JOURNAL OF ALGORITHMS 6, 200–212 (1985)

Pyramid Computer Solutions of the Closest Pair Problem

QUENTIN F. STOUT*

Mathematical Sciences, State University of New York, Binghamton, New York 13901, U.S.A.

Received March 1982

Given an $N \times N$ array of 0s and 1s, the closest pair problem is to determine the minimum distance between any pair of ones. Let D be this minimum distance (or $D = 2N$ if there are fewer than two 1s). Two solutions to this problem are given, one requiring $O(\log(N) + D)$ time and the other $O(\log(N))$. These solutions are for two types of parallel computers arranged in a pyramid fashion with the base of the pyramid containing the matrix. The results improve upon an algorithm of Dyer that requires $O(N)$ time on a more powerful computer. © 1985 Academic Press, Inc.

1. INTRODUCTION

In the field of computational geometry, the (2-dimensional) closest pair problem is to determine $\min\{d(x, y): x, y \text{ in } P,\ x \neq y\}$, where P is a collection of n points in 2-dimensional space and d is some metric, usually l_1 (the "taxi-cab" or "Manhattan" metric), l_∞ ("chessboard" metric), or l_2 ("Euclidean" metric). (The l_p distance from (a, b) to (c, d) is $[(a - c)^p + (b - d)^p]^{1/p}$ for $1 \leqslant p < \infty$, and is $\max(|a - c|, |b - d|\}$ for $p = \infty$.) For suitable metrics, including all l_p metrics, this problem can be solved in $\theta(n^*\log(n))$ time on a random access machine (RAM). (See Bentley and Shamos [2], Shamos [13], or Yuval [20]. If the floor function can be computed in unit time then even faster algorithms are possible: see Rabin [10] and Fortune and Hopcroft [6].) Dyer [4] has examined a related problem arising in image processing and pattern recognition: given a metric d and an $N \times N$ array A of 0s and 1s, find

$$\min\{d(p,q): p,q \text{ in } \{1 \cdots N\} \times \{1 \cdots N\}, p \neq q, A(p) = A(q) = 1\}.$$

*Current address: Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109, U.S.A. This research was partially supported by National Science Foundation Grant MCS-83-01019.

If A has fewer than two 1s then we arbitrarily say that the answer is $2N$, assuming that the metric is normalized so that its values are less than $2N$. From now on, the term "closest pair problem" will refer to this second version with the distance measured by the l_1 metric.

Notice that any RAM program for the closest pair problem must take $\Omega(N^2)$ time since each entry of A must be examined. Dyer showed that with a log-space pyramid cellular automation (log-space PCA, defined in Sect. 2), the closest pair problem can be solved in $\theta(N)$ time using $\theta(N^2)$ processors. We introduce two new, faster solutions to the closest pair problem. Both solutions are for parallel computers arranged in a pyramid fashion, using $\theta(N^2)$ processors. The first, given in Section 3, takes $\theta(\log(N) + D)$ time, where D is the answer. This solution uses a primitive model introduced in Section 2. The second solution appears in Section 4 and takes $\theta(\log(N))$ time on a pyramid cellular automation (PCA), also defined in Section 2.

By simulating the PCA, more powerful computers such as the log-space PCA or the paracomputer can also solve the closest pair problem in $\theta(\log(N))$ time. The **paracomputer** is a model in which each processor can transmit a value to any other processor in unit time, with arbitrarily many of these transmissions occurring simultaneously. It is easy to show that if write conflicts are prohibited (i.e., no two processors may simultaneously write to the same processor at the same time), then any paracomputer algorithm for the closest pair problem must take $\Omega(\log(N))$ time, no matter how many processors are used. Therefore our PCA solution is optimal for a wide range of models.

2. Models of Parallel Computation

We first recall definitions from Dyer [4]. A **pyramid cellular automation** (PCA) for an input array of size $2^n \times 2^n$ consists of a finite state automation (FSA) which is replicated at each node of a complete 4-ary tree of height n. The next state of an FSA at height k depends upon its current state and the current state of nine neighbors: four children at height $k - 1$, a parent at height $k + 1$, and the four adjacent nodes at level k. (For nodes along the sides of the pyramid, missing neighbors should be thought of as being nodes permanently in some "side" state.) The leaves form a **base array** which, if there were no parents, would be a cellular array. (A cellular array is also known as an iterative array or cellular automaton. See Cole [3], Levitt and Kautz [7], Moore [9], or von Neumann [18].) An **input configuration** is specified by defining the initial states of the base array, with all higher nodes in a quiescent state. A **step** of computation consists of a simultaneous state transition at each node. The base array is initialized so that each node contains the corresponding entry of the matrix. Since we think of each entry of the matrix as being a pixel (picture element) in a digitized picture, we use the term **pixel** to mean a node in the base array. A pixel which is initially 1 will be called an **active pixel**. PCAs have appeared in Dyer [4], Dyer and Rosenfeld [5], Sakoda [11], Stout [15], Tanimoto [16], and Tanimoto and Klinger [17].

A **log-space pyramid cellular automation** (log-space PCA) is a PCA in which each node contains a processor having a fixed number of registers, each of length $2k$, where k is the height of the node.

We introduce a new model of computation, a **centrally organized cellular array** (COCA). This is similar to a PCA, having an identical base, except that

(i) The apex of the pyramid is a RAM.

(ii) All intermediate nodes are simple switching devices connected only to their parent and four children. They can receive a signal from their parent and pass it to all four children in the next time unit, and they can receive signals from all four children and pass up their minimum in the next time unit.

(iii) The state of a pixel at time $t + 1$ is determined by its state, the states of its four adjacent pixels, and the value received from its parent, at time t.

For all of the models, it will often be more convenient to describe nodes sending and receiving values, instead of describing how a node's state depends upon those adjacent to it.

3. A COCA Solution

We first give a COCA solution to the closest pair problem. Throughout we let $n = \log_2(N)$. The algorithm is quite simple.

(i) The apex signals the start, and the signal reaches all pixels at time n.

(ii) Each active pixel starts a wave which spreads outward, at time $t + n$ consisting of all pixels at distance $t - 1$ from the active pixel.

(iii) When two waves collide a signal is sent up towards the apex, taking n units of time.

The apex determines the solution by timing the period until the first return

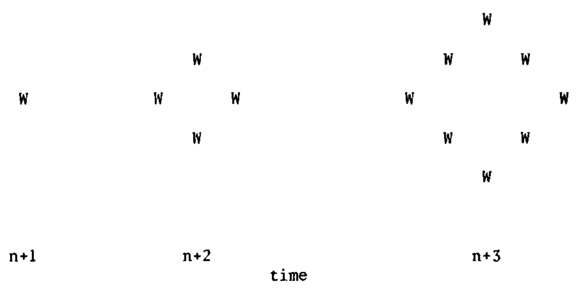

FIG. 1. A spreading wave for the l_1 metric.

signal, or noting that no signal has returned by time $2n + N - 1$, in which case there are not two active pixels and the answer is $2N$.

To complete the solution we need to specify how the waves propogate and collide. To propogate a wave we use a state W. At time n the pixels receive the state signal. At time $n + 1$ each active pixel goes into the W state and all others go into a start state. The wave propogates as in Fig. 1. If a pixel is not dormant at time $n + k$ $(k \geqslant 1)$ and one of its adjacent pixels is in state W, then the pixel goes into state W at time $n + k + 1$. Once a pixel reaches state W, in the next time cycle it goes into a dormant state, remaining dormant forever.

There are only two ways waves can collide.

(i) At time $n + k$ $(k \geqslant 1)$ a pixel and one of its four adjacent pixels are both in state W. In this case both pass upo an O (odd) signal at time $n + k + 1$.

(ii) At time $n + k$ $(k \geqslant 1)$ a nondormant pixel not in the W state has an opposite pair of pixels adjacent to it, both of which are in the W state. In this case it passes up an E (even) signal at time $n + k + 1$.

Notice that if there are two active pixels at distance one then at time $n + 2$ they both pass up an O. If there are two at distance two then at time $n + 2$ there is a pixel between them which passes up an E, unless it passes up an O because it is also active. In general, two active pixels at distance $2k + 1$ will have their waves collide at time $n + k + 1$ and result in an O signal, while two active pixels at distance $2k + 2$ will have their waves collide at time $n + k + 1$ and result in an E signal, unless their waves have been involved in earlier collisions.

Using the ordering $O < E$, at time $t + 1$ each intermediate node passes up the minimum of the signals to it at time t. If the apex first receives a return signal passed to it at time t, then the answer is $2(t - 2n) - 1$ if the signal is an O, while it is $2(t - 2n)$ if the signal is an E.

4. A PCA Solution

In this section we assume that the apex of the PCA is connected to some external computer. We initially give an algorithm which works in $O(\log(N))$ time if each node of the pyramid is a RAM with a fixed number of words storage, each word having $\Omega(\log(N))$ bits, and which can perform operations such as addition, comparison, and sending a word of data, in unit time. (This is known as the unit cost criterion. See Aho, Hopcroft, and Ullman [1].) This more powerful model is called the **pyramid computer**. Notice that the nodes of a pyramid computer can store the coordinates of pixels, while the nodes of a PCA cannot.

We use the informal term **neighbor** to mean a nearby node at the same height.

4.1 An Algorithm for the Pyramid Computer

The central point of the algorithm is that a node P at height $k \geqslant 1$ is responsible for determining if there are any active pixels beneath it which are less than 2^{k+1} from any other active pixels, and if so is to determine the minimum distance from any active pixel beneath P to any other active pixel. Notice that if there are two active pixels beneath P then their distance must be less than 2^{k+1}. Also notice that if P has active pixels beneath it, then it must check the active pixels beneath its neighbors to determine if they are within 2^{k+1} of P's active pixels.

More precisely, exactly one of the following will occur:

(a) P has no active pixels beneath it and passes up a message saying so.

(b) At least one of P's children has determined that it has an active pixel beneath it which is less than 2^k from some other active pixel, in which case P passes up the minimum such distance received from any child.

(c) P determines that the minimum distance from an active pixel beneath it to any other active pixel is greater than a distance being passed up by some other node at P's level, in which case P passes up a message that its pixels are irrelevant to the answer.

(d) P determines that the minimum distance between an active pixel beneath it and any other active pixel is at least 2^k, and less than 2^{k+1}, in which case it passes up the minimum distance.

(e) P has exactly one active pixel beneath it and all other active pixels anywhere are 2^{k+1} or further away, in which case P passes up the coordinates of its active pixel. Coordinates are relative so P uses k bits to specify each component of the coordinates.

1	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

a) base pixels

(0,0)	N	(0,1)	N
(1,1)	N	N	d=2
N	N	N	d=2
N	N	N	N

b) height k=1

X	d=2
N	d=2

c) height k=2

d=2

d) height k=3

FIG. 2. Values calculated at the nodes. N—no active pixels beneath; X—minimum distance cannot involve pixels below; (a, b)—one active pixel at relative position (a, b); $d = i$ —minimum distance from any active pixel below to any other active pixel.

The algorithm proceeds by having all nodes at height 1 compute their values in stage 1, then all at height 2 compute their values in stage 2, etc. There is a constant c such that each stage takes c units of time, resulting in $\theta(\log(N))$ total time. Figure 2 gives an example which shows the information passed up by each node.

Suppose that the correct answer is D. We will show that the algorithm works properly by showing that the following invariant is maintained: at the end of stage i, if $2^{i+1} > D$ then at least one node at height i sends up distance D, while otherwise every active pixel has its coordinates sent up, and no distances are sent up. We call this the **stage invariant**.

4.1.1 The Node Algorithm

To see how each node performs its calculations, suppose P is at height $k \geqslant 1$. (The pixels at height 0 merely pass up either 0 or 1, corresponding to

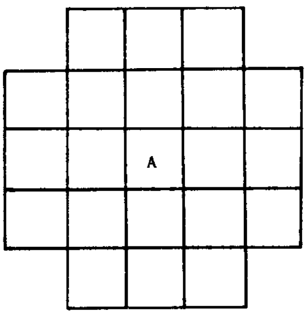

FIG. 3. Neighbors above pixels within 2**[height(A) + 1] of pixels beneath A.

their initial contents.) Each of P's children sends it one piece of information. If none of them had active pixels, then case (a) holds. If any of them pass up a type (b) or (d) message, i.e., a distance, then case (b) holds, since that distance must be less than 2^k. Notice that, because of the stage invariant, P knows that D was passed up by the previous stage, and has ensured that if it received D then it has passed D up.

If P receives no distances, but receives a type (c) message from a child, then it sends up a type (c) message. In this case it also knows that D was passed up in the previous stage, but it cannot possibly have received it.

Finally, suppose P receives only type (a) and (e) messages, with at least one child sending up the coordinates of an active pixel. P must determine whether cases (c), (d), or (e) apply. Any pixel within 2^{k+1} of a pixel beneath P must be beneath P or one of the twenty neighbors indicated in Fig. 3. Each of these neighbors exchanges with P the coordinates of all known subordinate active pixels. Since a node receives the coordinates of at most one active pixel from each child, P exchanges at most 4 coordinates with any other node. This occurs in parallel throughout P's level, taking a fixed amount of time independent of the height. P receives at most 80 coordinates from its neighbors, and in a fixed amount of time can compute the minimum distance between an active pixel beneath it and any other active pixel received. If this distance is less than 2^{k+1} then case (d) holds and the distance is passed up. Otherwise case (e) holds, P has exactly one active pixel beneath it, and passes up its coordinates. (Recall that if P has two active pixels beneath it then they must be less than 2^{k+1} from each other.)

Actually, it may be that not all of P's neighbors will agree to exchange coordinates of active pixels. If one of them had received a distance or type (c) message from a child, then it knows that the answer was computed at a previous stage, and so sends a message to P, telling it that its efforts are in vain. In this situation, P sends up a type (c) message.

4.1.2. Maintaining the Stage Invariant

To see that the stage invariant is maintained, suppose it was true for the $k - 1$st stage, and it will be shown to be true for the kth stage. If $D < 2^k$ then at least one of the nodes on the level below sent up the correct distance. The only way a distance sent at stage $k - 1$ can fail to be passed up to the $k + 1$st level is if it was received by a node which also received a smaller distance. Therefore the minimum distance D must be passed up to the next stage. (Since any distance created is correct, distances too small can never occur.)

If $D \geqslant 2^{k+1}$ then the coordinates of each active pixel were sent up by the previous stage, and it must be that every node at the previous stage passed up either a type (a) or (e) message. Further, no node at level k which received the coordinates of an active pixel received the coordinates of more than one active pixel, for otherwise they would have been within 2^{k+1} of each other. After checking its neighbors, such a node determines that case (e) holds, and it passes up the coordinates. Therefore every active pixel is passed up to the next level.

Finally, if $2^k \leqslant D < 2^{k+1}$ then all nodes of the level below passed up either type (a) or (e) messages, and all active pixels reach their ancestors at height k. Since each node at height k which received the coordinates of an active pixel checked its appropriate neighbors, each active pixel which is within 2^{k+1} of another will have its height k ancestor determine its minimal distance to any other active pixel. Therefore at least one node at level k computes D and passes it up, and the stage invariant is maintained.

One final technicality is that the first stage must be separately considered, since the above proof is inductive. For this we note that the pixels sent up their 0 or 1 values, so all active pixels reached their level 1 ancestors and the situation is as described in the last paragraph.

4.2 The PCA Algorithm

Using RAMs at each node with unit time operations, the above algorithm is finished in $\theta(\log(N))$ time. Since each operation involving a node at height k uses numbers of at most $k + 2$ bits, a log-space PCA can also run the algorithm. If a unit time criterion is used for the log-space PCA operations, then the algorithm will finish in $\theta(\log(N))$. If instead one assumes the nodes at the ith level need $\theta(i)$ time to perform their calculations, then it will take $\theta(i)$ time to perform the ith stage, and $\theta(\log(N)^2)$ total time. To convert this to a $\theta(\log(N))$ algorithm for a PCA without logarithmic space we need to revise the algorithm so that there is a uniform bound on the information being stored at any node and on the delay each level needs before starting to pass information up to the next level. We transform the algorithm piecemeal to achieve these requirements.

First, each node must compute and pass its answer one bit at a time. If an FSA is simultaneously given four numbers of equal but arbitrary length, one bit at a time with the high-order bits first, then it can output their minimum in a bitwise fashion, high-order bit first. Therefore all distances are passed up high-order bits first, synchronized so that all distances in the range 2^k to $2^{k+1} - 1$ (i.e., all distances first computed at stage k) start arriving at a node before any distances 2^{k+1} or greater. Since greater distances arrive later, they are ignored.

A node may need to compute distances between several pairs of points, though the number of such pairs does not depend on the node's height. To see how such calculations are performed, all in parallel, suppose P is a node at height k which needs to compute the distance between active pixels at positions (x_1, y_1) and (x_2, y_2). Since each pixel is either beneath P or one of the neighbors indicated in Fig. 3, only $k + 2$ bits are needed to specify x_1, y_1, x_2, and y_2. Coordinates will be passed starting with the low-order bit. If P knew that $x_2 \geqslant x_1$ and $y_2 \geqslant y_1$, then it could compute the distance bitwise, low-order bit first. Unfortunately, sometimes P can only be sure of the relative values of one set of coordinates. For example, if (x_1, y_1) is beneath P's lower left child and (x_2, y_2) is beneath P's right neighbor, then $x_2 > x_1$, but there is no initial knowledge about the ordering of the y's. In such cases P does two bitwise calculations, one assuming $y_2 \geqslant y_1$ and the other assuming $y_2 < y_1$. Only when P is finished receiving y_1 and y_2 will it know which calculation was correct.

As each bit is calculated P passes it to its lower left child, which in turn passes it to its lower left child, and so on. When a bit reaches a base node it is stored and the pixel sends a signal to its parent to stop passing bits for this calculation. The next bit is stored by this parent at height 1, which then informs its parent to stop sending, and so on.

P performs all of its distance calculations in parallel, and when they are finished it contains the high-order bits for each and, in those cases where two calculations were being done for the same pair of points, now knows which is correct. At this point it can determine if case (d) or (e) holds, and if case (d) holds then it starts computing the minimum of all correct distance calculations. Notice that the order of the bits has been reversed, so they are now in the correct order to be passed up as distances. These calculations introduce a delay of $\theta(\text{height}(P))$ before P can pass along distance information, but such a delay can happen only once on any path from a pixel to the apex. In particular, no distance expressible in k bits need wait for any calculations involving distances expressed in $k + 1$ or more bits, nor for any calculations involving distances expressed in $k - 1$ or fewer bits. Therefore the delays are not multiplicative, and the entire calculation is done in $\theta(\log(N))$ time.

There is another detail to consider. If P has two or more active pixels beneath it, and received no distances or type (c) messages, then P is certain that it will pass up a distance or type (c) message. However, if it has only one active pixel beneath it then it is initially unsure wheter cases (c), (d), or (e) apply. For example, there may be another active pixel beneath one of P's neighbors, but only after computing distances can P determine that the distance is too large and that case (e) applies, meaning that P should have been passing up the coordinates of its subordinate active pixel. To avoid introducing delays which would be multiplicative, whenever a node has only one active pixel beneath it (and receives no distances nor type (c) messages), it passes up the pixel's coordinates as it is also calculating distances. (A node receives the coordinates of an active pixel as $k - 1$ pairs of bits, low-order bits first, followed by a signal that they are finished. It passes these along in the order received, then adds a pair of bits indicating which child this pixel is under, and then passes the signal that the coordinates are finished.) If the distances are too large they are ignored, while otherwise they are starting up at their usual time.

This causes one final problem: since the distance calculations are temporarily stored in subordinate nodes, and since distance calculations could be proceeding concurrently at several levels on the same path, we need to be sure that there is a uniform bound on the amount of information any node is being asked to store. Notice that if a node at height k is computing the distance between two active pixels then their distance cannot exceed $5*2^k$. (See Fig. 3). Even if the distance exceeds $2^{k+1} - 1$, it will be in the correct range for the node's parent or grandparent. Therefore no node need store calculations for more than three levels, which will be the first three trying to store values. Any later attempts to store information will be ignored, since they will result in distances no less than the ones previously started.

The complete algorithm for the PCA is quite complicated, and can probably be simplified considerably. We have tried to outline its development in a natural manner, which seems useful for several PCA algorithms. This consists primarily of first developing an algorithm for the pyramid computer with RAMs at each node, and then modifying the algorithm to replace the RAMs with FSA. A similar approach has been used in Stout [15], where the "clerk" data structure aids in the modification. Using RAMs one can see a forest, while the FSA model seems to be one knotty tree after another.

5. Conclusion

We have shown that the centrally organized cellular array can solve the closest pair problem in $\theta(\log(N) + D)$ time, where D is the answer, and that the pyramid cellular automation can solve this problem in $\theta(\log(N))$ time. Since any parallel computer which prohibits write conflicts must take $\Omega(\log(N))$ time for this problem, this shows that, among the wide variety of parallel computers prohibiting write conflicts, the PCA is an optimal architecture for the problem. It seems to be the "simplest" optimal architecture, although we see no way of proving this, nor even of stating it precisely, since there does not even appear to be a good way of defining when one architecture is simpler than another.

Besides our models and Dyer's log-space PCA, this problem has been considered for other parallel computers. In Miller and Stout [8] an algorithm taking $\theta(N)$ time is given for an $N \times N$ mesh-connected computer. A mesh-connected computer is like a cellular array, except that the nodes are RAMs with unit time operations. In Stout [14] an algorithm taking $\theta(N^{2/3})$ time is given for an $N \times N$ mesh-connected computer with broadcasting. Broadcasting is a feature whereby a node can send a word of information to all other nodes in unit time, with the restriction that only one such message at a time can occur. Both of these algorithms are optimal for their respective models.

Our pyramid computer algorithm can be used to give an optimal algorithm for the uniprocessor RAM. As was mentioned in Section 1, the uniprocessor RAM must take $\Omega(N^2)$ time. To achieve this lower bound, the RAM simulates the pyramid computer. Since the pyramid has $(\frac{4}{3})N^2 - (\frac{1}{3})$ nodes, normally the RAM would need $\theta(T^*N^2)$ time to simulate an algorithm taking T time units on the pyramid computer. However, our algorithm has the property that only nodes at level i do any calculations during stage i. Using this, it takes the RAM CN^2 time to simulate the first stage for some constant C, $CN^2/4$ time to simulate the second stage, and so on, for $\theta(N^2)$ total time.

The algorithms in this paper are for the l_1 metric, but can be easily adapted to the l_∞ metric. For the COCA, this means that the waves spread as squares with edges parallel to the sides of the base, instead of the diamond pattern of the l_1 metric. To extend the algorithms to, say, the l_2 metric requires more effort. For the pyramid computer algorithm there are few changes, particularly if one compares the squares of distances, saving the square root operation until the very end. For the PCA there is additional difficulty in performing the multiplications, which can be overcome using clerks [15]. Clerks can also be used to help give a COCA solution to the l_2 closest pair problem. Such modifications would add a great deal of complication, and their details are of little value.

Pyramid-like architectures have been discussed for some time (Uhr [19]), but only recently has it become feasible to build such a structure. One such machine is already under construction (Schaefer [12]). Pyramids have a simple, regular structure which offers the potential of logarithmic algorithms, although, as we have seen, it sometimes requires a fair amount of effort to achieve that potential.

Acknowledgments

The author would like to thank Bette Warren and the reviewer for several useful suggestions.

References

1. A. V. Aho, J. C. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1975.
2. J. L. Bentley and M. I. Shamos, Divide and conquer in multidimensional space, *in* "Proc. 8th ACM Sympos. on Theory of Computing, 1976," pp. 220–230.
3. S. N. Cole, Real-time computations by n-dimensional iterative arrays of finite-state machines, *IEEE Trans. Comput.* **C-18** (1969), 349–365.
4. C. R. Dyer, A fast parallel algorithm for the closest pair problem, *Inform. Process. Lett.* **11** (1980), 49–52.
5. C. R. Dyer and A. Rosenfeld, "Cellular Pyramids for Image Analysis," TR-544, Computer Science Center, Univ. of Maryland, 1977.
6. S. Fortune and J. Hopcroft, A note on Rabin's nearest neighbor algorithm, *Inform. Process. Lett.* **8** (1976), 20–23.
7. K. N. Levitt and W. H. Kautz, Cellular arrays for the solution of graph problems, *Commun. ACM* **15** (1972), 789–801.
8. R. Miller and Q. F. Stout, Geometric algorithms for digitized pictures on a mesh-connected computer, *IEEE Trans. Pattern Anal. Mach. Intell.* **7** (1985), 216–228.
9. E. F. Moore, Machine models of self-reproduction, *in* "Proc. Sympos. in Applied Math.," No. 14, 1962, pp. 17–33.
10. M. O. Rabin, Probabilistic algorithms, *in* "Algorithms and Complexity: New Directions and Recent Results" (J. F. Traub, Ed.), pp. 21–39, Academic Press, New York, 1976.
11. B. Sakoda, "Parallel Construction of Polygonal Boundaries from Given Vertices on a Raster," Comput. Sci. Dept. Report CS81-21, Penn. State Univ., 1981.
12. D. H. Schaefer *et al.*, "A Pyramid of MPP Processing Elements," Tech. report, George Mason Univ., 1984.
13. M. I. Shamos, Geometric complexity, *in* "Proc. 7th ACM Sympos. on Theory of Computing, 1975," pp. 224–233.
14. Q. F. Stout, Broadcasting in mesh-connected computers, *in* "Proc. 1982 Conf. on Inform. Sci. Systems," Princeton Univ., pp. 85–90.
15. Q. F. Stout, Drawing straight lines with a pyramid cellular automaton, *Inform. Proc. Lett.* **15** (1982), 233–237.
16. S. L. Tanimoto, Towards hierarchical cellular logic: Design considerations for pyramid machines, Dept. of Comput. Sci. Tech. Report 81-02-01, Univ. of Washington, 1981.

17. S. L. Tanimoto and A. Klinger, "Structured Computer Vision: Machine Perception through Hierarchical Computation Structures," Academic Press, New York, 1980.
18. J. von Neumann, "The Theory of Automata: Construction, Reproduction, and Homogeneity," (A. Burks, Ed.), Univ. of Illinois Press, Urbana, 1966.
19. L. Uhr, Layered "recognition cone" networks that preprocess, classify, and describe, *IEEE Trans. Comput.* **C-21** (1972), 758–768.
20. G. Yuval, Finding nearest neighbors, *Inform. Process. Lett.* **5** (1976), 63–65.