Information Processing Letters 22 (1986) 303–306
North-Holland

30 May 1986

$O(n^{1/2})$ ALGORITHMS FOR THE MAXIMAL ELEMENTS AND ECDF SEARCHING PROBLEM ON A MESH-CONNECTED PARALLEL COMPUTER

Frank DEHNE

Lehrstuhl für Informatik I, Universität of Würzburg, Am Hubland, D-8700 Würzburg, Fed. Rep. Germany

Communicated by P.C. Poole
Received 12 July 1984
Revised 1 August 1985

Consider a set S of n points in the Euclidean plane. We obtain efficient parallel algorithms for the following two problems: (i) compute the contour spanned by the maximal elements of S, and (ii) compute the number of dominated points for every element of S (ECDF searching problem). Both algorithms run in $O(n^{1/2})$ time on a mesh of n processors, which is asymptotically optimal since any nontrivial computation requires $\Omega(n^{1/2})$ time on the mesh. The algorithms can be generalized to solve the d-dimensional maximal elements and ECDF searching problem in $O(n^{1/2+\log_2(d-2)})$ $(d > 2)$ time.

Keywords: Computational geometry, ECDF searching problem, maximal elements, mesh of processors, parallel processing

1. Introduction

Chazelle [1] has recently studied the parallel complexity of a number of problems in computational geometry. In this paper we explore two further problems from a similar point of view for implementation on a mesh of processors. A mesh-connected parallel computer of size n is a set of n synchronized processing elements (PEs) arranged in a $\sqrt{n} \times \sqrt{n}$ grid. Each PE is connected via bidirectional unit-time communication links to its four neighbors, if they exist (see Fig. 1).

Each processor has a fixed number of registers and can perform standard arithmetic and comparisons in constant time. It can also send the contents of a register to a neighbor and receive a value from a neighbor in a designated register in O(1) time units. Each PE in the leftmost column has an I/O port. Thus, we can 'load' S in $O(n^{1/2})$ time units such that each processor contains exactly one arbitrary point of S.

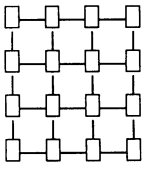

Fig. 1. A mesh-connected parallel computer of size 16.

We may think of these processors as individual VLSI chips or several chips each containing some part of the grid on a circuit board (cf. [4,9]).

The problems considered from computational geometry are the following. Let $S = \{s_1, \ldots, s_n\}$ be a set of n points in the Euclidean plane. (To simplify the exposition of our algorithms we assume that $n = 4^k$ for some integer k and all points have distinct x- and y-coordinates.)

Given a point p, p.x and p.y denote the x- and y-coordinate of p, respectively.

Definition 1.1. A point q *dominates* a point p, written $p \leqslant q$, iff $p.x \leqslant q.x$ and $p.y \leqslant q.y$. A point $s \in S$ is called *maximal* if there is no other $s' \in S$ with $s \leqslant s'$.

We are interested in computing the contour spanned by the maximal elements of S, called the m-*contour* of S.

From a more general point of view, maximal element determination is a special case of the ECDF searching problem. The ECDF searching problem consists of computing for each $p \in S$ the number $D(p, S) := |\{q \in S \mid q \leqslant p\}|$. (D is called the 'empirical cumulative distribution function'.)

It is well known that the time complexity for computing the maximal elements and the m-contour is $\theta(n \log n)$ using a sequential computer (see [3]). The ECDF searching problem has the same time complexity $\theta(n \log n)$ on a standard computer. Consult [7] for more details and applications (see also [6]).

In this paper we will give efficient algorithms for solving both problems on a mesh of n processors (with a constant number of registers for each processor) in $O(n^{1/2})$ time.

2. Computing the maximal elements and the m-contour

We use the well-known divide-and-conquer approach (cf. [4,8,9]) for computing the maximal elements of S on a mesh of n processors.

It is assumed that every processor of the mesh-connected computer contains exactly one arbitrary point of S. Each PE also has a boolean register MAXEL which denotes whether the point stored in this PE is a maximal element or not. The register MAXEL is initialized to 'false' for all PEs.

The preprocessing consists of sorting S according to the x-coordinate of the points, using a sorting technique of Thompson and Kung [8]. Since Thompson and Kung's algorithm computes a snake-like ordering of the points of S on the mesh-connected computer (see Fig. 3 on page 305), we can assign to each point the rank of this sorted order (called x-index in the remaining of this paper) in $O(n^{1/2})$ time units.

The m-contour will be represented as follows: every processor has a register NEXTEL such that, when it contains a maximal element, NEXTEL contains the x-index of the next point of the m-contour.

Algorithm A. Divide S into two disjoint subsets L and R of equal size with $\ell.x \leqslant r.x$ for all $\ell \in L$ and $r \in R$ (see Fig. 2) by computing the minimum (min_ind) and maximum (max_ind) x-index of all points of S (which takes $O(n^{1/2})$ time units, cf. [4]) and broadcasting med_ind $:= \frac{1}{2}$(min_ind + max_ind) to all PEs. Now, every PE knows whether its point belongs to L or R and the mesh of processors can shift all points of L and R to its left and right half, respectively, and (recursively) solve the problem for L and R in parallel.

In order to combine the solutions of both subproblems and solve the problem for $L \cup R$, the processor containing a point $p \in R$ with minimum x-index (x_min) with respect to R broadcasts p.y and x_min to all other PEs. All processors containing a point $t \in L$ that is maximal with respect to L but $t.y \leqslant p.y$ set their MAXEL register to 'false' and the processor containing the point $q \in L$ with $q.y \geqslant p.y$ and maximum x-index with respect to $\{\ell \in L \mid \ell.y \geqslant p.y\}$ sets its NEXTEL register to x_min.

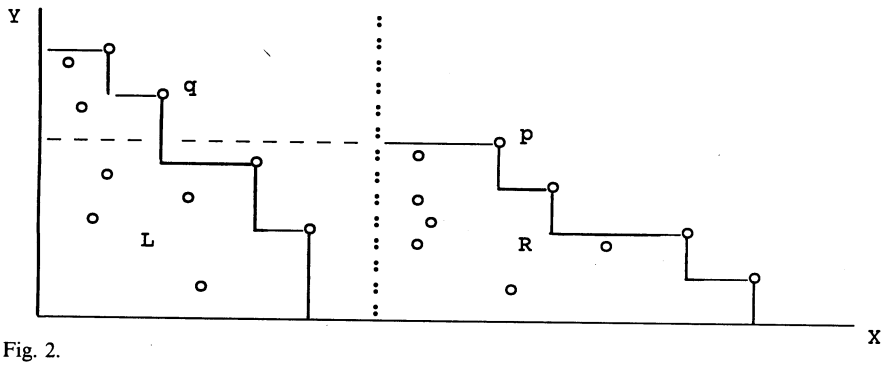

Fig. 2.

Theorem 2.1. *Algorithm* A *computes the maximal elements and the m-contour of a set of* n *points in* $O(n^{1/2})$ *time.*

Proof. Sorting n points takes $O(n^{1/2})$ time as described in [8]. Maximum/minimum determination, broadcasting, communication between two PEs, and data compression (moving the points of L and R into two subgrids of equal size) also takes $O(n^{1/2})$ time units. (For more details, consult [4,8,9].) Using T(n) to denote the time complexity of our algorithm, we get the following recurrence formula:

$$T(n) \leqslant T(\tfrac{1}{2}n) + c\, n^{1/2}.$$

Hence, $T(n) = O(n^{1/2})$. □

3. The ECDF searching problem

In addition to computing the empirical cumulative distribution function $D(p, S) := |\{q \in S \mid q \leqslant p\}|$ we shall also compute the function $B(p, S) := |\{q \in S \mid q.y \leqslant p.y\}|$, i.e., the number of points 'below' p (for all $p \in S$). The mesh-connected computer is initialized as before and the same preprocessing (sorting and computation of x-indices) is used as described in Section 2. In this case, the two registers MAXEL and NEXTEL of each processor used by Algorithm A are replaced by two registers D and B. These registers store the current value of the functions D and B. Initially, they are both set to zero. Each PE has two additional registers STATUS and VALUE and all PEs in the leftmost column have three more registers called READY, ROW_STATUS and ROW_VALUE.

Algorithm B. The structure of Algorithm B is essentially the same as for Algorithm A. Therefore, we shall only describe how to combine the solutions for two subsets L and R of S with $\ell.x \leqslant r.x$ for all $\ell \in L$ and $r \in R$ in $O(n^{1/2})$ time.

The final result of this procedure will be the following:

(1) $D(\ell, S) \leftarrow D(\ell, L)$ for all $\ell \in L$,

(2) $D(r, S) \leftarrow D(r, R) + \max\{B(\ell, L) \mid \ell.y \leqslant r.y\}$ for all $r \in R$,

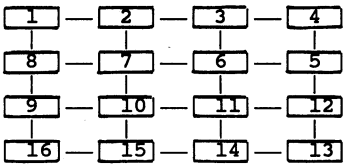

Fig. 3. Snake-like ordering of a mesh-connected computer.

(3) $B(\ell, S) \leftarrow B(\ell, L) + \max\{B(r, R) \mid r.y \leqslant \ell.y\}$ for all $\ell \in L$,

(4) $B(r, S) \leftarrow B(r, R) + \max\{B(\ell, L) \mid \ell.y \leqslant r.y\}$ for all $r \in R$.

To compute D(r, S) for all $r \in R$ (steps (3) and (4) can be implemented in essentially the same way) we first sort S according to the y-coordinates of its points (using the algorithm of Thompson and Kung [8]) and assign to each point its rank (called y-index) of this order. After this step, all points of S are sorted in a snake-like row-major indexing (see Fig. 3). Since for all $r \in R$ the number $\max\{B(\ell, L) \mid \ell.y \leqslant r.y\}$ is exactly $B(\ell, L)$ of the point $\ell \in L$ with maximum y-index and below r, the simultaneous computation of B(r, S) for all points $r \in R$ can be implemented as follows (see Fig. 4):

Each PE stores $B(\ell, L)$ into its VALUE register and sets STATUS := 1 if it contains a point $\ell \in L$ or the point with lowest y-index, otherwise both registers are set to zero. Each row of PEs computes the maximum of its STATUS (VALUE) registers and stores it into the ROW_STATUS (ROW_VALUE) register of its leftmost PE, respectively. Then all left-

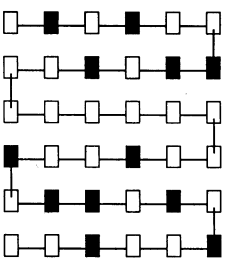

Fig. 4. Processors containing points of L (■) and R (□) in a snake-like ordering.

most PEs with ROW_STATUS = 1 sent the contents of their ROW_VALUE register upwards to PEs in the leftmost column with ROW_STATUS = 0. Now at least one PE of each row has the correct value $\max\{B(\ell, L) \mid \ell.y \leqslant r.y\}$ stored in its VALUE register. Thus, all B(r, S) can be computed simultaneously by shifting around these data in each row of PEs, respectively.

Theorem 3.1. *Algorithm* B *computes all values of the empirical cumulative distribution function* D(p, S) *in* $O(n^{1/2})$ *time.*

Proof. Passing the $B(\ell, L)$ upwards through the snake-like ordering would surely produce a correct answer, too. However, it could take more than $\theta(n^{1/2})$ time units since there might be several rows of PEs containing only points of R. This algorithm does essentially the same but only needs $O(n^{1/2})$ time since it detects such rows of PEs and passes the information upwards through the leftmost column of PEs first. Thus, Algorithm B has the same asymptotic time complexity as given for Algorithm A. □

4. Generalization to higher dimensions

To solve the maximal elements and ECDF searching problem in d-dimensional Euclidean space simply introduce one more divide-and-conquer step for each additional dimension and use the same algorithms as described in Sections 2 and 3 to combine the solutions of the subproblems. This yields algorithms running in $O(n^{1/2+\log_2(d-2)})$ $(d > 2)$ time, which is asymptotically optimal for $d = 3$, too.

References

[1] B.M. Chazelle, VLSI implementations of geometric tasks, 20th Ann. Allerton Conf. on Communication, Control and Computing (1982) 615–624.

[2] B.M. Chazelle, Computational geometry on a systolic chip, IEEE Trans. Comput. C-33 (9) (1984) 774–785.

[3] H.T. Kung, F. Luccio and F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (4) (1975) 469–476.

[4] R. Miller and Q.F. Stout, Computational geometry on a mesh-connected computer, Proc. 1984 Internat. Conf. on Parallel Processing (1984) 66–73.

[5] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, SIAM J. Comput. 9 (4) (1980) 744–757.

[6] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, J. Comput. System Sci. 23 (1981) 166–204.

[7] M.I. Shamos, Geometry and statistics: Problems at the interface, in: J.F. Traub, ed., Algorithms and Complexity (Academic Press, New York, 1976) 251–280.

[8] C.D. Thompson and H.T. Kung, Sorting on a mesh-connected parallel computer, Comm. ACM 20 (4) (1977) 263–271.

[9] J.D. Ullman, Computational Aspects of VLSI, Principles of Computer Science Series (Computer Science Press, Rockville, MD, 1984).