Mesh-Connected Computers with Broadcasting

QUENTIN F. STOUT, MEMBER, IEEE

Abstract—**We consider the effects of augmenting an arbitrary mesh-connected computer with a second communication system called broadcasting. In broadcasting, a processor sends a value to all the other processors simultaneously, taking unit time, with the restriction that only one broadcast occurs at any one time. We show that this significantly decreases the time to do sample problems such as semigroup calculations or finding the median, but it cannot significantly improve sorting. For example, in a one-dimensional mesh-connected computer without broadcasting, if there are n numbers, each stored separately in consecutive processors, then $\Theta(n)$ time is needed to find their minimum, find their median, or sort them, while with broadcasting, this can be done in $\Theta(n^{1/2})$, $\Theta((n \log n)^{1/2})$, and $\Theta(n)$ time, respectively.**

Index Terms—**Broadcasting, mesh-connected computer, parallel computing, selection, semigroup computations, sorting.**

I. Introduction

MESH-CONNECTED computers, or grid graphs, are an important class of physically realizable parallel computers. They can be represented as infinite, connected, point-symmetric undirected graphs whose vertices are computers and whose edges are bidirectional communication links through which information passes in unit time. (Point symmetry means that, given any two vertices, there is an isomorphism of the graph onto itself which maps the first vertex onto the second.) We require that each vertex have finite degree. A standard example has computers placed at the points (i, j) in the plane where i and j are integers, with the computer at (i, j) linked to the computers at $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$, and $(i, j - 1)$. There are more exotic possibilities, such as an infinite rootless tree where each vertex has degree three, or again having computers at the integer lattice points of the plane, but now the computer at (i, j) is connected to those at $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$, $(i, j - 1)$, $(i + 2, j + 2)$, $(i - 2, j - 2)$, $(i + 3, j - 3)$, and $(i - 3, j + 3)$. Specific mesh-connected computers include [7], [9]–[11], [15], [20], [21], [31], [32].

In mesh-connected computers, typical problems are as follows: there is a finite subset of n computers, each of which has a piece of information. One must either rearrange this information among the computers in a specified pattern, such as occurs when sorting, or have some designated computer evaluate a function whose parameters are these pieces of information, such as occurs when computing sums or minimums. Some problems, such as matrix multiplication, contain features of both types of problems. Examples of such problems occur in [1], [3], [5], [6], [8], [13], [14], [16], [17], [24], [27], [33].

We assume that each computer only has a finite number of registers to store information, prohibiting any one computer from being able to store all of the pieces of information. Technically, the size of each register must be at least lg (n) (lg means log base 2), and hence the size varies with n. Over a practical range of values, this variation can be ignored. Early studies of mesh-connected computers, e.g., [3], [9], [10], [13], [14], [18], [22], [23], [32], [33], assumed that the processors were copies of a fixed finite-state automaton, which is equivalent to our model with the restriction that the register size does not vary with n. While this is a natural model, it does not quite capture what is desired, and more recent studies, e.g., [19], [25], [30], use the model used here.

In many problems, the worst case solution time is constrained by the number of communication links along which a single item must pass. For example, if we have a computer at each integer i on a line, and if the computer at i is linked only to its neighbors at $i + 1$ and $i - 1$, then sorting n numbers into ascending order, one number per computer, must have a worst case time of $\Omega(n)$. (We use O for "order no greater," Ω for "order at least," and θ for "order exactly".) The worst case occurs when the numbers start in descending order, for then the smallest number must traverse at least $n - 1$ communication links. Further, if the numbers start in consecutive computers, then simple algorithms can be used to attain a worst case sorting time of $\theta(n)$. Abelson [1], Gentleman [8], and others have analyzed more complicated problems where again the worst case solution time is constrained by the data movement. We should mention that there are many other features to be considered, such as the relative merits of SIMD versus MIMD organization, but we will ignore these and concentrate on difficulties which arise solely due to properties of the connection graph.

Given that the solution times of many problems on a mesh-connected computer are constrained primarily by data movement considerations, a desirable goal is to design a computer which retains the natural mesh-connected configuration and supplements it with a faster mechanism for moving data long distances. Gentleman [8] was apparently the first to consider a supplemental mechanism called *broadcasting*. When a processor broadcasts a value, it is simultaneously received by all other processors. To avoid pandemonium, we allow only one broadcast at a time. We assume that there is a central clock, and that at the start of each time unit, a processor may send information along any or all of its links, as well as sending a broadcast. During the remainder of the time unit, each processor receives all values sent to it and can do a fixed amount of computation.

Gentleman's Theorem 3 seems to indicate that broadcasting is of little use, at least in O-notational analysis. In this paper, we show that, in a finite-dimensional setting, broadcasting is quite useful for problems which are sufficiently decomposable. Broadcasting is physically unrealistic in that it propagates information at infinite speed, but for practical situations this

Manuscript received March 2, 1982; revised August 13, 1982 and April 5, 1983.

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

0018-9340/83/0900-0826$01.00

may reasonably be ignored. This approach has been taken by Jordan and Sawyer [12] in their design of a mesh-connected computer with broadcasting. A similar combination of mixing realistic constraints, such as allowing only a fixed number of communication links per processor, with slightly unrealistic assumptions, such as allowing information to be transmitted arbitrarily far in a single time unit, is advocated for parallel computers such as the shuffle-connected computer [23], pyramid cellular array [29], etc.

In the next section, we show that semigroup operations can be computed more quickly using broadcasting, and that our results are the best possible. In the third section, we analyze sorting and show that broadcasting is of no significant use, but that it is useful for the related problem of selecting the median. Our final section has a few concluding remarks. Throughout, the term *k-dimensional lattice computer* means a parallel computer with individual processors at each point $(i_1, \cdots, i_k)$ where each i_l is an integer, with the processor at $(i_1, \cdots, i_k)$ connected to the one at $(j_1, \cdots, j_k)$ if and only if $\sum_{l=1}^{k} |i_l - j_l| = 1$. Whenever we do an analysis of an algorithm for a k-dimensional lattice computer, we assume that k is fixed and we only vary the size of the problem. Some papers analyze the effect of varying both k and the size of the problem, but this makes the analysis more complicated. Further, since a processor in a k-dimensional lattice computer is fundamentally different from one in a $(k+1)$-dimensional lattice computer, one must be extremely cautious when varying k.

II. Semigroup Computation

For a given mesh-connected computer, define the distance between two processors to be the minimum number of communication links needed to go from one to the other. Choose a processor and define $\sigma(t)$, t a nonnegative integer, to be the number of processors at distance t or less from the given one. Since the connection graph is point symmetric, σ is independent of the choice of processor. For the two-dimensional lattice computer, $\sigma(t) = 2t^2 + 2t + 1$, and the k-dimensional lattice computer has $\sigma(t) = 2^k\binom{t+k-1}{k} + 1$, which for any fixed k is $\theta(t^k)$. The infinite rootless tree where all vertices have degree k $(k > 2)$ has $\sigma(t) = 1 + k[(k-1)^t - 1]/(k-2)$, which for any fixed k is $\theta((k-1)^t)$. To be physically realistic, since each processor requires a certain minimum volume, the speed of light is finite, and we want the communication links to take unit time, one should have $\sigma(t) = 0(t^3)$. Currently, most real mesh-connected computers have $\sigma(t) = 0(t^2)$.

The σ function gives a lower bound to many computations, namely, if a processor is to compute a value which depends on the initial contents of n processors, then at least $\sigma^{-1}(n)$ time is needed if no broadcasting is used. When $\sigma(t) = \theta(t^k)$, $\sigma^{-1}(n) = \theta(n^{1/k})$. The equivalent lower bound for broadcasting-based computers is contained in the following theorem, which was simultaneously discovered by Bokhari [4].

Theorem 1: Let $*$ be a semigroup operation. Suppose that there are n distinct processors $P(1), \cdots, P(n)$ where $P(i)$ contains a number $s(i)$, and there is a processor P which is to compute $s(1) * s(2) * \cdots * s(n)$. If $\sigma(t) = \theta(t^k)$, then P has a worst case of time $\Omega(n^{1/k})$ using no broadcasting, but can finish in $\theta(n^{1/(k+1)})$ time if broadcasting is used and the processors are properly arranged. Further, this time is optimal.

Proof: The comments preceding this theorem show that $\Omega(n^{1/k})$ time is necessary when no broadcasting is used. To show that $\theta(n^{1/(k+1)})$ is possible using broadcasting, let $l = \lceil n^{k/(k+1)} \rceil$, $m = \lceil n/l \rceil$, and arrange the processors into m groups where i group, $1 \le i \le m$, contains processsor $P(l \cdot i - l + 1), \cdots, P(\min(l \cdot i, n))$. Let $Q(i)$ denote processor $P(l \cdot i - l + 1)$. Assume that group i is arranged around $Q(i)$ so that $Q(i)$ can compute $q(i) = s(l \cdot i - l + 1) * \cdots * s(\min(l \cdot i, n))$ in $\sigma^{-1}(l)$ time. Have all $Q(i)$ compute their values simultaneously, and at time $\sigma^{-1}(l) + i$, have $Q(i)$ broadcast $q(i)$. P computes $q(1) * \cdots * q(m) = s(1) * \cdots * s(n)$ in time $\sigma^{-1}(l) + m = \theta(n^{1/(k+1)})$.

To show that broadcasting requires $\Omega(n^{1/(k+1)})$ time, suppose that processor P is finished at time T. Through only standard communication links, P can use at most $\sigma(T)$ of the $s(i)$ in computing its value. If P receives a broadcast at time t, then the processor sending it can have used at most $\sigma(t-1)$ of the $s(i)$ which were not previously used to compute a broadcast message. Therefore, at most $\sigma(T) + \sum_{t=1}^{T}\sigma(t-1) = \theta(T^{k+1})$ of the $s(i)$ can have been used. If $*$ is nontrivial, then all n of the $s(i)$ must be used, so T must be $\Omega(n^{1/(k+1)})$. □

Notice that if σ grows so fast that $\sigma(T) = \Omega(\sum_{t=1}^{T-1}\sigma(t))$, then, in an O-notation analysis, broadcasting is useless. This is what Gentleman's theorem shows, but it is somewhat obscure. Further, this requires σ to grow exponentially, which is impossible for any finite-dimensional mesh-connected computer.

Theorem 1 shows that a k-dimensional lattice computer with broadcasting can do semigroup operations such as sums, products, minima, and maxima of n properly placed numbers in $\theta(n^{1/(k+1)})$ time. All that is required of the operation is associativity.

III. Sorting and Selecting

We will show that broadcasting does not significantly aid sorting. However, we must be somewhat careful about the values that can be broadcast. For example, if we are sorting a collection of natural numnbers, we want to require that only one element of our collection can be broadcast at a time. However, given two natural numbers i and j, we could broadcast the single number 2^i3^j and simultaneously convey both values. To prevent this, we assume that our items to be sorted come from some ordered set S, and the only operations available are to compare two of the given items or to transmit a single item.

In a one-dimensional lattice computer, if the processors at $1, 2, \cdots, n$ each start with an element of S, and if we are to sort these values so that the smallest value is in position 1 and the largest value is in position n, then, as was previously mentioned, without broadcasting the worst case time is $\Omega(n)$, and there are simple algorithms which need no more than $\theta(n)$ time. More generally, in a k-dimensional lattice computer, if the n

processors in locations $(i_1, \cdots, i_k)$, $1 \le i_j \le n^{1/k}$ each start with an element of S, and if these are to be sorted into some specified order (row-major, snake-like, etc.), then without broadcasting at least $\Omega(n^{1/k})$ time is needed, and indeed sorting can be accomplished in $\theta(n^{1/k})$ time. (See Thompson and Kung [30].) Theorem 2 shows that broadcasting is of little help for such problems.

Theorem 2: Suppose that $\sigma(t) = \theta(t^k)$, suppose that there are n different processors $P(1), \cdots, P(n)$, each of which contains a different element of S, and suppose that there are n different processors $Q(1), \cdots, Q(n)$, where perhaps some processors occur both in the P list and Q list. Then with or without broadcasting, the worst case time to sort the items in the P's so that the smallest item is in $Q(1)$, the second smallest is in $Q(2)$, etc., is $\Omega(n^{1/k})$.

Proof: Without loss of generality, we assume that $S = \{1, \cdots, n\}$. We construct a permutation π on $\{1, \cdots, n\}$ by induction. Since the Q's are distinct, there is an i such that $d(P(1), Q(i)) \ge \sigma^{-1}(n)$. Let $\pi(1)$ be defined to be i. Assuming that $\pi(1), \cdots, \pi(l)$ have been defined for $1 \le l < n$, we define $\pi(l+1)$ as follows: since the set $X = \{1, \cdots, n\} \backslash \{\pi(1), \cdots, \pi(l)\}$ has $n - l$ elements, there is an i in X such that $d(P(l+1), Q(i)) \ge \sigma^{-1}(n-l)$. Define $\pi(l+1)$ to be i.

Suppose that processor $P(i)$ initially contains $\pi(i)$ for $1 \le i \le n$. Notice that the item in $P(i)$ must be moved to $Q(\pi(i))$, and hence must travel at least $d(P(i), Q(\pi(i)))$ units. If no broadcasting is used, then sorting must take at least $\sigma^{-1}(n) = \Omega(n^{1/k})$ time. If broadcasting is used, assume that I broadcasts occur. If $I \ge n/2$, then the broadcasts themselves take $\Omega(n)$ time. Otherwise, there is an $i \le n/2$ such that the value $\pi(i)$ is never broadcast, which implies that the sorting takes at least $\sigma^{-1}(n-i) \ge \sigma^{-1}(n/2) = \Omega(n^{1/k})$ time. □

The next theorem shows that a closely related problem can be improved with broadcasting, but not as much as were the semigroup computations.

Theorem 3: If n consecutive processors of a one-dimensional lattice computer each contain an element of S, then any designated processor can determine the median of these values in $\theta((n \log n)^{1/2})$ time if broadcasting is used, while at least $\Omega(n)$ time is necessary if no broadcasting occurs.

Proof: For ease of exposition, we only treat the case where all values are distinct. Think of the n processors as being in m groups of n/m consecutive processors where $m = n/l_g(n)^{1/2}$. Sort all groups in parallel, taking $\theta(n/m)$ time. Throughout the following algorithm, we assume that each processor knows which group it is in, its relative position in the group, and the values of the items in its adjacent neighbors. Further, each processor keeps track of the proceedings, which requires keeping a fixed number of counters. This allows instructions such as "broadcast the median of the lth group" to be obeyed in unit time where the processor containing the value is the one which broadcasts it.

Through the algorithm, each processor starts in an active state, and eventually may become passive. A passive processor is one known to contain a value which is not the median. Within each group, the active processors are always contiguous, and each processor in the group can keep track of this interval of active processors. We proceed in a series of passes. In each pass, the groups, in order, transmit the number of active processors in the group and the median of the values in the active processors. (Notice that while all the processors in a group know which processor in their group contains the median of the active processor, most of them cannot know what that median value is since they cannot remember all the values in their group.) If $w(i)$ is the number of active processors in group i, and if $r(i)$ is the median of their values, then there is a unique I such that

$$\sum\{w(i): r(i) \le r(I)\} \ge \left(\sum_{i=1}^{m} w(i)\right)/2$$

and

$$\sum\{w(i): r(i) < r(I)\} < \left(\sum_{i=1}^{m} w(i)\right)/2.$$

We call $r(I)$ the *weighted median* of the broadcasted numbers. In each group j, the processor containing $r(j)$ is computing $\Sigma\{w(i): r(i) < r(j)\}$ and $\Sigma w(i)$ as the $w(i)$, $r(i)$ pairs are being broadcast. When this broadcasting is done, the processor containing the weighted median rebroadcasts it. Now the ith group, in turn, broadcasts the number $b(i)$ of items in the group that are less than or equal to $r(I)$. (If $r(I)$ is less than all items in the group, then the first processor broadcasts 0. Otherwise, there is a unique processor which contains a value no greater than $r(I)$, and which knows that the next processor is in a different group or contains a value greater than $r(I)$. This processor broadcasts its position within the group.) If $\Sigma_{i=1}^{m} b(i) = \lceil n/2 \rceil$, then the algorithm is finished. Otherwise, suppose that $\Sigma_{i=1}^{m} b(i) < \lceil n/2 \rceil$, the $>$ case being similar. Then $r(I)$ is smaller than the true median, and hence the first $b(i)$ processors in group i are now passive. Each processor in group i remembers $b(i)$ when it is broadcast, and all processors compute $\Sigma_{k=1}^{m} b(i)$, and so at the end of this pass, each processor knows the indexes of the active processor in its group.

Passes are repeated until the median is found. Each pass takes no more than Cm time for some constant C. To determine the number of passes, suppose that in one pass $r(I)$ was too small. Since $r(I)$ is a weighted median, at least half of all active processors are in a group where the median of the active processors is no greater than $r(I)$. In each such group, at least half of its active processors become passive at the end of the pass. Therefore, each pass reduces the number of active processors by at least one fourth; so the number of passes is no more than $\log_{4/3}(n)$. The time for all passes is at most $0(m \log(n)) = 0((n \log(n))^{1/2})$. Finally, it is easy to see that $\Omega(n)$ time may be required if no broadcasting occurs. □

The author does not know if Theorem 3 gives the best possible result, although there is not much room for improvement. There is also the question of designing an algorithm to find the median in optimal expected time.

We should mention that Theorem 3 was stated only for one-dimensional lattice computers because they provided a natural ordering to the processors within each group and to the groups themselves, and on such a computer, a contiguous group of G processors can easily sort their contents in $\theta(\sigma^{-1}(G))$ time. There are similar orderings and sort algorithms available for k-dimensional lattice computers (see Thompson and Kung [30]), and these can be used to find the median in $\theta((n \log(n)^k)^{1/(k+1)})$ time. Extending Theorem 3 to other mesh-connected computers first requires extending the necessary sorting results.

IV. Further Remarks

We have analyzed some of the improvements that one could attain by adding broadcasting to a mesh-connected computer. The improvement for a specific problem depends both on the σ-function of the mesh-connected computer, and on the ability of the problem to be sufficiently decomposed. For example, on a one-dimensional lattice computer for which $\sigma(t) = 2t + 1$, given n numbers, each initially in a different processor, with broadcasting, one can find the minimum, their median, or sort them in $\theta(n^{1/2})$, $\theta((n \log n)^{1/2})$, and $\theta(n)$ time, respectively, while all three problems require $\theta(n)$ time if no broadcasting is employed. For a standard single processor, these problems take $\theta(n)$, $\theta(n)$, and $\theta(n \log n)$ time, respectively. (See, e.g., [2].)

Apparently the first mesh-connected computer with broadcasting is the one outlined in Jordan and Sawyer [12]. They had analyzed the computational requirements of doing finite element analysis on a mesh-connected computer and they concluded that broadcasting provided a significant improvement. Further, finite element analysis is sufficiently important to deserve a special machine. Perhaps a similar machine will be built for image processing since this area is of considerable importance, and [25] and [26] show that there are several image processing problems for which broadcasting is useful. The decision to build machines with broadcasting depends on analyzing a range of algorithms to determine which speed-ups, if any, are possible.

Broadcasting seems to be of least use when the answer itself is composed of several pieces, each of which is to be in a separate processor. Examples of such problems are sorting, merging, and matrix multiplication. Even though broadcasting may speed up the computation of any single subproblem, its one-at-a-time nature prohibits it from significantly improving the simultaneous computation of all of them. On the other hand, if the subproblems are sufficiently related, then perhaps broadcasting is of use. Examples of this appear in [25] and [26]. This raises the following theoretical question, motivated by Gentleman's paper [8]: how long does it take a two-dimensional lattice computer with broadcasting to multiply two $N \times N$ real matrices? We assume a unit cost criterion for operations on real numbers, and we assume that each entry of each matrix is initially in a separate processor. We conjecture that the answer is $\theta(N)$, which is the same as the time without broadcasting. This problem is harder to analyze than was sorting because we are allowing values to be broadcast which are combinations (sums and products) of the original entries. For example, it was not until Strassen's work [28] that people realized that on a single processor, matrices could be multiplied faster by computing somewhat unusual combinations of the entries.

The form of broadcasting considered here is the most restrictive possible, and is analogous to a complete prohibition of write conflicts for parallel random access machines (PRAM's). Just as some authors permit write conflicts on a PRAM as long as all the processors writing to a given memory cell are writing the same value, so could one consider the effect of allowing simultaneous broadcasts as long as the values being broadcast are the same. Suppose that a one-dimensional lattice computer has this feature, and that processors 1–$n-1$ contain a 0 or a 1. To find the processor of lowest index which contains a 1 takes $\theta(n)$ time without broadcasting, $\theta(n^{1/2})$ time with standard broadcasting, but only $\theta(\log(n))$ time with this more liberal broadcasting. To accomplish this, at time 1, have any processor numbered 1–$n/2$ broadcast 1 if it contains a 1. At time 2, if a 1 was broadcast at time 1, then any processor numbered 1–$n/4$ should broadcast 1 if it contains a 1, while if no 1 were broadcast at time 1, then any processor numbered $n/2 + 1$–$3n/4$ should broadcast 1 if it contains a 1. Continuing this binary search gives a $\theta(\log(n))$ algorithm. While this more liberal broadcasting helps a few problems, it seems unable to improve upon the semigroup, sorting, or selection algorithms considered earlier. Even if we allow multiple values to be simultaneously broadcast, with the stipulation that the minimum value broadcast is the one which all processors receive, there are still very few situations where this form of broadcasting is superior to the original.

Finally, while we have assumed that a broadcast reaches all processors in unit time, in practice broadcasting may be implemented by a mechanism which takes $\theta(\log(n))$ time. (Physically, of course, broadcasting must take $\Omega(n^{1/3})$ time, but the logarithmic delay is probably a better model over a realistic range of n.) If each broadcast must wait $\theta(\log(n))$ time units after the previous one, then the algorithms will be somewhat slower. For example, on a one-dimensional lattice computer, by using groups of size $(n \log(n))^{1/2}$, one can do semigroup calculations in $\theta((n \log(n))^{1/2})$ time, and with groups of size $n^{1/2} \log(n)$, one can find the median in $\theta(n^{1/2} \log(n))$ time. However, if the broadcasts can be pipelined, with each starting one time unit after the previous one, then the semigroup and median computations can be completed as quickly (in θ notation) as before.

References

[1] H. Abelson, "Lower bounds in information transfer in distributed computation," *J. Ass. Comput. Mach.*, vol. 27, pp. 384–392, 1980.

[2] A. V. Aho, J. C. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[3] W. T. Beyer, "Recognition of topological invariants by iterative arrays," Ph.D. dissertation, M.I.T., Cambridge, 1969.

[4] S. H. Bokhari, "MAX: An algorithm for finding maximum in an array processor with a global bus," in *Proc. 1981 Int. Conf. Parallel Processing*, pp. 302–303.

[5] S. N. Cole, "Real-time computation by n-dimensional iterative arrays of finite-state machines," *IEEE Trans. Comput.*, vol. C-18, pp. 349–365, 1969.

[6] L. P. Cordella, M. J. B. Duff, and S. Levialdi, "An analysis of computational cost in image processing: A case study," *IEEE Trans. Comput.*, vol. C-27, pp. 904–910, 1978.

[7] M. J. B. Duff, "CLIP4: A large scale integrated circuit array parallel processor," in *Proc. 3rd Int. Joint Conf. Pattern Recognition*, 1976, pp. 728–832.

[8] W. M. Gentleman, "Some complexity results for matrix computations on parallel processors," *J. Ass. Comput. Mach.*, vol. 25, pp. 112–115, 1978.

[9] M. J. E. Golay, "Hexagonal parallel pattern transforms," *IEEE Trans. Comput.*, vol. C-19, pp. 733–740, 1969.

[10] S. B. Gray, "Local properties of binary images in two dimensions," *IEEE Trans. Comput.*, vol. C-21, pp. 551–569, 1971.

[11] J. Gregory and R. McReynolds, "The SOLOMON computer," *IEEE Trans. Comput.*, vol. C-13, pp. 774–781, 1963.

[12] H. F. Jordan and P. L. Sawyer, "A multimicroprocessor system for finite element structural analysis," *Comput. Struc.*, vol. 10, pp. 21–29, 1979.

[13] W. H. Kautz, K. N. Levitt, and A. Waksman, "Cellular interconnection arrays," *IEEE Trans. Comput.*, vol. C-17, pp. 443–451, 1968.

[14] S. R. Kosaraju, "Fast parallel processing array algorithms for some

graph problems," (Preliminary version), in *Proc. 11th ACM Symp. Theory Comput.*, 1979, pp. 231–236.
[15] B. Kruse, "A parallel processing machine," *IEEE Trans. Comput.*, vol. C-23, pp. 1057–1087, 1973.
[16] D. J. Kuck and A. H. Sameh, "Parallel computations of eigenvalues of real matrices," in *Information Processing 71*. Amsterdam: North-Holland, 1972, pp. 1266–1272.
[17] S. Levialdi, "On shrinking binary picture patterns," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 7–10, 1972.
[18] K. N. Levitt and W. H. Kautz, "Cellular arrays for the solution of graph problems," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 789–801, 1972.
[19] 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.
[20] K. Preston, "Feature extraction by Golay hexagonal transforms," *IEEE Trans. Comput.*, vol. C-21, pp. 551–569, 1971.
[21] A. P. Reeves, "A systematically designed binary array processor," *IEEE Trans. Comput.*, vol. C-29, pp. 278–287, 1980.
[22] A. Rosenfeld, "A characterization of parallel thinning algorithms," *Inform. Contr.*, vol. 29, pp. 286–291, 1975.
[23] J. T. Schwartz, "Ultracomputers," *ACM Trans. Programming Lang. Syst.*, vol. 2, pp. 484–521, 1980.
[24] H. S. Stone, "Parallel tridiagonal equatiion solvers," *ACM Trans. Math. Software*, vol. 1, pp. 289–307, 1975.
[25] Q. F. Stout, "Broadcasting in mesh-connected computers," in *Proc. 1982 Conf. Inform. Sci. Syst.*, Princeton Univ., 1982, pp. 85–90.
[26] ——, "Geometric algorithms for a mesh-connected computer with broadcasting," to be published.
[27] ——, "The use of clerks in parallel processing," in *Proc. 23 IEEE Symp. Foundations Comput. Sci.*, 1982, pp. 272–279.
[28] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, pp. 354–356, 1969.
[29] S. L. Tanimoto and A. Klinger, Eds., *Structured Computer Vision: Machine Perception Through Hierarchical Computational Structures*. New York: Academic, 1980.
[30] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 263–271, 1977.
[31] S. H. Unger, "A computer oriented toward spatial problems," *Proc. IRE*, pp. 1744–1750, 1958.
[32] ——, "Pattern detection and recognition," *Proc. IRE*, pp. 1737–1752, 1959.
[33] F. L. VanScoy, "The parallel recognition of classes of graphs," *IEEE Trans. Comput.*, vol. C-29, pp. 563–570, 1980.

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 in mathematics from Indiana University, Bloomington, in 1977.

Since 1976 he has been an Assistant Professor in the Department of Mathematical Sciences, State University of New York, Binghamton. His teaching and research interests include algorithms and data structures, particularly algorithms for parallel computers.