \ldots, S_{t_i}\}$, where $t_v > t_u$ if $v > u$. If A is empty, it completes the algorithm, else let $S'_j = S_{t_j}$ and do the $i+1$ phase as follows.

(i) If $i \geq \log n/32$, add the number t_i of the subproblems to the number $n(i)$ of the hull edges (bridges) found thus far to get l, which is a lower bound on the size h of the convex hull of S. If $l \geq n^{1/32}$, solve the problem using any optimal non-outputsize-size sensitive parallel algorithm. It completes the algorithm. Else do the following step.

(ii) For each S'_j $(1 \leq j \leq t_i)$, choose a random splitting point p from S'_j, and find a bridge above p by applying the bridge-finding-procedure. If the procedure for subset S'_j has not naturally terminated after a certain constant α steps, then terminate the procedure and say that subset S'_j has failed.

(ii) Use failure sweeping to compact those problems that failed in the previous step, so that each problem can be assinged $n^{3/4}$ processors (it is proved that the failed problems is less than $n^{1/4}$) with high probability, and use the bridge-finding-procedure to find bridges.

(v) For each j $(1 \leq j \leq t_i)$, regard the points of S'_j lying under the bridge as dead, and divide the remaining points of S'_j into two subsets which are numbered as $2j-1$ and $2j$, whose points lie in the left and the right of the birdge, resepctively.

The algorithm then continues at recursion level $i+1$, with above actions, until all points are dead, or until switch to using the other optimal non-outputsize-size sensitive parallel algorithm. □

Theorem 1: With probability $1 - n^{-b}$, the convex hull in $\mathcal{R}^2$ can be found in $O(\log n)$ time with $O(n \log h)$ work, where b is a constant and h is the size of the convex hull.

4. Robust Algorithms

Although many geometric algorithms have been developed so far, they cause surprising problems in practice. The major reason is that the basic geometric tests are unreliable or inconclusive when being implemented by imprecise computations such as ordinary floating point arithmetic. This uncertainty makes the solutions inaccuracy or even not satisfying the proposed geometric properties. Therefore, robust geometric algorithms whose correctness is not spoiled by numerical errors have attracted increasing attention recently. In this section, we show parallel methods for designing robust convex hull algorithms. Considering the situation that the output of one robust convex hull algorithm may become the input of another robust algorithm, we desire that the solution is not just weakly convex, even not just convex, but strongly convex so that many of the desirable properties are preserved in some fashion even they are tested with imprecise computations. We find a strongly convex approximate hull in two steps: (1) find a convex approximate hull of S, and (2) make it strongly convex. In the following we give some necessary concepts and the outline of the algorithms. The further details can be found in the original paper [22].

4.1 Primitive Operations

Let $l(pq)$ denote the straight line passing through segment pq, and $X(p)$ and Y(p) denote the x and y coordinates of point p, respectively. Two primitive operations are considered:

OP_I computing $d(z, pq)$, the signed distance from point z to line $l(pq)$, where $d(z, pq) > 0$ if z, p and q are in counter-clockwise, $d(z, pq) < 0$ if they are in clockwise, or $d(z, pq) = 0$ if they are collinear, and

OP_{II} computing $\sin(\theta(pq, p'q'))$, where $\theta(pq, p'q')$ equals the change in orientation from $l(pq)$ to $l(p'q')$ in counter-clockwise.

The slope of a line is usually defined by function *tangent*. To avoid large numerical errors caused by *tangent* whose value may reach ∞, *sine* is used to measure the slope. Let points $o = (0, 0)$ and $o_1 = (1, 0)$. The slope of line $l(pq)$ is defined to be $slop(pq) = \sin(\theta(oo_1, pq))$.

Let $\beta_1 > 0$ and $\kappa > 0$ be the error units caused by operation OP_I and OP_{II}, respectively. Inequalities $|d(z, pq) - (d(z, pq))_{\mathcal{I}}| \leq \beta_1$ and $|\sin(\theta(pq, p'q')) - (\sin(\theta(pq, p'q')))_{\mathcal{I}}| \leq \kappa$ hold, where the subscript $\mathcal{I}$ denotes the imprecise computations. A general error unit

is defined as $\beta = \max\{\beta_1, \beta_2\}$, where $\beta_2 = \kappa\mathcal{T}$ and $\mathcal{T}$ is a bound on the magnitude of the coordinates in the inputs. It is easily seen that β_2 represents the distance error caused by κ. Except operations OP_I and OP_{II}, comparison operation is used. As most related researches, a pure comparison operation is assumed to cause no numerical error.

Let P be a convex polygon, and u, v and w be any three contiguous vertices of P listing in counter-clockwise. Vertex v is said to be ϵ-*convex* in P if $d(v, uw) \geq 2\epsilon$. It is easily seen that if each vertex of P is ϵ-convex, then P is ϵ-strongly convex.

4.2 Finding A Convex Approximate Hull

Given two points p and q with $X(p) < X(q)$, let l_p and l_q be the vertical lines passing through points p and q, respectively. Lines l_p and l_q divide point set S into three subsets: $S_L(pq)$, $S_M(pq)$ and $S_R(pq)$ which consist of the points located in the left of l_p, between l_p and l_q (including the points on l_p and l_q), and in the right of l_q, respectively. In the following, the point sets S_1 and S_2 are assumed to be separated by a vertical line. The points of S_1 lie in the left of any points of S_2, and $S = S_1 \cup S_2$.

A line l is *the upper bridge* of S_1 and S_2, if l passes at least one point of each S_1 and S_2 and no point of S_1 and S_2 lies above l. The concept can be generalized as follows.

Definition 1:
1. Line l is a δ-upper support line of S_1, if l passes through at least one point of S_1 and any point of S_1 located above l lies at most δ away from l.
2. Line $l(pq)$ is a δ-upper bridge of S_1 and S_2, denoted as $UB(S_1, S_2, \delta)$, if points $p \in S_1$ and $q \in S_2$, and $l(pq)$ is a δ-upper support line of both S_1 and S_2.
3. A δ-upper bridge $l(pq)$ of S_1 and S_2 is ν-bounded, denoted as $BUB(S_1, S_2, \delta, \nu)$, if for each point $u \in S$ located above $l(pq)$ (i) $\sin(\theta(uq, pq)) \leq \nu$ holds if $u \in S_L(pq)$, and (ii) $\sin(\theta(pq, pu)) \leq \nu$ holds if $u \in S_R(pq)$ (Fig. 2). □

Symmetrical concepts of δ-lower support lines, δ-lower bridge, and δ-lower bridge bounded by ν can be defined similarly. We discuss the upper ones only. The following procedure *FindBridge-Or-DeletePoints*(S_1, S_2) either finds a segment pq such that $l(pq)$ is a $BUB(S_1, S_2, \delta, 4\kappa)$, where $\delta = \max\{\beta_1 + 3\beta_2, 4\beta_2\}$, or deletes about $n/4$ points of S which lie properly under the $BUB(S_1, S_2, 0, 0)$ in $O(\log n)$ time using n processors with imprecise computations in the EREW PRAM. Since the $BUB(S_1, S_2, 0, 0)$ exists and the points of S_1 or S_2 located on the $BUB(S_1, S_2, 0, 0)$ are never deleted, a $BUB(S_1, S_2, \delta, 4\kappa)$ can be finally found by calling the procedure $O(\log n)$ times.

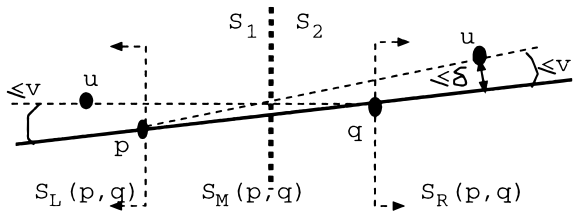

Fig. 2 A δ-upper bridge $l(pq)$ of S_1 and S_2 is ν-bounded.

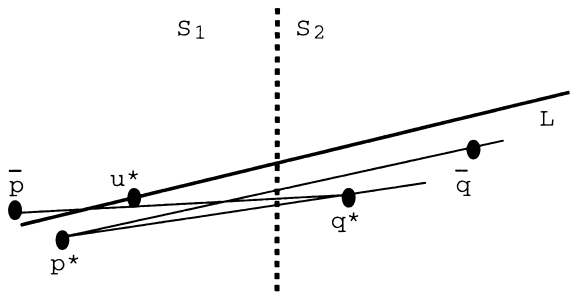

Fig. 3 Findig a $BUB(S_1, S_2, \delta, 4\kappa)$.

Procedure *FindBridge-Or-DeletePoints*(S_1, S_2)

(Step 1) If $S_1 = \{p\}$ and $S_2 = \{q\}$, then $l(pq)$ is $BUB(S_1, S_2, 0, 0)$. This completes the procedure. Else do the following steps.

(Step 2) (Decide whether deleting the points of S or finding a $BUB(S_1, S_2, \delta, 4\kappa)$)

(1) Find point u^* and line L such that L is a β_1-upper support line of S passing through u^* as follows (Fig. 3). Construct $\lceil n/2 \rceil$ pairs $e = (p, q)$, where $X(p) \leq X(q)$, by matching every two points of S in any way (if n is odd, the last point is matched twice). Let E denote the set of these pairs. Use operation OP_{II} to compute $(slop(e))_{\mathcal{I}}$ for each $e \in E$, and then find segment e^* such that $M = (slop(e^*))_{\mathcal{I}}$ is the median of $\{(slop(e))_{\mathcal{I}} \mid e \in E\}$. Use operation OP_I to compute $(d(u, e^*))_{\mathcal{I}}$ for each $u \in S$, and then find point u^* such that $(d(u^*, e^*))_{\mathcal{I}} = \max\{(d(u, e))_{\mathcal{I}} \mid u \in S\}$. Let L be the line passing through u^* with slope M.

In the following, assume that $u^* \in S_1$. The case that $u^* \in S_2$ can be treated symmetrically by exchanging the roles of the sets S_1 and S_2, the sets P and Q, the points p and q , p^* and q^*, and $\overline{p}$ and $\overline{q}$ in the following, respectively.

(2) Use operation OP_{II} to compute $(slop(u^*q))_{\mathcal{I}}$ for each $q \in S_2$, and then let $Q = \{ q \mid q \in S_2$ and $(slop(u^*q))_{\mathcal{I}} \geq M - 2\kappa\}$. If Q is empty, execute Step 3 else find q^* in Q such that q^* has the largest x coordinate and then execute Step 4.

(Step 3) (Delete one fourth points of S)

Find $E' = \{e = (a, b) \mid (a, b) \in E$ and $(slop(e))_{\mathcal{I}} \geq M\}$ and delete point a from S for each $e = (a, b) \in E'$.

This completes the procedure.

(Step 4) (Find a $BUB(S_1, S_2, \delta, 4\kappa)$)
Use operation OP_{II} to compute $(slop(pu^*))_{\mathcal{I}}$ for each $p \in S_1$. Find p^* such that it has the smallest x coordinate in $P = \{p \mid p \in S_1,\ X(p) < X(u^*)$ and $(slop(pu^*))_{\mathcal{I}} \leq M + \kappa\}$. If P is empty, set $p^* = u^*$. Use operation OP_{II} to compute $(slop(p^*q^*))_{\mathcal{I}}$.

[Case 1] $(slop(p^*q^*))_{\mathcal{I}} \leq M$.

Use operation OP_{II} to compute $(\sin(\theta(p^*q^*, p^*q)))_{\mathcal{I}}$ for each $q \in S_2$. Find $U = \{q \mid q \in S_2,\ X(q) \geq X(q^*)$ and $(\sin(\theta(p^*q^*, p^*q)))_{\mathcal{I}} \geq \kappa\}$, and find $\overline{q}$ such that $(\sin(\theta(p^*q^*, p^*\overline{q})))_{\mathcal{I}} = \max\{(\sin(\theta(p^*q^*, p^*q)))_{\mathcal{I}} \mid q \in U\}$. If U is empty set $\overline{q} = q^*$. Output line $l(p^*\overline{q})$. This completes the procedure.

[Case 2] $(slop(p^*q^*))_{\mathcal{I}} > M$.

Using operation OP_{II} to compute $(\sin(\theta(pq^*, p^*q^*)))_{\mathcal{I}}$ for each $p \in S_1$. Find $V = \{p \mid p \in S_1,\ X(p) \leq X(p^*)$ and $(\sin(\theta(pq^*, p^*q^*)))_{\mathcal{I}} \geq \kappa\}$. Find $\overline{p}$ such that $(\sin(\theta(\overline{p}q^*, p^*q^*)))_{\mathcal{I}} = \max\{(\sin(\theta(pq^*, p^*q^*)))_{\mathcal{I}} \mid p \in V\}$, if V is empty set $\overline{p} = p^*$. Output line $l(\overline{p}q^*)$. This completes the procedure. □

Now it is ready to construct a convex δ-upper hull of S. Let u and v be the leftmost and rightmost vertices of S. First we sort S in x-coordinate, and then construct a convex δ'-upper hull of S which contains u and v by the following algorithm *MakeConvexUpperHull(S)*, where $\delta = \max\{\beta_1 + 9\beta_2, 10\beta_2\}$. The algorithm is based on two-way divide-and-conquer. It (i) divides S into two separated subsets S_1 and S_2, (ii) finds an approximate bridge b of S_1 and S_2, i.e., a $BUB(S_1, S_2, \delta, 4\kappa)$, (iii) recursively finds F_1 and F_2, convex δ'-upper hulls of the points lie in the left and in the right of b respectively, in parallel, and (iv) combines F_1 and F_2 with bridge b into a convex upper δ'-hull of S. Since b can be found in $O(\log^2 n)$ time using n processors, the convex δ'-upper hull of S can be found in $O(\log^3 n)$ time using n processors with imprecise computations in the EREW PRAM. A convex δ'-lower hull of S containing u and v can be constructed similarly. A convex δ-hull of S is obtained by putting them together.

Algorithm *MakeConvexUpperHull*(S)
(Input) $S = (u_1, u_2, \ldots, u_n)$, a sequence of n points in the plane sorted by x coordinate in increasing order.
(Output) A convex δ'-upper hull of S, where $\delta' = \max\{\beta_1 + 9\beta_2, 10\beta_2\}$, which contains the leftmost point u_1 and the rightmost point u_n of S.

(Step 1) Divide S into two separated subsequences $S_1 = (u_1, u_2, \ldots, u_{\lceil n/2 \rceil})$ and $S_2 = (u_{\lceil n/2 \rceil + 1}, u_{\lceil n/2 \rceil + 2}, \ldots, u_n)$. Find a line $l(u_s u_t)$ which is a $BUB(S_1, S_2, \delta, 4\kappa)$, where $u_s \in S_1$ and $u_t \in S_2$. Let $S^1 = (u_1, u_2, \ldots, u_s)$ and $S^2 = (u_t, u_{t+1}, \ldots, u_n)$, where S^1 is a prefix of S_1 and S^2 be a suffix of S_2.

(Step 2) Recursively construct F_1 and F_2, the convex δ'-hulls of S^1 and S^2, respectively, in parallel. Notice that F_1 contains vertices u_1 and u_s, and F_2 containing vertices u_t and u_n (Fig. 4).

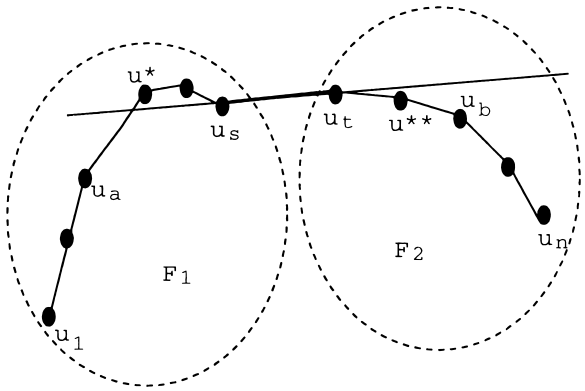

Fig. 4 Finding a convex approximate hull with two-way divide-and-conquer.

(Step 3) Use Operation OP_{II} to compute set $B_1 = \{u \mid (\sin(\theta(u_s u_t, u u_t)))_{\mathcal{I}} \geq 5\kappa,\ u \in F_1\}$ and find the rightmost vertex u_a of B_1. Similarly, compute $B_2 = \{u \mid \sin(\theta(u_s u, u_s u_t)))_{\mathcal{I}} \geq 5\kappa,\ u \in F_2\}$ and find the leftmost vertex u_b of B_2. Let u^* be the right neighbor of u_a in F_1 and u^{**} be the left neighbor of u_b in F_2, and let $\overline{F_1}$ be the left part of F_1 from u_1 to u^* and $\overline{F_2}$ be the right part of F_2 from u^{**} to u_n. Concatenate $\overline{F_1}$,u^*u^{**} and $\overline{F_2}$. The resulting polygon F is a convex δ-upper hull of S which contains the leftmost point u_1 the rightmost point u_n of S. □

4.3 Making a Convex Polygon Strongly Convex

Let P be a convex polygon with m vertices.

Definition 2: Let P' be a subpolygon of P, i.e., the vertices P' are taken from P. Polygon P is an (ϵ, δ)-ridge-ring of P, denoted as $r(P, \epsilon, \delta)$, if P' satisfies the following conditions: (i) at least one vertex is ϵ-convex in every three contiguous vertices of P', and (ii) P' is a δ-hull of P. □

An $r(P, \epsilon, 2\epsilon + 2\beta_1)$ can be found in $O(\log m)$ time using m processors in the EREW PRAM with imprecise computation. Let R be the resulted $r(P, \epsilon, 2\epsilon + 2\beta_1)$. By deleting the vertices which are not ϵ-convex from R, the resulted R' is ϵ-convex. From Definition 2, between any two adjacent ϵ-convex vertices of R, there are at most two vertices which are not ϵ-convex. It is easily proved that deleting these vertices make them lie at most $4\epsilon + 4\beta_1 + 4\beta_2$ outside of R'. Therefore, An ϵ-strongly convex $(6\epsilon + 6\beta_1 + 4\beta_2)$-hull of a convex m-gon can be computed in $O(\log m)$ time using m processors with imprecise computations in the EREW PRAM. Combining with the result of the previous subsection, an ϵ-strongly convex $(6\epsilon + \max\{7\beta_1 + 13\beta_2, 6\beta_1 + 14\beta_2\})$-hull (i.e., an ϵ-strongly convex $(6\epsilon + 20\beta)$-hull) of n points can be constructed in imprecise computations in $O(\log^3 n)$ time using n processors in the EREW PRAM.

5. Approaches on Other Models

Convex hull construction has been also considered in mesh, mesh of trees, hypercube, recofigurable mesh (array) and models of coarse grained multicomputers like BSP and LogP. Except reconfigurable mesh, the convex hull algorithms in these models are basically similar to those in the PRAM but the time used for communication is considered. In coarse grained multicomputers like BSP and LogP, super-steps are used to decrease the frequency of communication. In an ordinary mesh or its variants such as mesh of trees and hypercube, a routing strategy of choosing short but not busy paths is nessary for efficiently transmitting messages between processors. Readers can find more details from the papers we referred in Sect. 1. In the following, we introduce the convex hull algorithms only for recofigurable mesh.

In essence, a *reconfigurable mesh* consists of a mesh-connected multiprocessor augmented by the addition of a dynamic bus system whose configuration changes in response to computational and communication needs. More precisely, a reconfigurable mesh of size $n \times m$ consists of nm identical SIMD processors positioned on a rectangular array with n rows and m columns. As usual, it is assumed that every processor knows its own coordinates within the mesh.

Each processor is connected to its four neighbors provided by they exist local connections between these ports can be established at run time, under program control, creating a powerful *bus system* whose configuration changes dynamically to accommodate various computational needs.

In general, the bus system established as a result of setting local connections involves a number of disjoint *subbuses*. As it turns out, these subbuses can be used, in parallel, as powerful communication or computational devices.

On the reconfigurable mesh, the convex hull problem has been addressed in two different contexts: for *sparse input* and for *dense input*. While the sparse case allows one to use more processors than input points, in the dense case the number of processors and the number of input points are, essentially, the same. For sparse input, Olariu et al. [58] and Jang et al. [45] proposed $O(1)$ time algorithms to compute the convex hull of a set of $\sqrt{n}$ points on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. The algorithm is as follows. First, sort the $\sqrt{n}$ points by x-coordinate in $O(1)$ time. Then, partition the reconfigurable mesh into $n^{1/4}$ submeshes of size $\sqrt{n} \times n^{1/4}$ each. On each submesh, the convex hull of consecutive $n^{1/4}$ points can be computed in $O(1)$ time, because it has $(n^{1/4})^3 = n^{3/4}$ processors. After that, $n^{1/4}$ sub convex hulls are merged in $O(1)$ time. Nakano [56] extended this algorithm and showed that if the $\sqrt{n}$ points are sorted beforehand, then, for every fixed $\epsilon > 0$, the convex hull can be computed in $O(1)$ time on a reconfigurable mesh of size $\sqrt{n} \times n^{\epsilon}$.

In the dense case, Miller and Stout [53], and Olariu et al. [59] proposed an $O(\log^2 n)$ time algorithm computing the convex hull of a sorted set of n points, pretiled in proximity order on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. The idea of the algorithm is as follows. Partition the input into two subsets of $n/2$ consecutive points each. The convex hull of each subsets are computed recursively, and they are merged into one convex hull by finding the common tangent. Since the binary search can be used to find the common tangent, this merge can be done in $O(\log n)$ time. Thus, the total computing time is $O((\log n)^2)$. Nakano [56] showed that the convex hull of a set of $\sqrt{mn}$ points sorted in column major order can be computed in $O(\frac{\log^2 n}{\log m} + \log^2 m)$ time on a reconfigurable mesh of size $\sqrt{m} \times \sqrt{n}$. In particular, for $m = 2^{\log^{\frac{2}{3}} n}$ the computing time becomes $O(\log^{\frac{4}{3}} n)$. In this algorithm, the convex hull of the points pretiled on each column is computed independently in $O((\log m)^2)$ time. After that, the n convex hulls are merged into one in $O(\frac{\log^2 n}{\log m})$ time.

Hayashi et al. [43] showed that the convex hull of n points can be computed in $O((\log \log n)^2)$ time on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$ if the input is sorted by proximity order. The idea of this algorithm is as follows: First, partition the input into $\sqrt{n}$ subsets of $\sqrt{n}$ consecutive points, and find the convex hull recursively. After that, they are merged in $O(\log \log n)$ time. This merge is done by iteration of picking sample points and finding the convex hull of the sample points. They also proved that $\Omega(\log \log n)$ time is necessary to compute the convex hull for the dense input.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel computational geometry," Algorithmica, vol.3, pp.163–173, 1988.
[2] N.M. Amato and F.P. Preparata, "The parallel 3D convex hull problem revisited," Comput. Geom. Appl., vol.2, no.2, pp.163–173, 1992.
[3] N.M. Amato, M.T. Goodrich, and E.A. Ramos, "Parallel algorithms for higher-dimensional convex hulls," Proc. of the 36th Annual Symposium on Foundations of Computer Science, pp.683–694, 1995.
[4] N.M. Amato and F.P. Preparata, "A time-optimal parallel algorithm for three-dimensional convex hulls," Algorithmica, vol.14, no.2, pp.169–182, 1995.
[5] M.J. Atallah and D.Z. Chen, "Optimal parallel hypercube algorithms for polygon problems," IEEE Trans. Comput., vol.44, no.7, pp.914–922, 1995.
[6] M.J. Atallah and D.J. Chen, "Parallel geometric algorithms in coarse-grain network models," Proc. of 4th Annual International Computing and Combinatorics Conference, pp.55–64, 1998.
[7] M.J. Atallah and M.T. Goodrich, "Efficient parallel solutions to some geometric problems," J. Parallel Distrib. Comput., vol.3, pp.492–507, 1986.

[8] M.J. Atallah and M.T. Goodrich, "Parallel algorithms for some functions of two convex polygons," Algorithmica, vol.3, pp.535–548, 1988.
[9] C. Bajaj and M.S. Kim, "Convex hull of objects bounded by algebraic curves," Algorithmic, vol.6, pp.533–553, 1991.
[10] O. Berkman, B. Schieber, and B. Vishkin, "A fast parallel algorithm for finding the convex hull of a sorted point set," Comput. Geom. Theory Appl., vol.6, no.2, pp.231–241, 1996.
[11] B. Bhatacharya and J.E. Gindy, "A new linear convex hull algorithm for simple polygons," IEEE Trans. on Information Theory, vol.30, no.1, pp.85–88, 1984.
[12] J.D. Boissonnat, A. Cérézo, O. Deviller, J. Duquesne, and M. Yvinec,"An algorithm for constructing the convex hull of a set of spheres in dimension d," Comput. Geom., Theory Appl., vol.6, pp.123–130, 1996.
[13] H. Bröonnimann, B. Chazellel, and J. Matoušek, "Product range spaces, sensitive sampling, and derandomization," Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci., pp.400–409, 1993.
[14] C.D. Castanho, W. Chen, and K. Wada, "A parallel algorithm for constructing strongly convex superhulls of points," to appear in IEICE Trans. Fundamentals, vol.E83-A, no.4, April 2000.
[15] B. Chazellel, "An optimal convex hull algorithm in any fixed dimension," Discrete Comput. Geom., vol.10, pp.377–409, 1993.
[16] B. Chazellel and J. Matoušek, "Derandomizing an output-sensitive convex hull algorithm in three dimensions," Technical Report, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1980.
[17] D.Z. Chen, "Efficient geometric algorithms on the EREW PRAM," IEEE Trans. Parallel and Distributed Systems, vol.6, pp.41–47, 1995.
[18] W. Chen, X.W. Deng, K. Wada, and K. Kawaguchi, "Constructing a strongly convex superhull of points," Third Annual International Computing and Combinatorics Conference, Lecture Notes in Computer Science, no.1276, pp.4–51, 1997.
[19] W. Chen, K. Nakano, T. Masuzawa, and N. Tokura, "A parallel method for the prefix convex hulls problem," IEICE, Trans. Fundamentals, vol.E77-A, no.10, pp.1675–1683, Oct. 1994.
[20] W. Chen, K. Nakano, T. Masuzawa and N. Tokura, "Optimal parallel algorithms for finding the convex hull of a sorted point set," IEICE, Trans., vol.J74-D-I, no.12, pp.814–825, Dec. 1991.
[21] W. Chen and K. Wada, "Multi-level divide-and-conquer: A method for designing efficient parallel algorithms," Proc. of 2nd International Conference on Parallel and Distributed Computing and Networks, 1998.
[22] W. Chen, K. Wada, and K. Kawaguchi, "Parallel robust algorithms for constructing strongly convex hulls," Proc. of the 12th Annual ACM Symposium on Computational Geometry, pp.133–140, 1996.
[23] W. Chen, K. Wada, K. Kawaguchi, and D.Z. Chen, "Finding the convex hull of discs in parallel," J. of Computational Geometry and Applications, vol.8, no.3, pp.305–319, 1998.
[24] A.L. Chow, "Parallel algorithms for geometric problems," Ph.D. Thesis, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1980.
[25] K.L. Clarkson and P.W. Shor, "Application of random sampling in computational geometry, II," Discrete Comput. Geom., vol.4, pp.387–421, 1986.
[26] S.A. Cook, C. Dwork, and R. Reischuk, "Upper and lower time bounds for parallel random access machines without simultaneous writes," SIAM J. Comput., vol.15, pp.87–97, 1986.
[27] N. Dadoun and D.G. Kirkpatrick, "Parallel construction of subdivision hierarchies," Comput. Syst. Sci., vol.39, pp.153–165, 1989.
[28] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A.A. Khokhar, "A randomized parallel three-dimensional convex hull algorithm for coarse-grained multicomputers," Theory Comput. Syst., vol.30, no.6, pp.547–558, 1997.
[29] O. Devillers and M.J. Golin, "Incremental algorithms for finding the convex hulls of circles and the lower envelopes of parabolas," Proc. of the 6th Canadian Conference on Computational Geometry, pp.153–158, 1994.
[30] D.P. Dobkin and D.L. Souvaine, "Computational geometry in a curved world," Algorithmica, vol.5, pp.421–457, 1990.
[31] H. Edelsbrunner, "Algorithms in combinatorial geometry," vol.10 of EATCS Monographs on Theoretical Computer Science, Springer Verlag, Heidelberg, West Germany, 1987.
[32] H. Edelsbrunner and W. Shi, "An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem," SIAM J. on Computing, vol.20, pp.259–269, 1991.
[33] A. Ferreira, A. Rau-Chaplin, and S. Ueda, "Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers," Proc. of 7th IEEE Symposium on Parallel and Distributed Processing, pp.561–568, 1995.
[34] P-O. Fjällström, J. Katajainen, C. Levcopoulos, and O. Petersson, "A sublogarithmic convex hull algorithm," Bit, vol.30, pp.378–384, 1990.
[35] S. Fortune," Stable maintenance of point set triangulations in two dimensions," Proc. of the 30th Annu. Sympos. on Foundations of Computer Science, pp.494–499, 1989.
[36] M. Ghouse and M.T. Goodrich, "In-place techniques for parallel convex hull algorithms," in Proc. 3rd ACM Sympos. Parallel Algorithms Architect., pp.192–203, 1991.
[37] M.T. Goodrich, "Finding the convex hull of a sorted point set in parallel," Information Processing Letters, vol.26, pp.173–179, 1987.
[38] M.T. Goodrich, "Randomized fully-scalable BSP techniques for multi-searching and convex hull construction," Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp.767–776, 1997.
[39] L. Guibas, D Salesin, and J. Stolfi," Constructing strongly convex approximate hulls with inaccurate primitives," Algorithmica, vol.9, pp.534–560, 1993.
[40] N. Gupta and S. Sen, "Optimal output-sensitive algorithms for constructing planar hulls in parallel," Comput. Geom. Theory Appl., vol.8, no.3, pp.151–166, 1997.
[41] R.L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," Inform. Process. Lett., vol.1, pp.73–82, 1972.
[42] R.L. Graham and F. Yao, " Finding the convex hull of a simple polygon," J. of Algorithms, vol.4, pp.324–331, 1983
[43] T. Hayashi, K. Nakano, and S. Olariu, "An $O((\log \log n)^2)$ time convex hull algorithm for sorted points on reconfigurable meshes," IEEE Trans. on Parallel and Distributed Systems, vol.9, no.12, pp.1167–1179, 1998.
[44] J.A. Holey and O.H. Ibarra, "Iterative algorithms for the planar convex hull problem on mesh connected arrays," Parallel Comput., vol.18, no.3, pp.281–296, 1992.
[45] J. Jang, M. Nigam, V.K. Prasanna, and S. Sahni, "Constant time algorithms for computational geometry on the reconfigurable mesh," IEEE Trans. Parallel and Distributed Systems, vol.8, pp.1–12, 1997.
[46] C.S. Jeong and D.T. Lee, "Parallel convex hull algorithms in 2D and 3D on mesh-connected computers," Parallel Processing for Computer Vision and Display, pp.66–76, 1989.
[47] D.G. Kirkpatrick and R. Seidel, "The ultimate planar convex hull of algorithm?" SIAM J. Comput., vol.15, pp.287–

299, 1986.
[48] D.T. Lee, "On finding the convex hull of a simple polygon," International Journal of Computer and Information Sciences, vol.12, no.2, pp.87–98, 1983.
[49] D.T. Lee and R. Jou, "Efficient parallel geometric algorithms on a mesh of trees," Proc. of the 33rd Annual Southeast Conference, pp.213–218, 1995.
[50] Z. Li and V.J. Milenkovic, "Constructing strongly convex hulls using exact or rounded arithmetic," Algorithmica, vol.8, pp.345–364, 1992.
[51] J. Matoušek, " Linear optimization queries," J. Algorithms, vol.14, pp.432–448, 1993.
[52] D. McCallum and D. Avis, " A linear algorithm for finding the convex hull of a simple polygon," Information Processing Letters, vol.9, no.5, pp.201–206, 1979.
[53] R. Miller and Q.F. Stout, "Efficient parallel convex hull algorithms," IEEE Trans. Comput., vol.C-37, no.12, pp.1605–1618, 1988.
[54] R. Miller and Q.F. Stout, "Mesh computer algorithms for computational geometry," IEEE Trans. Comput., vol.C-38, no.3, pp.321–340, 1989.
[55] K. Nakano, "A bibliography of published papers on dynamically reconfigurable architectures," Parallel Processing Letters, vol.5, pp.111–124, 1995.
[56] K. Nakano, "Computing the convex hull of a sorted set of points on a reconfigurable mesh," Parallel Algorithms and Applications, vol.8, pp.243–250, 1996.
[57] F. Nielsen and M. Yvinec, "An output-sensitive convex hull algorithm for planar objects," Comput. Geom., Theory Appl. vol.8, no.1, pp.39–65, 1998.
[58] S. Olariu, J.L. Schwing, and J. Zhang, "Time-optimal convex hull algorithms on enhanced meshes," BIT, vol.33, pp.396–410, 1993.
[59] S. Olariu, J.L. Schwing, and J. Zhang, "Fast component labeling and convex hull computation on reconfigurable meshes," Image and Vision Computing Journal, vol.11, pp.447–455, 1993.
[60] F.P. Preparata and S.J. Hong, " Convex hulls of finite sets of points in two and three dimensions," Commun. ACM, vol.20, pp.87–93, 1977.
[61] F.P. Preparata and M.L. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.
[62] D. Rappaport, "A convex hull algorithm for discs, and applications," Computational Geometry: Theory and Applications, vol.2, pp.171–187, 1992.
[63] J.H. Reif and S. Sen, "Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems," SIAM J. Comput., vol.21, no.3, pp.466–485, 1992.
[64] A. Schäffer and C. Van Wyk, "Convex hulls of piecewise-Smooth Jordan curves," J. Algorithms, vol.8, no.1, pp.66–94, 1987.
[65] R. Seidel, "A convex hull algorithm optimal for point sets in even dimensions," M.Sc. Thesis, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, 1981.
[66] R. Seidel, "Constructing higher-dimensional convex hulls at logarithmic cost per face," Proc. 18th. Annu. ACM Sympos. Theory Comput., pp.404–413, 1986.
[67] R. Seidel, "Small-dimensional linear programming and convex hulls made easy," Discrete Comput. Geom., vol.6, pp.423–434, 1991.
[68] S. Shin and T. Woo, "Finding the convex hull of simple polygon in linear time," Pattern Recognition, vol.19, no.6, pp.453–458, 1986.
[69] I. Stojmenovic, "Computational geometry on a hypercube," Proc. of 1988 International Conference on Parallel Processing, vol.3, pp.100–103, 1988.
[70] H. Wagener, "Optimal parallel hull construction for simple polygons in $O(\log \log n)$ time," Proc. 33rd Annual Symposium on Foundations of Computer Science, pp.593–599, 1992.
[71] J. Zhou, X. Deng, and P. Dymond, "A 2-D parallel convex hull algorithm with optimal communication phases," Proc. of 11th International Parallel Processing Symposium, pp.596–602, 1997.

Wei Chen received the B.A. degree in mathematics from Shanghai Marine University in 1982, and received M.E., and Ph.D. degrees from the Department of Information Engineering, Faculty of Engineering Science, Osaka University in 1991 and 1994, respectively. Since 1994 she has been working at the Department of Electrical and Computer Engineering, Nagoya Institute of Technology. She is now an associate professor of that university. Her research interests include parallel and distributed algorithms, computational geometry and graph theory. She is a member of ACM, IEEE, LA Symposium and IPSJ.

Koji Nakano received the B.E., M.E. and Ph.D degrees from Osaka University, Japan in 1987, 1989, and 1992 respectively. In 1992–1995, he was a research scientist at Advanced Research Laboratory, Hitachi Ltd. Since 1995, he has worked at Nagoya Institute of Technology, Japan. He is currently an associate professor with the Department of Electrical and Computer Engineering. His research interests includes parallel algorithms and architectures, mobile computing, computational complexity, and graph theory.

Koichi Wada graduated in 1978 from the Department of Information Engineering, Faculty of Engineering Science, Osaka University, and received his M.S. and Ph.D. degrees both from the same university in 1980 and 1983, respectively. He was a research associate at Osaka University during 1983–1984. In 1984 he joined Nagoya Institute of Technology, where he is currently a professor in the Department of Electrical and Computer Engineering. He was a visiting associate professor at University of Minnesota, Duluth and University of Wisconsin, Milwaukee during 1987–1988. His research interests include graph theory, parallel/distributed algorithms and VLSI theory. Dr. Wada is a member of IEEE, ACM, LA Symposium, Japan SIAM and IPSJ.