SOLVING GEOMETRIC PROBLEMS ON MESH-CONNECTED AND ONE DIMENSIONAL PROCESSOR ARRAYS

F.Dehne

Lehrstuhl fuer Informatik I , Univ. of Wuerzburg
Am Hubland, 8700 Wuerzburg, W.-Germany

Abstract:
This paper presents $O(n^{1/2})$ time solutions of the planar maximal elements m-contour, and ECDF searching problem for n points stored in a $n^{1/2} \times n^{1/2}$ mesh of processors that can be extended to solve the d-dimensional maximal elements and ECDF searching problem in time $O(n^{1/2+\log_2(d-2)})$ $(d>2)$ which is asymptotically optimal for $d \in \{2,3\}$. We also describe O(n) time algorithms for the planar/3-dimensional largest empty rectangle/cube problem fcr n points stored in one dimensional array of processors which are asymptotically optimal, too.

1. Introduction

A mesh-connected processor array of size n is a set of n synchronized processing elements (PEs) arranged in a square lattice. Each PE is connected via bidirectional unit-time communication links to its four neighbors, if they exist. (see fig.1)

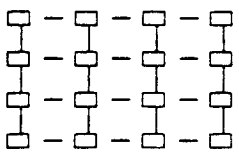

fig.1

Each processor has a fixed number of registers and can perform standard arithmetic and comparisons in constant time. It can also send or receive the contents of a register from a neighbor and test the existence of neighbors in O(1) time units.We may think of these processors as

F.Dehne, "Solving Geometric Problems on Mesh-Connected and One-Dimensional Processor Arrays", Proceedings of the 11th International Workshop on Graphtheoretic Concepts in Computer Science (WG'85), June 18-21, 1985, Castle Schwanberg, Wuerzburg, W.-Germany, Trauner Verlag 1985, pp. 43-59

individual VLSI chips or several chips each containing some part of the grid on a circuit board. For more details consult [MS] and [U].

Let $s=\{s_1,\ldots,s_n\}$ be a set of n points in the Euclidean plane. (To simplify exposition of our algorithms we assume w.l.o.g. $n=4^k$ for some integer k.). Given two points p and q, p.x [q.x] and p.y [q.y] denote the x-coordinate and y-coordinate of p [q], respectively.The point q dominates p, $p \leq q$, iff $p.x \leq q.x$ and $p.y \leq q.y$. A point s in S is called maximal, iff there is no other s' in S with $s \leq s'$. The contour spanned by the maximal elements of S is called m-contour of S.

Note, that the time comlexity for computing the maximal elements and m-contour is $\Theta(n \log n)$ using a sequential computer (see [KLP]).

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 $D(p,S)=|\{q \text{ in } S/\ q \leq p\}|$ (called 'empirical cumulative distribution function'). It has the same time complexity $\Theta(n \log n)$ on a standard computer. Consult [S] for more details and applications.

Chapter 2 of this paper will state algorithms solving both problems on a mesh-connected processor array in $O(n^{1/2})$ time and linear space (in the planar case), which is asymptitically optimal since any nontrivial computation requires $\Omega(n^{1/2})$ time on a mesh of processors.

Given a rectangle (with its edges parallel to the coordinate axes) containing the set S of n points, we will also consider the problem of finding the largest area subrectangle with sides parallel to those of the original rectangle which contains no point of S. [NHL] and [CD] gave $O(n^2)$ time, linear space, and $O(n \log^3 n)$ time, $O(n \log n)$ space, respectively, algorithms to solve this problem on a sequential computer.

In chapter 3 we describe a one-dimensional (systolic)

array of processors (see fig.2) solving this problem in linear time and space which is asymtotically optimal.

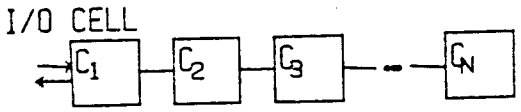

fig.2

Chapter 4 will extend these algorithms to higher dimensions and state some open problems.

2. $O(n^{1/2})$ Algorithms for the Planar Maximal Elements and ECDF Searching Problem on a Mesh-Connected Processor Array

2.1. Computing the Maximal Elements and M-Contour

We use a divide-and-conquer approach for computing the maximal elements of S as sketched by figure 3.
It is assumed that every processor of our mesh-connected processor array contains exactly one point of S; otherwise they can be loaded in $O(n^{1/2})$ steps. Each PE has also a boolean register MAXEL which denotes whether the point stored in this PE is a maximal element or not. The register MAXEL is initialised by 'false' for all PEs.

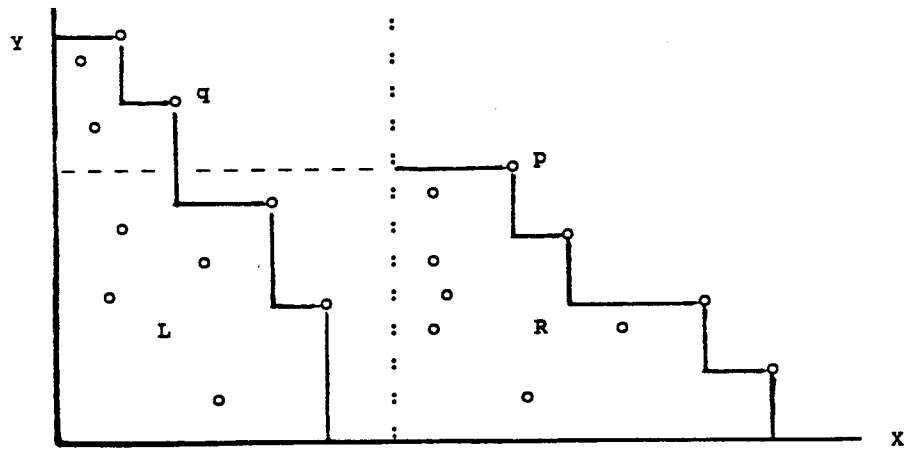

fig.3

Our preprocessing consists of sorting S according to the x-coordinate of the points as described by Thompson and Kung in [TK]. We assign to each point the index of this sorted order, called x-index in the remaining of this

paper. The m-contour will be represented as follows : each processor, storing a maximal element, has a register NEXTEL containing the x-index of the next point of the m-contour.

Now, we can state Algorithm **MAX:**

(1) Divide S into two disjoint subsets L and R of equal size with l.x $\leq$ r.x for all l in L and r in R:

(1a) Compute the minimum (min_ind) and maximum (max_ind) x-index of all points of S and broadcast med_ind := (min_ind + max_ind)/2 to all PEs.

(1b) Now, every PE knows, whether his point belongs to L or R. We shift all points of L and R to the left and right half (upper and lower half for recursion steps of even depth) in our current processor grid.

(2) Recursively, compute the maximal elements and m-contour of L and R on both parts of the mesh-connected processor array in parallel.

(3) Select from R the maximal element p with minimum x-index (and maximum y-coordinate).

(4) Broadcast p.y to all PEs in the other half of the grid (which store the points of L).
For all processors containing a point t in L
do(in parallel):
If t is maximal with rsp. to L and t.y $\leq$ p.y
then set MAXEL := 'false' and delete NEXTEL.

(5) Select from L the maximal element q (with rsp. to S) with maximum x-coordinate and store into the NEXTEL register of its processor the x-index of p if q.x$\neq$p.x. Otherwise, restore the NEXTEL reg. of q's PE by the x-index stored in the NEXTEL register of p's PE and set the MAXEL reg. of p's PE to 'false'.

Sorting n points takes time $O(n^{1/2})$ as described in [TK]. Maximum/ minimum determination, broadcasting, communication between two PEs, and data compression also takes $O(n^{1/2})$ time units. (For more details consult [MS] and

[TK].) Using T(n) as the time complexity of our algorithm, we get the following recurrence formula :

$$T(n) \leq T(n/2) + c \cdot n^{1/2} \quad .$$

Hence, $T(n) \in O(n^{1/2})$. Since one processor with a fixed number of registers is used for each point, the space requirement of the algorithm is linear with resp. to n.

2.2. The ECDF Searching Problem

In order to compute the empirical cumulative distribution function $D(p,S) := |\{q \text{ in } S / q \leq p\}|$ we additionally compute the function $B(p,S) := |\{q \text{ in } S / q.y \leq p.y\}|$ that is the number of points below p (for all p in S). The same initial state of our mesh-connected processor array and the same preprocessing (computation of x-indices) as described in chapter 2.1 is assumed. In this case, the two registers MAXEL and NEXTEL of each processor used by algorithm **MAX** are replaced by two registers D and B. These registers store the current value of the functions D and B, and they are initialized by zero.

Hence, there is the following Algorithm ECDF:

(1) Divide S into two disjoint subsets L and R of equal size with $l.x \leq r.x$ for all l in L and r in R; cf. (1a) and (1b) of algorithm **MAX**.

(2) Recursively compute D(l.L), B(l,L) for all l in L and D(r,R), B(r,R) for all r in R (in parallel).

(3) Sort S according to the y-coordinates of its points and assign to each point the index (y-index) of this sorted order. (cf. [TK])

(4) Update D:

(4a) $D(l,S) := D(l,L) \quad \forall\, l \in L.$

(4b) $D(r,S) := D(r,R) + \max\{B(l,L) / l.y \leq r.y\} \quad \forall\, r \in R.$

(5) Update B:

(5a) $B(l,S) := B(l,L) + \max\{B(r,R) / r.y \leq l.y\} \quad \forall\, l \in L.$

(5b) $B(r,S) := B(r,R) + \max\{B(l,L) / l.y \leq r.y\} \quad \forall\, r \in R.$

Now, we want to give some more details about step 4b; 5a and 5b can be done in the same way :

When the algorithm described in [TK] (step 3 of algorithm ECDF) terminates, then all points of S are sorted in a snake-like row-major indexing (see figure 4). Since for all r in R the number max{B(1,L)/ 1.y≤r.y} is exactly B(1,L) of the l in L with maximum y-index and below r , we compute step 4b as follows (see fig.5):

Let each PE have two additional registers STATUS and VALUE. All PEs in the leftmost column have three more registers called READY, ROW_STATUS and ROW_VALUE.

(4b.1) For each PE do (in parallel):
if the PE contains a point l of L
then STATUS := 1; VALUE := B(l,L);
else STATUS := 0; VALUE := 0 .

(4b.2) Set STATUS := 1 for the PE with lowest y-index.

(4b.3) For each row of PEs do (in parallel):
Set ROW_STATUS (ROW_VALUE) of the leftmost PE to the maximum of all STATUS (VALUE) registers of the PEs in the row. Set READY of the leftmost PE to ROW_STATUS.

(4b.4) With the leftmost column of PEs do:
Repeat
For all PEs in the leftmost column with READY=0 and READY=1 of the PE below do (in parallel):
Set VALUE to the VALUE of the PE below and READY:=1.
until all leftmost PEs have READY=1

(4b.5) For all rows do (in parallel):
If ROW_STATUS=0 for the leftmost PE then
set the VALUE register of the PE with lowest y-index of the row to ROW_VALUE, and set its STATUS register to 1.

(4b.6) Repeat
For all PEs with STATUS=0, where the STATUS register of its predecessor (in order of

increasing y-coordinate) is 1 **do** (in parallel):

Set VALUE to the predecessor's VALUE and STATUS:=1.

until all PEs have STATUS=1 .

(4b.7) **For** all PEs containing a point r in R **do** (in parallel):

Add VALUE to register D.

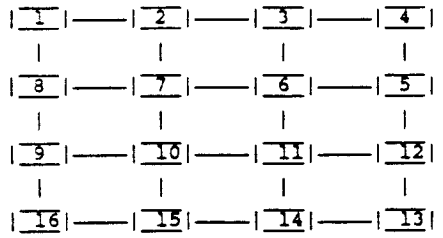

fig.4

Snake-like ordering of a mesh-connected processor array

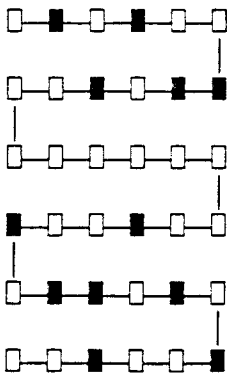

fig.5

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

Following chapter 2.1 it is known that step 1 and 3 of algorithm **ECDF** take $O(n^{1/2})$ time units.

Now, we want to analyse steps 4 and 5 by example of step 4b.

It is trivial to see that (4b.1), (4b.2), (4b.3), (4b.5), and (4b.7) have a time complexity of at most $O(n^{1/2})$. The hight of the leftmost column of PEs is at most $n^{1/2}$. Since at least the PE at the bottom has READY=1 (from 4b.2), the REPEAT...UNTIL loop of (4b.4) terminates after at most $n^{1/2}$ executions each taking O(1) time units. (Implementation of the REPEAT...UNTIL loop should use this fact.)

Note, that after executing step (4b.5) , there is at least

one PE with STATUS=1 in each row . Thus the REPEAT...UNTIL loop of (4b.6) finishes after at most $O(n^{1/2})$ executions, too.
Summarizing this, steps 1,3,4 and 5 of algorithm ECDF take $O(n^{1/2})$ time units and yield the same recurrence formula and asymptotic time complexity as given for algorithm **MAX** in chapter 2.1 .
Since we need exactly one processor with a fixed number of registers for each point of S space requirement is linear with resp. to n.

3. A One Dimensional Processor Array for the (Planar) Largest Empty Rectangle Problem

Our one dimensional (systolic) processor array for the largest empty rectangle problem (called **LER**) will support the following operations :

- Insert/delete a point
- Report the largest empty rectangle.

LER consists of N cells, so it can handle up to N points at a given time. For more details about one-dimensional systolic arrays consult [FKMD] and [C]. Note, that for **LER** all I/O-operations are performed by the leftmost cell C_1, since otherwise our algorithm would run in time linear with resp. to N (see fig.2).

3.1. Insertion, Deletion

To insert a new point, just put it into **LER** at its leftmost I/O-cell and let it move to the right, until it finds an empty cell. To delete a point, send an identifier to **LER**s I/O-cell C_1 and let it move to the right, until it finds the specified point. Delete this point and send a signal (special record) to its right neighbor , to let the following points shift to the left and close the gap.

3.2. Reporting The Largest Empty Rectangle in Linear Time

3.2.1. Basic Structure of Algorithm

Let $S=\{s_1,\dots,s_n\}$ be the current set of $n\leq N$ points sorted by their x-coordinates (sorting can be done in linear time applying the methods of [TK] to a one-dimensional array). Let x_{min}, x_{max}, y_{min}, y_{max} be the boundaries of the bounding rectangle.

Note that each edge of the largest empty rectangle is supported by either an edge of the bounding rectangle or at least one point of S (as described in [CDL]). We shall call these supporting edges or points "supporting elements with resp. to S".

To simplify exposition, we shall assume, that all points of S have distinct x-coordinates and distinct y-coordinates and do not lie on the boundary. Thus, the largest empty rectangle has exactly four supporting elements with resp. to S. As we shall see at the end of this paper, the existence of some more supporting elements will not change our algorithm signifficantly.

In order to compute the largest empty rectangle, we split S into two halves $S_L=\{s_1,\dots,s_{\lfloor n/2 \rfloor}\}$ and $S_R=\{s_{\lfloor n/2 \rfloor+1},\dots,s_n\}$ (with their bounding rectangles adjusted) and recursively solve the problem for S_L and S_R using the systolic cells $C_1,\dots,C_{\lfloor n/2 \rfloor}$ and $C_{\lfloor n/2 \rfloor+1},\dots,C_n$, respectively (see fig.6).

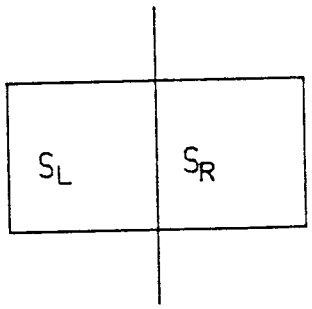

fig.6

Given the largest empty rectangles with resp. to S_L and

S_R, respectively, we have to compare the maximum area rectangle of these with the largest empty rectangle having at least one supporting element with resp. to S_L and S_R, respectively.
This "merging step" will be done by a second divide and conquer procedure.
After sorting S by y-coordinates, we split it into four subsets as described by fig.7 with $S_1 \cup S_2 = S_L$, $S_3 \cup S_4 = S_R$ and $\big|\,|S_2 \cup S_3| - |S_1 \cup S_4|\,\big| \leq 1$.

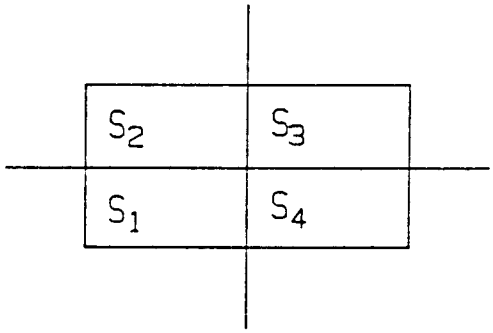

fig.7

With this we recursively compute the largest empty rectangle having at least one supporting element with resp. to S_2 (S_1) and S_3 (S_4) and none with resp. to S_1 (S_2) and S_4 (S_3), repectively.

3.2.2. The Final Merging Step

To solve the final merging step, we have to find the largest empty rectangle r, having the following property (*):

Let B_1 ($B_2,\ldots,B_4$) be the set of supporting elements of r with resp. to S_1 ($S_2,\ldots,S_4$), then

$|B_1|+|B_2|+|B_3|+|B_4|=4$
$|B_1|+|B_2|>0$
$|B_3|+|B_4|>0$
$|B_2|+|B_4|>0$
$|B_1|+|B_4|>0$.

Let $S_i' := S_i \cup \{p_i, q_i\}$ ($i=1,\ldots,4$) as sketched by fig.8 .

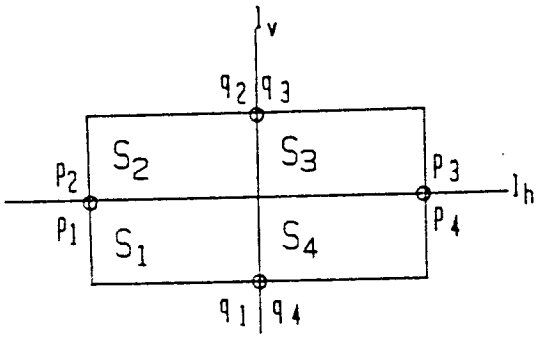

fig.8

Now, we can prove

Lemma 1:

If r is an empty rectangle with property (*) and e is an edge of r supported by a vertical (horizontal) boundary edge of S_i, then e is supported by p_i (q_i) , i=1...4 .

Proof:

From property (*) it is easy to see, that both vertical (horizontal) edges of r have to cross l_h (l_v), thus lemma 1 follows immediatly. ■

With this we can forget the bounding rectangles simply by adding the points p_i, q_i (i=1,...,4) during the final merging step and considering all empty rectangles with exactly four supporting points and property (*) with resp. to $S_1', \ldots, S_4'$.

Note, that we add at most 4n points simultaneously, thus every cell of **LER** has to store at most 5 points.

Definition 1:

Let (x_1,y_1), (x_2,y_2) be two points, then

$(x_1,y_1) <_{ur} (x_2,y_2)$:<=> $x_1<x_2$ and $y_1<y_2$

$(x_1,y_1) <_{ul} (x_2,y_2)$:<=> $x_1>x_2$ and $y_1<y_2$

$(x_1,y_1) <_{ll} (x_2,y_2)$:<=> $x_1>x_2$ and $y_1>y_2$

$(x_1,y_1) <_{lr} (x_2,y_2)$:<=> $x_1<x_2$ and $y_1>y_2$

Let M be a set of points and $x \in M$, then x is called a ur-maximal [ul-maximal, ll-maximal, lr-maximal] element of M :<=>

x is a maximal element of M with resp. to $<_{ur}$ [$<_{ul}$, $<_{ll}$, $<_{lr}$].

Let M_1 [M_2, M_3, M_4] be the ur [lr,ll,ul] -maximal elements of S_1' [S_2', S_3', S_4'], then we have

Lemma 2:

Let r be an empty rectangle supported by four points $\{t_1,\ldots,t_4\}\subseteq S_1'\cup S_2'\cup S_3'\cup S_4'$ with property (*), then $\{t_1,\ldots,t_4\}\subseteq M_1\cup M_2\cup M_3\cup M_4$.

Proof:

From property (*) it easy to see, that both vertical (horizontal) edges of r cross l_h (l_v). Since r has to be empty lemma 2 follows immediatly. ∎

Summarizing this, we have

Theorem 1:

The final merging step can be computed by finding the maximum area rectangle of all empty rectangles supported by four points $\{t_1,\ldots,t_4\}\subseteq B_1\cup B_2\cup B_3\cup B_4$ with $B_i\subseteq M_i$ $(i=1,\ldots,4)$, $|B_1|+|B_2|+|B_3|+|B_4|=4$, $|B_1|+|B_2|>0$, $|B_3|+|B_4|>0$, $|B_2|+|B_3|>0$ and $|B_1|+|B_4|>0$.

CASE	|B1|	|B2|	|B3|	|B4|	TYPE
1)	0	1 <	1 ^	2 v>	B
2)	0	1 <	2 ^>	1 v	B
3)	0	2 <^	0	2 v>	A
4)	0	2 <^	1 >	1 v	B
5)	1 <	0	1 ^	2 v>	B
6)	1 <	0	2 ^>	1 v	B
7)	1 <	1 ^	0	2 v>	B
8)	1 <	1 ^	1 >	1 v	C
9)	1 v	1 <	1 ^	1 >	C
10)	1 v	1 <	2 ^>	0	B
11)	1 v	2 <^	0	1 >	B
12)	1 v	2 <^	1 >	0	B
13)	2 <^	0	1 >	1 v	B
14)	2 <v	0	2 ^>	0	A
15)	2 <v	1 ^	0	1 >	B
16)	2 <v	1 ^	1 >	0	B

table_1

In table 1 all possible (16) cases are listed with <

[>,^,v] denoting that a point supports a left [right, upper, lower] edge of an empty rectangle.
There are essentially three types of empty rectangles which we have to consider.
A type A rectangle is supported by two points of M_1 [M_2] and M_3 [M_4], respectively.
A type B rectangle is supported by two points of one quadrant and one point each of two other quadrants, while a type C rectangle is supported by one point of each quadrant.
It is easy to see that the directions of support as given in table 1 are the only possible ones:

For both cases of type A rectangles this is trivial. Concerning type B rectangles let's for exemple look at case 5. There is only one supporting point in the left half, which must be a left support, since otherwise the rules of theorem 1 would be violated. There is only one supporting point in the upper quadrant of the right half, which must be an upper support for the same reason.

Having exactly one supporting point in each quadrant (type C), there are two possible cases. The supporting point sp_1 in the lower left quadrant M_1 must either be a left or lower support, since otherwise there would be no supporting point in M_2 or M_4, respectively. Assuming sp_1 to be a left [lower] support, it is easy to see that the other three directions of support are determined by this choice. With this we get exactly two cases of type C.

In order to compute the final merging step, LER will do 16 global shifts (called type A [B,C] shifts for cases of type A [B,C]) determining the largest empty rectangle for each case (if it exists) in linear time, respectively. The maximum area empty rectangle as descibed by theorem 1 is the largest of these 16 (or less) rectangles.
Before we can give the details of type A [B,C] shifts, we

need the following definition and lemma.

Definition 2:

Let (x_1,y_1), (x_2,y_2) be two elements of M_i $(i=1,...,4)$ with $x_1<x_2$.
(x_1,y_1) and (x_2,y_2) are called "close neighbors of M_i" :<=> there is no other point (x_3,y_3) in M_i with $x_1<x_3<x_2$.

Lemma 3:

Let r be an empty rectangle as described by theorem 1 and $\{t_1,t_2\}=B_i\subseteq M_i$ two supporting points in the same quadrant M_i $(i=1,...,4)$ => t_1 and t_2 are close neighbors of M_i.

Proof:

Let w.l.o.g. i=1 and r be an empty rectangle as described by theorem 1 supported by $\{t_1,t_2\}=B_1$ (see fig.9). Assuming there is a point $t'\in M_1$ with x-coordinate between t_1 and t_2, t' will lie inside r, since it is ur-maximal with resp. to S_1 and has distinct y-coordinate - a contradiction. ∎

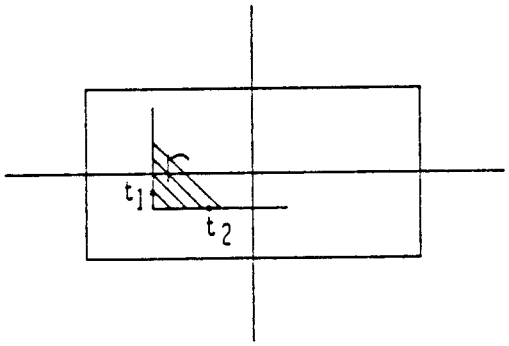

fig.9

With this we can describe the shifts as follows:

Type A shifts

Let's w.l.o.g. take case 3. From lemma 3 we know that all pairs of supporting points of M_3 [M_1] are close neighbors. Thus, we sort $M_1,...,M_4$ by x-coordinate and let each pair of close neighbors of M_3 (represented by the maximum x-coordinate and maximum y-coordinate of both points) shift through M_2 and M_4 to find the rightmost point with smaller y-coordinate and uppermost point with smaller x-coordinate, respectively.

With these two points "in mind", we shift each pair of close neighbors of M_3 through M_1 and determine the largest rectangle not containing any point of M_2 or M_4. Pipelining these processes yields linear running time.

Type B shifts

Let's w.l.o.g. take case 2. With the same arguments as given above, we sort $M_1,\ldots,M_4$ by x-coordinate. Taking each pair of close neighbors of M_3 as described above, it is easy to see, that the supporting lower point $l_4 \epsilon M_4$ and supporting left point $l_2 \epsilon M_2$ are determined. l_4 is the uppermost point of M_4 with smaller x-coordinate and l_2 is the rightmost point of M_2 with smaller y-coordinate. Thus we let each pair of close neighbors of M_3 shift through M_4 and M_2 and find these both points, respectyvely. With this, a shift through M_1 shows whether this rectangle is empty. Pipelining these processes yields linear running time, too.

Type C shifts

A type C shift is essentially the same, since given the left [lower] supporting point in M_1 the three other supporting points are determined.

3.2.3. Accumulated Running Time and Space Requirement

Since the final merging step can be done in linear time, the accumulated running time of all divide and merging steps is linear with resp. to n. Each cell has to store a constant amount of information yielding a linear space requirement, too.

4. Extensions, Open Problems

Consider the maximal elements and ECDF searching problem in d-dimensional Euclidean space.

Algorithms **MAX** and **ECDF** can easily be generalized by introducing one more recursion step for each additional coordinate axis. Thus, we get d-1 nested recursion proce-

dures yielding an accumulated running time of $O(n^{1/2+\log_2(d-2)})$ on a mesh-connected processor array, respectively.

In a similar way, our systolic array LER can be generalized to solve the largest empty cube problem (d=3) by introducing one additional recursion with resp. to the z-axis, yielding a linear (thus asymptotically optimal) running time, too.

A generalization of the largest empty cube algorithm to arbitrary dimensions does not seem to be very practical, since the number of shifts we have to compute increases exponentially with resp. to d. There may be another (optimal) solution of this problem which does not use such shifts and is easier to generalize.

Another open problem is the design of an optimal, $O(n^{1/2})$, solution of the largest empty rectangle problem on a mesh-connected processor array.

Acknowledgment, New Results

Thanks to N.Santoro (Carleton Univ.) and S.E.Hambrusch (Purdue Univ.) for helpfull comments on the first version of this paper which helped improve the results signiffi-cantly. Additionally, it turned out during WG'85 that algorithms **MAX** and **ECDF** can easily be generalized to compute the "k-th maximal elements" and "k-th m-contour".

References

[AH] M.J.Atallah, S.E.Hambrusch, SOLVING TREE PROBLEMS ON A MESH-CONNECTED PROCESSOR ARRAY, Report CSD-TR-518, Purdue Univ., West Lafayette, April 1985

[C] B.M.Chazelle, COMPUTATIONAL GEOMETRY ON A SYSTOLIC CHIP, IEEE Trans. on Computers, Vol. C-33, No.9, Sept. 1984

[CD] B.Chazelle, R.L.Drysdale, D.T.Lee, COMPUTING THE LARGEST EMPTY RECTANGLE, Proc. Symp. on Theoetical Aspects of Computer Science, 1984, pp 43-54

[FKMD] A.L.Fisher, H.T.Kung, L.M.Monier, Y.Dohi, A PROGRAMMABLE SYSTOLIC CHIP, Journal of VLSI and Computer Systems, Vol.I, No.2, 1984

[KLP] H.T.Kung, F.Luccio and F.P.Prparata, ON FINDING THE MAXIMA OF A SET OF VECTORS, J. of the ACM, Vol.22, No.4, Oct. 1975

[MS] R.Miller and Q.F.Stout, COMPUTATIONAL GEOMETRY ON A MESH- CONNECTED COMPUTER,Proc.1984 Int. Conf. on Parallel Proc.

[NHL] A.W.Naamad, W.L.HSU, D.T.Lee, ON THE MAXIMUM EMPTY RECTANGLE PROBLEM, Disc. Applied Math.

[NS] D.Nassami and S.Sahni, FINDING CONNECTED COMPONENTS AND CONNECTED ONES ON A MESH-CONNECTED PARALLEL COMPUTER, SIAM J. COMPUT., Vol.9, No.4, Nov. 1980

[S] M.I.Shamos, GEOMETRY AND STATISTICS: PROBLEMS AT THE INTERFACE, in J.F.Traub (Ed.): Algorithms and Complexity, Academic Press, New York 1976

[TK] C.D.Thompson and H.T.Kung, SORTING ON A MESH-CONNECTED PARALLEL COMPUTER, Comm. of the ACM, Vol.20,No.4, April 1977

[U] J.D.Ullman, COMPUTATIONAL ASPECTS OF VLSI, Principles of Computer Science Series, Computer Science Press, 1984