Proceedings of the 1982 Conference on Information Sciences and Systems
Department of Electrical Engineering and Computer Science, Princeton University

BROADCASTING IN MESH-CONNECTED COMPUTERS

Quentin F. Stout
Mathematical Sciences
State University of New York
Binghamton, NY 13901

Abstract

This paper considers the effect of augmenting the local communication capabilities of a mesh-connected computer with a global communication feature called broadcasting. For a variety of semigroup and picture processing problems this provides significant improvement, although it cannot significantly aid sorting algorithms.

Introduction

Mesh-connected computers, defined below, are an important class of physically realizable parallel computers. Their regular pattern makes them relatively simple to design and build, and also provides a natural basis for stating and solving problems in picture processing and matrix calculation. Several variations, under a variey of names, have been studied for some time (e.g., Cole [2], Kautz, Levitt, and Waksman [9], Levialdi [10], von Neumann [18]), and recently several such computers have been built (e.g., Duff [3], Gilmore et al [7], Reeves [12]). However, the extremely local nature of their interconnections often restricts their speed in solving certain problems, and in this paper we consider one mechanism for overcoming this obstacle.

Let n and k be positive integers. A k-dimensional mesh-connected computer (k-MCC) of size n^k is a parallel computer consisting of identical processors located at positions $(J_1, \ldots, J_k)$ in k-dimensional space, where for all i in $1 \le i \le k$, J_i is an integer and $1 \le J_i \le n$. The processor at position $(J_1, \ldots, J_k)$ is connected to the processor at position $(I_1, \ldots, I_k)$ if and only if $1 = \sum_{i=1}^{k} |J_i - I_i|$. That is, each processor is connected to its 2^k nearest neighbors (or fewer, for processors on the sides). Other schemes are possible, such as having connections whenever $1 = \max\{|J_i - I_i| : 1 \le i \le k\}$, but no significant change occurs in the speed of computations. (See Hamacher [8].) When k=1, 2, or 3, a k-MCC is a physically realizable model of parallel computation. We assume that each processor has a fixed number of words of storage, where each word has length log(n), and that fundamental operations, such as addition or sending a word of information to a neighbor, occur in O(1) time. Some authors assume the word size is fixed and independent of n, but since many problems involve answers or intermediate results of length log(n) we feel that our word size assumption is more aligned with practice. This point is discussed in Dyer and Rosenfeld [5].

It is well-known that for many problems (sorting [16], function minimization [1], matrix calculations [6], etc.) their solution time on a k-MCC equals the maximum time needed to move information from one processor to another. For example, suppose each processor contains a number and we wish to sum these numbers, placing the result in processor (1,1, ..., 1). This can easily be done in O(n) time, and further requires O(n) time since it takes that long for information from (n,n, ..., n) to reach (1,1, ..., 1). (Throughout our analyses assume that k is fixed, and the time is considered soley as a function of n. If both n and k are considered variables then the solution time is O(k*n).) In O(n) time a k-MCC of size n^k can do $O(n^{k+1})$ calculations, while only n^k are needed to form the sum. Some of the potential speed-up of the parallelism has been lost, and a natural question is whether one can somehow modify a k-MCC to better exploit all the processors, while still keeping the simplicity and naturalness of the k-MCC. Pyramids and cones do this by adding processors at different levels, but are significantly more complex. (See, for example, Tanimoto and Klinger [15] or Uhr [17].) We consider a solution which is intermediate between mesh-connected computers and pyramids.

Gentleman [6] briefly considered the effect of supplementing a k-MCC with broadcasting, whereby any processor can transmit a value which is simultaneously received by all other processors, with the broadcast taking only unit time. Only one broadcast can occur in any time period. We will not discuss here the question of how best to implement broadcasting. While it is unrealistic in that it transmits information at unbounded speed, for a wide range of values of n it can be closely approximated by various busing schemes. Gentleman's Theorem 3 seems to indicate that broadcasting is of little use, but in this paper we will show that it provides a significant improvement for certain problems. Since it is a relatively simple feature which provides faster algorithms, we feel that it deserves consideration.

In this expository paper we simplify our discussions by considering only 2-MCC's, which are the ones of greatest interest. We orient these so that (1,1) is at the upper left and (n,n) is at the lower right. Each processor stores its row and column coordinates, as well as n. Further, each processor uses one of its words of memory as a counter, adding 1 after each operation to act as a clock. This is possible since the word size is log(n), and enables processors to follow instructions such as "At time i processor j broadcasts its value". This feature is used extensively, without further mention. It can also be achieved by a SIMD discipline, but this extra requirement is unnecessary. Finally, for reasons that will be clear shortly, we assume that n is a perfect cube. This will simplify various algorithms, and the extension to the general case is quite straightforward.

Semigroup Calculations

From now on we will think of our 2-MCC as being composed of $n^{2/3}$ disjoint blocks, each a square of $n^{4/3}$ processors. These are numbered in row-major order, as illustrated in Figure 1. A block is just

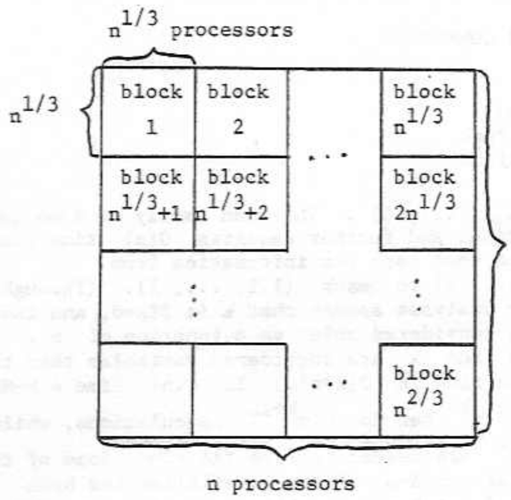

Figure 1. Block numbers for a 2-MCC of size n^2.

a smaller 2-MCC, and since each processor knows its position in the entire computer, it also knows which block it is in and its position within the block. For each block we will designate one of the processors as the block leader, with the choice of block leader depending on the problem. Block leaders use broadcasting for inter-block communication, with intra-block communication performed by the standard 2-MCC links. The following result is a simple example of this principle.

Theorem 1. Using broadcasting, the sum of n^2 numbers, stored one per processor in a 2-MCC of size n^2, can be found in $O(n^{2/3})$ time. Further, this time is optimal.

Proof. We give the desired algorithm as a sequence of steps. For this problem the block leader will be the one in the upper left corner.

Step 1. Simultaneously in each row of each block form the sum of the numbers in that row of that block, placing this value into the leftmost processor.

Step 2. In each block, in the block leader form the sum of the numbers calculated in Step 1. This value is the sum of all numbers in this block.

Step 3. Each block leader, in turn, broadcasts the number calculated in Step 2.

Step 4. As Step 3 proceeds, processor (1,1) calculates the sum of the broadcasted numbers, and when Step 3 is finished processor (1,1) contains the result.

Steps 1, 2, and 3 take $O(n^{2/3})$ time each. Since Step 4 requires only one addition per broadcasted number, it can be performed on-line without slowing down Step 3. Therefore the total time is $O(n^{2/3})$. The proof of optimality appears in [13], and consists of showing that an algorithm taking less than this time must ignore some numbers. □

This result has a natural extension to all dimensions, with the speed-up achieved by broadcasting decreasing as the dimension increases. Theorem 2 shows that, in O-notation, a k-MCC with broadcasting can sum N numbers in the same time as a (k+1)-MCC without broadcasting.

Theorem 2. Using broadcasting, the sum of n^k numbers, initially stored one per processor in a k-MCC, can be found in $O(n^{k/(k+1)})$ time. Further this time is optimal. □

Theorems 1 and 2 can be extended to any commutative semigroup operation, such as minimums, maximums, and products. Other calculations, such as averages and standard deviations, can essentially be reduced to a sequence of semigroup operations. Most problems do not have such a simple division into local computing followed by global computing, but instead tend to involve local computing, then global computing, and then a mixture of local and global. Several examples of this phenomenon appear in the next section.

Picture Processing

In this section we assume there is a {0,1}-valued matrix P(i,j), $1 \leq i,j \leq n$, which represents a digitized picture. The value of P(i,j) is initially stored in processor (i,j). Picture processing is a task ideally suited for a 2-MCC, so the speed-up available with broadcasting is of practical, as well as theoretical, significance. Our examples illustrate fundamental, nontrivial operations suitable for a wide range of applications.

We will use a basic fact about sorting in a 2-MCC of size k^2. We assume each processor contains a block of memory called its sort area. We will have each processor put a value into its sort area, and then these values will be sorted into snake-like order with the smallest value in processor (1,1), the next value in (1,2), and so on until (1,k), which is followed by (2,k), (2,k-1), ..., (2,1), (3,1), (3,2), ... See Figure 2. This sort can be done in O(k) time. (Thompson and Kung [16].)

1	2	3	4	5	6
12	11	10	9	8	7
13	14	15	16	17	18
24	23	22	21	20	19
25	26	27	28	29	30
36	35	34	33	32	31

Figure 2. Snake-like ordering for 36 processors.

First, suppose we want to find the median row of the 1's, that is, we want to find the row where about half the 1's are above it and about half are below it. Precisely, we want to find

$$\min\{x: \sum_{i=1}^{x} \sum_{j=1}^{n} P(i,j) \geq \sum_{i=x+1}^{n} \sum_{j=1}^{n} P(i,j) \}$$

The average row location of the 1's can be easily calculated using Theorem 1, but the median is slightly harder.

Proposition 3. Using broadcasting, a 2-MCC of size n^2 can find the median row in $O(n^{2/3})$ time.
Proof. For this problem the block leader will be at the lower left. For each row of blocks, the block leader of the leftmost block will also be called the block-row leader.
Step 1. In each row of each block, in the leftmost processor of that block's row form the number of 1's in that block's row.
Step 2. In each processor in the leftmost column of each block form the sum of all numbers calculated in Step 1 by that processor and all processors above which lie in that block. When done, the block leader contains the number of 1's within the block.
Step 3. Each block leader, in turn, broadcasts the number it formed in Step 2.
Step 4. While Step 3 proceeds, each block-row leadercomputes the total number of 1'2, the number of 1's in its block-row, and the number of 1's in blocks above its block-row.
Step 5. When Step 4 finishes, there is exactly one block-row leader for which the number of 1's in blocks above its block-row is less than half the total, but when the 1's in its block-row are added to this the number equals or exceeds half of the total. This block-row leader executes Steps 6 and 8.
Step 6. The block-row leader now starts a binary search among the rows of processors covered by its row of blocks. It initially broadcasts the row number of the middle row covered by this row of blocks.
Step 7. In each block in this block-row, each processor at the left edge of the block which is in the broadcasted row number now broadcasts, in turn, the number it calculated in Step 2.
Step 8. The block-row leader sums these broadcasted numbers, and then adds this to the number of 1's in blocks above this block-row. Using this, it knows whether the broadcasted row was too high or not, and it then broadcasts the appropriate new row number. Steps 7 and 8 are repeated until the median row has been found.

Steps 1, 2, and 3 each take $O(n^{2/3})$ time. In Step 4 each block-row leader has to do only a fixed number of calculations for each broadcasted number, so Step 4 slows Step 3 down by at most a constant multiple. Steps 5 and 6 take $O(1)$ time, and Steps 7 and 8 are repeated $\log(n^{2/3})$ times, where in each iteration Step 7 takes $O(n^{1/3})$ time and Step 8 takes $O(1)$. Therefore the total is $O(n^{2/3})$ □

This algorithm is typical in that the first round of broadcasting is used to determine which blocks need to continue.

Our next problem is to enumerate the extreme points of the convex hull of the 1's. That is, we consider each processor to be a single point and we want to identify those processors which are at the vertices of the smallest convex polygon containing all processors with a 1. Further, we want the processors at the extreme points to contain a number which indicates their relative position. The rightmost extreme point (or, if there are two rightmost extreme points then the lower of the two) is number 1, the next one in counter-clockwise order is number 2, and so on. We will use the following facts:

i) ([11]) In a 2-MCC of size k^2 the extreme points can be enumerated in $O(k)$ time.
ii) ([14]) In a 2-MCC of size k^2 suppose two disjoint convex sets have their extreme points enumerated. Then, by using broadcasting, the extreme points of their union can be enumerated in $O(\log(n))$ time.

Fact ii) comes from noticing that the extreme points of the union are the extreme points of the two regions minus an interval of extreme points from each. See Figure 3.

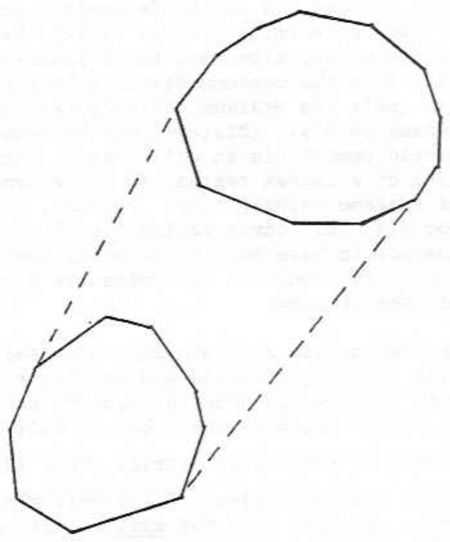

Figure 3. The convex hull of two disjoint convex regions.

Theorem 4. Using broadcasting, in a 2-MCC of size n^2 the extreme points of the set of 1's can be enumerated in $O(n^{2/3})$ time.
Sketch of Proof. Now the block leader is in the upper left.
Step 1. In each block, enumerate the extreme points of the 1's, if any.
Step 2. Each block leader, in turn, broadcasts "yes" if it is in a block containing 1's, and "no" otherwise.
Step 3. As Step 2 proceeds each block leader of a block containing 1's determines whether any processor in its block (not necessarily one containing a 1) could, if it contained a 1, be an extreme point. For example, if only blocks 1, 2, and 3 said "yes" then block 2 might have some extreme points, but if a block in the middle has its 8 nearest neighbors say "yes" then it cannot contain extreme points.
Step 4. Each block leader, in turn, now does the following: if the block has no 1's or cannot contain any extreme points, then it broadcasts "no", and otherwise broadcasts "yes". If this is the first "yes" of this step then this block is finished, while otherwise this block, and all preceeding ones, enumerates the extreme points of the 1's in this and all preceeding blocks.

Steps 1 and 2 take $O(n^{2/3})$ time. In [14] it is shown how to do Step 3 without slowing down Step 2 by more than a constant amount. Also in [14] it is shown that at most $4n^{1/3}$ block leaders can say "yes" in Step 4. This, plus fact ii) above, shows that Step 4 takes at most $O(n^{1/3}\log(n))$ time, so the entire algorithm takes $O(n^{2/3})$ time. □

When combined with broadcasting, the enumerated extreme points can be very useful. For example, once this numbering is known, the maximum distance between 1's can be determined in only O(# of extreme points) time by having the extreme points, in order, broadcast their location and noticing that the maximum distance between a pair of 1's equals the maximum distance between a pair of extreme points. (Distance may be measured in any metric computable in O(1) time.) Similarly the area of a convex region can be determined in O(# of extreme points) time. Further, if there are two disjoint convex regions, each of whose extreme points have been enumerated, then in only $O(\log(n))$ time one can determine the distance between the regions.

We now analyze more distance problems. Suppose the 1's are scattered and we wish to determine the minimum distance between any pair of 1's. Any ℓ_p metric could be used, but we will use only the simple ℓ_1 (taxi-cab) metric. That is, we will calculate $\min\{|i-\ell|+|j-m| : P(i,j)=P(\ell,m)=1 \text{ and } i\neq\ell \text{ or } j\neq m\}$. We call this the <u>minimum distance problem</u>, though in Dyer [4] it is called the closest pair problem.

First we show a simple solution for a 2-MCC without broadcasting, which will be used in our broadcasting-based algorithm. Our 2-MCC solution uses the simple spreading wave technique, for which we assume that each processor contains a memory cell called the d cell. At time 0 each processor containing a 1 sets d=0, while all other processors set d=∞. At time i, each processor where d is infinity which is adjacent to a processor where d equals i-1 now sets its d=i. This continues until time n, which is the time needed for a wave starting in one corner to reach a wave started in the opposite corner. Notice that no processor ever changes d once it is finite, and the value in d equals the minimum distance from that processor to a processor containing a 1.

A processor is at a collision point of two or more waves (four is the maximum possible) if an opposite pair of its neighbors have a d value less than or equal to its own. (See Figure 4.) If a processor at a collision point has d=i and two opposite neighbors have d=i-1, then it knows there are two 1's which are 2*i units apart, while otherwise it must be that one neighbor has d=i and the opposite one has d=i-1, in which case the two 1's are 2*i+1 units apart. Each processor at a collision point calculates the distance between the 1's and puts this value in its sort area, and all other processors put ∞ in their sort area. In O(n) time one can have the 2-MCC find the minimum of the values in the sort areas. This minimum is the answer.

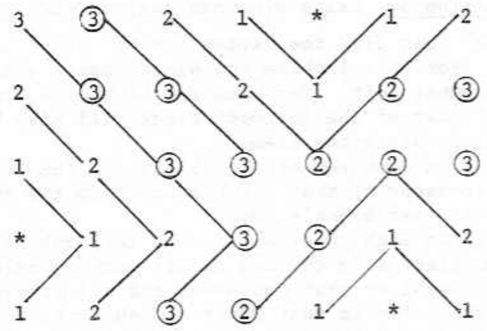

Figure 4. Spreading waves, emanating from the *'s. Circled numbers are collision points.

Now we show how to solve the minimum distance problem on a 2-MCC with broadcasting.

<u>Theorem 5</u>. Using broadcasting, a 2-MCC of size n^2 can solve the minimum distance problem in $O(n^{2/3})$ time.

<u>Proof</u>. Again the block leader will be in the upper left. Our initial local computing will go across block boundaries.

Step 1. Do the spreading waves for $n^{2/3}$ time units. This is the time needed for a spreading wave originating in one corner of a block to collide with a wave originating in the opposite corner of the block.

Step 2. Any processor at a collision point puts the distance between the 1's into its sort area, and all other processors put in ∞. (Notice that a processor may be at a collision point of two waves which originated in different blocks.) Within each block the minimum of the numbers in the sort areas is determined and put into the sort area of the block leader.

Step 3. The block leaders, in turn, broadcast the value in their sort area. If any finite number is broadcast then the answer is the minimum of all such numbers and the algorithm terminates.

Step 4. If the algorithm has not terminated then no block contains more than one 1. Each processor containing a 1 puts its coordinates into its sort area, while all others put in ∞, and the minimum of all values in sort areas within a block is put into the sort area of the block leader.

Step 5. The block leaders, in turn, broadcast the value in their sort area.

Step 6. During Step 5, each block leader having a finite position in its sort area determines the minimum distance between that position and all other broadcasted positions.

Step 7. The block leaders, in turn, broadcast either ∞, if their block contains no 1, or else the number determined during Step 6. The answer is the minimum of all numbers broadcasted.

Steps 1, 2, 3, 4, 5, and 7 take $O(n^{2/3})$ time each. Step 6 requires only a fixed number of calculations per broadcasted value, and hence cannot slow Step 5 by more than a constant multiple. Therefore the entire algorithm takes only $O(n^{2/3})$ time. □

Finally, we consider a distance problem requiring more global communication than the previous problem required. We call it the all points minimum distance problem, and it involves having each processor containing a 1 determine the minimum distance to any other processor containing a 1. In a 2-MCC without broadcasting one can also solve this by using spreading waves, where now a collision point sends a message back towards the 1's telling them of the distance. This message always goes from one processor to one of its neighbors with a lower d value, with each processor containing a 1 having as its answer the smallest message received. This takes at most $O(n)$ time.

As before, we modify the 2-MCC algorithm to utilize broadcasting whenever the answer is large.

Theorem 6. Using broadcasting, a 2-MCC of size n^2 can solve the all points minimum distance problem in $O(n^{2/3})$ time.

Sketch of Proof. Again the block leaders will be the upper left corner.

Step 1. Do the spreading wave technique, with collision points reporting back to the 1's, for $2*n^{2/3}$ steps. This is long enough to allow any two 1's in the same block to have their waves collide and report back. Any processor containing a 1 which receives a message back during this stage has as its answer the minimum of all such messages.

Step 2. Any processor containing a 1 which did not receive a message back during Step 1 places its coordinates into its sort area. This can happen only if this block contains no other 1's. A processor containing a 1 which did determine its answer in Step 1 puts 0 in its sort area, and all other processors put in ∞. The minimum of all values in sort areas is placed into the sort area of the block leader.

Step 3. The block leaders, in turn, broadcast the value in their sort area.

Step 4. As Step 3 proceeds, those block leaders which have a coordinate in their sort area calculate an upper bound on the minimum distance from that 1 to any other 1. They do this by calculating the minimum distance from the coordinates they have to any other broadcasted coordinates, and whenever a block broadcasts a 0 they calculate the furthest possible distance from their 1 to any processor in the block broadcasting the 0.

Step 5. If no coordinates were broadcast during Step 3 then the algorithm is finished. Otherwise, each block leader, in turn, either broadcasts a null message, if its sort area has anything other than coordinates, or else rebroadcasts the coordinates in its sort area, followed by the value calculated in Step 4.

Step 6. As Step 5 proceeds, each processor containing a 1 calculates the distance from it to each broadcasted coordinate, and if this is less than the broadcasted distance bound then the coordinates and distance are stored as a pair in its sort area. It can be shown ([14]) that at most 7 pairs are stored.

Step 7. Each processor treats its sort area as 7 separate areas, and fills any area not used in Step 6 with infinity. The sort areas are sorted into snake-like order, where the major key is the coordinates and the minor key is the distance, and where the first 7 pairs are stored in the block leader, the next 7 in the next processor, and so on.

Step 8. Notice that all distance information relating this block to a given broadcasted location is now grouped together, and the first pair in each group contains the minimum distance from a 1 in this block to that broadcasted 1. Each pair which is not the first in its group is replaced by infinity, and the sort areas are resorted. It can be shown that at most 7 pairs remain, so, assuming $n^{4/3} \geq 7$, now the sort only places one pair per processor.

Step 9. Each block leader, in turn, broadcasts the pair in its sort area. If this is not infinity then it passes a message to the next processor in the snake-like ordering, which then broadcasts the pair in its sort area. This continues until an infinity is broadcast, which signals the next block to start.

Step 10. While Step 9 proceeds, each processor containing a 1 which did not determine its answer during Step 1 keeps track of the minimum distance which is paired with its coordinates in a broadcast. The minimum such value is its answer, or else, if no such pair is broadcasted for it, then its answer is the distance its block leader calculated in Step 4.

Steps 1, 2, 3, 5, 7, 8, and 9 each take $O(n^{2/3})$ time. Steps 4, 6, and 10 can be performed on-line while the preceeding step is being performed without slowing it down more than a constant multiple, resulting in $O(n^{2/3})$ total time. □

Conclusions

We have shown that, for certain representative problems, a mesh-connected computer with broadcasting can perform significantly faster than one without it. Several problems which take $O(n)$ time on a k-MCC of size n^k can be done in $O(n^{k/(k+1)})$ time if broadcasting is used, an effect equivalent to using a (k+1)-MCC without broadcasting. Broadcasting is most useful when a short period of local computing followed by broadcasting significantly reduces the information remaining to be determined. As we have shown, several geometric problems have this property. Since these problems are among the ones to which mesh-connected computers are being applied, we feel that broadcasting may be of practical use. Broadcasting also raises a great many theoretical questions in trying to determine how to apply it to various problems.

We should mention that broadcasting does not cure all data movement ills. For example, in [13] it is shown that sorting algorithms cannot be improved. The difficulty here is that many values may have to travel far, and for broadcasting to assist in this the number of broadcasts becomes excessive. A related, open problem is the determine the time needed to find the median of n^2 numbers, stored one per processor in a 2-MCC of size n^2. It is easy to show that at least $O(n^{2/3})$ time is necessary, and by modifying a 1-MCC algorithm of [13] one can obtain an algorithm which uses $O((n^2*\log(n))^{1/3})$ time. It is an open question as to whether this time is optimal.

Acknowledgment

This research was partially supported by a grant from the S. U. N. Y. University Awards Committee.

References

1. H. Abelson, Lower bounds in information transfer in distributed computation, J. ACM 27 (1980), 384-392.

2. S. N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, IEEE Trans. Computers 18 (1969), 349-365.

3. M. J. B. Duff, CLIP4: A large scale integrated circuit array parallel processor, 3rd Int'l. Joint Conf. on Pattern Recognition, 1976, 728-732.

4. C. R. Dyer, A fast parallel algorithm for the closest pair problem, Info. Proc. Let. 11 (1980), 49-52.

5. C. R. Dyer and A. Rosenfeld, Parallel image processing by memory-augmented automata, IEEE Trans. Pat. Anal. and Mach. Intel. 3 (1981), 29-41.

6. W. M. Gentleman, Some complexity results for matrix computations on parallel processors, J. ACM 25 (1978), 112-115.

7. Gilmore et al, Massively parallel processor, Goodyear Aerospace Company, 1979.

8. V. C. Hamacher, Machine complexity versus interconnection complexity in iterative arrays, IEEE Trans. Computers 21 (1971), 321-323.

9. W. H. Kautz, K. N. Levitt, and A. Waksman, Cellular interconnection arrays, IEEE Trans. Computers 17 (1968), 443-451.

10. S. Levialdi, On shrinking binary picture patterns, Comm. ACM 15 (1972), 7-10.

11. R. Miller and Q. F. Stout, Mesh-connected computer algorithms for some topological problems, to appear.

12. A. P. Reeves, Asystematically designed binary array processor, IEEE Trans. Computers 29 (1980), 278-287.

13. Q. F. Stout, Mesh-connected computers with broadcasting, submitted.

14. Q. F. Stout, Geometric algorithms for a mesh-connected computer with broadcasting, to appear.

15. S. L. Tanimoto and A. Klinger (eds.), Structured Computer Vision: Machine Perception Through Hiearchical Computational Structures. Academic Press, New York, 1980.

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

17. L. Uhr, Layered recognition cone networks that preprocess, classify, and describe, IEEE Trans. Computers 21 (1972), 758-768.

18. J. von Neumann, The Theory of Automata: Construction, Reproduction, and Homogeneity, A. Banks, ed. Univ. Illinois Press, Urbana, 1966.