PAPER *Special Section on Discrete Mathematics and Its Applications*

Polynomially Fast Parallel Algorithms for Some *P*-Complete Problems*

Carla Denise CASTANHO[†a)], **Wei CHEN**[††], **Koichi WADA**[††], *and* **Akihiro FUJIWARA**[†††], *Regular Members*

SUMMARY P-complete problems seem to have no parallel algorithm which runs in polylogarithmic time using a polynomial number of processors. A P-complete problem is in the class EP (Efficient and Polynomially fast) if and only if there exists a cost optimal algorithm to solve it in $T(n) = O(t(n)^\epsilon)$ ($\epsilon < 1$) using $P(n)$ processors such that $T(n) \times P(n) = O(t(n))$, where $t(n)$ is the time complexity of the fastest sequential algorithm which solves the problem. The goal of our research is to find EP parallel algorithms for some P-complete problems. In this paper first we consider the convex layers problem. We give an algorithm for computing the convex layers of a set S of n points in the plane. Let k be the number of the convex layers of S. When $1 \le k \le n^{\frac{\epsilon}{2}}$ ($0 \le \epsilon < 1$) our algorithm runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, and it is cost optimal. Next, we consider the envelope layers problem of a set S of n line segments in the plane. Let k be the number of the envelope layers of S. When $1 \le k \le n^{\frac{\epsilon}{2}}$ ($0 \le \epsilon < 1$), we propose an algorithm for computing the envelope layers of S in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, and $\alpha(n)$ is the functional inverse of Ackermann's function which grows extremely slowly. The computational model we use in this paper is the *CREW-PRAM*. Our first algorithm, for the convex layers problem, belongs to EP, and the second one, for the envelope layers problem, belongs to the class EP if a small factor of $\log n$ is ignored.

key words: *parallel algorithm, P-complete problems, convex layers problem, envelope layers problem*

1. Introduction

In parallel computational theory, one of the primary measures of parallel complexity is the class NC. Let n be the input size of a problem. The problem is said to be in the class NC if there exists an algorithm which solves the problem in polylogarithmic time using polynomial number of processors. Many problems in the class P, which is the class of problems solvable in polynomial time sequentially, are also in the class NC.

On the other hand, there are some problems in P which do not seem to admit parallelization readily. These problems, which we refer to as hardly parallelizable ones, form the class so-called P-complete problems. In other words, the class of P-complete problems consists of the most likely candidates of P that are not in NC. If a parallel algorithm which runs in polylogarithmic time using a polynomial number of processors could be found for at least one P-complete problem then a similar solution would exist for any other one.

However, polylogarithmic time complexity is not so important when considering practical parallel computation. Actually, the number of processors is usually small in comparison with the size of a problem. Thus, in practice cost optimality turns to be the most important measure for parallel algorithms. The cost of a parallel algorithm is defined as the product of the running time and the number of processors required by the algorithm. A parallel algorithm is called cost optimal if its cost is of the same order as the time complexity of the fastest known sequential algorithm for the same problem. In other words, the cost optimal parallel algorithm achieves optimal speedup, which is equal to the number of processors.

Therefore, one way to parallelize P-complete problems is to find a cost optimal parallel algorithm. Assume that $O(n^k)$ is the upper bound of the fastest known sequential time complexity for a P-complete problem A. It seems that the problem A has no parallel algorithm which runs in polylogarithmic time since A is P-complete. However, the problem A may have a parallel algorithm which runs in $O(n^{k-\epsilon})$ time using n^ϵ processors for some constant ϵ, $0 < \epsilon < k$. It means that, in practice, the algorithm achieves optimal speedup if the number of processors is not larger than n^ϵ.

Kruskal et al. [9] proposed the class EP. The EP means "Efficient and Polynomially fast," and a problem is in EP if and only if there exists a cost optimal algorithm to solve it in $T(n) = O(t(n)^\epsilon)$ ($\epsilon < 1$) using $P(n)$ processors such that $T(n) \times P(n) = O(t(n))$, where $t(n)$ is the time complexity of the fastest sequen-

Manuscript received September 4, 2000.
Manuscript revised November 22, 2000.

†The author is a Ph.D. student at Nagoya Institute of Technology, Nagoya-shi, 466-8555 Japan.
††The authors are with the Department of Electrical and Computer Engineering, Nagoya Institute Techonology, Nagoya-shi, 466-8555 Japan.
†††The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
a) E-mail: caca@phaser.elcom.nitech.ac.jp

*This research was partly supported by the Hori Information Promotion Foundation (1999), a Scientific Research Grant-In-Aid from the Ministry of Education, Science, Sports and Culture of Japan, under grant No. 10205209, and the Sasakawa Scientific Research Grant from the Japan Science Society.

tial algorithm which solves the problem.

In this paper, we consider EP algorithms for two famous P-complete problems, the convex layers problem and the envelope layers problem, in the *CREW-PRAM* computational model.

First we consider the convex layers problem which requires to partition the input set S of n points in the Euclidean plane into a set of convex polygons defined as follows: (i) compute the convex hull of S and remove its points from S, (ii) then repeat instruction (i) until no point remains in S. This problem is a natural extension of the convex hull problem.

Chazelle [2] proposed an optimal sequential algorithm for the convex layers problem which runs in $O(n \log n)$ time. The sequential algorithm is time optimal because the computation of a convex hull, which is the first hull of the convex layers, requires $\Omega(n \log n)$ time [13]. Dessmark et al. [4] proved that the problem is P-complete. In [5] Fujiwara et al. considered the problem under a very strong constraint. They proved that if all points of S lie on d horizontal lines, when $d \leq n^{\delta}$ $(0 < \delta \leq \frac{1}{2})$ the problem is still P-complete. They proposed an EP algorithm for the problem which runs in $O(\frac{n \log n}{p})$ time using p processors if $d \leq n^{\epsilon}$ $(0 < \epsilon \leq \frac{1}{2})$ and $1 \leq p \leq n^{\epsilon}$ in the *EREW-PRAM*. That is, to achieve cost optimality, there must be d horizontal lines such that each line must pass through more than $n^{\frac{1}{2}}$ points in average, what is unlikely in most cases. Besides, the parameter d does not represent the substantial complexity of the problem.

In this paper we present a new EP parallel algorithm for computing the convex layers of a set S of n points. Let k be the number of the convex layers of S. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ our algorithm runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, in the *CREW-PRAM*, and it is cost optimal.

As the number of layers never exceeds the number of horizontal lines in which the input points lie, i.e. $k \leq d$, the problem considered by Fujiwara et al. [5] which have been proved to be P-complete when $d \leq n^{\delta}$ $(0 < \delta \leq \frac{1}{2})$, can be reduced to the convex layers problem we consider here. Thus, when $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ the problem of finding the convex layers of S is also P-complete.

The second problem considered here is the envelope layers problem. The envelope layers of a set of (opaque) line segments are analogous to the convex layers of a set of points, with convex hulls replaced by upper envelopes. The upper envelope of a set of line segments in the plane is the collection of segment portions visible from the point $(0, +\infty)$. To find the envelope layers, we repeatedly compute the upper envelope of the set and discard the segments that appear on it (if any piece of a segment appears on the envelope, we discard the whole segment). The envelope layers problem is to label each segment with the iteration number at which it appears on the envelope.

Let S be a set of n (opaque) line segments in the plane. Hershberger [7] gave an $O(n\alpha(n) \log^2 n)$ algorithm for computing the envelope layers of S, where $\alpha(n)$ is the functional inverse of Ackermann's function which grows extremely slowly. Hershberger [7] also proved that the problem of finding envelope layers is P-complete.

Here, we also give an algorithm for the envelope layers problem. Let k be the number of envelope layers of S. When $0 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ our algorithm runs in $O(\frac{n\alpha(n) \log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, in the *CREW-PRAM*. If we ignore a factor of $\log n$ our algorithm belongs to the class EP.

The problem of finding the envelope layers of lines, which is dual to the convex layers problem, can be reduced to the problem of finding the envelope layers of line segments. Thus, when $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ the problem of computing envelope layers of S is also P-complete.

This paper is organized as follows. Section 2 states the definitions of the problems. In Sect. 3 we give a cost optimal parallel algorithm for constructing the convex layers. Section 4 shows an algorithm for computing the envelope layers. Some conclusions are given in Sect. 5.

2. Preliminaries

Definition 1 (Convex hull of points): Let S be a set of n points in the Euclidean plane. The convex hull of S, denoted as $CH(S) = (p_1, p_2, \ldots, p_m)$ $(m \leq n)$ where p_1 is the rightmost vertex of $CH(S)$, is the smallest convex polygon that contains all the points of S. □

Definition 2 (Convex layers problem): Let S be a set of n points in the Euclidean plane. The convex layers of S, denoted as $CL(S)$, consists of a sequence of convex hulls, $(CL_1(S), CL_2(S), \ldots, CL_k(S))$ $(1 \leq k \leq n)$, which satisfies the following two conditions:
(1) $P(CL_1(S)) \cup P(CL_2(S)) \cup \ldots \cup P(CL_k(S)) = S$, where $P(CL_i(S))$ $(1 \leq i \leq k)$ denotes the set of the vertices of $CL_i(S)$;
(2) Each $CL_i(S)$ $(1 \leq i \leq k)$ is a convex hull of a set of points $P(CL_i(S)) \cup P(CL_{i+1}(S)) \cup \ldots \cup P(CL_k(S))$, which is referred to as the ith convex layer of S. □

The size of $CL(S)$ is defined to be the total number of the vertices in $CL_1(S), CL_2(S), \ldots, CL_k(S)$, which is obviously $O(n)$.

Definition 3 (Upper envelope): Let S be a set of n (opaque) line segments in the plane. The upper envelope of S is the collection of segment portions visible from the point $(0, +\infty)$. □

The size of an upper envelope is defined to be the number of distinct pieces of segments that appear on it.

The size of the upper envelope of a set of n line segments is $\Theta(n\alpha(n))$, where $\alpha(n)$ is the functional inverse of Ackermann's function which grows extremely slowly [6].

Definition 4 (Envelope layers problem): Let S be a set of n (opaque) line segments in the plane. The envelope layers of S, denoted as $EL(S)$, consists of a sequence of upper envelopes, $(EL_1(S), EL_2(S), \ldots, EL_k(S))$ $(1 \le k \le n)$, which satisfies the following two conditions:
(1) $L(EL_1(S)) \cup L(EL_2(S)) \cup \ldots \cup L(EL_k(S)) = S$, where $L(EL_i(S))$ $(1 \le i \le k)$ denotes the set of the line segments that appear on $EL_i(S)$;
(2) Each $EL_i(S)$ $(1 \le i \le k)$ is the upper envelope of a set of line segments $L(EL_i(S)) \cup L(EL_{i+1}(S)) \cup \ldots \cup L(EL_k(S))$, which is called as the ith envelope layer of S. □

The size of $EL(S)$ is defined to be the total size of $EL_1(S), EL_2(S), \ldots, EL_k(S)$, which is $O(n\alpha(n))$ [7].

Let S_1 and S_2 be two sets of line segments. We say that sets S_1 and S_2 are *separated* if the x-coordinate of the rightmost endpoint of S_1 is smaller than or equal to the x-coordinate of the leftmost endpoint of S_2.

3. The Convex Layers Problem

3.1 The 2-3 Tree for Supporting Operations on the Convex Layers

When solving the convex layers problem we use a balanced tree, say a 2-3 tree, to support the operations on the convex layers. A 2-3 tree is a rooted tree in which each internal node has two or three children. Every path from the root to a leaf is of same length. Therefore, if the number of leaves is n, the height of the tree is $\Theta(\log n)$.

Let P be a convex polygon of n vertices, and let $(p_1, p_2, \ldots, p_n)$ denote the sequence of vertices of P listed counterclockwise, where p_1 is the rightmost vertex of P. Let e_i $(1 \le i \le n)$ be the edge of P whose endpoints are p_i and p_{i+1} $(p_{n+1} = p_1)$, and let s_i denote the slope of edge e_i. Let $p_r = p_1$ and p_l denote the rightmost and leftmost points of P, respectively. The line passing through points p_r and p_l divides P into two parts: the upper part $UP(P) = (p_1, p_2, \ldots, p_l)$ and the lower part $LP(P) = (p_l, p_{l+1}, \ldots, p_n, p_1)$. We store the upper part of P in a 2-3 tree as follows (Fig. 1). The lower part is stored in another 2-3 tree similarly. The pairs $(e_1, s_1), (e_2, s_2), \ldots, (e_{l-1}, s_{l-1})$ are placed at the leaves in a left-to-right order. Since P is convex, it holds that $s_i < s_{i+1}$. Each internal node v holds three data items, $L[v]$, $M[v]$ and $R[v]$, where $L[v]$, $M[v]$ and $R[v]$ are the pairs (e_x, s_x), (e_y, s_y) and (e_z, s_z) with the largest slope s_x, s_y, and s_z, stored in the first (leftmost), the second, and the third (rightmost) subtrees of v, respectively. If v does no have the third subtree then $R[v]$ is empty.

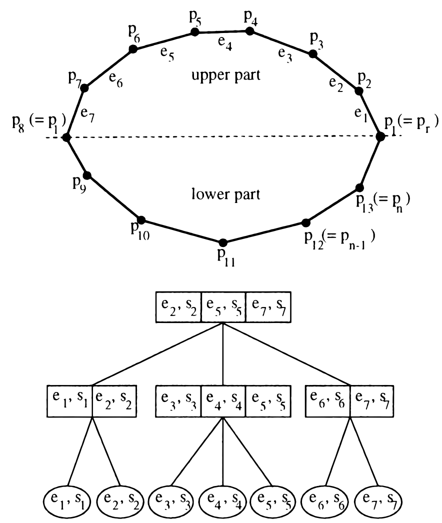

Fig. 1 A convex polygon and the 2-3 tree storing its upper part.

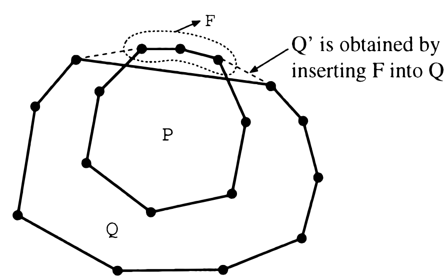

Fig. 2 Inserting a sequence F of contiguous edges in a convex layer P into a convex layer Q.

In our algorithm for finding convex layers, the following operations are performed on the 2-3 tree: (I) storing a convex layer whose vertices are taken from the points of S, (II) searching for an edge of a convex layer, (III) finding the common tangents of two convex layers, (IV) deleting a set of contiguous edges from a convex layer, and finally (V) given two convex polygons P and Q and a set F of contiguous edges in P, inserting F into Q, where P, Q and F satisfy the following condition: assuming that Q' is the polygon consisting of the vertices of Q and F, then Q' is convex, and the vertices of F lie contiguously in Q' (Fig. 2).

For a convex layer with n elements, operation (I) can be done in $O(\log n)$ time using n processors, and operation (II) can be done in $O(\log n)$ time using a single processor as well [8]. For two convex polygons with n vertices each one, operation (III) can be done in $O(\log n)$ time using a single processor [10]. In the following we consider operations (IV) and (V).

Let us consider the following two lemmas which are necessary to describe operations (IV) and (V).

Lemma 1: Let T_1 and T_2 be two 2-3 trees of height h_1 and h_2 respectively, where $(s_1, s_2, \ldots, s_f)$ and $(t_1, t_2, \ldots, t_g)$ are the elements stored in the leaves of T_1 and T_2 respectively, and $s_1 \le s_2 \le$

$\ldots \leq s_f \leq t_1 \leq t_2, \leq \ldots \leq t_g$. Constructing a 2-3 tree T storing $(s_1, s_2, \ldots, s_f, t_1, t_2, \ldots, t_g)$, denoted by $merge((T_1, T_2; h_1, h_2) : T)$, can be done in $O(\max\{h_1, h_2\} - \min\{h_1, h_2\} + 1)$ sequentially.

Proof: Without loss of generality we assume $h_1 \geq h_2$. From the condition that $s_1 \leq \ldots \leq s_f \leq t_1 \leq \ldots \leq t_g$, T can be constructed by inserting all leaves of T_2 to the right side of the rightmost leaf of T_1. This can be easily done by considering the root of T_2 as a rightmost leaf to be inserted into T_1 at height h_2. Such insertion can be done in $O(h_2 - h_1 + 1)$ time sequentially [1]. Thus, T can be constructed in $O(\max\{h_1, h_2\} - \min\{h_1, h_2\} + 1)$ time sequentially. □

Lemma 2: Let T be a 2-3 tree of height h and let $(t_1, t_2 \ldots, t_l)$ be a sequence of contiguous leaves in T. the operation of constructing a 2-3 tree T' storing $(t_1, t_2 \ldots, t_l)$, denoted by $build((T, h, (t_1, t_2 \ldots, t_l)) : T')$, can be done in $O(h)$ time sequentially.

Proof: Let $FR = (T_1, T_2, \ldots, T_k)$ $(k \leq l)$ be the forest consisting of the largest subtrees of T whose leaves store only the elements of $(t_1, t_2 \ldots, t_l)$. FR can be constructed from T in $O(h)$ time sequentially and it holds that $k = O(h)$. T' can be obtained by constructing the 2-3 tree of FR as follows.

Let T_g $(1 \leq g \leq k)$ be the highest tree in FR. From the fact that $(t_1, t_2 \ldots, t_l)$ are contiguous leaves in T then $h_1 \leq h_2 \leq \ldots \leq h_{g-1} < h_g$ and $h_g \geq h_{g+1} \geq \ldots \geq h_k$ where h_i denotes the height of tree T_i in FR $(1 \leq i \leq k)$. T_g divides FR into two sub-forests, say FR' and FR'' such that the trees in FR' and FR'' are listed, respectively, in an increasing and decreasing order of height from left to right (Fig. 3). If $g = 1$ let FR' be empty and let $FR'' = FR$, otherwise let $FR' = (T_1, T_2, \ldots, T_{g-1})$ and $FR'' = (T_g, T_{g+1}, \ldots, T_k)$.

We can obtain T' in two steps:

(i) First we construct the 2-3 tree of FR'', that is, merge $T_g, T_{g+1}, \ldots, T_k$ into T_g as follows. Sequentially from $i = k - 1$ down to g we perform the operation $merge((T_i, T_{i+1}; h_i, h_{i+1}) : T_i)$.

(ii) Then we merge the trees of FR' and T_g as follows. Sequentially from $i = 1$ to $g - 1$ we perform the operation $merge((T_i, T_{i+1}; h_i, h_{i+1}) : T_{i+1})$, and let T' be T_g.

From Lemma 1 step (i) takes $O((h_g - h_{g+1} + 1) + (h_{g+1} - h_{g+2} + 1) + \ldots + (h_{k-1} - h_k + 1)) = O(\max\{h_g - h_k + 1, k - g + 1\})$ time, and step (ii) takes $O((h_2 - h_1 + 1) + (h_3 - h_2 + 1) + \ldots + (h_g - h_{g-1} + 1)) = O(\max\{h_g - h_1 + 1, g\})$ time. Thus, the total time to construct T' is $O((\max\{h_g - h_k + 1, k - g + 1\}) + (\max\{h_g - h_1 + 1, g\})) = O(\max\{h_g - \min\{h_1, h_k\} + 1, k + 1\}) = O(h)$. □

Now let us consider operation (IV). We show that deleting a set of contiguous edges from a convex layer with n vertices can be done in $O(\log n)$ time sequentially.

Contiguous edges of a convex layer are stored in the leaves of a 2-3 tree contiguously. Let $P = (p_1, p_2, \ldots, p_n)$ be a convex layer and $G = (g_1, g_2, \ldots, g_m)$ $(1 \leq m < n)$ be a set of contiguous vertices in P listed counterclockwise, such that g and g' are the vertices of P located immediately before and after G, respectively. Let $BT_1[P]$ and $BT_2[P]$ denote, respectively, the two 2-3 trees storing $UP(P)$ and $LP(P)$. For simplicity let us assume that all edges of G belong to the upper part of P, $UP(P)$. If we delete the edges of G from P, $BT_1[P]$ has to be updated, that is, all the edges (g_i, g_{i+1}) $(1 \leq i \leq m-1)$ of G must be deleted from $BT_1[P]$, and a new edge connecting g_1 to g_m has to be inserted in $BT_1[P]$.

We update $BT_1[P]$ as follows. $BT_1[P] - H$ consists of at most two parts of consecutive leaves of $BT_1[P]$. Let G_l and G_r be the consecutive leaves of $BT_1[P]$ located to the left and right, respectively, of the leaves storing the edges of G (Fig. 4). Then we construct the 2-3 trees of G_l and G_r by $build((BT_1[P], h(BT_1[P]), G_l) : T_l)$ and $build((BT_1[P], h(BT_1[P]), G_r) : T_r)$ respectively, where $h(BT_1[P])$ denotes the height of $BT_1[P]$. This can be done in $O(\log n)$ time from Lemma 2. Finally by $merge((T_l, T_r; h(T_l), h(T_r)) : BT_1[P])$ we merge T_l and T_r in $O(\log n)$ time from Lemma 1, where $h(T_l)$ and $h(T_r)$ denotes the height of T_l and T_r respectively.

The new edge connecting g_1 to g_m can be inserted into $BT_1[P]$ in $O(\log n)$ time sequentially [1]. Therefore, deleting G from P can be done in $O(\log n)$ time sequentially.

Now let us consider operation (V) which inserts a set F of contiguous edges in a convex layer P into a convex layer Q. For simplicity we assume that the

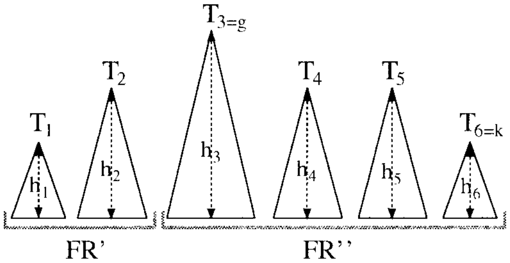

Fig. 3 The forest FR.

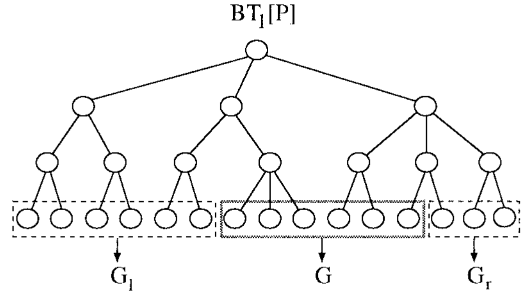

Fig. 4 G, G_l and G_r in $BT_1[P]$.

edges of F belong to $UP(P)$, and will be inserted into $UP(Q)$. Let $BT_1[P]$ and $BT_1[Q]$ be the two 2-3 trees storing $UP(P)$ and $UP(Q)$ respectively. First we construct the 2-3 tree of F, denoted as $T[F]$, by $build((BT_1[P], h(BT_1[P]), F) : T[F])$ in $O(\log n)$ time from Lemma 2. Then we delete F from $BT_1[P]$ in $O(\log n)$ time by using operation (IV) explained above.

If F has to be inserted before the leftmost leaf of $BT_1[Q]$ then we directly perform $merge((T[F], BT_1[Q]; h(T[F]), h(BT_1[Q])) : BT_1[Q])$, where $h(T[F])$ and $h(BT_1[P])$ denotes the height of $T[F]$ and $BT_1[P]$ respectively. Otherwise, if F has to be inserted after the rightmost leaf of $BT_1[Q]$ then we directly perform $merge((BT_1[Q], T[F]; h(BT_1[Q]), h(T[F])) : BT_1[Q])$. In both cases the merging process can be done in $O(\log n)$ time from Lemma 1.

Now assume that F has to be inserted between two consecutive leaves of $BT_1[Q]$, say e and e'. Leaves e and e' divide $BT_1[Q]$ into two parts of contiguous leaves. Let Q_l be the consecutive leaves of $BT_1[Q]$ located to the left of e inclusive, and Q_r be the consecutive leaves of $BT_1[Q]$ located to the right of e' inclusive. Then we construct the new $BT_1[Q]$ containing F in three steps as follows.

(i) We construct the 2-3 trees of Q_l and Q_r by $build((BT_1[Q], h(BT_1[Q]), Q_l) : T_l)$ and $build((BT_1[Q], h(BT_1[Q]), Q_r) : T_r)$ respectively, where $h(BT_1[Q])$ denotes the height of $BT_1[Q]$.

(ii) By $merge((T_l, T[F]; h(T_l), h(T[F])) : T_l)$ we merge T_l and $T[F]$, where $h(T_l)$ and $h(T[F])$ denote the height of T_l and $T[F]$ respectively.

(iii) Finally we obtain the new $BT_1[Q]$ by merging T_l (updated in (ii)) and T_r by $merge((T_l, T_r; h(T_l), h(T_r)) : BT_1[Q])$.

Step (i) can be done in $O(\log n)$ time from Lemma 2. Steps (ii) and (iii) can be done in $O(\log n)$ time from Lemma 1. Thus, operation (V) can be done in $O(\log n)$ time sequentially.

3.2 Outline of the Algorithm

Let S be a set of n points in the Euclidean plane. We sort S by its x-coordinates, which can be done in $O(\log n)$ time using n processors [8]. Let k be the number of the convex layers of S and ϵ $(0 \le \epsilon < 1)$ be a constant. When $1 \le k \le n^{\frac{\epsilon}{2}}$ holds, the following algorithm computes the set of convex layers of S in $O(\frac{n \log n}{p})$ time using p processors, $1 \le p \le n^{\frac{1-\epsilon}{2}}$.

Algorithm *ComputeCL*(S)

[Input] A set $S = (p_1, p_2, \ldots, p_n)$ of n points in the Euclidean plane sorted by their x-coordinates.
[Output] A set $CL(S) = (CL_1(S), CL_2(S), \ldots, CL_k(S))$ $(1 \le k \le n)$ of the convex layers of S, where $CL_i(S)$ $(1 \le i \le k)$ is the ith convex layer of S.

(Step 1) Divide S into $S_1, S_2, \ldots, S_{n^{\frac{1-\epsilon}{2}}}$ subsets such that S_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ contains $n^{\frac{1+\epsilon}{2}}$ points and the x-coordinate of any point of S_i is less than the x-coordinate of any point of S_{i+1}.
(Step 2) In parallel, for each subset S_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ compute the convex layers of S_i, denoted as $CL(S_i) = (CL_1(S_i), CL_2(S_i), \ldots, CL_{k_i}(S_i))$ by Chazelle's sequential algorithm, where k_i is the number of layers in S_i. Store each $CL_j(S_i)$ $(1 \le j \le k_i)$ $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ in two 2-3 trees, denoted as $BT_1[CL_j(S_i)]$ and $BT_2[CL_j(S_i)]$, as stated in Sect. 3.1, that is, $BT_1[CL_j(S_i)]$ for the upper part and $BT_2[CL_j(S_i)]$ for the lower part of the convex layer.
(Step 3) Let $S' = S$ and for each i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ let $S'_i = S_i$. Let $x = 1$. While S' is not empty, find $CL_x(S)$, that is, the xth convex layer of S, repeatedly as follows.

(a). Find the convex hull of the outermost layers of $S'_1, S'_2, \ldots, S'_{n^{\frac{1-\epsilon}{2}}}$, which obviously is $CL_x(S)$.

(b). Revise S' and S'_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$.

(i). Delete the vertices of $CL_x(S)$ from S'.

(ii). For each i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ in parallel, delete the vertices of $CL_x(S)$ from S'_i. Then reconstruct the convex layers of S'_i on the 2-3 trees.

(c). Set $x = x + 1$. While S' is not empty return to (a).

□

Obviously, in the above algorithm, Step 1 can be executed in $O(1)$ time using n processors if S is stored in an array. In Step 2, the size of S_i is $n^{\frac{1+\epsilon}{2}}$ $(1 \le i \le n^{\frac{1-\epsilon}{2}})$. The convex layers of each subset S_i, $CL(S_i) = (CL_1(S_i), CL_2(S_i), \ldots, CL_{k_i}(S_i))$, can be computed by the known Chazelle's sequential algorithm [2] in $O(n^{\frac{1+\epsilon}{2}} \log n)$ time. Therefore, all the convex layers of $S_1, S_2, \ldots, S_{n^{\frac{1-\epsilon}{2}}}$ can be computed in $O(n^{\frac{1+\epsilon}{2}} \log n)$ time using $n^{\frac{1-\epsilon}{2}}$ processors. Then we store each $CL_j(S_i)$ $(1 \le j \le k_i)$ $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ in two balanced trees $BT_1[CL_j(S_i)]$ and $BT_2[CL_j(S_i)]$, that is, $BT_1[CL_j(S_i)]$ for the upper part and $BT_2[CL_j(S_i)]$ for the lower part of the convex layer. This can be done in $O(\log |S_i|)$ time using $|S_i|$ processors as stated in Sect. 2. That is, all the layers of all S_i can be stored in $O(\log n)$ time using n processors.

Now let us consider Step 3. The convex layers of S, $CL(S) = (CL_1(S), CL_2(S), \ldots, CL_k(S))$ $(1 \le k \le n)$, is constructed repeatedly in Step 3. After $CL_x(S)$ $(1 \le x \le k-1)$ is computed in Step 3(a), S' and S'_i $(1 \le i \le n^{\frac{1+\epsilon}{2}})$ must be revised in Step 3(b)(i) and Step 3(b)(ii), respectively, by deleting its points that appear as vertices of $CL_x(S)$, and then the convex layers of each revised S'_i must be reconstructed in Step 3(b)(ii).

Let S be stored in an array. Step 3(b)(i) can be easily done in $O(1)$ time using n processors. In Sect. 3.3 we show that Step 3(a) can be done in $O(\log n)$ time using $O(n^{1-\epsilon})$ processors. In Sect. 3.4 we show that Step 3(b)(ii) can be done in $O(k \log n)$ time using $O(n^{\frac{1-\epsilon}{2}})$ processors. Since the instructions of Step 3 are repeated k times, where k is the number of convex layers of S, Step 3 takes totally $O(k^2 \log n)$ time using $O(n^{\frac{1-\epsilon}{2}})$ processors. The whole algorithm $ComputeCL(S)$ can be executed in $O(\max(n^{\frac{1+\epsilon}{2}} \log n, k^2 \log n))$ time using $n^{\frac{1-\epsilon}{2}}$ processors. Thus by using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, it takes $O(\max(\frac{n \log n}{p}, \frac{n^{\frac{1-\epsilon}{2}} k^2 \log n}{p}))$ time. Therefore, when $1 \le k \le n^{\frac{\epsilon}{2}}$ $(0 \le \epsilon < 1)$, the algorithm $ComputeCL(S)$ is cost optimal and runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$.

Theorem 1: Let ϵ $(0 \le \epsilon < 1)$ be a constant. Algorithm $ComputeCL(S)$ computes the convex layers of a set S of n points in the plane in $O(\max(\frac{n \log n}{p}, \frac{n^{\frac{1-\epsilon}{2}} k^2 \log n}{p}))$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$. □

Corollary 1: Let k be the number of the convex layers of S. When $1 \le k \le n^{\frac{\epsilon}{2}}$ $(0 \le \epsilon < 1)$ the convex layers of S can be computed optimally in $O(\frac{n \log n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, in the *CREW-PRAM*. □

3.3 Constructing the Outermost Convex Layer

In this section, we explain how to find the convex hull of the outermost layers of $S'_1, S'_2, \ldots S'_{n^{\frac{1-\epsilon}{2}}}$ (Step 3(a) of the algorithm $ComputeCL$). We call the upper part and the lower part of a convex hull as the upper hull and lower hull, respectively. Let OL_i be the outermost convex layer of S'_i, that is $OL_i = CL_1(S'_i)$ $(1 \le i \le n^{\frac{1-\epsilon}{2}})$. We compute the upper hull of $OL_1, \ldots, OL_{n^{\frac{1-\epsilon}{2}}}$, denoted as UH, as follows. The lower hull LH of $OL_1, \ldots, OL_{n^{\frac{1-\epsilon}{2}}}$ can be computed similarly.

Procedure $UHull$ **(**$OL_1, \ldots, OL_{n^{\frac{1-\epsilon}{2}}}$**)**

(Step 1) For any pair of i, j $(1 \le i < j \le n^{\frac{1-\epsilon}{2}})$ find T_{ij}, the common tangent of OL_i and OL_j, in parallel. Let $T_{ij} = (l_{ij}, r_{ij})$, meaning that points l_{ij} and r_{ij} are the left and right endpoints of T_{ij}, respectively.

(Step 2) For each OL_i we now have $n^{\frac{1-\epsilon}{2}} - 1$ incident tangents. The tangents can be partitioned into two subsets: one for which the tangents are incident to OL_i from the left, and the other for tangents incident from the right (Fig. 5). Among the first subset we find the tangent, let us call it L_i, which lies uppermost, that is, with the smallest slope. Similarly, among the second subset we find the tangent R_i, which lies uppermost, that is, with the largest slope. Notice that L_1 and $R_{n^{\frac{1-\epsilon}{2}}}$ do not exist because OL_1 and $OL_{n^{\frac{1-\epsilon}{2}}}$ are the respective leftmost and rightmost ones among all OL_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$.

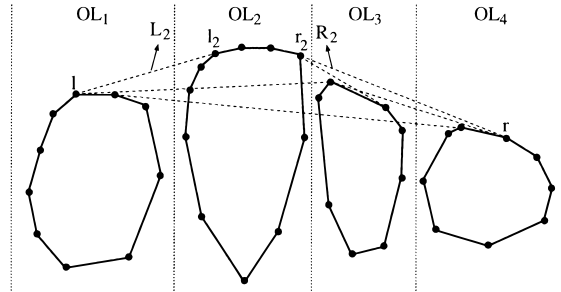

Fig. 5 Finding the upper common tangents.

(Step 3) For $2 \le i \le n^{\frac{1-\epsilon}{2}} - 1$, let $L_i = (l, l_i)$ and $R_i = (r_i, r)$, where l_i and r_i refer to points belonging to OL_i, and l and r refer to points belonging to convex layers lying left and right of OL_i respectively (Fig. 5). If the point r_i lies to the left of the point l_i, that is, if $r_i < l_i$, then no point of OL_i belongs to UH. Otherwise, the points of OL_i located counterclockwise between r_i and l_i, inclusive, belong to UH. For $i = 1$ let the points of OL_1 located counterclockwise between r_1 and the leftmost point of OL_1 inclusive, belong to U. For $i = n^{\frac{1-\epsilon}{2}}$ let the points of $OL_{n^{\frac{1-\epsilon}{2}}}$ located counterclockwise between the rightmost point of $OL_{n^{\frac{1-\epsilon}{2}}}$ and $r_{n^{\frac{1-\epsilon}{2}}}$ inclusive, belong to UH. □

Now let us consider the computational complexity of procedure $UHull(OL_1, \ldots, OL_{n^{\frac{1-\epsilon}{2}}})$. As mentioned in Sect. 3.1 the tangent of two convex layers can be found in $O(\log n)$ time sequentially [10]. Therefore, Step 1 can be done in $O(\log n)$ time using $n^{1-\epsilon}$ processors. In Step 2 finding L_i and R_i for each OL_i can be done by using the maxima finding algorithm. As OL_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ is stored in a 2-3 tree, and the number of incident tangents in each OL_i is less than $n^{\frac{1-\epsilon}{2}}$, finding L_i and R_i requires $O(\log n^{\frac{1-\epsilon}{2}}) = O(\log n)$ time and $O(n^{\frac{1-\epsilon}{2}})$ processors. Thus, all L_i and R_i can be found in $O(\log n)$ time using $n^{1-\epsilon}$ processors. In Step 3, the part of OL_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ which belongs to UH can be found in constant time using $n^{\frac{1-\epsilon}{2}}$ processors. Therefore, the procedure $UHull(OL_1, \ldots, OL_{n^{\frac{1-\epsilon}{2}}})$ can be done in $O(\log n)$ time using $O(n^{1-\epsilon})$ processors.

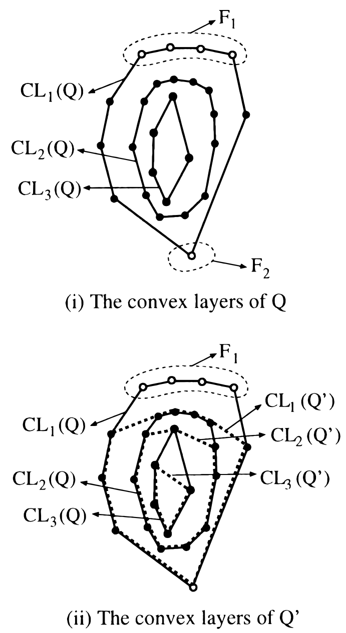

Fig. 6 Reconstructing the convex layers.

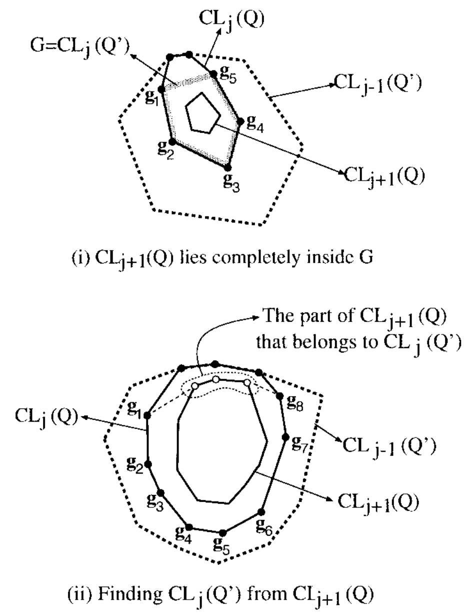

Fig. 7 Finding $CL_j(Q')$.

3.4 Reconstructing the Convex Layers

Let Q be a set of n points and $CL(Q) = (CL_1(Q), CL_2(Q), \ldots, CL_h(Q))$ $(1 \leq h \leq n)$ be the convex layers of Q, where $CL_i(Q)$ $(1 \leq i \leq h)$ is the ith convex layer of Q, and h is the number of the layers of Q. Let each convex layer of Q be stored in two 2-3 trees as explained in Sect. 3.1.

Let F_1 and F_2 be two contiguous sub-polygonal chains of $CL_1(Q)$, such that F_1 and F_2 have no common parts (Fig. 6(i)). Suppose we want to delete the points of F_1 and F_2 from Q and reconstruct the convex layers Q. If $F_1 \cup F_2 = CL_1(Q)$, that is, F_1 and F_2 consist of all the points of $CL_1(Q)$, then we just delete the points of F_1 and F_2 from Q, delete $CL_1(Q)$ from $CL(Q)$, and no reconstruction needs to take place. Otherwise, we revise Q and its convex layers as follows. As F_1 and F_2 share no common parts we can delete both of them at the same time and reconstruct the convex layers of Q. In order to simplify the explanation, in the following we show how to reconstruct the the convex layers of Q after deleting F_1. The procedure for deleting F_2 is similar.

Let Q' be the set obtained by deleting the points of F_1 from Q. We reconstruct the convex layers $CL(Q') = (CL_1(Q'), CL_2(Q'), \ldots, CL_{h'}(Q'))$, where $h' \leq h$ is the number of convex layers of Q', based on the convex layers of $CL(Q)$. It is easily seen that the points of $CL_l(Q')$ $(1 \leq l \leq h')$ come from the points of $CL_l(Q)$, or $CL_l(Q)$ and $CL_{l+1}(Q)$ (see Fig. 6(i) where the layers $CL_1(Q)$, $CL_2(Q)$, and $CL_3(Q)$, are drawn with thick lines and Fig. 6(ii) where the layers $CL_1(Q')$, $CL_2(Q')$, and $CL_3(Q')$, are drawn with dotted lines).

We construct $CL(Q')$ from $CL_1(Q')$ to $CL_{h'}(Q')$ repeatedly. Assuming that we have found $CL_1(Q')$, $CL_2(Q')$, $\ldots$, $CL_{j-1}(Q')$ $(1 \leq j \leq h'-1)$ we find $CL_j(Q')$ as follows. Let $G = (g_1, g_2, \ldots, g_v)$ consist of the points of $CL_j(Q)$ which do not belong to $CL_{j-1}(Q')$ (F_1 when $j=1$) listed in counterclockwise. If $CL_{j+1}(Q)$ lies completely inside the convex polygon G, then $CL_j(Q') = G$ (Fig. 7(i)). Else, $CL_j(Q')$ consists of not only the points of G but also some points of $CL_{j+1}(Q)$. To find these points of $CL_{j+1}(Q)$, we draw the tangents from the endpoints of G into $CL_{j+1}(Q)$ (Fig. 7(ii)). The points of $CL_{j+1}(Q)$ intersected by the tangents will determine a sub-chain of $CL_{j+1}(Q)$ that will become the part of $CL_j(Q')$ (see Fig. 7(ii)). To construct $CL_j(Q')$, we first delete the vertices which do not belong to $CL_j(Q')$ from $CL_{j+1}(Q)$ (these vertices are contiguous in $CL_{j+1}(Q)$). Then by inserting G into $CL_{j+1}(Q)$, we get $CL_j(Q')$.

Since the layers are stored in 2-3 trees, the tangents from the endpoints of G into $CL_{j+1}(Q)$ can be found in $O(\log n)$ time sequentially [10]. By using the operations (IV) and (V) stated in Sect. 3.1, the 2-3 trees storing $CL_j(Q)$ can be updated to the ones storing $CL_j(Q')$ in $O(\log n)$ time sequentially. Since we find $CL_i(Q')$ repeatedly from $i=1$ to $i=h'$, $CL(Q') = (CL_1(Q'), CL_2(Q'), \ldots, CL_{h'}(Q'))$ can be found in $O(h' \log n) = O(h \log n)$ time sequentially.

Now let us go back to Step 3(b)(ii) of the algorithm *ComputeCL* in Sect. 3.2, where for each i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in parallel, we delete the vertices of $CL_x(S)$ from S'_i, and then reconstruct the convex layers of S'_i. The vertices of $CL_x(S)$ which belong to S'_i form two (or one) contiguous subchains in $CL_x(S)$, denoted as F'_1 and F'_2 respectively. Let k'_i be the number of layers in S'_i. Considering S'_i as Q, F'_1 and F'_2 as F_1 and F_2 respectively, and k'_i as h, we can reconstruct the convex layers of S'_i as above. Therefore, each S'_i can be processed in $O(k'_i \log n)$ time sequentially. As the ith convex layer of S never contains the points of the jth ($j > i$) convex layer of S'_i then $k'_i \le k$, where k is the number of convex layers of S. Therefore, all S'_i ($1 \le i \le n^{\frac{1-\epsilon}{2}}$) can be processed in $O(k \log n)$ time using $n^{\frac{1-\epsilon}{2}}$ processors.

4. The Envelope Layers Algorithm

Let S be a set of n (opaque) line segments in the plane. Our algorithm for computing the set of envelope layers of S, is basically as follows. First we use a segment tree to divide S into $m = O(\log n)$ groups, $G_1, G_2, \ldots, G_m$, and reduce the envelope layers problem of G_i ($1 \le i \le m$) into the convex layers problem of points in the plane. Next, we find the envelope layers of G_i ($1 \le i \le m$) by our algorithm for the convex layers given in Sect. 3. Then we cut the envelope layers of all groups $G_1, G_2, \ldots, G_m$ by $n^{\frac{1-\epsilon}{2}}$ vertical lines into $n^{\frac{1-\epsilon}{2}}$ separate groups $H_1, H_2, \ldots, H_{n^{\frac{1-\epsilon}{2}}}$, and for each i ($1 \le i \le n^{\frac{1-\epsilon}{2}}$) in parallel, we compute the envelope layers of H_i, by Hershberger's sequential algorithm [7]. Finally, we concatenate the envelope layers of all H_i from $i = 1$ to $n^{\frac{1-\epsilon}{2}}$ into the envelope layers of S.

4.1 The Segment Tree to Divide Segments

Let S be a set of n line segments. The plane can be partitioned into slabs by drawing vertical lines through all the segments endpoints. This can be accomplished by sorting the endpoints of the segments in S by x-coordinates in increasing order, and then partitioning the x-axis by the x coordinates of the endpoints into $2n+1$ slabs. The segment tree of S, denoted as $ST(S)$, is built as follows [11] (Fig. 8):

(i) Construct a complete binary tree with $2n$ leaves. (We assume that $2n$ is a power of two, if not add some dummy leaves.)
(ii) Let each leaf of $ST(S)$ represent one slab taken in left-to-right order, and each internal node represent the union of its descendants' slabs. Each region associated with a node, whether an original slab or a union of them, is a *canonical slab*.
(iii) In a top-down fashion, that is, descending from the root to the leaves, associate with each node $v \in ST(S)$ a subset $S[v]$ of S, where $S[v]$ consists of the segments or subsegments of S that have its endpoints on the boundary of the canonical slab represented by node v, which have not been associated with any of v's ancestors in $ST(S)$.

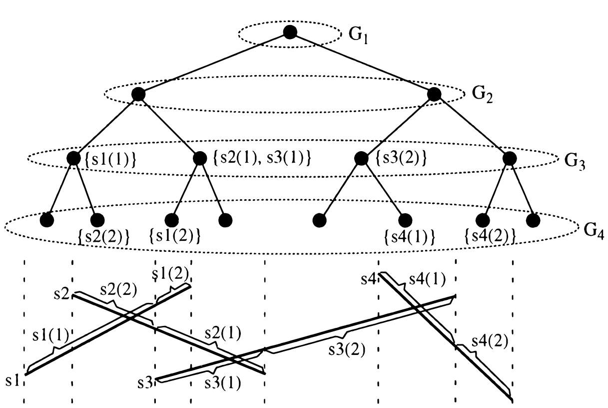

Fig. 8 The segment tree $ST(S)$.

It can be seen from step (iii) that for each segment in S, at most two subsegments may appear in one level of the segment tree $ST(S)$. Thus, every segment is decomposed into at most $2 \log n + 1 = O(\log n)$ subsegments, each with its endpoints on the boundary of some canonical slab.

Property 1: [7] The segment tree $ST(S)$ divides set S into $4n - 1$ subsets which satisfy the following property: for any two nodes x and y on the same level of $ST(S)$, the subsets associated with nodes x and y are separated. □

The segment tree $ST(S)$ can be constructed in $O(\log n)$ time using $O(n)$ processors, by slightly modifying the algorithm of Chen et al. [3].

Let i ($1 \le i \le \log n + 1$) denote each the level of ST, such that root is located at level $i = 1$ and the leaves at level $i = \log n + 1$. On each level i of $ST(S)$, let the nodes be numbered from 1 to $g_i (= 2^{i-1})$. We define group $G_i = (L_i^1, L_i^2, \ldots, L_i^{g_i})$, where L_i^j ($1 \le j \le g_i$) is the set associated with node j on level i of $ST(S)$. Therefore, the n segments of S are divided into $O(n \log n)$ subsegments which belong to $m = O(\log n)$ groups, $G_1, G_2, \ldots, G_m$ (Fig. 8). Obviously group G_i ($1 \le i \le m$) has $O(n)$ subsegments. From Property 1, for each group G_i, $EL(L_i^1), EL(L_i^2), \ldots, EL(L_i^{g_i})$ are separated from each other.

Given i and j, let us consider the property of L_i^j. In L_i^j, the right endpoints of all the segments have the same x-coordinates, and the left endpoints also have the same x-coordinates. This implies that L_i^j can be considered as a set of lines. From the duality of points and lines, finding the envelope layers of n lines can be reduced into finding the convex layers of n points in the plane. Since the size of the convex layers of n points is $O(n)$, the size of $EL(L_i^j)$ is $O(|L_i^j|)$ segments. We summarize the above property as follows.

Property 2: (i) Given i and j, ($1 \le i \le m$, $1 \le j \le g_i$), $EL(L_i^j)$ can be found by computing $CL(L_i^j)$. (ii) the size of the envelope layers of each subset L_i^j, i.e. $|EL(L_i^j)|$, is $O(|L_i^j|)$ segments, the size of the envelope layers of all subsets in each group G_i, i.e. $\sum_{j=1}^{g_i} |EL(L_i^j)| = O(|G_i|)$, is $O(n)$ segments, and the size of the envelope layers of all groups $G_1, G_2, \ldots, G_m$, i.e. $\sum_{i=1}^{m} \sum_{j=1}^{g_i} |EL(L_i^j)| = O(\sum_{i=1}^{m} |G_i|)$, is $O(n \log n)$ segments. □

4.2 The Outline of the Algorithm

Let S be a set of n (opaque) line segments in the plane. Let k be the number of the envelope layer of S, and let ϵ $(0 \leq \epsilon < 1)$ be a constant. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ the following algorithm computes the envelope layers of S in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$.

Algorithm *ComputeEL*(*S*)

[Input] A set S of n (opaque) line segments in the plane.
[Output] A set $EL(S) = (EL_1(S), EL_2(S), \ldots, EL_k(S))$ $(1 \leq k \leq n)$ of the envelope layers of S, where k is the number of layers, and $EL_i(S)$ $(1 \leq i \leq k)$ is the ith envelope layer of S.
(Step 1) Use a segment tree, denoted as $ST(S)$, which has $m = O(\log n)$ levels, to divide the n segments of S into $O(n \log n)$ subsegments which belong to m groups, say $G_1, G_2, \ldots, G_m$, where group G_i $(1 \leq i \leq m)$ corresponds to the subsegments associated on level i of $ST(S)$ (Fig. 8). We reduce the envelope layers problem of G_i to the convex layers problem of $O(|G_i|)$ points in the plane.
(Step 2) For each i $(1 \leq i \leq m)$ in parallel, find $EL(G_i)$ the envelope layers of G_i by using our algorithm for the convex layers.
(Step 3) Cut the envelope layers of all $G_1, \ldots, G_m$, i.e. $EL(G_1), \ldots, EL(G_m)$, by $n^{\frac{1-\epsilon}{2}}$ vertical lines into $n^{\frac{1-\epsilon}{2}}$ separate groups $H_1, H_2, \ldots, H_{n^{\frac{1-\epsilon}{2}}}$ (Fig. 9). Then, for each i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in parallel, compute the envelope layers of H_i, $EL(H_i) = (EL_1(H_i), EL_2(H_i), \ldots, EL_{h_i}(H_i))$, where h_i is the number of layers in H_i, by Hershberger's sequential algorithm [7].
(Step 4) Let $k = \max(h_1, h_2, \ldots, h_{n^{\frac{1-\epsilon}{2}}})$. Obtain $EL_t(S)$ $(1 \leq t \leq k)$, i.e. the tth envelope layer of S, by concatenating $EL_t(H_1)$, $EL_t(H_2)$, $\ldots$, $EL_t(H_{n^{\frac{1-\epsilon}{2}}})$ ($EL_t(H_i)$ is empty if $t > h_i$). □

In Sect. 4.1 we have shown that Step 1 can be done in $O(\log n)$ time using n processors. It was also shown that the envelope layers problem of G_i can be reduced to the convex layers problem of $O(|G_i|)$ points in the plane. In Sect. 4.3 we show that when $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$, Step 2 can be done in $O(\frac{n\log^2 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$. Finally, in Sect. 4.4 we show that when $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ $EL(S)$ can be constructed from $EL(G_1), EL(G_2), \ldots, EL(G_m)$ (Step 3 and Step 4), in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$.

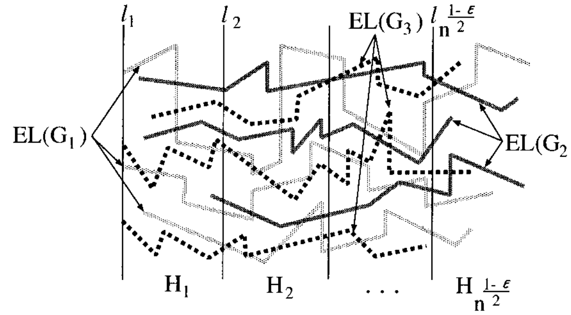

Fig. 9 Cutting the envelope layers of $G_1, \ldots, G_m$ by $n^{\frac{1-\epsilon}{2}}$ vertical lines into $n^{\frac{1-\epsilon}{2}}$ parts.

Theorem 2: Let k be the number of the envelope layers of a set S of n (opaque) line segments. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 \leq \epsilon < 1)$ the envelope layers problem of S can be solved in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$. □

4.3 Constructing the Layers of Each Group

In Step 2, we compute the convex layers of $G_i = (L_i^1, L_i^2, \ldots, L_i^{g_i})$ for each i $(1 \leq i \leq m)$, which consists of two subtasks. First we find $EL(L_i^1), EL(L_i^2), \ldots, EL(L_i^{g_i})$. Then we concatenate $EL(L_i^j)$ from $j = 1$ to $j = g_i$ into the envelope layers of G_i (Fig. 10). The details of these two subtasks are given below.

From Property 2(i) we use the algorithm *ComputeCL* in parallel, in Step 2, to find the envelope layers of L_i^j. Now we show how this can be accomplished having p processors available. Recall that from Step 1 the total number of subsegments of all subsets L_i^j is $O(n \log n)$. Then in Step 2 each processor is responsible for processing $\frac{n \log n}{p}$ subsegments. As the size of L_i^j may be larger or smaller than $\frac{n \log n}{p}$, we may need to assign more than one processor to process it, or assign several of them to one processor. Let l_i^j be the number of subsegments in subset L_i^j. If $l_i^j = \frac{n \log n}{p}$, then we assign one processor to compute the envelope layers of L_i^j. Otherwise, if $l_i^j > \frac{n \log n}{p}$, then we assign $z = \left\lfloor \frac{l_i^j}{\frac{n \log n}{p}} \right\rfloor$ processors to L_i^j such that each one corresponds to $O(\frac{n \log n}{p})$ subsegments of L_i^j. If $l_i^j < \frac{n \log n}{p}$ then we consider L_i^j as a small subset of G_i. In this case, we let one processor to be in charge for several elements such that it is responsible for processing a total

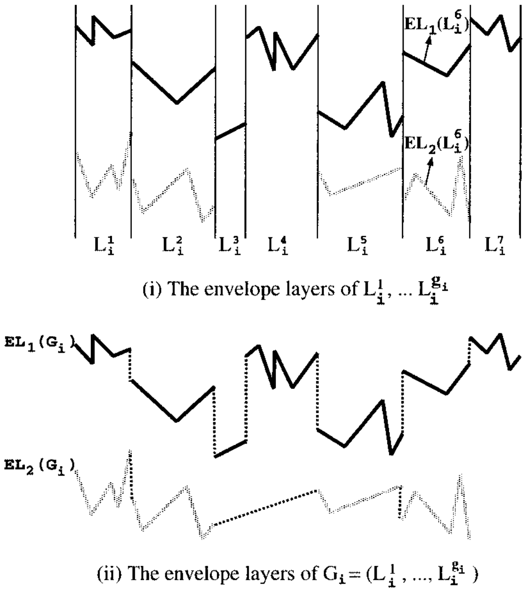

Fig. 10 Concatenating the envelope layers of $L_i^1, L_i^2, \ldots, L_i^{g_i}$ into the convex layers of $G_i = L_i^1, L_i^2, \ldots, L_i^{g_i}$.

of $O(\frac{n \log n}{p})$ subsegments.

Now let us consider the running time for computing $EL(L_i^j) = (EL_1(L_i^j), EL_2(L_i^j), \ldots, EL_{k_{ij}}(L_i^j))$ $(1 \le i \le m, 1 \le j \le g_i)$, where k_{ij} is the number of the envelope layers of L_i^j. If $l_i^j > \frac{n \log n}{p}$, L_i^j is processed in parallel using z processors. If $l_i^j \le \frac{n \log n}{p}$, L_i^j is processed sequentially as explained above. Since from Property 2(ii) the total size of all L_i^j $(1 \le i \le m, 1 \le j \le g_i)$ is $N = O(n \log n)$, by using the algorithm for the convex layers in Sect. 3, when $1 \le k \le n^{\frac{\epsilon}{2}}$ we can find all $EL(L_i^j)$ $(1 \le i \le m, 1 \le j \le g_i)$ in $O(\frac{N \log N}{p}) = O(\frac{n \log^2 n}{p})$ time using p processors.

Since $EL(L_i^1), EL(L_i^2), \ldots, EL(L_i^{g_i})$ are separated, we can concatenate them into $EL(G_i) = (EL_1(G_i), EL_2(G_i), \ldots, EL_{k_i}(G_i))$, where k_i is the number of envelope layers of G_i, as follows (Fig. 10). For each i $(1 \le i \le m)$ let k_{ij} be the number of the envelope layers of L_i^j. We have $k_i = \max(k_{i1}, k_{i2} \ldots, k_{ig_i})$. $EL_t(G_i)$ can be obtained by concatenating $EL_t(L_i^j)$ from $j = 1$ to $j = g_i$ ($EL(L_i^j)$ is empty if $t > k_i$). Refer to Fig. 10(ii) where dotted lines are used to show the layers of G_i $(k_i = 2)$ obtained from the concatenation of the layers of $EL(L_i^1), EL(L_i^2), \ldots, EL(L_i^7)$. From Property 2(ii), $\sum_{j=1}^{g_i} |EL(L_i^j)| = O(n)$. Therefore, if the envelope layers of $EL(L_i^j)$ are saved in arrays, the concatenation of the layers of $EL(L_i^1), EL(L_i^2), \ldots, EL(L_i^{g_i})$ into the layers of $EL(G_i)$ can be done in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors by using prefix sums computation. Thus, in Step 2 $EL(G_i)$ can be computed in $O(\frac{n \log^2 n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$.

4.4 Constructing the Envelope Layers of S

Let $v = \max(k_1, k_2, \ldots, k_m)$, where k_i $(1 \le i \le m)$ is the number of layers in $EL(G_i)$. Then, the total number of layers in $EL(G_1), \ldots, EL(G_m)$ is $K = O(vm)$. Since $v \le k$, where k is the number of envelope layers of S, and $m = O(\log n)$, $K = O(k \log n)$. On the other hand, from Property 2(ii) the total size of $EL(G_1), EL(G_2), \ldots, EL(G_m)$ is $O(n \log n)$.

Now we show how to construct the envelope layers of S from $EL(G_1)$, $EL(G_2)$, $\ldots$, $EL(G_m)$. First, we use $n^{\frac{1-\epsilon}{2}}$ vertical lines to cut K envelope layers of $EL(G_1), \ldots, EL(G_m)$ into $n^{\frac{1-\epsilon}{2}}$ separate parts $H_1, H_2, \ldots, H_{n^{\frac{1-\epsilon}{2}}}$ (Fig. 9). The $n^{\frac{1-\epsilon}{2}}$ vertical lines, $l_1, l_2, \ldots, l_{n^{\frac{1-\epsilon}{2}}}$ can be decided as follows. Let X be the set consisting of the endpoints of the segments in $EL(G_1), \ldots, EL(G_m)$. Sort the points of X by their x-coordinates in increasing order, and then divide X into $n^{\frac{1-\epsilon}{2}}$ parts, i.e. $X_1, X_2, \ldots, X_{n^{\frac{1-\epsilon}{2}}}$, such that each X_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ contains $O(n^{\frac{1+\epsilon}{2}} \log n)$ points of X and the x-coordinates of the points of X_i are less than the x-coordinates of the points of X_{i+1}. Notice that the endpoints of a same segment may belong to different X_i parts. For each i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$, define l_i to be the vertical line passing through the leftmost point of X_i. After cutting, each H_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ contains $O(n^{\frac{1+\epsilon}{2}} \log n)$ endpoints. In the cutting process some segments may be partitioned by the vertical lines into subsegments that belong to several parts (Fig. 9). As each vertical line cuts the K layers of $EL(G_1), \ldots, EL(G_m)$, it may produce $K = O(k \log n)$ segments. Thus, each H_i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ consists of $O(n^{\frac{1+\epsilon}{2}} \log n + k \log n)$ segments.

Then, for each i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$ in parallel, we compute the envelope layers of H_i by using Hershberger's sequential algorithm [7].

Sorting the $O(n \log n)$ points in X can be done in $O(\log n)$ time using n processors. For each i $(1 \le i \le n^{\frac{1-\epsilon}{2}})$, $EL(H_i)$ can be computed in $O(|H_i| \alpha(|H_i|) \log^2 |H_i|)$ time sequentially [7]. Therefore, $EL(H_1)$, $EL(H_2)$, $\ldots$, $EL(H_{n^{\frac{1-\epsilon}{2}}})$ can be computed in $O(|H_i| \alpha(|H_i|) \log^2 |H_i|)$ time using $n^{\frac{1-\epsilon}{2}}$ processors, where $|H_i| = O(n^{\frac{1+\epsilon}{2}} \log n + k \log n)$. Thus, Step 3 can be done in $O(n^{\frac{1+\epsilon}{2}} \alpha(n) \log^3 n)$ time using $n^{\frac{1-\epsilon}{2}}$ processors, i.e. it can be done in $O(\frac{n \alpha(n) \log^3 n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$ $(0 \le \epsilon < 1)$.

Finally, we concatenate the layers of $EL(H_1)$, $EL(H_2)$, $\ldots$, $EL(H_{n^{\frac{1-\epsilon}{2}}})$ into the layers of $EL(S) = (EL_1(S), EL_2(S), \ldots, EL_k(S))$. Let $EL(H_i) = (EL_1(H_i), EL_2(H_i), \ldots, EL_{h_i}(H_i))$, where h_i is the number of the layers of H_i. Obviously, $k =$

$\max(h_1, h_2, \ldots, h_{n^{\frac{1-\epsilon}{2}}})$. $EL_t(S)$ $(1 \le t \le k)$ can be obtained by concatenating $EL_t(H_1)$, $EL_t(H_2)$, $\ldots$, $EL_t(H_{n^{\frac{1-\epsilon}{2}}})$ ($EL_t(H_i)$ is empty if $t > h_i$). As stated in Sect. 2 the size of the envelope layers of H_i is $O(\alpha(|H_i|)|H_i|)$. Thus, $|EL(S)| = \sum_{i=1}^{n^{\frac{1-\epsilon}{2}}} |EL(H_i)| = O(\sum_{i=1}^{n^{\frac{1-\epsilon}{2}}} \alpha(|H_i|)|H_i|) = O(n\alpha(n)\log n)$ when $k < n^{\frac{\epsilon}{2}}$. If the envelope layers of each $EL(H_i)$ are saved in arrays and $k < n^{\frac{\epsilon}{2}}$, concatenating $EL(H_1)$, $EL(H_2)$, $\ldots$, $EL(H_{n^{\frac{1-\epsilon}{2}}})$ into $EL(S)$ (Step 4) can be done in $O(\log n)$ time using $O(\frac{n\alpha(n)\log n}{\log n})$ processors by using prefix sums computation.

5. Conclusion

In this paper we have proposed an *EP* parallel algorithm for computing the convex layers of a set S of n points. Let k be the number of the convex layers of S. When $1 \le k \le n^{\frac{\epsilon}{2}}$ $(0 \le \epsilon < 1)$ our algorithm runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, in the *CREW-PRAM*, and it is cost optimal.

We have also considered the envelope layers problem. We presented an algorithm which solves the envelope layers problem of a set S of n (opaque) line segments. Let k be the number of the envelope layers of S. When $1 \le k \le n^{\frac{\epsilon}{2}}$ $(0 \le \epsilon < 1)$ our algorithm runs in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \le p \le n^{\frac{1-\epsilon}{2}}$, in the *CREW-PRAM*. If we ignore a factor of $\log n$ our algorithm for the envelope layers belongs to the class *EP*.

To simplify the explanation we used the *CREW-PRAM* parallel computational model, although the results are also generalized to the *EREW-PRAM* model.

We expect that our methodology can be generalized to solve other *P*-complete problems as well.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, Data Structures and Algorithms, Addison-Wesley, 1983.
[2] B. Chazelle, "On the convex layers of a planar set," IEEE Trans. Inf. Theory, vol.31, no.4, pp.509–517, 1995.
[3] W. Chen and K. Wada, "On computing the upper envelope of segments in parallel," Proc. 27th International Conference on Parallel Processing, pp.253–260, 1998.
[4] A. Dessmark, A. Lingas, and A. Maheshwari, "Multi-list ranking: Complexity and applications," 10th Annual Symposium on Theoretical Aspects of Computer Science, LNC vol.665, pp.306–316, 1993.
[5] A. Fujiwara, M. Inoue, and T. Masuzawa, Practical parallelizability of some P-complete problems, Technical Report of IPSF, 1999.
[6] D. Hart and M. Sharir, "Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes," Combinatorica, vol.6, pp.151–177, 1989.
[7] J. Hershberger, "Upper envelope onion peeling," Computational Geometry: Theory and Applications, vol.2, pp.93–110, 1992.
[8] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, 1992.
[9] C.P. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," Theoretical Computer Science, vol.71, pp.95–132, 1990.
[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] M. Sharir and P.K. Agarwal, Davenport-Schinzel Sequences and Their Geometric Applications, Cambridge University Press, 1995.
[13] A.A. Yao, "A lower bound to finding convex hulls," J. ACM, vol.28, no.4, pp.780–787, 1981.

Carla Denise Castanho received the B.S. degree in Computer Science from Passo Fundo University, Brazil, in 1994. She received the M.E. degree in Computer Science from Nagoya Institute of Technology, Japan, in 1998. She is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Nagoya Institute of Technology. Her research interests include parallel algorithms and computational geometry.

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. 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.

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.

Akihiro Fujiwara received the B.E. degree in Osaka University in 1993, and received the M.E. and Ph.D. degrees in Nara Institute of Science and Technology (NAIST) in 1995 and 1997, respectively. He is now an associate professor of Kyushu Institute of Technology. His main research interests are parallel algorithms, parallel complexity theory and cluster processing. He is a member of IEEE and IPSJ.