Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer

RUSS MILLER, STUDENT MEMBER, IEEE, AND QUENTIN F. STOUT, MEMBER, IEEE

Abstract—Although mesh-connected computers are used almost exclusively for low-level local image processing, they are also suitable for higher level image processing tasks. We illustrate this by presenting new optimal (in the O-notational sense) algorithms for computing several geometric properties of figures. For example, given a black/white picture stored one pixel per processing element in an $n \times n$ mesh-connected computer, we give $\theta(n)$ time algorithms for determining the extreme points of the convex hull of each component, for deciding if the convex hull of each component contains pixels that are not members of the component, for deciding if two sets of processors are linearly separable, for deciding if each component is convex, for determining the distance to the nearest neighboring component of each component, for determining internal distances in each component, for counting and marking minimal internal paths in each component, for computing the external diameter of each component, for solving the largest empty circle problem, for determining internal diameters of components without holes, and for solving the all-points farthest point problem. Previous mesh-connected computer algorithms for these problems were either nonexistent or had worst case times of $\theta(n^2)$. Since any serial computer has a best case time of $\theta(n^2)$ when processing an $n \times n$ image, our algorithms show that the mesh-connected computer provides significantly better solutions to these problems.

Index Terms—Computational geometry, convexity, image processing, mesh-connected computer.

I. Introduction

MESH-connected computers (MCC's) have long been proposed for image processing. Images can be naturally mapped onto an MCC so that neighboring pixels are mapped onto neighboring (or the same) processing elements. Because of this, local operations on the image, such as edge detection or median filtering, can be performed by local operations on the mesh, enabling such algorithms to efficiently exploit the massive parallelism available. Many discussions of actual MCC's, such as the SOLOMON, ILLIAC III, CLIP4, or MPP, emphasize their speed on local operations [4], [11], [18], [20], and most of the early papers on MCC's, such as those of Unger [30]-[32] and Golay [8], similarly emphasized local operations.

In this paper we consider higher level image processing and pattern recognition tasks which require combining information globally. We concentrate on geometric problems involving convexity, internal distance, and external distance. For some problems, notably the Euler number [9], [15], connectivity [3], [13], and skeletization [23], it is possible to iterate local operations to achieve an optimal solution to a global problem. However, this approach seems to work only in isolated problems. (For example, the Beyer and Levialdi "shrinking" approach to connectivity problems of digitized pictures [3], [13] does not extend to higher dimensions. A solution strategy similar to ones presented in this paper was needed for optimal algorithms in dimensions greater than two [24].) Instead of local operations, our algorithms emphasize the use of sorting and graph-theoretic algorithms. The same approach was used by Nassimi and Sahni [17] in their optimal MCC algorithm for labeling the connected components of digitized pictures. The algorithms presented in this paper are always optimal in the O-notational sense, and we have tried to make them as general as possible.

The paper is organized as follows. In Section II, we define the MCC model and review standard MCC algorithms that will be used throughout the paper. In Section III, given an $n \times n$ black/white picture stored one pixel (picture element) per processing element in an $n \times n$ MCC, we give $\theta(n)$ time algorithms for computing internal distances, marking minimal internal paths, and counting the number of these paths for each component of the picture. These problems arose in an image processing task considered in [10], and our algorithms are faster than their $\theta(n^2)$ algorithms. In Section IV, we give a $\theta(n)$ time algorithm for marking the extreme points of the convex hull for every labeled set of processors. We also give $\theta(n)$ time algorithms for deciding if the convex hull of each component contains pixels that are not members of the component, for deciding if two sets of processors are linearly separable, for solving the smallest box problem, and for deciding if each black figure is convex [21], [29]. Previously, [5] and [12] described algorithms that decided convexity of figures, but their algorithms require $\theta(n^2)$ time in the worst case. In Section V, we show how to compute the distance between components in $\theta(n)$ time, where the distance can be measured by almost any metric. A $\theta(n)$ solution also appears in Dyer and Rosenfeld [5], but their solution can only be used with the l_1 ("taxicab" or "city block") and l_∞ ("chessboard") metrics. Section V also contains optimal solutions to nearest neighbor, radius query, and farthest point problems [21].

Manuscript received January 20, 1984; revised July 23, 1984. This work was supported by the National Science Foundation under Grant MCS-83-01019.

R. Miller is with the Department of Mathematical Sciences, State University of New York, Binghamton, NY 13901.

Q. F. Stout was with the Department of Mathematical Sciences, State University of New York, Binghamton, NY 13901. He is now with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

II. Preliminaries

A. Notation

We use Ω to mean "order at least," O to mean "order no greater," and θ to mean "order exactly."

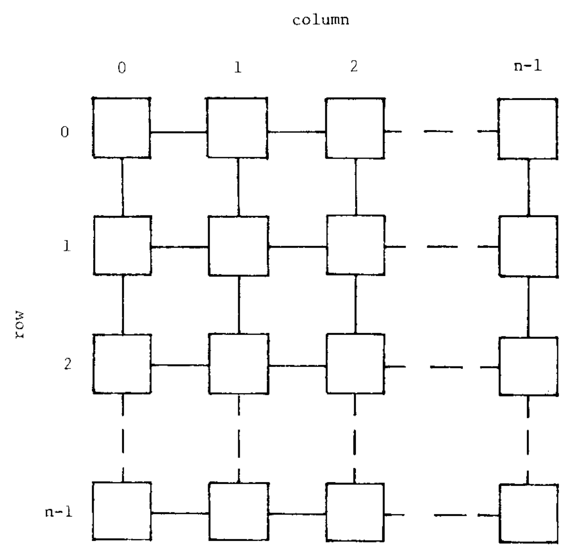

Fig. 1. An $n \times n$ MCC.

B. Definition of MCC

The *mesh-connected computer* (*MCC*) is a single instruction stream-multiple data stream (SIMD) computer. It consists of n^2 *processing elements* (*PE's*) arranged in an $n \times n$ grid. To simplify exposition, we assume that $n = 2^c$ for some integer c. PE(i, j) will represent the processing element in row i, column j. The mesh is oriented so that PE$(0, 0)$ is situated in the northwest corner of the MCC, while PE$(n - 1, n - 1)$ resides in the southeast corner. (See Fig. 1.) For all i, j in $\{0, \cdots, n - 1\}$, PE(i, j) is connected via unit-time communication links to its four *neighbors*, PE's$(i \pm 1, j \pm 1)$, assuming that they exist. Each PE has a fixed number of registers (words), each of size $\theta(\log(n))$, and each PE can perform standard arithmetic and Boolean operations on the contents of its registers in $\theta(1)$ time. Each PE can send or receive a word of data from a neighbor in $\theta(1)$ time. Each PE contains a unique identification register (ID) whose contents correspond to that PE's row-major index (i.e., PE(i, j) has an ID of $n * i + j$). This model is the same as that used in [1], [5], [17], and [28], and is sometimes called a *memory-augmented cellular array*.

There are some variations on the MCC which deserve mention. Early studies considered other interconnection schemes, such as Moore's pattern of connecting each PE to its eight nearest neighbors [16] or Golay's use of a hexagonal decomposition of two-dimensional space where each PE communicates with its six nearest neighbors [8]. In an O-notational sense, such differences are easily seen to be irrelevant. A more significant change is to require that the word size be $\theta(1)$ instead of $\theta(\log(n))$. This model is known as a *mesh automata, iterative array, parallel processing array*, or *cellular array*. It is equivalent to requiring that all PE's be copies of some fixed finite state automaton. For any fixed automaton, once n is sufficiently large, a PE does not have enough memory to store its ID or coordinates, which seriously complicates matters. Mesh automata have been widely studied (e.g., [3], [8], [9], [13], [16], [24], [25], [31]-[33]), but currently there seems to be much greater interest in the more powerful MCC. To the best of the authors' knowledge, all real mesh-connected computers have PE's capable of storing their coordinates, so it seems that the MCC is the more realistic model. There are also more powerful variations, such as the mesh-connected computer augmented with broadcasting [27], but their study is outside the bounds of this paper.

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

Fig. 2. Snake-like ordering of a 4×4 MCC.

C. Standard MCC Algorithms

We now describe well-known MCC algorithms that will be used throughout this paper.

1) Rotating Data Within a Row (Column): For each row (column), every PE can transmit a fixed number of pieces of information to all other PE's in its row (column) in $\theta(n)$ time. Copies of the necessary pieces of information from each PE move towards the easternmost (northernmost) PE of their row (column). Once there, the copies of information reverse themselves until they reach the westernmost (southernmost) PE of the row (column) where they reverse themselves again. The algorithm terminates when all PE's simultaneously receive copies of their original data.

2) Passing a Row (Column) Through the MCC: In $\theta(n)$ time, every PE of the MCC can view all of the data from every PE of a given row (column). For the given row (column), rotate all columns (rows) simultaneously so that there is a copy of the row's (column's) data in every row (column) of the MCC. Now, simply rotate all of the rows (columns) simultaneously.

Besides organizing data in rows or columns, we will often order it via the *snake-like ordering*, illustrated in Fig. 2. Notice that in the snake-like ordering, each PE is adjacent to its predecessor and its successor. Other orderings, such as the proximity ordering in [25], also have this property, but the snake-like ordering seems to be the simplest.

3) Rotating Data in Snake-Like Order: Suppose that each PE contains a record with a key and data part, and suppose that all PE's with the same key form an interval in the snake-like ordering. If it is known that there are no more than D PE's with the same key, then in $\theta(D)$ time, each PE can pass its data to all other PE's with the same key. This can be done by first having each PE check the keys of its neighbors to determine if it is the first or last PE with its key (according to the snake-like ordering). Then the data are rotated just as in rotating data within a row, with each PE passing data to adjacent PE's, where the first PE with a given key acts just as the westernmost PE of a row, and the last PE with a given key acts just as the easternmost PE of a row. Notice that the data may traverse more than a single row.

4) Sorting: Thompson and Kung [28] have shown that n^2 elements, distributed one element per PE, can be sorted in $\theta(n)$ time by using a recursive merging procedure. In particular, the elements can be sorted into snake-like order in $\theta(n)$ time.

Two other common data movement operations for the MCC are the random access read (RAR) and random access write (RAW). These operations involve two sets of PE's, the *sources* and the *destinations*. Source PE's send a record consisting of a key and one or more data parts. (A record may also be null.) Destination PE's receive a record sent by a source PE, or else receive a null record. We allow the possibility that a PE is both a source and a destination.

5) Random Access Read (RAR): In a RAR, we require that no two records sent by source PE's have the same key. Each destination PE specifies the key of the record it wishes to receive, or it specifies a null key, in which case it receives nothing. Several destination PE's can request the same key. A destination PE may specify a key for which there is no source record, in which case it receives a null message.

6) Random Access Write (RAW): In a RAW, the destination PE's do not specify the keys they want to receive. They merely indicate their willingness to receive. At the end of the RAW, each destination PE receives either a null record or a record sent by a source PE. Each key is received by exactly one destination PE. If two or more source PE's send records with the same key, then a destination PE will receive the record with the minimum data field. (In other circumstances, one could replace minimum with any other commutative, associative, binary operation.)

Both the RAR and RAW can be completed in $\theta(n)$ time by using the algorithms in [17]. The RAR (RAW) could be performed by a straightforward sort-like step if it were not for the possibility that several destination PE's (source PE's) request (send) the same key. To overcome this problem, sorting into snake-like order is used to bring all the requests (records sent) with the same key together. Next, a single representative of each key is chosen. A second sort-like step allows each representative to complete its operation, and then, for the RAR, each representative must return to the others, distribute the value obtained to them, and then a sort step returns all records to their destination PE's.

For the algorithms presented in this paper, it is assumed that a digitized picture initially exists in the MCC. The picture consists of n^2 pixels distributed one per PE. The pixels are in one of two states: black or white. It is helpful to think of this digitization as being a black picture on a white background. Two black pixels in neighboring PE's are said to be connected. Given a black PE A, the *connected component* that A is a member of is the set of PE's that can be reached from A via a connected path.

7) Component Labeling: Nassimi and Sahni [17] have shown that the black connected components of a digitized picture can be labeled in $\theta(n)$ time. Upon completion of the labeling algorithm, each PE will contain the label of its pixel's component, where the label of a component is chosen to be the minimum ID of any PE containing a pixel in the component. Many of our algorithms consist of labeling the components of a picture and then having each component decide some property, where by a *component deciding* we mean that each PE of the component has the answer to the query.

8) Transitive Closure: Let $G = (V, E)$ be a directed graph, where $V = \{1, \cdots, n\}$ is the set of vertices and E is the set of edges. G can be represented by its adjacency matrix A, where $A(i, j)$ is 1 if there is an edge from vertex i to vertex j, and is 0 otherwise. A related matrix is the connectivity matrix A^*, where $A^*(i, j)$ is 1 if there is a path in G from vertex i to vertex j, and is 0 otherwise. A^* is sometimes called the *transitive closure of* A. Warshall's algorithm to compute the transitive closure of A is [19], [35]

```
for $k$ := 1 to $n$ do
    for $i$ := 1 to $n$ do
        for $j$ := 1 to $n$ do
            $A_k(i, j) := A_{k-1}(i, j)$ or
                $[A_{k-1}(i, k)$ and $A_{k-1}(k, j)]$
```

where A_0 is A and A_n is A^*. The interpretation of $A_k(i, j)$ is quite simple: $A_k(i, j)$ is 1 if there is a path from vertex i to vertex j using no intermediate vertex greater than k, and is 0 otherwise. Given this interpretation, the assignment statement in Warshall's algorithm merely states that there is a path from vertex i to vertex j using no intermediate vertex greater than k if and only if either there is a path from i to j using no intermediate vertex greater than $k - 1$, or there is a path from i to k using no intermediate vertex greater than $k - 1$ and there is a path from k to j using no intermediate vertex greater than $k - 1$.

Van Scoy [33] showed that if PE(i, j) initially stores $A(i, j)$, then in $\theta(n)$ time one can compute A_n, where PE(i, j) contains $A_n(i, j)$ when the algorithm terminates. For us, the importance of this result is that Van Scoy's algorithm easily extends to show that any recurrence of the form

$$f_k(i, j) = g(f_{k-1}(i, j), f_{k-1}(i, k), f_{k-2}(k, j))$$

can be solved for all $f_n(i, j)$ in $\theta(n)$ time if the function g can be computed in $O(1)$ time by a single PE and if $f_0(i, j)$ is initially stored in PE(i, j). Upon termination of the algorithm, $f_n(i, j)$ will be stored in PE(i, j).

III. Internal Distances

For black PE's A and B, an *A-B path* is a sequence of connected black PE's originating at A and terminating at B. A *minimal A-B path* is an A-B path containing the minimum number of PE's over all possible A-B paths. The *internal distance* from A to B, denoted DIST(A, B), is defined to be one less than the number of PE's in a minimal A-B path. (Note: while the minimal A-B path may not be unique, the internal distance between A and B is.)

Assume that the digitized picture has had its black components labeled, in $\theta(n)$ time, as described in Section II. Given a special marked black pixel M, it is clear that in $\theta(n)$ time, every PE of the MCC can be informed as to M's component label. The main problem of this section is to determine DIST(S, M) for each pixel S in the same component as M. This problem occurs in image processing [10], and from its solution one can find an internal spanning tree in $\theta(1)$ additional time. (A spanning tree of a graph G is a connected acyclic subgraph containing every vertex of G. They are important to many problems in graph theory [1], and are also used in [5] to solve several image processing problems.) Reference [10] gives $\theta(n^2)$ time algorithms for several of the problems considered in this section.

Our internal distance algorithm is based on using the generalized transitive closure operation described in Section II. Given a directed graph G of n vertices, if we define $A_k(i,j)$ to be the minimal length of a path from i to j using no intermediate vertex greater than k, then we see that the A_k satisfy the recurrence

$$A_k(i,j) = \min\{A_{k-1}(i,j), A_{k-1}(i,k) + A_{k-1}(k,j)\}$$

where $A_0(i,j)$ is 0 if $i=j$, 1 if there is an edge from i to j, and ∞ otherwise. Notice that $A_n(i,j)$ is DIST(i,j).

Unfortunately, we cannot just blindly use the transitive closure operation to find internal distances since there may be $\theta(n^2)$ black pixels (vertices), which would require a matrix with $\theta(n^4)$ entries. To reduce the matrix to $O(n^2)$ entries, we must make use of the underlying geometry of the digitized picture. The solution to the all-points minimum distance problem will be described as a two-phase algorithm, with both phases being implemented via a recursive divide-and-conquer strategy.

At a given stage i of the divide and conquer, let $k = 2^i$. The *outer border elements* of a $k \times k$ square are defined to be those PE's in rows and/or columns O and $k - 1$ of the square that contain the same label as that of the marked PE. The *inner border elements* of a $k \times k$ square are defined to be those PE's in rows and/or columns $(k+1)/2$ and $(k+1)/2 - 1$ of the square that contain the same label as that of the marked PE. (That is, the outer border elements of the four $(k/2) \times (k/2)$ subsquares that have neighbors in a different subsquare of the square.) The term *border elements* shall be used to refer to the collection of inner and outer border elements of a $k \times k$ square. (See Fig. 3.) Note: we assume that the $k \times k$ squares are aligned so that PE's $(c*k, d*k)$, $1 \leqslant c, d \leqslant \lceil n/k \rceil$, mark the southeast PE of each $k \times k$ square.

The objective of the first phase of the algorithm is to obtain the distance to the marked PE for all of the border elements of the $n \times n$ MCC. This phase is implemented using a bottom-up divide-and-conquer solution strategy. The objective of the second phase of the algorithm is to obtain the distances to the marked PE for the remaining PE's that are in the same component as the marked PE. The second phase will be implemented via a top-down divide-and-conquer solution strategy where each iteration of the solution requires applications of phase 1.

Internal Distance Algorithm

We begin by describing the first phase of the algorithm at an arbitrary stage i of the divide-and-conquer solution. Let $k = 2^i$.

Phase 1 Description: At the conclusion of stage $i - 1$, each $(k/2) \times (k/2)$ square contains a distance matrix, where the entries of the matrix represent the *restricted* internal distances between the border elements of the $(k/2) \times (k/2)$ square. These internal distances are measured using paths that are restricted solely to the square in question. The matrix also contains entries representing the restricted internal distances between these border elements and the marked PE (if the marked PE is not contained in the $(k/2) \times (k/2)$ square, then these distances are infinity).

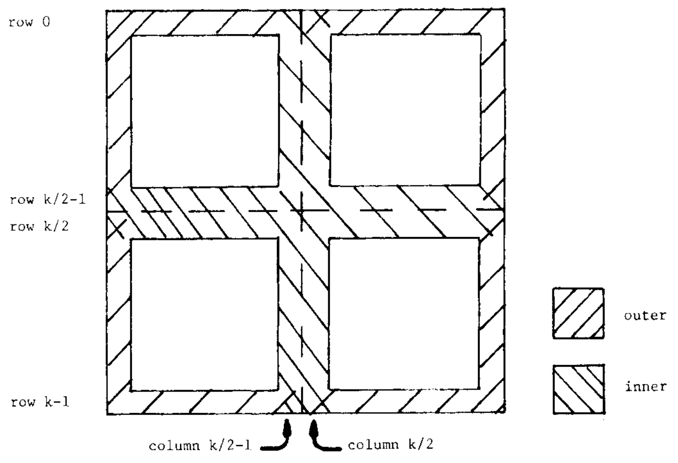

Fig. 3. Possible border elements of a $k \times k$ subsquare.

At stage i, squares of size $(k/2) \times (k/2)$ are merged to form squares of size $k \times k$. Using the aforementioned matrices from the four $(k/2) \times (k/2)$ subsquares, the restricted internal distance from every border element of a $k \times k$ square to the marked PE and all other border elements of that $k \times k$ square can be computed.

After $\theta(\log(n))$ stages of the divide-and-conquer solution, a minimum internal distance matrix containing the internal distances among all of the border elements of the entire MCC will exist. (In fact, this matrix will also contain information pertaining to the outer border elements of the four $(n/2) \times (n/2)$ subsquares of the MCC.) Furthermore, each of these border elements will know its distance to the marked pixel.

Phase 1 Assumptions: Before performing computations at stage i on a $k \times k$ square A, the following must hold for each of the four $(k/2) \times (k/2)$ subsquares of A at the completion of stage $i - 1$.

1) A $(4k - 15) \times (4k - 15)$ matrix exists that contains the restricted internal distances between the border elements and the marked PE of the subsquare. By convention, we will let the last row and column of the matrix be those distances pertinent to the marked PE.

2) Every entry in the matrix contains the unique ID's of the PE's that the distance represents. Recall that the ID is the row-major index of the PE.

3) Every border element has a register containing its restricted internal distance to the marked PE. This can be obtained from the last row or column of the matrix by performing a RAW.

Phase 1 Procedure: For all $k \times k$ squares A, set up the distance matrix for the border elements and the marked PE. Since there must be a row and column for the marked PE and for each one of the $2k - 4$ border elements from the four subsquares of square A, this matrix can be of size at most $(8k - 15) \times (8k - 15)$. For simplicity, an $(8k) \times (8k)$ pseudomachine is used to represent this matrix. (That is, each PE of the $k \times k$ machine will simulate 64 PE's.)

In each of the four subsquares, compress the $(4k - 15) \times (4k - 15)$ matrices to the northwest by logically deleting the rows and columns that are not needed for the computations in square A. Once each of the subsquares has compressed its

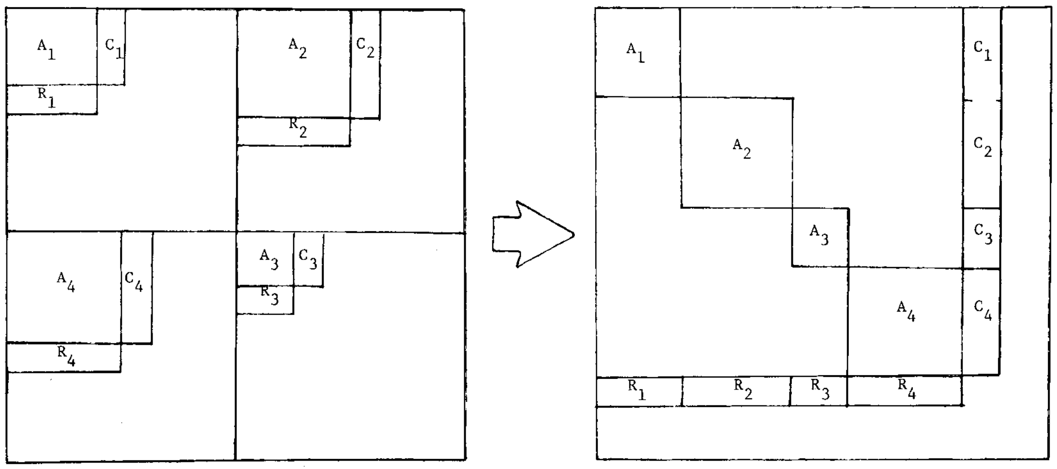

Fig. 4. Rearranging distance matrices.

matrix, move the matrices to the regions as illustrated in Fig. 4. This can be accomplished via a RAW in $\theta(k)$ time since the only information necessary is the size of each of the four submatrices, which can be computed in $\theta(k)$ time.

In each row and column, rotate the coordinates of the PE's represented so that the new entries can know which PE's they represent. If an entry detects that it represents the distance between two inner border elements that were in different squares at stage $i - 1$, then replace the entry of ∞ with a distance of 1.

We now use the generalized transitive closure operation to determine, in $\theta(k)$ time, the minimal path lengths between vertices. (Recall that the vertices in a square correspond to its border elements and the marked PE.) Next, pass the row representing the marked PE through the $k \times k$ subsquare so that every border element can obtain and record its (perhaps infinite) distance to the marked PE.

After $O(\log(n))$ iterations, phase 1 will be complete and each of the border elements of the $n \times n$ MCC will have its "correct" internal distance to the marked PE.

Phase 2 Description: To obtain the "correct" internal distances to the marked PE for the border elements of the $(n/2) \times (n/2)$ subsquares, simply apply phase 1 to each of the $(n/2) \times (n/2)$ subsquares of the MCC. The only difference in the reapplication of the algorithm to each of the subsquares is that the distances just obtained from the outer border elements of the $(n/2) \times (n/2)$ squares to the marked PE must be used in order to obtain the "correct" internal distance for all of the border elements of the subsquares. To obtain the correct distance for every PE in the same component as the marked PE, simply continue this process recursively for $\theta(\log(n))$ iterations.

Analysis: The time to initially label the picture and pass the label of the marked PE to all PE's can be completed in $\theta(n)$ time. The time to complete phase 1 is $\theta(n)$ since the time to complete each phase i of the divide-and-conquer is $\theta(2^i)$. The time to complete phase 2 is again $\theta(n)$ since the time to compute the distances for the border elements of a $k \times k$ square is the time to complete phase 1 on that $k \times k$ square, which is $\theta(k)$. Therefore, we have the following.

Theorem 1: Given an $n \times n$ digitized black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, and given a marked processor M, in $\theta(n)$ time each processor can compute its (possibly infinite) internal distance to M. □

This improves upon the $\theta(n^2)$ worst case time for algorithms in [5] and [10].

In addition to knowing the internal distance between PE's, it is sometimes desirable to mark minimal internal paths and to count the number of such paths.

Theorem 2: Given an $n \times n$ black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, and given marked processors A and B, if the distance from A to B is finite, then in $\theta(n)$ time

a) all minimal A-B paths can be marked,

b) a single minimal A-B path can be marked, and

c) the number of minimal A-B paths can be determined.

Proof: For a), we perform the all-points internal distance algorithm twice, first with A as the marked processor, and then with B as the marked processor. Inform all PE's that are in the same component as A and B of the distance from A to B. (This distance is contained in both A and B.) Every PE C such that $\text{DIST}(A, C) + \text{DIST}(C, B) = \text{DIST}(A, B)$ is on some minimal A-B path. If every such C now creates an edge (C, D) to each neighbor D such that $\text{DIST}(D, B) = \text{DIST}(C, B) - 1$, then all minimal A-B paths will be marked.

For b), first perform part a). Each PE now contains between zero and four edges. In order to mark a single minimal path, each PE discards all but one of its edges. Now there exists exactly one directed, minimal, A-B path. In order to mark this path, perform part a) again using only the directed edges that were just created.

For c), first mark all minimal A-B paths. Assume that for some k, there exist $k \times k$ squares in the MCC, each containing a $(4k - 2) \times (4k - 2)$ matrix M that represents the number of minimal paths between A, B, and the border elements of each $k \times k$ square, where only paths within the square are considered. Entry $M(i, j)$ represents the number of distinct minimal paths from vertex i to vertex j.

Merge the appropriate four $k \times k$ squares to create an $(8k - 14) \times (8k - 14)$ matrix. Place a "1" in $M(i, j)$ if the (C, D) edge exits, where PE C is the ith row in the matrix, and

PE D is the jth column of the matrix, and C and D are neighbors from two of the different $k \times k$ squares that are merged. Next, compute the number of minimal paths from each entry of the matrix to each other by the following formula:

$$f_k(i,j) = \begin{cases} f_{k-1}(i,j) + f_{k-1}(i,k) * f_{k-1}(k,j), \\ \qquad \text{if } \operatorname{DIST}(i,k) + \operatorname{DIST}(k,j) = \operatorname{DIST}(i,j). \\ f_{k-1}(i,j), \quad \text{otherwise.} \end{cases}$$

This requires a slight modification of the transitive closure algorithm as presented in [33]. When one PE must pass an arbitrary $f_{k-1}(i,j)$ to another PE, it must also pass to that PE DIST(i,j), since this information is necessary in order to ensure the proper evaluation of the function.

Compress the matrix by deleting the rows and columns that do not represent border elements of the $2k \times 2k$ square or marked PE's. The result is a matrix of size $(4(2k) - 2) \times (4(2k) - 2)$. Continue this merging, computing, and compressing process until the computations have been performed on the entire $n \times n$ MCC. The entry $M(a, b)$, where A is the ath row of the matrix and B is the bth column, contains the number of minimal paths from A to B. □

In the algorithms just presented, it is not necessary to restrict A and B to single PE's. With only minor modifications to the algorithms of this section, A and B can be arbitrary sets of PE's. Given sets A and B of PE's, we define the internal distance from A to B to be min $\{\operatorname{DIST}(x,y): x \in A, y \in B\}$. The most crucial modification is that of letting the last row and column of the distance matrix represent the set B of PE's instead of just a single marked PE. With this change, we obtain the following result.

Theorem 3: Given an $n \times n$ digitized black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, and given (not necessarily disjoint) sets A and B of marked processors, in $\theta(n)$ time each processor can compute its (perhaps infinite) distance to the set B, and the distance from A to B can be determined. Further, if the distance from A to B in finite, then in $\theta(n)$ time

a) all minimal A-B paths can be marked,

b) a single minimal A-B path can be marked, and

c) the number of minimal A-B paths can be determined. □

(If A and B overlap, then define DIST(A, B) to be 0, and define marking minimal paths to mean indicating which PE's are in both sets.)

One application of Theorem 3 is where each black component has exactly one of its pixels in A (for example, A may contain the pixel whose ID is that of the component). Then by applying part a), in $\theta(n)$ time one can construct a breadth-first spanning tree of each component where a breadth-first spanning tree of a graph is a spanning tree such that each vertex is at the minimal possible distance from the root.

Another generalization comes from noting that all of the above algorithms work equally well if the edges between pixels are directed and have arbitrary positive weights. If negative edge weights are allowed, then if there is a cycle with a negative total weight, the cycle can be repeated arbitrarily often to make distances as negative as desired. Therefore, any path touching such a cycle should be given a total distance of $-\infty$. With a little work, we can accommodate negative weights.

Theorem 4: Given an $n \times n$ mesh-connected computer such that each processor contains a directed weighted edge to each of its neighbors, where the weights can be $+\infty$ or any real number, and given (not necessarily disjoint) sets A and B of processors, in $\theta(n)$ time each processor can compute its (perhaps infinite) distance to B, DIST(A, B) can be determined, and it can be decided if all cycles have a positive total distance. Further, if all cycles have a positive total distance and DIST(A, B) is finite, then in $\theta(n)$ time

a) all minimal A-B paths can be marked,

b) a single minimal A-B path can be marked, and

c) the number of minimal A-B paths can be determined.

Proof: If there is a cycle with negative total weight, then the recurrence used to calculate internal distances is no longer correct. To remedy this, when working in any square, first run the algorithm as before, except that all diagonal entries are initialized to be $+\infty$. The (i, i) entry of the resulting matrix is negative if and only if vertex i is on a cycle of total negative distance. (The entry is 0 if and only if the vertex is on a cycle of zero total distance, and is not on any negative cycles.) If the (i, i) entry is negative, it is replaced with $-\infty$, as is any entry, other than $+\infty$, in the ith row and ith column. (This is because any path leading into or out of a negative cycle should have a path length of $-\infty$.) Now the generalized transitive closure algorithm is run again, where we define $(-\infty) + (+\infty) = +\infty$. It can be shown that the resulting matrix has the correct distances, and is therefore ready for the next stage. □

We note that if there are cycles of nonpositive total distance, then it is difficult to make sense of marking or counting all minimal paths. We leave the questions of giving the proper definitions and finding fast algorithms corresponding to the definitions as open problems.

The *internal diameter* of a set A of processors is defined to be max $\{\operatorname{DIST}(x,y): x, y \in A\}$. (External diameters appear in Section V.) Fischler [6] shows how the internal diameter can be used to classify cracks in an industrial inspection application. For an arbitrary set A of PE's, we do not know of efficient algorithms for determining the internal diameter. However, for the important case where A is a component without holes, we can rapidly determine its internal diameter. (This includes the case of interest to Fischler.)

We only outline our solution, which is based on the algorithm in Theorem 1. When working on finding distances in some square, for each black PE (vertex) x on the border of the square, we also have another black PE (vertex), denoted $F(x)$, which is one of the furthest PE's from x in the square, subject to the condition that $F(x)$ is connected to x in the square. (It may be that $F(x)$ is another border PE, in which case it was already a vertex, and it may be that $F(x)$ and $F(y)$ are the same, even though x and y are not. In these cases, the redundant vertices are eliminated.) The important fact is that, in a component with no holes, $F(x)$ can be selected from among $\{F(y): y$ is a border element of a subsquare$\}$. Further, it can be shown that the largest finite internal distance ever calculated during any stage is the internal diameter. Incorporating these facts, we obtain the following.

Theorem 5: Given an $n \times n$ digitized black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, then simultaneously for all components, in $\theta(n)$ time each black component without a hole can determine its internal diameter. □

We note that, using techniques from [3], [13], and [15], in $\theta(n)$ time each component can decide if it has any holes.

IV. Convexity

In this section we solve several convexity problems. Central to our work is the identification of the PE at position (i, j) with the integer lattice point (i, j). A set of PE's is said to be *convex* if and only if the corresponding set of integer lattice points is convex, i.e., the smallest convex polygon containing them contains no other integer lattice points. While this is the proper notion of convexity for integer lattice points, it does have the annoying property that some disconnected sets of points, such as $\{(0, 0), (2, 3)\}$, are convex.

The relationship between the convexity of PE's and the convexity of the original figures is complicated by the digitization process. If the integer lattice corresponding to the PE's is placed atop a black/white picture, and each lattice point is given the color of the point it covers, then a convex black figure will yield a convex set of black lattice points. For each convex set of black lattice points, there is a convex black figure whose digitization is the given set, but this figure is never unique. Further, there are nonconvex black figures which also yield the given set. If, instead, each lattice point is thought of as being at the center of a closed unit square (with adjacent squares overlapping on their edges), and if a lattice point is colored black when all of its square is black, then once again a convex black figure will yield a convex set of black lattice points, and for any convex set of black lattice points, there are both convex and nonconvex black figures which yield the given set.

For some other digitization schemes, the correspondence is not quite as nice. For example, suppose that each lattice point is again viewed as the center of a closed unit square, but now it is colored black if any part of its square is black. Convex black figures will yield convex connected sets of black lattice points, but not all convex connected sets of black lattice points can arise as the digitization of a black convex figure. (See Fig. 5.) We can alter our definitions of lattice convexity and alter our algorithms to decide this new convexity with the same time bounds, but we will not do so here. Readers interested in pursuing the relationship between convexity and digitization should read [12] and the references therein.

Given a set S of PE's, the *convex hull of* S, denoted hull (S), is the smallest convex set of PE's containing S. Just as for standard planar convexity, it is easy to show that hull (S) is the intersection of all convex sets containing S. A PE P in S is an *extreme point of* S if $P \notin$ hull $(S - P)$. The extreme points of S are the corners of the smallest convex polygon containing S. We say that we have *identified the extreme points of* S if each PE in S has decided whether or not it is an extreme point of S.

We will show that many queries concerning S can be reduced to questions concerning the extreme points of S. On an $n \times n$ mesh, S may have $\theta(n^2)$ points, but since S has at most two extreme points in any row, it has $O(n)$ extreme points. In fact, by using a little number theory, one can show that S has only $O(n^{2/3})$ extreme points [34]. Since it takes $\Omega(n)$ time to move data across an $n \times n$ mesh, this latter bound does not help in most algorithms, but it is crucial to the algorithm of Theorem 10. (See also [14] and [27].)

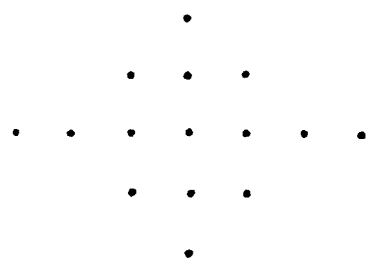
Fig. 5. A convex set of black lattice points which, in some digitization schemes, cannot arise as the digitization of a convex black figure.

For the following theorem, we will use the fact that there are constant time algorithms enabling a PE to decide if one integer lattice point is on the line segment between two others, if one integer lattice point is on the line determined by two others, if one integer lattice point is in the angle determined by three others (one of which is designated as the vertex), and if one integer lattice point is in the closed triangle determined by three others. The theorem assumes that each PE has some label, with ∞ used as a "DON'T CARE" label. The labels may arise as the labels of connected components, but there is no requirement that they do so.

Theorem 6: In an $n \times n$ mesh-connected computer, simultaneously for all labels A, in $\theta(n)$ time one can identify the extreme points of the processors labeled A.

Proof: First, each PE determines if it is either the leftmost or rightmost PE containing its label in its row. This is done in $\theta(n)$ time by rotating in each row the label and position of all PE's in the row. When finished, each PE that is either a leftmost or rightmost PE for its label within its row places its label and position into its sort field, while all other PE's put infinity and their position into their sort field. These values are sorted into snake-like order using the label as the key.

For all finite labels, we now rotate in snake-like fashion, as described in Section II, the position information in the sort field among all PE's with the same label in their sort field. Since for each finite label there are at most $2n$ positions, this can be done in $\theta(n)$ time. As the information rotates, each PE determines whether or not the position in its sort field is an extreme point. This is done as follows: suppose that X is the position in the sort field. The positions of at most two more PE's of the same label will be stored. As each new position Y arrives, if no other position has been stored, then Y is copied. If only one other position U has been stored, then the PE checks if X is on the line segment between Y and U. If it is, then X is not an extreme point; otherwise, Y and U are stored, unless Y is on the line determined by X and U, in which case only U is kept.

Finally, if two positions U and V are being stored when Y arrives, then the PE checks if X is in the (perhaps degenerate) closed triangle formed by Y, U, and V. If so, then X is not an extreme point. Otherwise, the PE needs to determine which two of Y, U, and V are to be stored. One of these must be in the angle formed by the other two with X at the vertex, and it is this one which is eliminated. (If Y lies on the line determined by X and one of the others, then Y is eliminated.)

There is some constant C such that after Cn time units, the information is finished rotating. It is easy to show that at any time in the computation, if a PE has not yet determined that the position in its sort field is not an extreme point, then the position is an extreme point of the set of points passing through that PE so far. Using this fact, we see that when the information is finished rotating, if a PE has not yet determined that the position in its sort field is not an extreme point, then the position must be an extreme point. A random access write sends this information back to the position. □

Corollary 1: Given an $n \times n$ digitized black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, simultaneously for all components, in $\theta(n)$ time each black connected component can decide if it is convex.

Proof: We use the observation that a connected set of PE's S is convex if and only if for each PE P, such that $P \notin S$ and P is the right or left neighbor of a PE in S, $P \notin$ hull (S). We will check if $P \notin$ hull (S) by checking if P is an extreme point of $S \cup \{P\}$.

First use the labeling algorithm of [17] to uniquely label the PE's in each connected component. Then, by rotating information within rows, each component determines if its restriction to each row is convex, e.g., an "O" shaped component would find rows in which it is not convex, but a "Z" shaped one would not. If each of its rows is convex, then we follow the algorithm in Theorem 6 to identify the extreme points of the component. As the information is rotating and the PE's are determining if the position in their sort field is an extreme point, they are also determining if the PE to the right of that position (if the position is the rightmost PE with its label in its row) or to the left of that position (if the position is the leftmost PE) is an extreme point. That is, the coordinates of these extra PE's are not rotated, but the algorithm of Theorem 6 is performed for them. When done, if any one of these extra PE's is not an extreme point, then the component is not convex, while otherwise it is. Finally, rotations can be used to ensure that each black PE knows if its component is convex or not. □

More than 20 years ago, Unger [32] gave a $\theta(n)$ time algorithm for detecting horizontal and vertical concavities, i.e., concavities detectable by traveling along a horizontal line or a vertical line. Any figure with such concavities is not convex, but there are nonconvex figures without such concavities, so Unger's algorithm does not decide convexity. Convexity decision algorithms for a single component appeared in [5] and [12], finishing in time proportional to the perimeter of the component. (These algorithms are based on the work of Sklansky [22].) Since a component may have a perimeter of $\theta(n^2)$ points, this gives a worst case time of $\theta(n^2)$. No convex region can have more than $4n - 4$ PE's on its perimeter, and since the size of the perimeter can be determined in $\theta(n)$ time, the algorithms in [5] and [12] can easily be modified to decide convexity of a single figure in $\theta(n)$ time. However, we see no easy way to modify these algorithms to produce the extreme points in $\theta(n)$ time.

Using ideas similar to those used in Corollary 1, the following result on intersecting convex hulls follows routinely.

Corollary 2: Given an $n \times n$ digitized black/white picture stored one pixel per processor in an $n \times n$ mesh-connected computer, simultaneously for all components, in $\theta(n)$ time each black component can decide if its convex hull contains any black pixels not in the component. Further, simultaneously for all components, in $\theta(n)$ time each black component can decide if any processors in it are in the convex hull of another black component. □

A closely related problem concerns linear separability [29]. Suppose that each PE has a label, with A and B as possible labels (there may be additional possibilities). Then the PE's labeled A are *linearly separable* from the PE's labeled B if there exists a straight line in the plane such that all the lattice points corresponding to PE's labeled A lie on one side of the line, and all of the lattice points corresponding to PE's labeled B lie on the other side. The crucial observation is that two sets are linearly separable if and only if their convex hulls are disjoint. Using this fact, we easily obtain the following result.

Corollary 3: In an $n \times n$ mesh-connected computer, in $\theta(n)$ time it can be decided if the PE's labeled A are linearly separable from the PE's labeled B. □

Given a set P of points in the plane, a *smallest box* [7], [29] containing P is a rectangle of smallest area containing P. It can be shown that the area of a smallest box is unique, that each of its sides contains an extreme point of P, and that at least one side contains two extreme points of P [7].

Corollary 4: In an $n \times n$ mesh-connected computer, simultaneously for all labels A, in $\theta(n)$ time each processor labeled A can determine a smallest box containing all of the processors labeled A. Further, in $\theta(n)$ time, simultaneously for all labels A, every processor labeled A can determine the same smallest box containing all of the processors labeled A, as every other processor labeled A.

Proof: First perform the algorithm of Theorem 6, except that the extreme values are not written back to the PE's. If a PE P has position X in its sort field, and if X is an extreme point, then at the conclusion of the algorithm, the other two points being stored (call them U and V) are also extreme points. (If no other points are being stored, then only one PE has its label, while if only one other point is being stored, then the PE's with that label form a straight line segment.) By using the angle UXV, P can determine whether traveling from X to U or from X to V will produce a counterclockwise traversal around the convex hull. For convenience, we will assume that it is from X to U.

P now tries to determine the corners of the rectangle in Fig. 6. It does this by finding R, S, and T, where R is the point furthest from the line XU, S is the point whose projection onto the line XU is the most negative (where X is the origin and U is at a positive location), and T is the point whose projection onto XU is the most positive. To enable each PE to find its R, S, and T, rotate the position information again. When finished, each PE having an extreme point in its sort field now knows R, S, and T, and hence can compute the corners of its box. Now

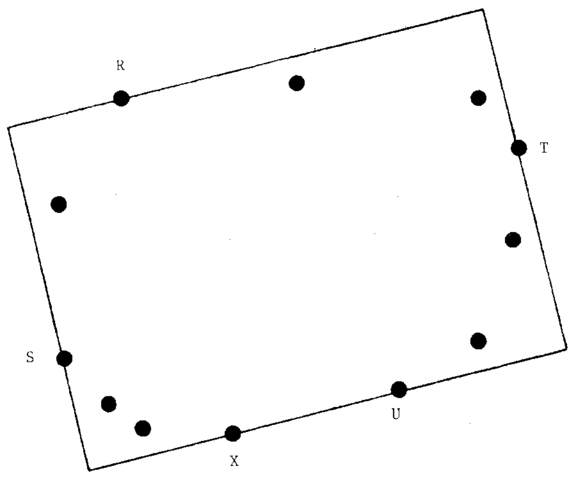

Fig. 6. A smallest box.

the corners of these boxes are rotated in snake-like fashion among the PE's with the same label in their sort field. When finished, each PE knows a smallest box for the label in its sort field. A RAR and row rotation informs each PE labeled A of a smallest box containing all PE's labeled A. A final RAR will allow all PE's labeled A to know the same smallest box as all other PE's labeled A, simultaneously for all labels A. □

V. External Distances

In this section we will consider some problems involving (external) distances between PE's where again PE(i, j) is identified with the integer lattice point (i, j). The most common distance measures used are the l_p metrics, where for $1 \leq p < \infty$, the l_p distance from (a, b) to (c, d) is $[|a - c|^p + |b - d|^p]^{1/p}$. (The l_∞ distance from (a, b) to (c, d) is $\max\{|a - c|, |b - d|\}$.) The connection scheme of the MCC is based on the l_1 ("taxi cab" or "city block") metric, so problems are usually easiest when expressed in terms of this metric. Further, simple tricks can also be used to solve problems in terms of the l_∞ metric. However, for other metrics, such as the important l_2 (Euclidean) metric, slightly more sophisticated methods are needed.

We will assume that there is a function $d(x, y)$ which computes, in unit time, the distance between points x and y. d cannot be completely arbitrary, for then there would be no connection between the metric and the underlying geometry of the mesh. For example, suppose that there is a special PE P, and suppose d is such that $d(P, Q) = 9$ for all other PE's Q, and $d(Q, R) = 10$ whenever $Q \neq R$, $Q \neq P$, and $R \neq P$. While this gives a metric, problems like, "for each pixel, find the distance to the closest black pixel," must be answered by examining P alone, rather than having each PE look at the pixels stored in nearby PE's. To avoid this, we will consider only monotone metrics, where a metric d is said to be *monotone* if for all PE's P, Q, and R, if Q and R are neighbors and the l_1 distance from P to R exceeds the l_1 distance from P to Q, then $d(P, R) \geq d(P, Q)$. (See Fig. 7.) All l_p metrics are monotone, and it seems that monotone metrics are the only ones ever encountered in practice.

Let P be a set of PE's. The *external diameter* of P is defined to be $\max\{d(x,y): x, y \in P\}$. Dyer and Rosenfeld [5] showed how to compute the external diameter in $O(n)$ time when d

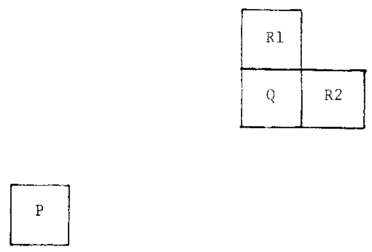

Fig. 7. For a monotone metric d, $d(P, Q) \leq d(P, R1)$ and $d(P, Q) \leq d(P, R2)$.

was either the l_1 or l_∞ metric, but their solution is unusable for other metrics. Fischler [6] gives an image processing application using external diameter measured with the Euclidean metric.

Theorem 7: In an $n \times n$ mesh-connected computer, for any monotone metric, simultaneously for all processors, in $\theta(n)$ time each labeled processor can determine the external diameter of the processors with its label.

Proof: Let S denote the set of all PE's with a given label. If the metric were an l_p metric, then we could use the fact that the external diameter of S is equal to

$$\max\{d(x, y): x, y \text{ are extreme points of } S\}.$$

For arbitrary monotone metrics this is no longer true, but it is true that the external diameter of S is equal to

$$\max\{d(x, y): x, y \text{ are the rightmost or leftmost elements of } S \text{ in their rows}\}.$$

As in Theorem 6, we first have each PE determine if it is either a leftmost or rightmost PE of its label in its row. Each such PE puts its label and coordinates into its sort field and all other PE's put infinity and their coordinates into their sort field. These elements are then sorted with the label as primary key. For each finite label, the coordinates are now rotated (in snake-like fashion), and each PE keeps track of the maximum distance from the coordinates in its sort field to the received coordinates. When the coordinates are done rotating, these maxima are rotated. The solution is the largest of the maxima. A RAR then ensures that each labeled PE knows the external diameter for its label. □

A closely related problem is the *all-points farthest point problem* [21], in which for each labeled PE we are to find the greatest distance to a PE with the same label. With a slight change to the preceding algorithm, we obtain the following.

Corollary 5: In an $n \times n$ mesh-connected computer, for any monotone metric, in $\theta(n)$ time the all-points farthest point problem can be solved. □

Theorem 7 was concerned with finding distances among PE's with the same labels, while the following theorem is concerned with finding distances between PE's with different labels.

Theorem 8: In an $n \times n$ mesh-connected computer, for any monotone metric, simultaneously for all processors, in $\theta(n)$ time each processor can determine the distance to the nearest processor with a different label, if such a processor exists.

Proof: Let P be an arbitrary PE and let Q be a closest labeled PE with a label different from P's. Let R be the PE in P's column and Q's row. Since d is monotone, it must be that

R and all PE's between R and Q are either unlabeled or have the same label as P. This forms the basis of a simple algorithm. In each row, we rotate the labels, and as the labels rotate each PE determines the closest (in the l_1 sense) labeled PE on its right. (If the PE is itself labeled, it is the closest labeled PE.) Since this label might be the same as P's, each PE also finds the closest labeled PE on the right with a label different from the first found one. This procedure is also performed for the left side. When finished, each PE has determined at most four label/coordinate records, which are now rotated within the columns. Each PE determines the minimal distance to a record with a different label, completing the algorithm. □

One use of this theorem is to have a black/white image where each white is unlabeled and each black is labeled by its coordinates. In this case, the result is known as the *all-points closest point problem* or the *all-points nearest neighbor problem* [21], [29]. Another use of the theorem is to first label the components, then apply Theorem 8, and then perform a RAW to determine the distance between black components where by the *distance between components A and B* we mean

$$\min \{d(P, Q): P \in A, Q \in B\}.$$

Corollary 6: In an $n \times n$ mesh-connected computer, for any monotone metric, simultaneously for all components, in $\theta(n)$ time each black component can find the distance to its nearest black component. □

Yet another application of Theorem 8 is to the *largest empty circle problem* [21], in which each PE is marked or unmarked and we want to find a PE P which maximizes $\min \{d(P, Q)$: Q is marked$\}$, subject to the additional constraint that P must lie in the convex hull of the marked PE's. (Technically, we should call this a circle problem only if the Euclidean metric is used.) By combining Theorem 8 and the algorithm in Theorem 6, we have the following.

Corollary 7: In an $n \times n$ mesh-connected computer, for any monotone metric, the largest empty circle problem can be solved in $\theta(n)$ time. □

Given a nonempty set S of PE's, there are several natural definitions of the center of S. If S is connected, then an *internal center* of S is a PE P of S which minimizes $\max \{\text{DIST}(P, Q)$: $Q \in S\}$, where DIST is the internal distance. For any metric d, a *planar center* of S is a point x in the real plane which minimizes $\max \{d(x, Q)$: $Q \in S\}$, and a *restricted planar center* of S is a PE P in S which minimizes $\max \{d(P, Q)$: $Q \in S\}$. For each definition of center, there is also a corresponding definition of radius. For any l_p metric ($1 < p < \infty$), the planar center is unique, but the restricted planar center may not be. For example, the four indicated points in Fig. 8 are restricted planar centers, and all of the points of the figure are internal centers.

The proof of the following theorem is similar to that of Theorem 7 and will be omitted.

Theorem 9: In an $n \times n$ mesh-connected computer, simultaneously for all labels, for any monotone metric, in $\theta(n)$ time each processor can determine if it is a restricted planar center among the processors with its label. Further, it can determine the restricted planar radius of the processors with its label. □

The following theorem is proven only for the Euclidean metric, contrary to our attempt to make the theorems as general as possible. Our proof uses facts which are specific to the Euclidean metric, and for which we have been unable to find a usable generalization.

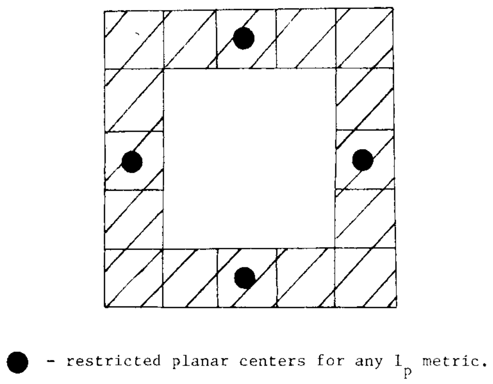

Fig. 8. All black pixels are internal centers.

Theorem 10: In an $n \times n$ mesh-connected computer, simultaneously for all components, in $\theta(n)$ time each black component can determine its Euclidean planar center and Euclidean planar radius.

Proof: For the Euclidean metric, this problem is known as the *smallest enclosing circle problem* [21]. The following facts will be used to prove the theorem.

a) If a set has only one or two points, then the smallest enclosing circle can be found in $\theta(1)$ time.

b) For a set of three points, either all three points are on the boundary of the smallest enclosing circle or else two of the points form a diameter of the circle. In either case, the center and radius of the circle can be found in $\theta(1)$ time.

c) For a set S of three or more points, there is a three-element subset T of S such that the smallest enclosing circle of T is the smallest enclosing circle of S. The radius of the smallest enclosing circle of T is the maximum radius of any smallest enclosing circle of a three-element subset of S. Further, T can be taken to be a subset of the extreme points of S, except when all of S lies on a straight line, in which case T contains the two endpoints and any third point.

These facts are straightforward and the proofs will be omitted.

Our strategy is quite simple: given a set, find its extreme points, and then for each three-element subset of the extreme points, find the smallest enclosing circle. However, if there are e extreme points, there will be $\binom{e}{3} = \theta(e^3)$ calculations. As was mentioned at the beginning of Section IV, on an $n \times n$ mesh, the worst case value of e is $\theta(n^{2/3})$, requiring $\theta(n^2)$ calculations. If these calculations must be done in the e PE's, they will require at least $\Omega(n^{4/3})$ time. We have prevented this by considering only connected components. If a component has p PE's, then $e = O(\min(p, n)^{2/3})$, so at most $O(\min(p, n)^2)$ calculations are required. By suitably dividing these calculations among the p PE's, they can be completed in $O(n)$ time. □

We should mention that finding the l_1 and l_∞ planar radii and planar centers are particularly easy. For these two metrics, the planar radius is half of the diameter, which we (and Dyer and Rosenfeld [5]) have shown can be found in $\theta(n)$ time. The l_1 (and l_∞) planar centers form a straight line segment (see

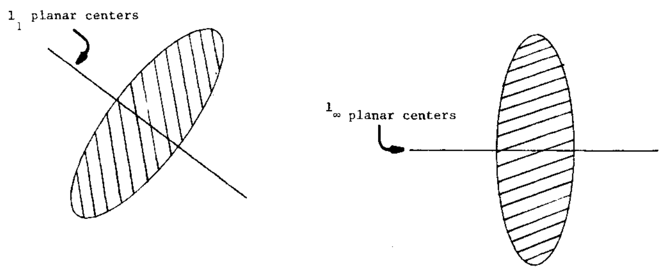

Fig. 9. Figures with nonunique planar centers.

Fig. 9) which may degenerate to a single point. We leave the details of finding the endpoints of these segments to the reader.

Our final distance problem is called an *all-points radius query*. (It is also known as an *all-points fixed radius near neighbor problem* [2].) Given a radius r, for each pixel we are to determine the number of black pixels at distance r or less. The set of PE's at distance r or less from a PE P is called an *r-ball centered at P*.

To perform the all-points radius query efficiently, we need to impose an additional restriction on the metric. A metric is a *vector metric* if it is monotone and if $d(P, Q)$ is dependent only on the vector from P's position to Q's position. Vector metrics have the property that for any radius r and any PE's P and Q, the r-ball centered at P is just a rigid translation (with no rotation) of the r-ball centered at Q, i.e., the metric looks the same everywhere. All l_P metrics are vector metrics, and it seems that all metrics encountered in practice are vector metrics.

Theorem 11: In an $n \times n$ mesh-connected computer, for any vector metric and for any radius, an all-points radius query can be solved in $\theta(n)$ time.

Proof: Suppose that the radius r is sufficiently small so that the r-ball centered at PE$(n/2, n/2)$ lies entirely within the $n \times n$ mesh. The monotonicity guarantees that to traverse the perimeter of the r-ball, one will visit at most $4n$ PE's. (Fig. 10 shows a typical r-ball.) Suppose that each PE has a value, denoted B, which is the number of black pixels in its row to its left. Consider a traversal of the perimeter of an r-ball during which a running total will be kept. Initially the total is 0, and as the traversal reaches a PE which is rightmost in its row (among those in the r-ball), one adds the B value, plus 1 if the pixel there is black. At each PE which is leftmost in its row (among those in the r-ball), one subtracts the B value. The total at the end of the traversal is the number of black pixels in the r-ball.

Using the above procedure is quite simple. To ensure that the traversal does not try to move off the $n \times n$ mesh, we think of the $n \times n$ mesh as being in the center of a $3n \times 3n$ mesh, where all of the added pixels are white and each real PE must simulate nine PE's. We redefine the r-ball centered at a PE P to be

$$\{Q: d(P, Q) \leqslant r \text{ and the } l_\infty \text{ distance from } P \text{ to } Q \text{ is } \leqslant n\}.$$

Notice that the new r-ball centered at a PE in the original mesh lies entirely in the $3n \times 3n$ mesh and contains the same

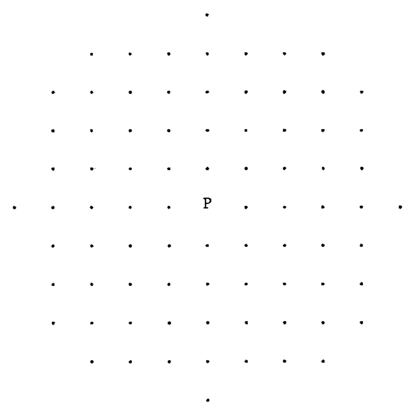

Fig. 10. A five-ball about P, using the Euclidean metric.

PE's of the original mesh as does the original r-ball. In particular, it contains exactly the same number of black pixels.

To start, use a row rotation to have each PE determine its B value. Then all PE's in the original mesh create a record which acts as their representative in the traversal. Since the r-balls are identical, these representatives can be passed along in a lockstep fashion as they perform the traversal and return to their originating PE. No matter what the value of r, the modified r-ball has a perimeter of $O(n)$, so the algorithm is finished in $\theta(n)$ time. □

VI. Conclusion

Good solutions to higher level tasks increase the usefulness of mesh-connected computers. In a hybrid image processing system in which a mesh-connected computer is connected to a standard computer, much of the time is spent moving data from one machine to the other. If more of the higher level processing can be done in the mesh-connected computer, then its massive parallelism will be better exploited and the total computation time of the system will be significantly reduced.

Towards this end, we have shown that a large number of geometric problems can be solved in $\theta(n)$ time on an $n \times n$ mesh-connected computer. Many of these problems involve combining information from PE's far apart, in which case the use of sort-like data movements is crucial to the development of optimal algorithms. Often, people performing image processing try to avoid sorting, which may account for the previous $\theta(n^2)$ solutions for these problems.

Since it takes $\theta(n)$ time for data to travel across an $n \times n$ mesh, all of our algorithms have optimal worst case times. However, there may be situations where the answer can be found faster. For example, suppose that no black component has an l_∞ external diameter greater than D. Then by partitioning the mesh into subsquares of size $\theta(D)$, and sharing data between adjacent squares, in $\theta(D)$ time each black component can determine its extreme points. In the appropriate situations, this technique can be used with all of our results, reducing them to $\theta(D)$ time. One particularly interesting application of this technique occurs when we apply it to Theorem 11, when the l_∞ metric is used with a radius of D. In $\theta(D)$ time, each PE will known the number of blacks in a square centered at the PE. If each PE then becomes black if and only if more than half of the PE's in its square were previously black, then we have performed bilevel median filtering with a window of edge length $2D + 1$ in $\theta(D)$ time. Median filtering with a window of edge length $2D + 1$ on an arbitrary grey level picture can also be accomplished in $\theta(D)$ time, but the algorithm is far more complicated [26].

The results of this paper suggest many additional questions, most of which are still open. Some obvious ones are whether there is a $\theta(n)$ time algorithm for locating all internal centers, and whether for any p there are $\theta(n)$ time algorithms for locating l_P planar centers and computing l_P planar radii. If one considers three-dimensional "pictures," then almost all of the questions are open. That is, suppose that there is an $n \times n \times n$ three-dimensional picture stored one pixel per processor in an $n \times n \times n$ three-dimensional mesh-connected computer. How fast can one locate extreme points, determine internal distances, compute diameters, etc.? Many of our convexity and external distance algorithms reduce the amount of data by one dimension, reducing an $n \times n$ picture to $\theta(n)$ points, giving $\theta(n)$ time algorithms. In three dimensions, this would give $\theta(n^2)$ time, which is not necessarily optimal. We have a $\theta(n)$ time algorithm for finding the distance to the nearest processor with a different label, but for most other three-dimensional problems, we do not yet have $\theta(n)$ time algorithms.

For internal distances, the difficulty in extending to three dimensions is particularly acute. Our two-dimensional algorithms considered $k \times k$ subsquares, ignoring all of the square except for the $\theta(k)$ border elements, and constructed a distance matrix with $\theta(k^2)$ entries. This matrix was able to fit in the original square. In three dimensions, $k \times k \times k$ subcubes have $\theta(k^2)$ border elements, which would require a distance matrix containing $\theta(k^4)$ entries. This matrix will not fit in the original cube, so the method fails. One could use the simple method of having each marked processor tell its neighbors that they are at distance one, each of them tells their neighbors they are at distance two, and so on, but this has a worst case time of $\theta(n^3)$.

Finally, we should mention connections to mesh automata. As was noted in Section II, for any given finite state automaton, once the mesh becomes large enough, the individual PE's do not have enough memory to store their coordinates or, say, distances to other PE's. This means that some of the problems solved herein, such as determining the external diameter of each component, must be modified if they are to have mesh automata solutions. For example, one may take a black/white picture and want to compute the external diameter of the black pixels, where the answer is emitted by PE(0, 0) one bit at a time. Except for problems involving internal distances, each problem discussed herein, or an appropriately modified version, can be solved in $\theta(n)$ time on a mesh automaton by using clerks to simulate the solution given here. (*Clerks* appear in [24] and [25] and can be viewed as a systematic use of counters.) The problems involving internal distances cause difficulties because our solutions create arrays having $\theta(n^2 \log(n))$ bits of information, which cannot be held in an $n \times n$ mesh automaton. Beyer [3] considered the problem of having a mesh automaton mark a minimal internal path between two given PE's in the same component, and it is still an open question whether there is a $\theta(n)$ time algorithm for this problem.

References

[1] M. J. Atallah and S. R. Kosaraju, "Graph problems on a mesh-connected processor array," Johns Hopkins Univ., Baltimore, MD, Rep. JHU 81-16, 1981.
[2] J. L. Bentley, "Multidimensional divide-and-conquer," *Commun. ACM*, vol. 23, pp. 214-229, 1980.
[3] W. T. Beyer, "Recognition of topological invariants by iterative arrays," Ph.D. dissertation, M.I.T., Cambridge, 1969.
[4] P. E. Danielsson and S. Levialdi, "Computer architectures for pictorial information systems," *IEEE Computer*, vol. 14, pp. 53-67, 1981.
[5] C. R. Dyer and A. Rosenfeld, "Parallel image processing by memory augmented cellular automata," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 29-41, 1981.
[6] M. A. Fischler, "Fast algorithms for two maximal distance problems with applications to image analysis," *Pattern Recognition*, vol. 12, pp. 35-40, 1980.
[7] H. Freeman and R. Shapira, "Determining the minimal-area encasing rectangle for an arbitrary closed curve," *Commun. ACM*, vol. 18, pp. 409-413, 1975.
[8] M. J. E. Golay, "Hexagonal parallel pattern transformations," *IEEE Trans. Comput.*, vol. C-18, pp. 733-740, 1969.
[9] S. B. Gray, "Local properties of binary images in two dimensions," *IEEE Trans. Comput.*, vol. C-20, pp. 551-561, 1971.
[10] A. Hubler, R. Klette, and G. Werner, "Shortest path algorithms for graphs of restricted in-degree and out-degree," *EIK*, vol. 18, pp. 141-151, 1982.
[11] K. Hwang and K.-S. Fu, "Integrated computer architectures for image processing and database management," *IEEE Computer*, vol. 15, pp. 51-60, 1982.
[12] C. E. Kim and A. Rosenfeld, "Digital straight lines and convexity of digital regions," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 149-153, 1982.
[13] S. Levialdi, "On shrinking binary picture patterns," *Commun. ACM*, vol. 15, pp. 7-10, 1972.
[14] R. Miller and Q. F. Stout, "The pyramid computer for image processing," in *Proc. 7th Int. Conf. Pattern Recognition*, 1984, pp. 240-242.
[15] M. Minsky and S. Papert, *Perceptrons.* Cambridge, MA: M.I.T. Press, 1969.
[16] E. F. Moore, "Machine models of self-reproduction," in *Proc. Symp. Appl. Math.*, 1962, vol. 14, pp. 17-33.
[17] D. Nassimi and S. Sahni, "Finding connected components and connected ones on a mesh-connected parallel computer," *SIAM J. Comput.*, vol. 9, pp. 744-757, 1980.
[18] A. P. Reeves, "Parallel computer architectures for image processing," *Comput. Vision, Graphics, Image Processing*, vol. 25, pp. 68-88, 1984.
[19] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice.* Englewood Cliffs, NJ: Prentice-Hall, 1977, ch. 8.
[20] A. Rosenfeld, "Parallel image processing using cellular arrays," *IEEE Computer*, vol. 16, pp. 14-20, 1983.
[21] M. I. Shamos, "Computational geometry," Ph.D. dissertation, Yale Univ., New Haven, CT, 1978.
[22] J. Sklansky, "Recognition of convex blobs," *Pattern Recognition*, vol. 2, pp. 3-10, 1970.
[23] R. Stefanelli and A. Rosenfeld, "Some parallel thinning algorithms for digital pictures," *J. ACM*, vol. 18, pp. 255-264, 1971.
[24] Q. F. Stout, "Using clerks in parallel processing," in *Proc. 23rd IEEE Symp. Foundations of Comput. Sci.*, 1982, pp. 272-279.
[25] —, "Topological matching," in *Proc. 15th ACM Symp. Theory of Comput.*, 1983, pp. 24-31.
[26] —, "Sorting, merging, selecting, and filtering on tree and pyramid machines," in *Proc. Int. Conf. Parallel Processing*, 1983, pp. 214-221.
[27] —, "Geometric algorithms for a mesh-connected computer with broadcasting," to be published.
[28] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. ACM*, vol. 20, pp. 263-271, 1977.
[29] G. T. Toussaint, "Pattern recognition and geometrical complexity,"

in *Proc. 5th Int. Conf. Pattern Recognition*, 1980, pp. 1324–1347.

[30] S. H. Unger, "A computer oriented towards spatial problems," *Proc. IRE*, vol. 46, pp. 1744–1750, 1958.

[31] —, "Pattern detection and recognition," *Proc. IRE*, vol. 47, pp. 1737–1752, 1959.

[32] —, "Pattern recognition using two-dimensional bilateral, iterative, combinatorial switching circuits," in *Proc. Symp. Math. Theory of Automata*, 1962, pp. 577–591.

[33] F. L. Van Scoy, "The parallel recognition of classes of graphs," *IEEE Trans. Comput.*, vol. C-29, pp. 563–570, 1980.

[34] K. Voss and R. Klette, "On the maximum number of edges of convex digital polygons included into a square," Friedrich-Schiller-Universitat Jena, Forschungsergebnisse, no. N/82/6.

[35] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, pp. 11–12, 1962.

Russ Miller (S'82) was born in Flushing, NY, on January 8, 1958. He received the B.S. and M.A. degrees in computer science/mathematics from the Department of Mathematical Sciences, State University of New York, Binghamton, in 1980 and 1982, respectively, and the Ph.D. degree, also from SUNY, Binghamton, in 1985.

His current research interests include algorithms, data structures, and software engineering, particularly for parallel computers.

Dr. Miller is a member of the IEEE Computer Society, the Association for Computing Machinery, and Phi Beta Kappa.

Quentin F. Stout (M'82) was born in Cleveland, OH, on September 23, 1949. He received the B.A. degree in mathematics from Centre College, Danville, KY, in 1970, and the Ph.D. degree, also in mathematics, from Indiana University, Bloomington, in 1977.

From 1976 to 1984 he was an Assistant and then an Associate Professor of Mathematical Sciences at the State University of New York, Binghamton. He is currently an Associate Professor in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His research interests include algorithms (particularly those for parallel computers), data structures, and image processing.