Univ. u Novom Sadu
Zb. Rad. Prirod.–Mat. Fak.
Ser. Mat. 23, 1 (1993), 189 - 197

Review of Research
Faculty of Science
Mathematics Series

MULTILOCATION OF POINTS IN A VORONOI SUBDIVISION IN PARALLEL

Ivan Stojmenović
Computer Science Department, University of Ottawa
Ottawa, Ontario, Canada K1N 9B4

Ljubomir Jerinić [1]
Institute of Mathematics, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Yugoslavia

Abstract

The paper presents a parallel algorithm for location of $O(n)$ points in a subdivision induced by a Voronoi diagram in the plane. The technique is based on the chain method [10] for sequential location of multiple points in a planar subdivision. It runs in $O(\log^2 n)$ time using $O(n)$ processors on a hypercube or cube-connected cycle. The algorithm leads to $O(\log^3 n)$ time algorithm for the construction of Voronoi diagram of a set of n points on a hypercube or cube-connected cycle with $O(n)$ processors, which improves a former result by A. Chow [4].

AMS Mathematics Subject Classification (1991): 86U05, 68Q10
Key words and phrases: Computational geometry, parallel computation, point location, hypercubes, cube-connected cycles, Voronoi diagram.

[1] Work supported by Fund for Science of Serbia.

1. Introduction

Let S be a set of n points in the plane. To simplify the exposition, we assume that no four points are cocircular. Given any point p in S, the (open) Voronoi cell $V(p,S)$ consists of all points in the plane closer to p than 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 bisector of two points from S. The first sequential method for computing the Voronoi diagram $Vor(S)$ is presented in [12].

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 one can easily obtain the closest neighbor of each site (point), or the Euclidean minimum spanning tree of the n sites, or the Delaney triangulation, or the minimal distance between two convex sets, or the largest point-free circle with center inside their convex hull, etc.

We consider the following problem, called "multilocation": Given a plane subdivision R induced by a Voronoi diagram graph $G = (V, E)$, with $|V| = n$, and a set S of $m = O(n)$ query points in plane, we wish to locate in parallel S in R, i.e., to find the region of R containing Z for each query point Z in S.

The multilocation problem was studied in literature for general planar subdivision. The chain method that solves the problem in $O(\log^2 n)$ time per query point (after suitable preprocessing in $O(n \log n)$ time) on a sequential computer is described in [6,10]. That algorithm transforms an arbitrary subdivision into a monotone one and then locates query point in such subdivision. A parallelization of the chain method that runs in $O(\sqrt{n} \log n)$ time on a mesh connected computer with $O(n)$ processor is given in [8]. In fact, the algorithm [8] is designed to work correctly only for a special class of monotone subdivisions that include Voronoi diagrams. There exist several sequential techniques to locate given query point in $O(\log n)$ time after $O(n \log n)$ preprocessing (see [10] for references). In [2] a multilocation data structure based on the segment tree was constructed in parallel and the multilocation problem is solved in optimal $O(\log n)$ time on a CREW PRAM with $O(n)$ processors.

Recently, Lee and Preparata in [7] described an $O(log^2 n)$ time with

$O(n \log n)$ processors, or $O(\log^3 n)$ time with $O(n)$ processors algorithm to solve the general multilocation problem on a CCC (cube-connected cycles) model of parallel computation. Their algorithm is also based on the segment tree data structure.

In this paper we propose an algorithm that solves the multilocation problem for monotone subdivisions induced by Voronoi diagrams in $O(\log^2 n)$ time on a CREW PRAM, on a CCC or on a hypercube with $O(n)$ processors and constant memory per processor. The algorithm is based on the chain method algorithm [6] and is a more sophisticated parallelization (because of the nature of interconnection networks used in the paper) than the one given in [6] for mesh-connected computers.

It is well known that any algorithm that runs in time $T(n)$ on a theoretical CREW PRAM model can be simulated on a hypercube or a CCC and will run in $O(T(n) \log^2 n)$ time. Using this it immediately follows that the segment tree approach of [2] leads to an $O(\log^3 n)$ algorithm to solve the multilocation on a hypercube or a CCC with $O(n)$ processors. Since the parallel chain method described in this paper runs in $O(\log^2 n)$ on a CREW PRAM, the straightforward simulation would lead to $O(\log^4 n)$ time solutions on a hypercube and a CCC. However, we are able to keep the same time complexity $O(\log^2 n)$ on two interconnection network models. This is an example of an interesting paradigm: the best algorithms for CREW PRAM does not necessarily lead to the best algorithms on interconnection network model like hypercube or CCC.

Using our algorithm the Voronoi diagram of a set of n points can be solved in $O(\log^3 n)$ time on a CCC or a hypercube with $O(n)$ processors, which improves a former result by A. Chow [4].

2. Models of parallel computation

One computational model mentioned in the paper is the synchronous shared memory SIMD (single instruction multiple data) model in which concurrent reads are allowed, but no two processors should attempt to simultaneously write in the same memory location. We refer to this theoretical model as the CREW PRAM (concurrent read exclusive write parallel random access machine).

We use two practical interconnection network models: a hypercube and

a CCC (cube-connected cycles).

A d-dimensional hypercube is a set of $n = 2^d$ synchronized processing elements, called nodes, linked together in a d-dimensional binary cube network. Each node has associated a constant size memory and is given a unique d-bit identification number. Two nodes i and j ($0 \leq i, j < 2^d$) are connected by a bidirectional communication link if and only if the binary representations of i and j differ in exactly one bit. Thus, each node has d communication links.

The other interconnection network SIMD model is known as cube-connected cycles (CCC), and is described as follows. Consider a d-dimensional cube. Each of the 2^d corners of the cube is a cycle of d processors; each processor in a cycle is connected to a processor in a neighboring cycle in the same dimension. The number of nodes is therefore $n = d2^d$.

We described the basic data communication techniques used in our algorithm; our algorithm does not use other data communication techniques.

Sorting. Given an element per processor, the sorting can be done in $O(\log^2 n)$ time on a hypercube [3,14] and CCC [11]. After sorting the elements are kept in nodes in the lexicographic order; for CCC we assume a hypercube order with the cycles of d processors forming intervals of consecutive elements.

Merging. Given two sorted arrays A and B, stored one element per processor, and in two consecutive intervals of processors, their merging can be done in $O(\log n)$ time on a hypercube [14] and CCC [11].

Compression. Some nodes contain "active" elements while others do not. Compress the active elements, i.e. store them in nodes $0, 1, 2, \ldots, s-1$ where s is the number of active elements, and store the non-active elements in nodes $s, s+1, \ldots, n-1$, keeping in both cases their relative order. A $O(\log n)$ solution is described in [9].

Interval broadcast. Certain of nodes $0, 1, \ldots, n-1$ are leaders; they possess data that they must share with all the higher numbered nodes, up to but not including the next leader (the interval of nodes between two leaders). Interval broadcasting can be done in $O(\log n)$ time on a hypercube [9,14] or CCC.

3. Multilocation

A planar stright line graph $G = (V, E)$ is determined by its vertices and edges (that are stright line segments). If G contains no vertex of degree less then 2, then all the bounded regions of the plane subdivision R induced by G are simple polygons. Without loss of generality, G is assumed to be connected. The input of a multilocation problem therefore consists of $O(n)$ vertices and edges of a planar subdivision R such that each edge is assigned the labels of two regions that it separates, and a set S of $m = O(n)$ query points. The task is to find the region of R containing Z for each query point Z in S.

A simple polygon is said to be monotone (with respect to y axis) if its intersection with any vertical line is connected (consists of one interval only). A plane subdivision is monotone if all of its regions are monotone polygons. The solution to the multilocation to be presented in this paper requires that the subdivision is monotone. Monotone subdivisions include all triangulations, and subdivisions into convex regions among others. An important instance of the later subdivision is the Voronoi diagram of a set of points in the plane. Multilocation is a part of the merging step in divide-and-conquer construction of the Voronoi diagram.

The sequential multilocation algorithm [6,10] refines first an arbitrary subdivision into one which is monotone by a sweep-line procedure, and then locates query points inside the monotone subdivision.

For two regions r_i and r_j of R, $r_i < r_j$ if they share a common edge and $p.x < q.x$ for any two point p from r_i and q from r_j that belong to the same horizontal line (i.e. $p.y = q.y$); here $p.x$ and $p.y$ are $x-$ and $y-$coordinates of a point p, respectively. The relation "$<$" is acyclic if applied to the regions of a monotone subdivision. This means that a linear order of all regions can be found that is compatible with the partial order relation "$<$". If the subdivision R is a Voronoi diagram of a set S of points in the plane then the linear order is exactly the same as obtained by sorting the points from S by their $x-$coordinates (such an ordering is applied in [8]). Otherwise (for arbitrary monotone subdivision) a more sophisticated procedure is required.

First we sort regions by $x-$coordinates of their centers. The median center in the sorted list is used to divide centers and corresponding regions into two halves. There is a chain of edges which separate half regions to the left from half to the right. The chain is a single monotone chain, i.e. chain

of edges such that any horizontal line intersect it in exactly one point. We apply recursively the same division to the left and right halves of regions and obtain 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). Thus, chains are nodes of a balanced binary tree the leaves of which correspond to the regions of planar subdivision. Chains may share common edges. If an edge e belongs to more that one chain it belongs to all members of a set (an interval) of consecutive chains. This interval corresponds to the interval from one center of e to the other (in sorted list of centers). There is unique member c of this interval which, in the binary search tree, is a common ascendant of all members of the interval (i.e. hierarchically the highest chain to which e belongs). We assign e to such a member c. Then the discrimination (deciding on which side of c_1 the query point Z lies) of point Z against c_1 is preceeded by the discrimination of Z against such a member c. By $O(\log n)$ discriminations each query point can be located.

Each chain has its level and index (the rank of the chain in the set of chains of given level). Each edge is assigned to exactly one chain by the rule described in [8]: find the "bit exclusive **or**", say μ, of the binary indices of centers of e. The level of e (denoted by e') can be found by $|\log \mu| = e'$. Then,

$$\frac{2's \text{ complement } (2^{e'}) - 2^{e'} \text{ bitwise-}\mathbf{and} \text{ (index of center of } e \text{)}}{2^{e'+1}}$$

is the index of the chain to which e belongs. For an example, if centers of e are 0010 and 0101, "bit exclusive **or**" of 0010 and 0101 is 0111, $|\log 0111| = (2)_{10}$, so e is of level 2. Furthermore, $2's$-complement (2^2) $-$ 2^2 $=$ $2's$ - complement (0100) $-$ 0100 $=$ 1100 $-$ 0100 $=$ 1000. (1000 bitwise-**and** 0010 $=$ 0 (**or** 1000 bitwise-**and** 0101 $=$ 0); so e indexed as 0 in the chains of level 2. Thus level and index of each edge can be found in constant time in parallel.

The parallel solution proceeds as follows.

Sort all edges by their level as primary key, their index as the secondary key, and the $y-$coordinate of the lower endpoint of edge as the ternary key. Also, we sort the set S of all query points by their $y-$coordinates. Initially, all query points are assigned highest level $\log n$ and index 0. Next, do the following loop.

For each level i, from $i = \log n$ **to** $i = 0$ **do**:

(i) Merge the set of edges with the set of query points, using current indices of query points; note that all query points have the same level, equal to i, and therefore the merging will be done practically only with edges having level i,

(ii) Perform interval broadcast to find, for each query point Z, the corresponding edge e the query point should be disciminated against. If the $y-$coordinate of Z is not between $y-$ coordinates of endpoints of e then Z has been discriminated at level before. Depending on which side of e the query point Z is, Z calculates the index of chain at the next level it should be descriminated (we refer to this as the new index of given query point). If Z is to the left of corresponding edge we call it "left" query point, otherwise we call it "right" one; if the current index of query point is k then the new index is either $2k$ or $2k+1$, for "left" and "right" query point, respectively.

(iii) Unmerge the set of edges and the set of query points by applying the compression technique (edges are considered active elements for the compression purpose), i.e. return them to the position at the beginning to the step (i).

(iv) Re-sort query points by their new indices, by applying the following two steps:

- a compression technique, where the active elements are "left" query points in step (ii); note that both sets of active and non-active elements are sorted according to their indices (both current and new),
- a merging technique, using the new indices of query points (i.e. merging "left" and "right" query points);

(v) Assign the next level to all query points and return to the step (i).

All query points will be located in the step (ii) when $i = 0$. Since all steps (i)-(v) take $O(\log n)$ time, the time complexity of planar point location algorithm is $O(\log^2 n)$. This is valid for all three considered models: CREW PRAM, hypercube and CCC.

4. Voronoi diagram

An $O(\log^3 n)$ time algorithm to construct Voronoi diagram of a set of n planar points on a hypercube or CCC with n processors and constant memory per processor can be obtained by using Jeong and Lee [5] algorithm to solve the problem on mesh-connected computers, multilocation technique and data communication techniques on these models. It can be shown in a rather straightforward way that all operations in the merge step of the algorithm [5] can be implemented in $O(\log^2 n)$ time. Note that Chow [4] has proposed an $O(\log^4 n)$ algorithm for the construction of Voronoi diagram on CCC with $O(\log n)$ storage per processor. Since the algorithm can be implemented on hypercube or CCC models with no penalty or benefit, our proposed solution improves both time complexity and storage per processor, on boath interconnection networks.

References

[1] Akl, S. G., The Design and Analysis of Parallel Algorithms. Prentice-Hall, New York, 1989.

[2] Atallah, M. J., Cole, R., Goodrich, M. T., Cascading divide-and-conquer: a technique for designing parallel algorithms. In Proc. 28th IEEE symp. on Found. Comp. Sci., 1987, 151-169.

[3] Batcher, K. E., Sorting networks and their applications. Proc. Spring Joint Computer Conf., 32, AFIPS Press, Montvale, New York, 1968, 307-314.

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

[5] Jeong, C. S., Lee, D. T., Parallel geometric algorithms on mesh-connected computers, Proc. Fall Joint Computer Conf., 1987.

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

[7] Lee, D. T., Preparata, F. P., Parallel batched planar point location on the CCC, Inform. Process. Lett., 33 (1989), 175-179.

[8] Lu, M., Constructing the Voronoi diagram on a mesh-connected computer. Proc. of IEEE Int. Conf. on Parall. Proc., 1986, 806-811.

[9] Nassimi, D., Shani, S., Data broadcasting in SIMD computer, IEEE Trans. Comput. C-30, 2, Feb. (1981), 101-106.

[10] Preparata, F. P., Shamos, M. I., Computational Geometry, An Introduction. Springer-Verlag, New York, 1985.

[11] Preparata, F. P., Vuillemin, J., The cube-connected cycles: a versatile network for parallel computation, Comm. ACM 24(5), (1981), 300-309.

[12] Shamos, M. I., Hoey D., Closest point problems. Proc. of the 16th IEEE Symp. on Found. of Comp. Sci., 1976, 208-215.

[13] Stojmenović, I., Computational geometry on a hypercube. Proc. Int. Conf. on Prallel Processing, Vol. III, 1988, 100-103.

[14] Ullman, J. D., Computational aspects of VLSI. Comp. Sci. Press, Potomac, MD, 1984.

REZIME

PARALELNA LOKACIJA VIŠE TAČAKA U VORONOI DIJAGRAMIMA

Rad prikazuje jedan paralelni algoritam za lokaciju $O(n)$ tačaka u ravni izdeljenoj dijagramom Voronoja. Metod je zasnovan na sekvencijalnom lančanom metodu i izvešava se u $O(\log^2 n)$ koraka sa $O(n)$ procesora na modelu višedimenzionalne kocke ili ciklusa povezanih kockama. Na osnovu tog algoritma može se konstruisati dijagram Voronoja u $O(\log^3 n)$ koraka sa $O(n)$ procesora na modelu višedimenzionalne kocke ili ciklusa povezanih kockama, što predstavlja poboljšanje rezultata A. Chow-a [4].

Received by the editors February 1, 1987, revised September 10, 1989.