ZBORNIK RADOVA
Prirodno-matematičkog fakulteta
Univerziteta u Novom Sadu
Serija za matematiku, 16,2(1986)

REVIEW OF RESEARCH
Faculty of Science
University of Novi Sad
Mathematics Series, 16,2(1986)

PARALLEL COMPUTATION OF VORONOI DIAGRAMS

Ivan Stojmenović and Marija Kulaš

University of Novi Sad, Faculty of Science,
Institute of Mathematics, Dr I. Djuričića 4,
21000 Novi Sad, Yugoslavia

ABSTRACT

The paper presents algorithms for solving some computational geometry problems on a shared memory parallel computer, where concurrent reads are allowed but no two processors can simultaneously attempt to write in the same memory location. We present new $O(\log^3(n))$ time with $O(n \log(n))$ processors algorithm for construction of the Voronoi diagrams and show that the algorithm given in [1] for solving the same problem does not work correctly. Also, we make the convex hull algorithm described in [1] more efficient by omitting their last step which is superfluous due to a too weak lemma they have given.

1. INTRODUCTION

Since they involve asking basic questions about sets of points, lines, polygons, etc, geometric problems arise often in many applications [9]. We are interested in finding parallel algorithms solving some of these problems which are efficient both in terms of their running time and in the number of processors used. In this paper we shall consider two basic geometric notions: the Voronoi diagram and the convex hull of a set of n points.

AMS Mathematical Subject Classification (1980): Primary 68U05,
Secondary 68Q10.
Key words and phrases: Parallel computation, computational geometry, Voronoi diagram, convex hull.

Throughout this paper, the computational model used is the shared memory in which concurrent reads are allowed, but no two processors should attempt to simultaneously write in the same memory location. We shall henceforth refer to this model as the CREW PRAM (concurrent read exclusive write parallel RAM).

In [9] it is pointed out that the Voronoi diagram can be used as a powerful tool to give efficient algorithms for a wide variety of other geometric problems. For example, from the Voronoi diagram we can easily obtain the closest neighbour of each site (point), or the Delaney triangulation, or the Euclidean minimum spanning tree of the n sites, or the minimal distance between two convex sets, or the largest point free circle with center inside their convex hull, et cetera.

Geometric algorithms for the CREW PRAM model of computation have been studied in literature. We shall mention two optimal parallel algorithms for computing the planar convex hull [1,3] and 3-dimensional convex hull [4,1] , an efficient algorithm for selecting the closest pair of points [3] and an algorithm for computing the minimal circumscribing triangle to a convex n-gon [1] .

2. PARALLEL COMPUTATION OF VORONOI DIAGRAMS

In [1] a $O(\log^3(n))$ time with $O(n)$ processors algorithm to compute the planar Voronoi diagram of a set of n points is presented. We show that the algorithm does not work correctly because the main problem in the merge step of the construction is not solved (or even mentioned) and give a correct algorithm.

Let S be a set of n points. Given any p in S, the (open) Voronoi cell V(p,S) consists of all the points in the plane closer to p than to any other point in S; it is well-known that this is a planar graph with O(n) straight-line edges, O(n) vertices, and the n cells are bounded by convex polygons. In nondegenerate cases each Voronoi vertex is defined by three points from S, and each Voronoi edge is the bisector of two points from S. First sequential method for computing Vor(S) is presented in [10].

Suppose that S is partitioned into P U Q (i.e. there is a vertical line L which has all of P to its left and all of Q to its right). This can be accomplished by sorting the points in O(log(n)) time and O(n) processors [2,6]. Recursively find Vor(P) and Vor(Q). The merging of Vor(P) and Vor(Q) is defined by by a polygonal path t, called the contour, separating P from Q. The contour t cuts the plane into a left portion t(l) and a right portion t(r). Then the Voronoi diagram Vor(S) is the union of Vor(P) ∩ t(l) and Vor(Q) ∩ t(r).

The main task of a parallel algorithm is to identify those edges (and hence, those cells) of Vor(P) and Vor(Q) which intersect the contour. As claimed in [1], an edge in Vor(P) intersects the contour if and only if one end is closer to P and the other end is closer to Q, so to identify such edges it is enough to locate their endpoints in the appropriate cells of Vor(Q) and compare distances [1].

The main problem in the algorithms given in [1] is that they assume that all the vertices of Vor(P) lying inside H(P) (the convex hull of P) are certainly vertices of Vor(P ∪ Q) (similarly for vertices of Vor(Q)). Following the assumption, they locate only the vertices of Vor(P) lying outside H(P) into sectors (see definition in [1]) of R^2-H(Q) (the complement of H(Q)) and into the correct Voronoi cell for Q if the cell is not completely inside H(Q). This enables them to use the binary tree structure of Vor(P)-H(P) to give simpler constructions for locating points. Let us consider the counterexample on Fig. 1.

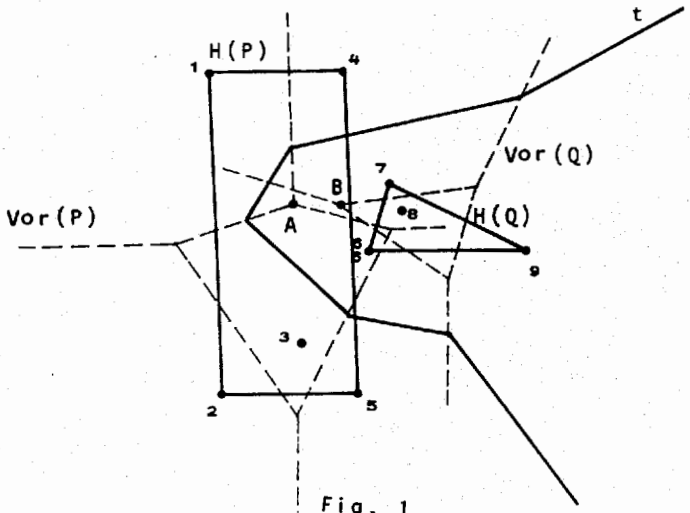

Fig. 1

The vertex A of Vor(P) lies inside H(P), but it is not a vertex of Vor(P ∪ Q) while vertex B of Vor(Q) lies inside H(P) but still remains as a vertex of Vor(P ∪ Q).

Thus, it is necessary to determine for each vertex of Vor(P) the appropriate cell in Vor(Q) and similarly for each vertex of Vor(Q) the appropriate cell in Vor(P) in order to determine the edges of contour t and the endpoints of the edges.

In order to locate a point into Voronoi cells of P or Q we shall use the chain method described by Lee and Preparata in [5,9] for planar straight-line graphs. First we shall describe a data structure to support the planar location of a point into appropriate Voronoi cell of Vor(S).

We can organize all the contours that appear in the divide-and-conquer solution as nodes of a balanced binary tree (Fig. 2). Each contour is a single monotone chain (cf. [10,9]. Here a monotone chain is a chain of edges such that any horizontal line intersects it in exactly one point. Also, in the process contours will define a monotone complete set of chains (i.e. the set of chains so that for any two chains c(1) and c(2) the vertices of c(1) that are not on c(2) are on the same side of c(2)). We shall label each contour by the number

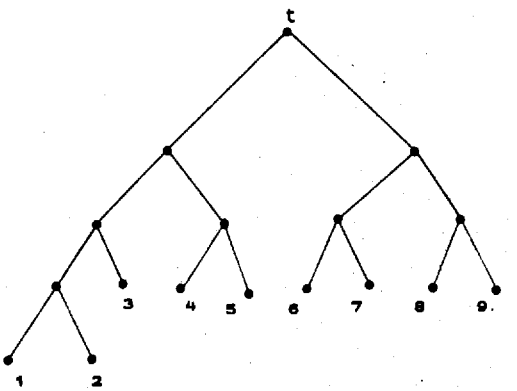

Fig. 2.

of points from S that lie to their left. So, each Voronoi cell is situated between two neighbouring contours. In the divide--and-conquer process, each edge of a new contour is assigned to the contour. However, each new contour can intersect some of the previous ones. In this case all the intersected contours are modified by discarding their edges that are eliminated from the Voronoi diagram and including corresponding part of the new contour instead. Thus not all the edges that belong to a contour are necessarily assigned to the contour because each edge of the Voronoi diagram is assigned exactly ones at the time when it appears in a contour by a merge. Afterwords the edge can only be intersected or discarded but it cannot be assigned to another contour. If an edge e belongs to more than one Voronoi chain (i.e. contour) it belongs to all the members of a set (an interval) of consecutive chains. Each edge e stores the endpoints of the corresponding interval [C1(e),C2(e)] of Voronoi chains. Then there is a unique member C(e) of the interval [C1(e),C2(e)] which, in the search tree, is a common ascendant of all the other members of the interval (i.e. hierarchically the highest chain to which e belongs). Let C'be any such other member; then the discrimination (deciding on which side of C' the query point P lies) of a point P against C' is preceded - in the binary search scheme - by the discrimination of P against C(e). The computing of the common predecessor for the interval of chains in a standard hierarchy is described in [5,7] .

We have preprocessed our subdivision as in [5,7,8,9] . Note that the discrimination of a point against a chain C is effected in time O(log(n)) since each chain is the header of an adjacency list which contains the edges assigned to C. There are at most log(n) such discriminations since chains are organized in a balanced binary tree. So, the entire point location can be achieved in $O(\log^2(n))$ time (see [5,7,8,9] for details). Apparently the technique can be used for the location of O(n) points stored one point per processor in the same planar subdivision in $O(\log^2(n))$ time in parallel [7].

Now, we can describe the merge step of our algorithm. Infinite rays of the contour can be computed as bisectors of tangent-lines of H(P) and H(Q) ([4,1,3]). We assign a processor to each edge of Vor(P) and Vor(Q). Let e=AB be an edge of Vor(P). Using the above technique we can locate A and B into Voronoi cells Vor(A1,Q) and Vor(B1,Q) of Vor(Q), respectively. Infinite rays can just compare their slopes with the slopes of infinite rays of contour. Then A and B compute their distances to the points A1 and B1 which are respectively their nearest points in P. In [1] it is claimed that there are three possible cases:

(1) both A and B are closer to P than to Q. Then AB remains a Voronoi edge [1];

(2) both A and B are closer to Q than to P. Then AB can be discarded from the Voronoi diagram [1] ;

(3) A is closer to P than to Q, but B is closer to Q than to P (the remaining case is discussed in a similar way). Then AB intersect the contour of the Voronoi diagram. The point B is discarded from Vor(S). Suppose AB is a bisector of points P1 and P2 in P. Then a new Voronoi vertex R is obtained as described below.

The set of all the new vertices R found in (3) can be sorted by their y-coordinates [2,6] and every edge defined by two neighbouring vertices in the sorted order is a Voronoi edge. The points that an edge UV is bisector of are determined by the common points associated with both U and V. For each new edge the endpoints of corresponding interval of chains it belongs to are just labels (ranks) of points the edge is bisector of. All the contours and edges of Vor(Q) that are included in Vor(S) increase their labels by |P| (the number of points from P).

The algorithm is complete and works in $O(\log^3(n))$ time with $O(n \log(n))$ processors.

We are now going back to (3). Let the marked edges in Vor(P) or Vor(Q) be the edges for which one end is closer to P and the other end is closer to Q. A subtechnique for computing new Voronoi vertices, i.e. the points where marked edges intersect the contour presented in [1] is the next inaccurate place in [1].

Let P' and Q' denote the sets of marked points in P and Q respectively, i.e. the points in P and Q whose cells in Vor(Q) intersect the contour (the boundaries of these cells contain at least one marked edge). In [1] it is supposed that the marked points are sorted according to the order in which their cells intersect the contour. However, computing such an order is not a trivial problem, since this order is not necessarily the same as the order obtained by sorting according to the y-coordinates of points. Moreover, some points can appear twice in the sorted list of cells which intersect the contour. In the example in Fig. 1 corresponding orders are P'=(4,1,3,5), Q'=(9,7,6,9). Each marked edge determines a pair of neighbouring marked points in the sorted list P' (or Q').

We give the correct technique for ordering the marked points into a sorted list P'. A neighbouring pair of marked points is determined by a marked edge. Choose the endpoints of a neighbouring pair (of marked points) as the first and the last point in P'(and discard the pair from considerations in steps below). We have to compute the rank of each point in P'. Let the distance between two points in P' be the difference between their ranks in P'. For simplicity, we shall assume $n=2^k$ although the method can be easily modified to run for arbitrary n. In log(n) steps we proceed as follows:

For i=1 to i=log(n)-1 we can compute all the pairs of marked points that are at distance 2^i from each other. One such pair EF is determined by a point G such that G is at distance 2^{i-1} from both E and F.

Then, for i=log(n)-1 down to i=1 the points with ranks $m*2^i$ ($1 \leq i<n/2$, m odd) are found since for each of these points the ranks of two points which are at distance 2^i from the point are already computed.

For example let n=8 we are given circuit edges EF, CD, GH, DE, IJ, FG, HI where the ranks of C and J are 1 and 8 respectively. For i=1 we generate CE, EG, GI, DF, FH, HJ and for i=2 we get CG, EI, DH, FJ. Then, for i=2 G has rank 5 and F has rank 4. For i=1 we compute the ranks of D, E, H, I to be 2,3,6,7, respectively.

Our method requires O(log(n)) steps and O(n*log(n)) space on a CREW PRAM.

We present a correct method for finding new Voronoi vertices. Choose the marked edge e determined by two median points A and B from P'. Let U and V be the endpoints of e such that U is closer to P than to Q. By assigning one processor to each point T' from Q' it is possible to determine in constant time for each T', the intersection T between e and the bisector of AT', and the distance of T to U. Then a point R' from Q' is chosen so that the considered distance RU is the smallest (we shall consider only the intersection points that lie between U and V). R is a new Voronoi vertex. It is easy to see that marked edges from the sorted list P' above (respectively, below) e can only interact with points above (below) R' in the sorted list Q'. This partitioning is iterated log(n) times, after which all the contour edges can be computed.

3. PARALLEL CONVEX HULL IN THE PLANE

In [1] and [3] two similar convex hull algorithms which run in O(log(n)) time on a CREW PRAM with O(n) processors are described. We shall follow the algorithm in [1] and show that their last step is superfluous because they gave too weak a lemma.

Their algorithm divides the sorted list of n points from the given set S into $n^{1/2}$ sets of $n^{1/2}$ points, recursively the upper convex chain of each set computes, and then merges all these chains together in O(log(n)) time (similarly for the lower convex hull chains). The merge step begins by assigning a processor for each pair of $n^{(1/2)}$ upper chains and finding the "upper tangent line segments" of them. Consider the graph G whose vertices are the points of S and whose O(n) edges are either in the upper chains or among the just-computed upper tangent segments. After sorting the edges by the x-coordinate of their first components, all the edges with the same x-coordinate in the first component form a contiguous block. Similarly we can sort all the edges by the x-coordinate of their second component (this step is not mentioned in[1]). For each point $v \in S$, if E(v) denotes the edges incident to v, assign $|E(v)|$ processors to v, and in O(log(n)) parallel time, sort E(v) by the slope of the

edges. Notice that if v is a vertex on the upper chain of H(S), but not one of the two extremal vertices, then among the edges adjacent to v there are two cyclically adjacent edges which form a reflex angle with the interior pointing upward. Delete all the vertices (except the extremal vertices) which do not have two such incident edges, delete all but these two edges from those which do, and delete all the edges from the leftmost (respectively rightmost) extremal vertex except that with maximal (respectively minimal) slope. Denoting the trimmed graph by G´ in [1] the following lemma is stated.

LEMMA. Graph G´consists of one or more vertex chains of edges. One of these chains connects the extremal vertices, and this is the upper chain for H(S).

However, this assertion is too weak. We can prove a stronger result:

LEMMA. Graph G´is exactly the upper convex chain of S.

PROOF. It is obvious that all the vertices of G´ are vertices of the upper convex chain of S and for each vertex of G´ there are exactly two edges from G´ incident to the vertex (except the rightmost and leftmost vertices which are incident to only one edge). Therefore, no edge can be discarded from G´in order to obtain the upper convex hull.

Thus, the last step described in [1] is superfluous.

REFERENCES

[1] Aggarval,A., Chazelle,B., Guibas,L., O´Dunlaing,C., Yap,C., *Parallel computational geometry, IEEE Annual Symp. Found. Comp. Sci. 1985, 468-477.*

[2] Ajtai ,M., Komlos,J., Szemeredi,E., *An O(n log(n)) sorting network, Combinatorica 3, 1, 1983, 1-19.*

[3] Atallah,M.,J., Goodrich,M.T., *Efficient parallel solutions to geometric problems, IEEE Symp. Parallel Processing, 1985, 411-417.*

[4] Chow,A.L., *Parallel algorithms for geometric problems, Ph. D. Thesis, Comp. Sci. Dept., Univ. of Illinois at Urbana-Champaign, 1980.*

[5] Lee,D.T., Preparata,F.P., *Location of a point in a planar subdivision and its applications, SIAM J. Comput., 6,3, 594-606, 1977.*

[6] Leighton,T., *Tight bounds on the complexity of parallel sorting, IEEE Trans. Comput., C-34, 4, 1985, 344-354.*

[7] Lu,M., *Constructing the Voronoi diagram on a mesh-connected computer, IEEE Int. Conf. on Parallel Processing, 1986, 806-811.*

[8] Preparata,F.P., *A note on locating a set of points in a planar subdivision, SIAM J. Comput., 8, 4, 1979, 542-545.*

[9] Preparata,F.P., Shamos,M.I., *Computational Geometry, An Intoduction, Springer-Verlag, New York, 1985.*

[10] Shamos,M.I., Hoey,D., *Closest point problems, Proc. of the 16th IEEE Symp. on Found. of Computing, 1975, 151-162.*

REZIME

PARALELNO IZRAČUNAVANJE DIJAGRAMA VORONOJA

Rad prikazuje geometrijske algoritme za model paralelnog kompjutera u kome je dozvoljeno čitanje iz iste lokacije, ali ne i upis u isti registar. Prikazan je novi algoritam za paralelnu konstrukciju Voronoi dijagrama i pokazano da je raniji algoritam dat u [1] netačan. Takodje, algoritam za izračunavanje konveksnog omotača dat u [1] je napravljen efikasnijim izbacivanjem jednog suvišnog koraka.

Received by the editors November 4, 1986.