Optimal, Output-Sensitive Algorithms for Constructing Upper Envelope of Line Segments in Parallel

Neelima Gupta, Sumit Chopra, and Sandeep Sen

[1] Department of Computer Science, Hansraj College, Delhi University, New Delhi 110007, India. neelima_research@yahoo.com
[2] Department of Computer Science, Hansraj College, Delhi University, New Delhi 110007, India. spchopra@mantraonline.com
[3] Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India. ssen@cse.iitd.ernet.in

Abstract. In this paper we focus on the problem of designing very fast parallel algorithms for constructing the upper envelope of straight-line segments that achieve the $O(n \log H)$ work-bound for input size n and output size H. Our algorithms are designed for the arbitrary CRCW PRAM model. We first describe an $O(\log n \cdot (\log H + \log\log n))$ time deterministic algorithm for the problem, that achieves $O(n \log H)$ work bound for $H = \Omega(\log n)$. We present a fast randomized algorithm that runs in expected time $O(\log H \cdot \log\log n)$ with high probability and does $O(n \log H)$ work. For $\log H = \Omega(\log\log n)$, we can achieve the running time of $O(\log H)$ while simultaneously keeping the work optimal. We also present a fast randomized algorithm that runs in $\tilde{O}(\log n/\log k)$ time with nk processors, $k > \log^{\Omega(1)} n$. The algorithms do not assume any input distribution and the running times hold with high probability.

1 Introduction

The upper envelope of a set of n line segments in the plane is an important concept in visibility and motion planning problems. The segments are regarded as opaque obstacles, and their upper envelope consists of the portion of the segments visible from the point $(0, +\infty)$. The complexity of the upper envelope is the number of distinct pieces of segments that appear on it. If the segments are nonintersecting, then the complexity of their upper envelope is linear in n. On the other hand, if the segments are allowed to intersect, then the worst case complexity of the upper envelope increases to $O(n\alpha(n))$, where $\alpha(n)$ is the functional inverse of Ackermann's function [2].

There exists an $O(n \log n)$ time algorithm to compute the upper envelope of n line segments, and this is worst case optimal [15]. However, this is true only if the output size, i.e., the number of vertices (or the edges) of the upper envelope, is large. More specifically, the time-bound of $O(n \log n)$ is tight when the ordered output size is $\Omega(n)$. However if the output size is small then we should be able to do much better. The output-size of a problem is an important parameter in measuring the efficiency of an algorithm and one can get considerably superior algorithms in terms of it. There exists an $O(n \log H)$ algorithm for the problem [17], where H is the output size, which implies a linear time algorithm for constant output size.

We are aiming at designing an output size sensitive parallel algorithm that speeds up optimally with output size in the sub-logarithmic time domain.

In the context of sequential algorithms, it has been observed that the upper envelope of n line segments can be computed in $O(n\alpha(n) \log n)$ time, by a straight forward application of divide-and-conquer technique. Hershberger describes an optimal $O(n \log n)$ algorithm by reorganizing the divide-and-conquer computation [15].

Clarkson describes a randomized $O(n\alpha(n) \log n)$ algorithm for computing a face in an arrangement of line segments [8]. The upper envelope of line segments can be viewed as one such face. Guibas et al. [11] , gave a deterministic algorithm which computed a face for a collection of line segments in $O(n\alpha^2(n) \log n)$ time.

The process of gift wrapping amounts to locating an extreme segment of the upper envelope and then "walking" along successive segments of the envelope which are defined by the points of intersection of the lines in the given arrangement. This results in a simple $O(nH)$ time algorithm. Franck Nielsen and Mariette Yvinec presents a deterministic sequential output-sensitive algorithm that computes the upper envelope in $O(n \log H)$ time [17]. Their algorithm is based on the marriage-before-conquest paradigm to compute the convex hull of a set of fixed planar convex objects, which they also apply to compute the upper envelope of line segments. They use the partitioning technique of Hershberger to get an $O(n \log H)$ time for upper envelope. They claim that their algorithms are easily parallelizable onto EREW PRAM, following the algorithm of S. Akl [3]. This implies a parallel output-sensitive algorithm which is optimal for number of processors bounded by $O(n^z)$, $0 < z < 1$.

Recently Wei Chen and Koichi Wada gave a deterministic algorithm that computes the upper envelope of line segments in $O(\log n)$ time using $O(n)$ processors [7]. If the line segments are nonintersecting and sorted, the envelope can be found in $O(\log n)$ time using $O(n/ \log n)$ processors. Their methods also imply a fast sequential result: the upper envelope of n sorted line segments can be found in $O(n \log \log n)$ time.

We present algorithms whose running times are output-sensitive even in the sub-logarithmic time range while keeping the work optimal. For designing fast output-sensitive algorithms, we have to cope with the problem that the output-size is an unknown parameter. Moreover, we also have to rapidly eliminate input line segments that do not contribute to the final output without incurring a high cost - see Gupta and Sen [13] for a more detailed discussion.

We present one deterministic and two randomized algorithms that construct the upper envelope of n line segments. Both our randomized algorithms are of

Las Vegas type, that is we always provide a correct output and the bounds hold with high probability[1].

We first describe a deterministic algorithm for the problem that takes $O(\log n \cdot (\log H + \log\log n))$ time using $O(n/\log n)$ processors. The algorithm is based on the marriage before conquest approach and we use the ideas presented by Neilsen and Yvinec [17] to bound the size of the sub-problems. Our algorithm achieves $O(n \log H)$ work bound for $H = \Omega(\log n)$.

Next we present fast randomized algorithm. The expected running times hold with high probability. The algorithm runs in $O(\log H)$ expected time using n processors for $H > \log^{\varepsilon} n$, $\varepsilon > 0$. For smaller output sizes the algorithm has an expected running time of $O(\log H \cdot \log\log n)$ keeping the number of operations optimal. Therefore for small output sizes our algorithm runs very fast. The algorithm uses the iterative method of Gupta and Sen [13].

We also describe a randomized algorithm that solves the problem in $O(\log n/\log k)$ time with high probability using nk processors for $k > \log^{\Omega(1)} n$. This algorithm is based on the general technique given by Sen to develop sub-logarithmic algorithms [21].

2 Deterministic Algorithm for Upper Envelope

Our algorithm is based on the Marriage-before-conquest technique and it uses the ideas presented by Neilsen and Yvinec [17] to bound the size of the sub-problems.

Let S be the set of n line segments. Let its upper envelope be denoted by UE(S). Then, if we know that there exists a partition $\cup_{i=1}^{k} P_i$ of S into k subsets such that each subset P_i, for $i \in [1, k]$ is a set of non-overlapping line segments, then in the marriage-before-conquest procedure we can bound the size of the sub-problems by $(n/2 + k)$. Thus we transform the set S into a set T of line segments partitioned into subsets, each subset consisting of a set of non-overlapping line segments, such that UE(S) = UE(T). We now apply the marriage-before-conquest procedure on the set T. To compute the set T, we partition the set S using a partition tree and use the communication of J. Hershberger to make the size of T linear. With a good estimate of k, we run the marriage-before-conquest procedure on the set T (the number of stages of this procedure depends on k), and reduce the total size of the problem to $n/\log n$, after which the problem is solved directly.

2.1 Transforming the Set S of Line Segments into the Set T

The Vertical Decomposition. Let us call two line segments to be x-separated if there exists a vertical line such that the two line segments lie entirely on the different sides of the vertical line. The vertical decomposition will give rise to a set of x-separated line segments.

[1] The term high probability implies probability exceeding 1-1/n^c for any predetermined constant c where n is the input size.

Let $Q = \{l_1, l_2, \ldots, l_{\mathrm{m}}\}$ be a subset of S. Through each vertex of the upper envelope of Q, we draw a line parallel to y-axis. These parallel lines induce a decomposition of the line segments of Q into smaller line segments called the tiny line segments. We only keep the tiny line segments that participate in the upper envelope. Note that the part of a line segment not contributing to such an upper envelope will not contribute to the final upper envelope. The tiny line segments are x-separated i.e. they are non-overlapping. The number of these tiny line segments is equal to the complexity of the upper envelope. Each tiny line segment is defined by a single line segment.

The Partition Tree. To generate the set T from the set S, we group the line segments of the set S into k groups and compute the vertical decomposition of each group. Thus the tiny line segments formed in this decomposition form the set T, which is partitioned into k subsets, each consisting of non-overlapping segments. However such a grouping technique does not guarantee that the size of the set T i.e. the total number of such tiny line segments, is linear.

We use the technique of J. Hershberger to group the line segments of S. The idea is to create groups so that the size of the vertical decomposition of each group remains linear. Let p be a given integer. We first construct the interval tree of depth $\log p$ on the endpoints of the line segments, as presented by Neilsen and Yvinec [17]. Next we load this tree with the line segments, by allocating n line segments to the nodes of this tree, according to the lowest common ancestor of their end points. By the communication of J. Hershberger [15], the upper envelope of the line segments belonging to an internal node is linear in the number of line segments (He proves this from the fact that line segments belonging to a node cross a vertical line) and the upper envelope of the line segments belonging to different nodes at the same internal level is also linear in the number of line segments since they are x-seperated. We group the line segments of each internal level of the interval tree into groups of size p forming $O(n/p + \log p)$ groups. Also, two line segments belonging to different leaves of the tree are x separated. We group the line segments belonging to the leaves by picking one segment from each leaf and forming a group, obtaining additional n/p groups each of size p. Compute the vertical decomposition of the upper envelope of each group, obtaining $O(2n/p+\log p)$ groups each consisting of $O(p)$ x-seperated tiny line segments. The set T is the union of all these tiny line segments.

Thus we obtain a new set T, which is partitioned into $O(2n/p+\log p)$ subsets, each consisting of p x-seperated tiny line segments, such that $\mathrm{UE}(T) = \mathrm{UE}(S)$. We choose p such that $p \log p < n$.

Lemma 1. *For a fixed integer p, the set T partitioned into $O(2n/p + \log p)$ subsets each of p x-seperated line segments from the set S, such that UE(S) = UE(T) can be generated in $O(\log n \cdot \log p)$ time using $O(n/\log n)$ processors.*

Proof. Computing T requires computing the median, the lowest common ancestor of the endpoints, and $O(2n/p + \log p)$ upper envelopes each of size $O(p)$. All these can be done in the required bounds. $\square$

Remark 1. In fact the same procedure runs in expected $O(\log p)$ time using n processors with high probability.

2.2 The Marriage-Before-Conquest Procedure

We assume that the partition $\cup_{i=1}^{k} P_{\mathrm{i}}$ of the set T into k subsets such that each subset P_{i}, for $i \in [1, k]$ is a set of non-overlapping line segments, is given. Call the two vertical lines passing through the end points of a line segment as the *walls.* Let W be the set of walls and denote by $|W| = 2n$ its cardinality. We define a *slab* as the portion of the Euclidean plane E^2 between two walls. The upper envelope $\mathrm{UE}(S)$ can be described as the x-ordered sequence of edges. The Marriage-before-conquest (*MBC*) procedure computes a subsequence of the edges of $\mathrm{UE}(S)$ included in the slab B formed by the extreme right and extreme left walls. We say that a line segment spans a slab, if it intersects the slab but does not have a wall inside the slab.

Find the median w_{m} of the walls W. Compute the edge b of the upper envelope intersecting w_{m}. Split W into two subsets W_1 (Resp. W_2) consisting of walls that are to the left (Resp. right) of the edge b and not intersecting it. Call the slab formed by W_1 (Resp. W_2) as B_1 (Resp. B_2). The line segments to be considered in a slab are those that have a wall inside the slab and those that span the slab. If a slab has no walls then stop, retain the line segments spanning such a slab (we will take care of these line segments later), else recurse.

Observation 1. At every stage, each sub-problem has at least one output edge.

The Analysis. Our partitioning scheme guarantees that each sub-problem has at most one spanning line segment from a partition (since the line segments belonging to a partition are x-separated). Thus for each sub-problem, there can be at most k spanning line segments. Thus the size of the sub-problems is bounded by $O(n/2 + k)$, where $k = O(2n/p + \log p) = O(n/p)$ for $p \log p < n$ from Lemma 1.

Lemma 2. *From observation 1, if we choose the integer p such that* $H \log n \leq p$ *the size of the problem reduces to* $n/\log n$ *after* $O(\log p)$ *stages of the MBC procedure.*

2.3 The Main Algorithm

The algorithm works in two phases: *The estimation phase.* During this phase we find a good estimate of p ($p < (H \log n)^2$ and $p \log p < n$). We start with a constant value of p, run *MBC* and see if the size of the problem reduces to $n/\log n$ after $\log p$ stages. If this happens for small p our job is done. However otherwise, from Lemma 2 we know that once p is big enough that is $p \geq H \log n$, we are guarenteed this to happen. Thus p is always $< (H \log n)^2$. For every p we also check if $p \log p > n$ ($n < p \log p$, $p < (H \log n)^2, \Rightarrow H > n^{\varepsilon}$). If so, then use an $O(n, \log n)$ algorithm to solve the problem directly and stop. *The terminating phase.* During this phase we solve the problem (of size $n/\log n$) directly.

The Analysis. For a fixed p from Lemma 1, the set T can be generated in $O(\log n \cdot \log p)$ time using $O(n/\log n)$ processors. The total time spent during the *MBC* procedure is $O(\log n \cdot \log p)$ using $O(n/\log n)$ processors.

Let p_{e} denote the estimate of p, then $p_{\mathrm{e}} < (H \log n)^2$. If p_{i} is the i^{th} estimation of p, then the total time spent in this phase is $< O(\log n \sum \log p_{\mathrm{i}}) = O(\log n \cdot \log p_{\mathrm{e}}) = O(\log n \cdot (\log H + \log\log n))$. The problem of size $n/\log n$ can be solved in $O(\log n)$ time. Thus we have the following theorem.

Theorem 1. *The upper envelope of n line segments can be constructed in $O(\log n \cdot (\log H + \log\log n))$ time using $O(n/\log n)$ processors in a deterministic CRCW PRAM.*

3 The Randomized Optimal Algorithm

In this section we present a randomized algorithm using $O(n)$ processors. The algorithm is an iterative one rather than recursive. The underlying algorithm is similar to the algorithm described for the planar convex hulls by Gupta and Sen [13]. Hence the entire analysis of their algorithm goes through in our case also. However here we detail out the steps that are specific to our problem. We assume for simplicity that no more than two line segments intersect at the same point and the abscissa and ordinate of the end points of no two line segments and that of points of intersection are the same. To take care of the gaps we also introduce an extra line segment L lying below all the line segments.

The idea is to construct the upper envelope of the random sample R of the set S of staright line segments and filter out the redundant segments that do not contribute to the upper envelope of S. We pickup a sample of line segments of constant size, compute its upper envelope, discard all the redundant line segments and iterate on the reduced problem. In each successive iteration we square the size of the sample to be chosen. We iterate until either the size of the problem reduces to some threshold or the sample size becomes greater than some threshold. At this point, if the problem size reduces to less than n^{ε}, we use a brute force algorithm to solve the problem in constant time, else we use the algorithm described by Chen and Wada [7] to compute the upper envelope directly.

To prove any interesting results we must determine how quickly the problem size decreases. The Random Sampling Lemmas discussed in the next section guarantee that when a sample of size $\Omega(H^2)$ is chosen, the problem size reduces fast.

3.1 The Random Sampling Lemmas

For any subset $P \subseteq S$ consider the the two adjacent edges of the upper envelope of P. Draw two vertical lines, one through the left end point of the left edge and the other through the right end point of the right edge. We define a slab as the portion of the Euclidian plane E^2 between these two vertical lines. A configuration σ (which we call region) is the region lying above the upper envelope in such a slab. We say that the line segments adjacent to σ define σ. Notice that we include the line segment L (defined previously) in every subset P of S. From now onwards, whenever we talk of a subset P of S we would actually be meaning $P \cup L$.

We define a configuration space $\Pi(S)$ as the set of all such configurations defined by all the subsets of S. The set of line segments intersecting a region σ is called the conflict list of σ and is denoted by $L(\sigma)$ and its cardinality $|L(\sigma)|$ denoted by $l(\sigma)$ is called the conflict size. Let $\Pi^i(S)$ denote the set of configurations in $\Pi(S)$ with $l(\sigma) = i$.

Let R be a random subset of S. We define a subspace $\Pi(R)$ as the set of all such configurations $\sigma \in \Pi(S)$ whose defining line segments are a subset of R. The conflict list in R for such configurations is $L(\sigma) \cap R$. Define $\Pi^*(R)$ to be the set of configurations in $\Pi(R)$ which are in the slabs formed by the pair of edges of the upper envelope of R scanned left to right. Clearly for a region $\sigma \in \Pi^*(R)$, the set of line segments of R, conflicting with σ is empty, i.e. $L(\sigma) \cap R$ is empty. That is $\Pi^*(R) \subseteq \Pi^o(R)$.

Clearly our configuration space $\Pi(S)$ has bounded valence, since the number of regions defined by the same set of line segments is bounded (constant). In fact this number is exactly equal to one. Thus the following random sampling results due to Clarkson and Shor hold for our problem.

Lemma 3 **([9, 16]).** *For some suitable constant k and large n,*

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

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

The above lemma gives a bound on the total size of the sub-problems.

Lemma 4 **([9, 16]).** *For some suitable constant k_1 and large n,*

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

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

This lemma gives a bound on the maximum size of each sub-problem.

A sample is "good" if it satisfies the properties of Lemmas 3 and 4 simultaneously. Infact we have the following.

Lemma 5. *We can find a sample R which satisfies both Lemmas 3 and 4 simultaneously with high probability. Moreover, this can be done in $O(\log r)$ time and $O(n \log r)$ work with high probability.*

Proof. This can be done using Resampling and Polling [18]. □

We call a region "critical" if it contains at least one output vertex. Let $\Pi^*(R)$ denote the set of regions introduced by the random sample R and $\Pi_{\mathrm{h}}(R)$ denote the set of critical regions. Since $|\Pi_{\mathrm{h}}(R)| \leq H$, a good sample clearly satisfies the following property also.

Lemma 6. *For a good sample R, with $|R| = r$,*

$$\sum_{\sigma \in \Pi_{\mathrm{h}}(\mathrm{R})} l(\sigma) = O(nH \log r/r) \enspace .$$

This will be used repeatedly in the analysis to estimate the non-redundant line segments whenever $H \leq r/\log r$.

We say that a line segment is redundant if it does not intersect any critical region, i.e. either it lies entirely below the upper envelope of R or it intersects some region lying above the upper envelope but not the critical region. Consider a region that does not contain any output vertex. Clearly only one line segment is useful in this region, which is the line segment that has the maximum y-coordinate in that region. Such a line segment must intersect at least one of the regions containing an output point and is therefore retained.

3.2 The Algorithm

Let n_{i} (respectively r_{i}) denote the size of the problem (respectively sample size) at the i^{th} iteration with $n_1 = n$. Repeat the following procedure until $r_i > n^{\varepsilon}$ (this condition guarantees that the sample size is never too big) or $n_{\mathrm{i}} < n^{\varepsilon}$ for some fixed ε between 0 and 1. If $n_{\mathrm{i}} < n^{\varepsilon}$ then find out the upper envelope of n_{i} line segments directly using a brute force algorithm, else do one more iteration and find out $n_{\mathrm{i+1}}$. If $n_{\mathrm{i+1}} = O(n^{\varepsilon})$ then we use a brute force algorithm, otherwise we use the algorithm of Chen and Wada [7] to find the upper envelope of $n_{\mathrm{i+1}}$ line segments. The following is the description of the i^{th} iteration of the algorithm.

Rand-UE (i)

1. Choose a "good" sample R of size $r_{\mathrm{i}} =$ constant for $i = 1$ and $r_{\mathrm{i-1}}^2$ for $i > 1$. Find out the upper envelope of R.
2. (a) For every line segment find out the regions that it intersects.
   (b) Discard the redundant line segments lying below the upper envelope of R. (Sect. 4 discusses this in detail).
   (c) If the sum taken over all the remaining line segments, of the regions intersecting a line segment is $O(n)$ then continue else go to Step 1.
3. Filter out the segments not conflicting with any critical region as follows.
   (a) Compute $\Pi_{\mathrm{h}}(R)$ as follows.
   For every region σ do the following:
      i. Find out the line segments intersecting σ and assign as many processors to it (see Lemma 3 and Step 2(c) above).

   ii. Consider the points of intersection of the line segments with the vertical lines defining the region (the boundaries of the slab lying above the edges of the upper envelope) including the intersection points by the edges of the upper envelope. If the points with maximum y-coordinate belong to the same line segment say s, and there does not exist any segment above s lying entirely in σ, then $\sigma \notin \Pi_{\mathrm{h}}(R)$ else $\sigma \in \Pi_{\mathrm{h}}(R)$.

   (b) Delete a line segment if it does not belong to $\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)$.

4. The set of line segments for the next iteration is $\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)$ and its size is $n_{\mathrm{i}+1} = |\cup_{\sigma \in \Pi_{\mathrm{h}}(R)} L(\sigma)|$. Increment i and go to 1.

The analysis of Gupta and Sen [13] goes through here also. However detecting and deleting the redundant segments require further explanation which is done in Sect. 4. Hence we have the following results.

Lemma 7. *The upper envelope of n possibly intersecting straight line segments can be constructed in* $O(\max\{\log H, \log\log n\})$ *time with high probability using a linear number of processors.*

Remark 2. For $\log H = \Omega(\log\log n)$, this attains the ideal $O(\log H)$ running time with n processors, keeping the work optimal.

Using the standard slow-down method the algorithm is made optimal for all values of H. That is, we use $p = n/\rho$ processors where $\rho = \log\log n$, instead of n processors.

Theorem 2. *The upper envelope of n possibly intersecting straight line segments can be constructed in* $O(\log H \cdot \log\log n)$ *expected time and* $O(n \log H)$ *operations with high probability in a CRCW PRAM model where H is the size of the upper envelope.*

Proof. Refer to Gupta and Sen [13]. □

4 Finding the Redundant Line Segments

For the purpose of this section we call a segment redundant if it lies entirely below the upper envelope of the random sample. Otherwise, we call it non-redundant. Consider an upper envelope of the random sample of line segments, divided into regions (as defined in Sect. 3.1). If a line segment intersects the upper envelope at any point then it is non-redundant. However, if it does not intersect, then two cases arise: 1. Either it lies entirely above the upper envelope in which case, it is non-redundant or, 2. It lies entirely below the upper envelope in which case, it is redundant. So to determine if the line segment is redundant or not, we need to determine if it intersects the upper envelope. Let the vertices of the upper envelope spanned by a line segment be called the defining points for that line segment. If a line segment intersects the part of the envelope defined only by the defining points, then the line segment intersects the upper envelope and hence, it is declared non-redundant. But if it does not intersect this part, then if it lies above the upper envelope of its defining vertices, then it is non redundant. Else, if it lies below the upper envelope of its defining vertices, we consider the two extreme slabs which the line segment intersects. If the line segment intersects the part of the envelope in these extreme slabs then it is non-redundant, else it is redundant. Thus determining whether a line segment is redundant or not reduces to determining whether it intersects the upper envelope formed by its defining vertices.

Lemma 8. *If a line segment intersects the lower convex chain of its defining vertices then it intersects the upper envelope defined by them.*

Proof. Ommitted. □

The task of determining whether a segment is redundant or not, reduces to finding whether it intersects the lower convex chain of its defining vertices. Checking whether it lies above or below, in case it doesn't, can be done by checking it against one of the vertices of the lower convex chain in constant time. Clearly, a segment that does not intersect the lower convex chain and lies above it is non-redundant (whether it intersects the upper envelope or not). For the segment that does not intersect the lower convex chain and lies below it, we will examine the extreme slabs. Checking the extreme slabs can be done in constant time because we need to consider only two regions having a constant number of edges; check whether the line segment intersects any of these edges. The case when a line segment belongs entirely to one region is a special one in which the two extreme regions coincide. Hence, this again can be handled in constant time.

4.1 The Locus Based Approach

We use a locus-based approach to find whether the line segment is redundant. We define an equivalence relation where two segments are equivalent if and only if they have the same set of defining points. The number of such equivalence classes is $O(m^2)$. For each of these equivalence classes we take a representative line segment and precompute the lower convex chain of its defining vertices. Using a brute force algorithm we can compute the lower convex chain of the vertices associated with all the equivalence classes in constant time using $O(m^6)$ processors. Hence we have the following lemma.

Lemma 9. *Given an upper envelope of size m, we can preprocess its vertices in constant time using $O(m^6)$ processors, such that given any arbitrary line segment with k processors, we can determine whether it is redundant or not in $O(\log m/\log k)$ time.*

Proof. Since convex chains have been precomputed, the result follows from the fact that, given a convex chain C of size c and a line segment 'l', whether the line segment intersects C can be determined in $O(\log c)$ time or in $O(\log c/\log k)$ time by a k-way search using k processors. □

5 The $\tilde{O}(\log n/\log k)$ Time Algorithm

The general technique given by Sen [21] to develop sub-logarithmic algorithms can be used to design an $O(\log n/\log k)$ algorithm for the problem. Let the number of line segments be n and the number of processors be (nk), $k > \log^{\Omega(1)} n$.

1. Pick up a "good" sample R of size $r = nk^{1/4}$.
2. Compute the upper envelope of the sample using a brute force algorithm.
3. Remove all the redundant line segments (those line segments that lie completely below the upper envelope of the sample computed in Step 2).
4. For every remaining line segment find out the number of regions that it intersects. If the sum taken over all the line segments is $O(n)$ then continue else goto Step 1.
5. For every region σ do the following
   (a) Find out the line segments intersecting the region and assign k times as many processors to it. (see Lemma 3, 4 and Step 4).
   (b) If the size of the sub problem is $> \log^{\Omega(1)} n$, then recurse, else
   (c) Solve the sub-problem in constant time. With $k > \log^{\Omega(1)} n$ this can be done in constant time.
6. Merge the upper envelopes formed in all the sub problems.

Theorem 3. *Given n straight line segments and nk ($k > \log^{\Omega(1)} n$) processors, we can find their upper envelope in expected $O(\log n/\log k)$ time with high probability.*

Proof. Refer to Sen [21]. □

6 Remarks and Open Problems

One of the issues that remains to be dealt with is that of speeding up the algorithm further using a superlinear number of processors such that the time taken is $\Omega(\log H/\log k)$ with nk processors, where $k > 1$. Designing an algorithm that takes $O(\log H)$ time and does optimal work for all values of H is another open problem.

The other issue is to design a sub-logarithmic time algorithm using superlinear number of processors for $k > 1$. The technique of Sen [18, 21] to filter the redundant line segments to control the blowup in the problem size and the processor allocation scheme used by them is not particularly effective here. In that, we allocate one processor for every output vertex contributed by the segment. As the number of output vertices contributed by a segment can't be predicted, we don't know how many processors to allocate to a given line segment and the requirement of a segment as it goes down the levels of recursion may increase.

References

1. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C. K. Yap, Parallel computational geometry. in: Proc. 25th Annual Sympos. Found. Comput. Sci. (1985) 468-477; also: Algorithmica 3 (3) (1988) 293-327.
2. P. K. Aggarwal and M. Sharir, Davenport-Schinzel Sequences and their Geometric Applications. (1995).
3. S. Akl, Optimal algorithms for computing convex hulls and sorting, Computing 33 (1) (1984) 1-11.
4. H. Bast and T. Hagerup, Fast parallel space allocation, estimation and integer sorting, Technical Report MPI-I-93-123 (June 1993).
5. B. K. Bhattacharya and S. Sen, On simple, practical, optimal, output-sensitive randomized planar convex hull algorithm, J. Algorithms 25 (1997) 177-193.
6. B. Chazelle and J. Fredman, A deterministic view of random sampling and its use in geometry, Combinatorica 10 (3) (1990) 229-249.
7. W. Chen and K. Wada, On computing the upper envelope of segments in parallel, Proc. International Conference on Parallel Processing. (1998).
8. K. L. Clarkson, A randomized algorithm for computing a face in an arrangement of line segments.
9. K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387-421.
10. M. Goodrich, Geometric partitioning made easier, even in parallel, in: Proc. 9th ACM Sympos. Comput. Geom. (1993) 73-82.
11. L. J. Guibas, M. Sharir, and S. Sifrony, On the general motion planning problem with two degrees of freedom, Discrete Comput. Geom., 4 (1989), 491-521
12. N. Gupta, Efficient Parallel Output-Size Sensitive Algorithms, Ph.D. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Delhi (1998).
13. N. Gupta and S. Sen, Optimal, output-sensitive algorithms for constructing planar hulls In parallel, Comput. Geom. Theory and App. 8 (1997) 151-166.
14. N. Gupta and S. Sen, Faster output-sensitive parallel convex hull for $d \leq 3$: optimal sub-logarithmic algorithms for small outputs, in: Proc. ACM Sympos. Comput. Geom. (1996) 176-185.
15. J. Hershberger, Finding the upper envelope of n line segments in $O(n \log n)$ time, Inform. Process. Lett. 33 (1989) 169-174.
16. M. Ketan, Computational Geometry: An Introduction through Randomized Algorithms. Prentice Hall, Englewood Cliffs, NJ, (1994).
17. F.Neilsen and M.Yvinec, An output-sensitive convex hull algorithm for convex objects. Institut National De Recherche En Informatique Et En Automatique. (1995).
18. S. Rajasekaran and S. Sen, in: J. H. Reif, Ed., Random Sampling Techniques and Parallel Algorithm Design (Morgan Kaufmann Publishers, San Mateo, CA, 1993).
19. J. H. Reif and S. Sen, Randomized algorithms for binary search and Load Balancing on fixed connection networks with geometric applications, SIAM J. Comput. 23 (3) (1994) 633-651.
20. J. H. Reif and S. Sen, optimal parallel randomized algorithms for 3-D convex hull, and related problems, SIAM J. Comput. 21 (1992) 466-485.
21. S. Sen, Lower bounds for parallel algebraic decision trees, parallel complexity of convex hulls and related problems, Theoretical Computer Science. 188 (1997) 59-78.