アルゴリズム 41－3
（1994. 9. 21）

円集合の凸包を求める効率の良い並列アルゴリズム

陳慰 和田幸一 川口喜三男

名古屋工業大学電気情報工学科

本稿では円集合の凸包を求める効率の良い並列アルゴリズムを提案する. n 個の円の集合 Sが与えられたとき, このアルゴリズムから次の２つの結果が得られる. (1) Sの凸包が $O(n/\log^{\epsilon} n)$ プロセッサを用いて $O(\log^{1+\epsilon} n)$ ($\epsilon > 0$ は任意の定数) 時間で求められる. (2) Sの凸包が $O(n \log n)$ プロセッサを用いて $O(\log n \log\log n)$ 時間で求められる. その中, (1) はコスト最適である. (1) と比べて多くのプロセッサの利用される (2) はより高速である。本研究では, 計算モデルとして $CREW PRAM$モデルが採用される.

An efficient parallel algorithm for finding the convex hull of discs

Wei Chen Koichi Wada Kimio Kawaguchi

Department of Electrical and Computer Engineering, Nagoya Institute of Technology
Showa, Nagoya 466, Japan

In this paper, we present an efficient parallel algorithm for finding the convex hull of a set of discs. Two results are yielded from the algorithm. One is that the convex hull of n discs can be computed in $O(\log^{1+\epsilon} n)$ ($\epsilon > 0$ is an arbitrary constant) time using $O(n/\log^{\epsilon} n)$ processors, and another is that the convex hull can be computed in $O(\log n \log\log n)$ time using $O(n \log n)$ processors. The first one achieves cost optimal and the second one runs faster. The computational model used is the CREW PRAM model.

1 Introduction

Given a set S of geometric objects in the plane, the convex hull of S is the smallest convex region containing the all objects of S. Much work has been done in computing the convex hull efficiently, both in serial computers and in parallel computers, especially for a set of points in the plane [1]~[7][9]. In this paper we consider the CHD problem, the problem of computing the convex hull of n discs, in which the sizes of the discs can be different. The CHD problem is theoretically interesting and also has many applications[12]. Recently, D. Rappaport [12] gives an $O(n \log n)$ sequential algorithm for the CHD problem. It is optimal since the problem for finding the convex hull of n points in the plane has an $\Omega(n \log n)$ sequential lower bound.

Algorithms for the CHD problem have also been developed in the CREW (Concurrent Read Exclusive Write) PRAM (Parallel Random Access Machine). The PRAM is a synchronous parallel model of computations. This model employs a number of processors which share a common memory. The CREW PRAM is one of the PRAM models where concurrent reads are allowed but no two processors can simultaneously write to the same memory cell. Given two parallel algorithms for the same problem one is more efficient than the other if: (1) primarily, its time-processor product is smaller, and (2) secondarily (but important), its parallel time is smaller. We call the time-processor product as the cost of the algorithm. A PRAM algorithm is said to be *cost optimal* if the cost is of the same order as the time complexity of the fastest known sequential algorithm. K.V. Wering [13] presents an algorithm which solves the CHD problem in $O(\log^2 n)$ time using $O(n)$ processors in CREW PRAM. M. Yoshimori [14] improves Wering's result. He describes a cost optimal algorithm in CREW PRAM which runs in $O(\log^2 n)$ time using $O(n/\log n)$ processors. In this paper, we present an efficient parallel algorithm for finding the convex hull of a set of discs. Two results are yielded from the algorithm. One is that the convex hull of n discs can be computed in $O(\log^{1+\epsilon} n)$ ($\epsilon > 0$ is an arbitrary constant) time using $O(n/\log^{\epsilon} n)$ processors, and another is that the convex hull can be computed in $O(\log n \log\log n)$ time using $O(n \log n)$ processors. The first one achieves cost optimal and the second one runs faster.

The paper is organized as follows. In Section 2, we give some definitions and discuss several lemmas. In Section 3, we explain the basic idea of our algorithm. In Section 4, we present the algorithm for finding the convex hull of discs.

2 Preliminaries

Let S be a set of closed planar discs, in which the radiuses of discs can be different. We say u is an arc (of circle) of disc s if u is an arc on the boundary of s. The convex hull of S, denoted as $CH(S)$, is the smallest convex region containing S. It is easily to see that the boundary of $CH(S)$ is consisting of arcs (of circles) and straight line segments connecting these arcs. We represent the convex hull of S by a sequence of arcs, that is, $CH(S) = (s_1, s_2, \ldots, s_m)$, such that the arcs appear in the boundary in clockwise. A disc $u \in S$ is said to be the leftmost disc of S, if all the discs of S are in the right of the vertical line L which passes through the leftmost point of u. Similarly, a disc $v \in S$ is said to be the rightmost disc of S, if all the discs of S are in the left of the vertical line R which passes through the rightmost point of v. Without loss of generality, in $CH(S)$, let arc s_1 be an arc of leftmost disc u and arc s_a be an arc of rightmost disc v. The straight line D passing through the centers of u and v divides each of s_1 and s_a into two arcs: s_{11},s_{12} and s_{a1}, s_{a2}, respectively. Line D divides $CH(S)$ into two parts: an upper hull, $UH(S)$, above D consisting of the arcs from s_{11} to s_{a1} inclusive, and a lower hull, $LH(S)$, below D consisting of the arcs from s_{a2} to s_{12} inclusive, in the clockwise. We discuss how to construct the upper hull only, since the lower hull can be considered in the same way. With the upper hull and the lower hull, the convex hull can be computed simply. To simplify the presentation, in this paper, we omit the floor and the ceiling operations to ensure that any constant or variable takes an integer value. In the rest of this section, we discuss several lemmas.

Lemma 1 [12] *Let S be a set of n closed planar discs. Convex hull $CH(S)$ contains at most $2n-1$ arcs.* ∎

Lemma 2 [14] *Let S be a set of n closed planar discs. Convex hull $CH(S)$ can be computed in $O(\log^2 n)$ time using $O(n/\log n)$ processors.* ∎

From Lemma 2, $CH(S)$ can be computed in $O(t)$ time using $O(n \log n/t)$ processors for any $t \geq \log^2 n$. Therefore, the following corollary holds.

Corollary 1 *Let S be a set of n closed planar discs. Convex hull $CH(S)$ can be computed in in $O(n)$ time using $O(\log n)$ processors.* ∎

Two upper hulls are said to be separated by a straight line if the line separates these two hulls into its either side. Given a sequence $A = a_1, a_2, \ldots, a_n$ of arcs (of circles), A is said to be sorted in x coordinates if for any i $(1 \leq i \leq n-1)$ there exits a vertical line such that arc a_i is in the left side of the line and a_{i+1} is in the right side of the line. For sequence A, the following lemma holds.

Lemma 3 *If sequence $A = a_1, a_2, \ldots, a_n$ of arcs (of circles) is sorted in x coordinates, the convex hull of A can be constructed in $O(\log n)$ time using $O(n/\log n)$ processors.*

Proof: First, consider a point set P which is sorted in x coordinates. Goodrich's algorithm [9] uses divide-and-conquer to compute the upper hull of P as follows. Divide P into several subsets, then construct the upper hull of each subset, finally merge these separated upper hulls into the upper hull of P. In the merge step, the basic operation is to find the upper common tangent of two separated upper hulls. See that when A is sorted in x coordinates, both divide-and-conquer and the basic operation can be used for arc set A without any problem. Therefore, by considering the points as the arcs, we can compute the upper hull of A by Goodrich's algorithm. The proof follows from the fact that Goodrich's algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors. ∎

The following corollary comes from Lemma 3.

Corollary 2 *Given m upper hulls $q_1, q_2, \ldots, q_m$ each with at most δ arcs, if any two upper hulls of them are separated by a vertical line, then the upper hull of $q_1, q_2, \ldots, q_m$ can be constructed in $O(\log(\delta m))$ time using $O(\delta m/\log(\delta m))$ processors.* ∎

3 The basic idea

In this section, we explain the basic idea of our algorithm. We construct the upper hull $UH(S)$ by divide-and-conquer. If we can partition n discs of S into several equally-sized parts such that any two parts are separated by a straight line, then we can construct $UH(S)$ simply by, first, finding the upper hull of each part, then, merging these upper hulls into $UH(S)$. But such a partition seems to be impossible. In fact, when we partition S, say, by $m-1$ vertical lines, we obtain m parts which may contain $\Theta(mn)$ arcs totally (Fig. 1). It makes the problem difficult. Therefore, in our algorithm, we avoid to partition S directly. We first, divide S into d subsets such that each subset contains n/d discs and construct the upper hull of each subset. Note that these upper hulls may intersect each other. Then, instead of partitioning S, we partition these d upper hulls into n/d separated parts by $n/d - 1$ vertical lines and find the upper hull of each part. Finally, merge their upper hull into $UH(S)$. Why our algorithm works is because all these n/d separated parts contain only $O(n)$ arcs. To show it, see that when we partition these upper hulls by vertical lines, one line increases at most d arcs. We summarize the basic idea for computing the upper hull of set S of discs as follows.

Phase 1 Divide S into d equall-sized subsets $S_1, S_2, \ldots, S_d$. For each i $(1 \leq i \leq d)$, recursively construct upper hull $UH(S_i)$, in parallel (Fig. 2 (i)).

Phase 2 Let N be the number of the arcs in $UH(S_i)$ for all i, that is, $N = \Sigma_{i=1}^{d} |UH(S_i)|$. Partition upper hulls $UH(S_1)$, $UH(S_2)$, …, $UH(S_d)$ into n/d separated parts by $n/d + 1$ vertical lines as follows (Fig. 2(ii)).

(i) Let $P = \{p \mid$ the left endpoint or the right endpoint of arc $s \in UH(S_i)$, $1 \leq i \leq d\}$. Sort P (P has $2N$ points) in x coordinate.

(iii) Denote the vertical lines passing through the 1st, d-th, $2d$-th, …,nth points of sorted P as $L_0, L_1, \ldots, L_{n/d}$, resepectively. For each i $(1 \leq i \leq d)$, partition $UH(S_i)$ into the subhulls by vertical lines $L_0, L_1, \ldots, L_{n/d}$. Obviously, each subhull consists of the contiguous arcs of $UH(S_i)$, except the leftmost and the rightmost arcs of the subhull which may be the subarcs (Note that an arc of $UH(S_j)$ may be partitioned into many subarcs by these vertical lines.) For each j $(1 \leq j \leq n/d)$, let $E_j = \{s \mid s$ is an arc (or subarc)) of $UH(S_i)$, $1 \leq i \leq d$, and s is located between line L_{j-1} and $L_j\}$.

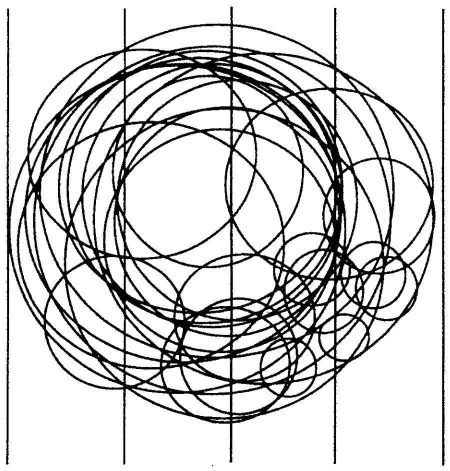


Figure 1: A set of discs difficult to separate

Phase 4 For each j $(1 \leq j \leq n/d)$, recursively construct upper hull $UH(E_j)$, in parallel.

Phase 5 Construct $UH(S)$ by merging $UH(E_j)$ for all j $(1 \leq j \leq n/d)$. ∎

4 The convex hull algorithm for discs

In this section, we present algorithm *MakeDiskUH* which computes the upper hull of a set of discs. The basic idea was explained in Section 3. To make the algorithm efficiently, we handle the problem more delicately. In the algorithm, we uses δ divide-and-conquer. What we must pay attention to is that the value δ varies with the size of the input. For example, we may use δ_i divide-and-conquer for the problem with size $n_i \leq n < n_{i+1}$, where $1 \leq i \leq k$, $n_1 = n$ and $n_{k+1} = 0$. That is, when we use δ_i divide-and-conquer to solve an problem with size $n_i \leq n < n_{i+1}$, we divide the problem into δ_i equally-sized subproblems recursively until the size of the subproblems is at most n_{i+1}, then we use δ_{i+1} divide-and-conquer to solve the subproblems. In addition, to save the cost, in the algorithm we reduce the points of P and number of these vertical lines which are used for separating the upper hulls. The algorithm is as follows.

Input: $S = (s(1), s(2), \ldots, s(n))$, a set of closed planar discs.

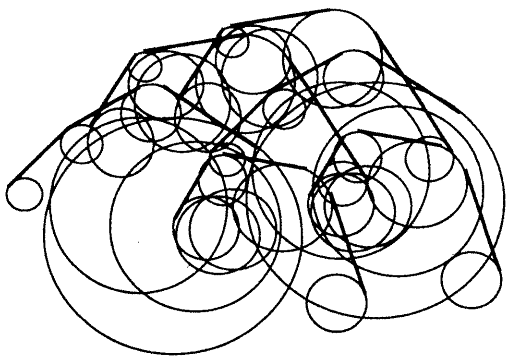
Output: $UH(S) = (u_1, u_2, \ldots, u_m)$, the upper hull of S.

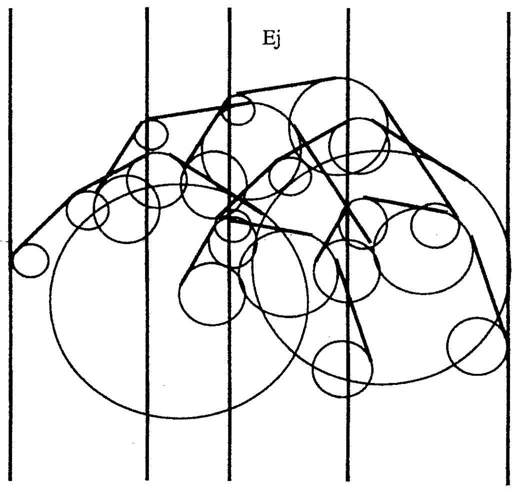
In the algorithm, let $d = \log^{1/c} n$ and let $d_i = 2^{d^i}$ $(= 2^{\log^{i/c} n}$, $1 \leq i \leq c)$. We let c be a parameter and determine it later.

Algorithm *MakeDiscsUH(S)*

Phase 1 If $n < d_1$, compute $UH(S)$ in $O(n)$ time using $O(\log n)$ processors by Corollary 1, or if $n = d_1$, compute $UH(S)$ in $O(\log^2 n)$ time using $O(n/\log n)$ processors by Lemma 2. This completes the algorithm. Else do the following steps.

Phase 2 Determine t such that $d_t < n \leq d_{t+1}$. Let $\delta_t = \min(d_t, n/d_t)$. Divide S into δ_t equall-sized subsets $S_1, S_2, \ldots, S_{\delta_t}$ such that $S_i = \{s((i-1)n/\delta_t + 1), s((i-1)n/\delta_t + 2), \ldots, s(in/\delta_t)\}$ $(1 \leq i \leq \delta_t)$. For each i, call $MakeDiscsUH(S_i)$ to construct upper hull $UH(S_i)$, in parallel.


(i) upper hulls UH(S1), UH(S2),... drawn in thick lines



(ii) The vertical lines partioning upper hulls UH(S1), UH(S2),...

Figure 2: Divide-and-conquer technique used in algorithm MakeDiscUH(S)

Phase 3 Let U_i be the sequence of the endpoints of the arcs in $UH(S_i)$ listed from the left to the right. For each i $(1 \le i \le \delta_t)$, pick out an endpoint from every $\log d_1$ endpoints, that is, pick out 1st, $(\log d_1 + 1)$-th, $(2\log d_1 + 1)$-th, ...,$(2|UH(S_i)|)$-th endpoints of U_i. Include all these endpoints to a set P, that is, $P = \{p \mid (k \log d_1 + 1)\text{-th endpoint of } U_i,\ 0 \le k \le (2|UH(S_i)| - 1)/\log d_1,\ 1 \le i \le \delta_t\}$. Let N be the number of the arcs of P. Partition upper hulls $UH(S_1)$, $UH(S_2)$, ...,$UH(S_{\delta_t})$ into $\overline{N} = N/\delta_t$ separate parts by $\overline{N} + 1$ vertical lines as follows.

(i) Compute $N = \Sigma_{i=1}^{\delta_t}(2(|UH(S_i)| - 1)/\log d_1 + 1)$.

(ii) Sort P in x coordinate.

(iii) Denote the vertical lines passing through the 1st, δ_tth, $(2\delta_t)$-th,... , $(\overline{N}\delta_t)$-th points of sorted P as L_0,L_1,... ,$L_{\overline{N}}$, resepectively. For each i, divide $UH(S_i)$ into the subhulls by L_0,L_1,... ,$L_{\overline{N}}$. Gather the arcs of the subhulls which are located between L_{j-1} and L_j into arc set E_j, i.e., for each j $(1 \le j \le N)$, $E_j = \{s \mid s \text{ is an arc (or subarc) of } UH(S_i), 1 \le i \le \delta_t, \text{ and located between line } L_{j-1} \text{ and } L_j\}$.

Phase 4 For each j $(1 \le j \le \overline{N})$, recursively construct $UH(E_j)$, in parallel.

Phase 5 Construct $UH(S)$ by merging $UH(E_j)$ $(1 \le j \le \overline{N})$. ∎

Let us analyse the complexity of algorithm MakeDiscsUH(S). Discuss each step of Phase 3. Step (i) computes N, the size of P. It can be computed in $O(\log \delta_t)$ time using $O(\delta_t / \log \delta_t)$ processors by the sum computing algorithm[8]. By Lemma 1, $UH(S_i)$ has at most $2n/\delta_t$ arcs. Each arc has 2 endpoints, therefore, $N \le 4n/\log d_1$ holds. Step (ii) sorts point set P in x coordinate. It can be executed in $O(\log(n/\log d_1))$ time using $O(n/\log d_1)$ processors by the sorting algorithm[8]. In step (iii), it is obvious that $UH(S_j)$ can be cut into the subhulls by vertical lines L_0,L_1,... ,$L_{\overline{N}}$ in $O(\log |UH(S_j)|)$ time with $O(\overline{N})$ processors. From $\overline{N}\delta_t \le 4n/\log d_1$, all $UH(S_i)$ $(1 \le i \le \delta_t)$ can be cut into the subhulls in $O(\log n)$ time using $O(n/\log d_1)$ processors. Gathering all the arcs of $UH(S_i)$ $(1 \le i \le \delta_t)$ which are located between lines L_{j-1} and L_j, we get E_j. In Phase 4, we recursively compute the upper hull of E_1,E_2, ...,$E_{\overline{N}}$, in parallel. Let us consider the size of E_j. First we see that vertical lines L_{j-1} and L_j are not the same line, i.e., L_j is of a larger x coordinate than that of L_{j-1}. Recall that L_{j-1} and L_j pass through the $j - 1 \times \delta_t$th and $j \times \delta_t$th endpoints of sorted point set P, respectively. Therefore, if L_{j-1} and L_j are the same line, there must be at least $\delta_t + 1$ endpoints of P which have the same x coordinate as that of L_{j-1}. This is a contradiction, since for each i, at most one arc of $UH(S_i)$ can have an endpoint whose x coordinate is the same as that of L_{j-1}. Therefore, L_{j-1} is really in the left of L_j. Recall that E_j is defined as a set of arcs located between line L_{j-1} and line L_j. An arc of E_j is in one of the following three cases: (1) it is cut by neither L_{j-1} nor L_j, (2) it is cut by either L_{j-1} or L_j, and (3) it is cut by both L_{j-1} and L_j. We say that an arc is not cut by L_{j-1} or L_j if its end points are exactly on L_{j-1} or L_j. By the definition of L_0,L_1, ...,$L_{\overline{N}}$, the number of arcs in (1) and (2) is at most $\log d_1 (N/\overline{N}) = \delta_t \log d_1$. The number of the arcs in (3) is at most δ_t since for each i $(1 \le i \le \delta_t)$ $UH(S_i)$ can have at most one arc which is cut by both lines L_{j-1} and L_j. Therefore, the size of E_j is at most $2\delta_t \log d_1$. Finally, by Corollary 2, all $UH(E_j)$ can be merged in to $UH(S)$ in $O(\log n)$ time using $O(n/\log n)$ processors.

Denote the running time and number of processors of the algorithm as $T(n)$ and $P(n)$, respectively. Phase 2 runs in $T(n/\delta_t)$ time using $\delta_t P(n/\delta_t)$ processors, and Phase 4 runs in $T(2\delta_t \log d_1)$ time using $\overline{N}P(2\delta_t \log d_1)$ processors, respectively. Considering together with Phase 1, the complexity of the algorithm can be summarized as following inequalites (A) and inequalites (B), where k_1 is a constant.

(A) When $\delta_t = d_t$, *i.e.*, $n/d_t \ge d_t$, the following inequalites hold.

$$T(n) \le \begin{cases} T(n/d_t) + T(2d_t \log d_1) + k_1 \log n & if\ d_t < n \le d_{t+1}\ and\ (1 \le t \le c-1) \\ k_1 \log^2 n & if n = d_1 \\ k_1 n & if n < d_1 \end{cases}$$

$$P(n) \le \begin{cases} \max\{k_1 n/\log d_1, d_t P(n/d_t), n/(d_t \log d_1) P(2d_t \log d_1)\} & if d_t < n \le d_{t+1}\ and\ 1 \le t \le c-1 \\ k_1 n/\log n & if n = d_1 \\ k_1 \log n & if n < d_1 \end{cases}$$

(B) When $\delta_t = n/d_t$, *i.e.*, $n/d_t < d_t$, the following inequalites hold.

$$T(n) \leq \begin{cases} T(d_t) + T(2(n/d_t)\log d_1) + k_1 \log n & if d_t < n \leq d_{t+1} \ and \ 1 \leq t \leq c-1 \\ k_1 \log^2 n & if n = d_1 \\ k_1 n & if n < d_1 \end{cases}$$

$$P(n) \leq \begin{cases} \max\{k_1 n/\log d_1, n/d_t P(d_t), d_t/\log d_1 P(2n\log d_1/d_t)\} & if d_t < n \leq d_{t+1} \ and \ 1 \leq t \leq c-1 \\ k_1 n/\log n & if n = d_1 \\ k_1 \log n & if n < d_1 \end{cases}$$

First, let us compute the term $T(2d_t \log d_1)$ in inequalites (A). Note $2\log d_t < d_t$ holds. From inequalites (B), there exits a constant k_2 such that the following inequalites (C) hold.

(C)

$$T(2d_t \log d_1) \leq T(d_t) + T(4\log^2 d_1) + k_2 \log d_t$$

$$P(2d_t \log d_1) \leq \max\{k_2 d_t, 2\log d_1 P(d_t), d_t/\log d_1 P(4\log^2 d_1)\}$$

Since $4\log^2 d_1 < d_1$, $T(4\log^2 d_1) \leq 4k_1 \log^2 d_1$ and $P(4\log^2 d_1) \leq 4k_1 \log\log d_1$ hold. There exits a constant k_3 such that the following inequalites (C') hold.

(C')

$$T(2d_t \log d_1) \leq T(d_t) + k_3 \log d_t + k_3 \log^2 d_1$$

$$P(2d_t \log d_1) \leq \max\{k_3 d_t, 2\log d_1 P(d_t)\}$$

When n is large enough, from inequalites (A) and (C') there exits a constant k_4 such that the following inequalites (A') hold.

(A')

$$T(n) \leq \begin{cases} T(n/d_t) + T(d_t) + k_4 \log n + k_4 \log^2 d_1 & if d_t < n \leq d_{t+1} \ and \ 1 \leq t \leq c-1 \\ k_4 \log^2 n & if n = d_1 \end{cases}$$

$$P(n) \leq \begin{cases} \max\{k_4 n/\log d_1, 2n/d_t P(d_t), d_t P(n/d_t)\} & if d_t < n \leq d_{t+1} and 1 \leq t \leq c-1 \\ k_4 n/\log n & if n = d_1 \end{cases}$$

In the following, we use (A') to show that $T(d_t) \leq A_t \log d_{t+1}$ and $P(d_t) \leq B_t d_t/\log d_1$ hold for all $1 \leq t \leq c$, where $A_t = 1 + 2(t-1)k_4$ and $B_t = 2^{t-1}k_4$. Recall $d = \log^{1/c} n$, $d_t = 2^{d^t}$. Obviously, they hold for $t = 1$, where $A_1 = k_4$ and $B_1 = k_4$. Assume that $T(d_i) \leq A_i \log d_{i+1}$ and $P(d_i) \leq B_i d_i/\log d_1$ hold. The following inequalites hold for $d_i < n \leq d_{i+1}$.

$$\begin{cases} T(n) \leq T(n/d_i) + T(d_i) + k_4 \log n + k_4 \log^2 d_1 \\ T(d_i) \leq A_i \log d_{i+1} \end{cases}$$

$$\begin{cases} P(n) \leq \max\{k_4 n/\log d_1, d_i P(n/d_i), 2n/d_i P(d_i)\} \\ P(d_i) \leq B_i d_i/\log d_1 \end{cases}$$

To compute $T(d_{i+1})$, we iterate the recursive inequality of $T(n)$ from $n = d_{i+1}$ down to $n = d_i$. Assuming that h is the times for iterating recurrence, $d_{i+1}/(d_i)^h = d_i$ hold, i.e., $h = \log d_1 - 1$. Therefore, $T(d_{i+1}) \leq A_i \log d_{i+1} + (\log d_1 - 1)((A_i + k_4)\log d_{i+1} + k_4 \log^2 d_1) \leq (A_i + 2k + 4)\log d_{i+2}$. Let $A_{i+1} = A_i + 2k_4 = 1 + 2ik_4$. We obtain the inequality $T(d_{i+1}) \leq A_{i+1} \log d_{i+2}$. In the similar way, we can prove $P(d_{i+1}) \leq B_{i+1} d_{i+1}/\log d_1$, where $B_{i+1} = 2B_i = 2^i k_4$. Therefore, $T(n) = T(d_c) = (1 + 2(c-1)k_4)\log^{1+1/c} n$ and $P(n) = P(d_c) = 2^{c-1}k_4 n/\log^{1/c} n$ hold. If let $c = \log\log n$, $T(n) = O(\log n \log\log n)$ and $P(n) = n\log n$ hold. If let c be a constant, $T(n) = O(\log^{1+1/c} n)$ and $P(n) = O(n/\log^{1/c} n)$ hold. Therefore, we have the following theorems.

Theorem 1 *The convex hull of a set of n discs can be computed in $O(\log^{1+\epsilon} n)$ ($\epsilon > 0$ can be an arbitrary constant) time using $O(n/\log^{\epsilon} n)$ processors in CREW PRAM.*

Theorem 2 *The convex hull of a set of n discs can be computed in $O(\log n \log\log n)$ time using $O(n\log n)$ processors in CREW PRAM.*

5 Conclusion

We have shown an efficient algorithm which solves the convex hull problem for a set of n discs in the plane. The algorithm yields two results: one is that the convex hull of n discs can be computed in $O(\log^{1+\epsilon} n)$ ($\epsilon > 0$ is an arbitrary constant) time using $O(n/\log^{\epsilon} n)$ processors, and another is that the convex hull can be computed in $O(\log n \log\log n)$ time using $O(n \log n)$ processors. The first one achieves cost optimal and the second one runs faster. The computational model used is the *CREWPRAM* model. Whether the probelm can be solved in $O(\log n)$ time using $O(n)$ processors in CREW PRAM still remains to be an open problem.

References

(1) A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap: Parallel computational geometry. Algorithmica, vol.3, pp.293-327, 1988.

(2) M. J. Atallah and M. T. Goodrich: Efficient parallel solutions to some geometric problems. Journal of Parallel and Distributed Computing, vol.3, pp.492-507, 1986.

(3) M. J. Atallah and M. T. Goodrich: Parallel algorithms for some functions of two convex polygons. Algorithmica, vol.3, pp.535-548, 1988.

(4) W. Chen, K. Nakano, T. Masuzawa and N. Tokura: Optimal Parallel Algorithms for Finding the Convex Hull of a Sorted Point Set, Trans.IEICE, vol.J74-D-I, no.12, pp.814-825, 1991.

(5) W. Chen,K. Nakano, T. Masuzawa and N. Tokura: Optimal Parallel Algorithm for Computing the Prefix Convex Hulls of A Sorted Points Set, Tech.Rep. IPSJ, vol.AL28-10, pp.77-84, 1992.

(6) R. Cole and M. T. Goodrich: Optimal parallel algorithms for polygon and point set problems. In Proc. of the 4th Annual ACM Symposium on Computational Geometry,1988.

(7) P-O. Fjällström, J. Katajainen, C. Levcopoulos and O. Petersson: A sublogarithmic convex hull algorithm. Bit, vol.30, pp.378-384,1990.

(8) A. Gibbons and W. Rytter: Efficient parallel algorithms. Cambridge University Press, 1988.

(9) M. T. Goodrich: Finding the convex hull of a sorted point set in parallel. Information Processing Letters, vol.26, pp.173-179, 1987.

(10) M. H. Overmars and J. V. Leeuwen: Maintenance of configurations in the plane. J. Comput. System Sci. vol.23, pp.166-204,1981.

(11) F.P.Preparata and M.L.Shamos; Computational geometry: an introduction. Springer-Verlag,1985.

(12) D. Rappaport: A convex hull algorithm for discs, and applications. Computational Geometry: Theory and Applications no.1, pp.171-187, 1992.

(13) K. V. Weringh: Algorithms for the Voronoi diagram of a set of disks. M.Sc.thesis, Department of Computing and Information Science, Queen's University, Kingston, Ontario (1990).

(14) M. Yoshimori: An Efficient Convex Hull Parallel Algorithm for discs. M.Sc.thesis, Department of Computing and Information Sciences, Faculty of Engineering Science, Osaka University, (1994).