SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

SOLVING TRIDIAGONAL SYSTEMS ON ENSEMBLE ARCHITECTURES*

S. LENNART JOHNSSON†

Abstract. The concurrent solution of tridiagonal systems on linear and 2-dimensional arrays, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes by elimination methods are devised and analyzed. The methods can be obtained by symmetric permutations of some rows and columns, and amounts to cyclic reduction or a combination of Gaussian elimination and cyclic reduction (GECR). The ensembles have only local storage and no global control. Synchronization is accomplished via message passing to neighboring processors.

The parallel arithmetic complexity of GECR for N equations on a K processor ensemble is $O(N/K + \log_2 K)$, and the communication complexity is $O(K)$ for the linear array, $O(\sqrt{K})$ for the 2-dimensional mesh, and $O(\log_2 K)$ for the networks of diameter $O(\log_2 K)$. The maximum speed-up for the linear array is attained at $K \approx (N/\alpha)^{1/2}$ and for the 2-d mesh at $K \approx (N/2\alpha)^{2/3}$, where $\alpha =$ (the time to communicate one floating-point number)/(the time for a floating-point arithmetic operation). For the binary tree the maximum speed-up is attained at $K = N$, and for the perfect shuffle and boolean k-cube networks, $K = N/(1+\alpha)$ yields the maximum speed-up. The minimum time complexity is of order $O(N^{1/2})$ for the linear array, of order $O(N^{1/3})$ for the mesh, and of order $O(\log_2 N)$ for the binary tree, the shuffle-exchange, the perfect shuffle and the boolean k-cube.

The relative decrease in computational complexity due to a truncation of the reduction process in a highly concurrent system is much greater than on a uniprocessor. The reduction in the arithmetic complexity is proportional to the number of steps avoided, if the number of processing elements equals the number of equations. So also is the reduction in the communication complexity for ensembles configured as binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes.

Partitioning the ensemble into subsets of processors is shown to be more efficient for the solution of multiple independent problems than pipelining the solutions over the entire ensemble. A *balanced* cyclic reduction algorithm is presented for the case where each system is spread uniformly over the processing elements, and its complexity is compared with Gaussian elimination.

Key words. odd-even cyclic reduction, Gaussian elimination, tridiagonal systems, parallel computers, distributed computing, ensemble architectures, MIMD

AMS(MOS) subject classifications. 15A06, 65F05, 65N05, 65Q25, 68Q35

1. Introduction.

The rapidly developing integrated circuit technology already allows hundreds of thousands of devices on a single chip or, equivalently, a few microprocessors with local storage. With a small amount of storage per processor, on the order of 10 16-bit processors can fit on a single chip in state-of-the-art technology [51]. It is expected that this number will increase by one to two orders of magnitude within a decade. The low cost of reproduction of an integrated circuit makes architectures consisting of a large number of identical processing elements sparsely and regularly interconnected a viable alternative to large mainframe computers. We refer to such architectures as *ensemble architectures.* A high nominal performance is attained by using a large number of processing elements built in standard technology. An excellent overview of the principles guiding VLSI architectures can be found in [52]. An assessment of the impact of VLSI on computer architecture is also given in [17].

We assume for convenience that the architectures are of the MIMD type (Multiple Instruction streams Multiple Data streams) [7], but a SIMD (Single Instruction Multiple Data streams) architecture is equally appropriate in most cases. Synchronization is obtained via message passing. There is a high degree of uniformity in the algorithms described in this paper. There are only a few different types of codes. The

* Received by the editors December 12, 1984; accepted for publication (in revised form) February 11, 1986. This work was supported by the Office of Naval Research under contract N00014-84-K-0043.

† Departments of Computer Science and Electrical Engineering, Yale University, New Haven, Connecticut 06520.

uniformity conceptually simplifies concurrent algorithms, enhances program clarity, simplifies verification of correctness, and makes program loading more efficient. Identical codes for different processors can be reproduced within the ensemble. If the encoding of what processors should receive which code can be made compactly, then the program loading may be particularly efficient [38]. With a unique program for each processor the loading time might be of a complexity that is of the same order as the arithmetic complexity of sequential algorithms. The time for program loading may be a significant fraction of the total time, if the ensemble serves as an attached processor, and only a few problems are solved for each loading of the program.

In the ensemble architectures we consider the processing elements with their storage are interconnected as linear and 2-dimensional arrays, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes. These interconnection schemes have different characteristics with respect to wiring complexity, extensibility, and communication capabilities between arbitrary pairs of nodes. For a feasibility evaluation of these interconnection schemes, it is necessary to investigate the mapping of typical computations on to the ensembles. Several efficient algorithms for matrix multiplication, FFT, sorting, and many other problems are known. In this report we develop efficient distributed algorithms for the solution of tridiagonal systems.

The solution of tridiagonal systems of equations is part of several methods for the solution of partial differential equations. For instance, fast Poisson solvers employing the method of Fourier Analysis-Cyclic Reduction (FACR) [19], [20] and the Alternate Direction Implicit (ADI) method both include the solution of tridiagonal systems. If the number of processing elements is large enough to allow for one grid point per processing element, then a mapping may be made such that a processing element stores only one equation and a tridiagonal system is solved in a distinct subensemble. However, in general, multiple systems have to be solved in a set of processing elements. In some instances it may be possible to allocate the systems freely, but in other cases each system may be distributed evenly over a subensemble [24].

We first give a brief characterization of a few model ensemble architectures, then review Gaussian elimination and odd-even cyclic reduction [5] in the context of concurrent computation, and define the optimization of communication in an ensemble architecture as a graph embedding problem. In § 4 we investigate the arithmetic and communication complexity of Gaussian elimination and odd-even cyclic reduction on a parallel computer with ideal communication capabilities, but finite computational resources. We also investigate the concurrent solution of a tridiagonal system with multiple right-hand sides, and the solution of multiple independent systems. In § 5 we present algorithms for the mapping of odd-even cyclic reduction onto linear and 2-dimensional arrays, followed by a complete binary tree algorithm of arithmetic and communication complexity $O(\log_2 N)$ for a system of N irreducible equations and N processing elements. We use this tree algorithm to obtain an algorithm of comparable complexity for the shuffle-exchange network. Odd-even cyclic reduction on a perfect shuffle is discussed next. The boolean cube algorithms we present have a complexity that is of minimum order, and use properties of Gray codes in the local control of communication operations. All algorithms have entirely distributed control. Section 6 contains a summary and conclusions.

Throughout the analysis we consider the performance gain possible through truncated cyclic reduction, and the solution of multiple tridiagonal systems as well as the solution of a tridiagonal system with multiple right-hand sides.

Independently and concurrently, Gannon and Van Rosendale [9] have undertaken a similar study. In the case of finite resources, i.e., multiple equations per processor,

they perform local eliminations according to an algorithm by Sameh et al. [49], [35], which yields a reduced pentadiagonal system with two equations per processor. In the case of tridiagonal systems this difference is insignificant with respect to computational complexity if $N \gg K$. However, in the generalization of the algorithms described here to arbitrary banded systems the difference becomes significant. With the partial elimination order determined by a symmetric permutation of rows and columns, as in nested dissection and the partitioning method of Wang, not only is the arithmetic complexity lower, but the scheme also preserves symmetry and positive definiteness, should the original matrix have these properties [29]. We also treat the case where multiple independent tridiagonal systems shall be solved and describe a *balanced* cyclic reduction algorithm. We also compare the complexity of this algorithm with Gaussian elimination and derive the conditions under which one or the other is of minimum complexity.

In the following, $N = 2^n - 1$ denotes the number of equations in the tridiagonal system, K the number of processors in the ensemble, P the number of tridiagonal systems to be solved, NR the number of right-hand sides, t_a the time for local data fetching and arithmetic, t_c the time for interprocessor communication, $\alpha = t_c/t_a$, and τ the time for a communication start-up. For our complexity estimates we assume that a processor can perform one communication on two of its ports concurrently with arithmetic operations.

A few sample programs in pseudo code are contained in the Appendix. The programs serve to illustrate a programming style for multiprocessor systems in which synchronization is obtained via message passing. All sample programs are written for the case $N = K$ and $NR = 1$. The programs also illustrate the degree of program uniformity for cyclic reduction on the various ensembles.

2. Ensemble architectures. One important property of an ensemble architecture with respect to algorithm performance is the diameter of its corresponding graph. The diameter gives a lower bound for the time to perform global communication [10]. Global communication is required for the solution of irreducible systems of equations, since any component of the solution vector depends on all elements of the right-hand side, and all matrix elements. However, note that the global dependence may be sufficiently weak to allow for good local approximations, and thereby eliminate the requirement for global communication. In the case of cyclic reduction the reduction process may be truncated after a few steps if the system is sufficiently diagonally dominant. Communication is everywhere local.

The diameter of a K node linear array is $K/2$ and that of a 2-dimensional mesh with end-around connections $\sqrt{K}$. The lower bound of the communication complexity of those two ensemble configurations is of a higher order than the arithmetic complexity for the solution of tridiagonal systems on a PRAM [8] model of computation.

The diameter of a complete binary tree of $2^k - 1$ nodes is $2(k-1)$, the diameter of a 2^k node shuffle-exchange graph is $2k-1$ and that of a boolean cube of 2^k nodes is k. The lower bound for the communication complexity for these ensembles is the same as the order of the arithmetic complexity for the solution of tridiagonal systems on a PRAM model. The ensembles with a diameter of order $O(k)$ are potentially capable of solving tridiagonal systems in a time proportional to the lower bound for limited *fan-in* circuitry. In this paper we devise mappings of cyclic reduction onto these ensembles such that the system is solved in a time proportional to the lower bound.

In light of the evolving VLSI technology it is conceivable that dedicated circuitry can be used for tridiagonal solvers. For the design of hardware for tridiagonal solvers

in VLSI the area requirement of the various ensemble configurations is also important. Another important characteristic is the number of communication channels needed between chips when the ensemble is implemented on several chips, which is the case in today's technology. A complete binary tree of K nodes requires an area of $O(K)$ based on the Thompson grid model [56], [57], [4], if there is no restriction on the placement of nodes. The area is increased to $O(K \log_2 K)$ if all leaf nodes are placed on the boundary [3]. A shuffle-exchange network requires a layout area of order $O(K^2/\log_2^2 K)$ and a boolean cube an area of order $O(K^2)$ [36]. The complete binary tree not only requires the smallest area, but also allows for a partitioning such that only 4 off-chip channels are required independent of the size of the subtree that fits on a single chip and the size of the tree being assembled [37], [2]. It also has the smallest lower bound, $O(\sqrt{K}/\log_2 K)$ for the maximum length of any wire interconnecting nodes [41], [48]. For a boolean cube the maximum wire length is of order $O(K)$ [36] and the number of channels required for interchip connections is equal to the number of nodes implemented on-chip $\times$ the number of off-chip dimensions.

The binary tree has definite advantages over the shuffle-exchange and boolean cube configurations from a designer's point of view. Moreover, the shorter wires may allow the clock rate for the binary tree to be higher than for the other two configurations, giving it a possible edge also with respect to running time.

3. Elimination methods on parallel architectures.

3.1. Sparse elimination. In this paper we will make use of Gaussian elimination as well as odd-even cyclic reduction. Odd-even cyclic reduction is equivalent to Gaussian elimination performed on a permuted system of equations. The relationship between the two methods is easily seen from the graph representation of elimination [40]. In this representation a nonzero matrix element is represented by a directed edge between two vertices labeled with the indices of the matrix elements. For a symmetric matrix the edges may be considered as undirected. In the graph model the elimination of variable i corresponds to the removal of edges incident on vertex i. New edges are inserted, or the values associated with existing edges are changed, so that after the elimination of variable i there exist, for each edge ji incident upon i, edges from node j to all nodes upon which edges emanating from i terminate.

The graph corresponding to a symmetric tridiagonal matrix of order N is a path of N nodes. The graph corresponding to a tridiagonal matrix is a *perfect elimination graph* [46], and the elimination order defined by sequentially eliminating the variables from one end of the path to the other (or from both ends towards the middle) is a *perfect elimination order*, i.e., no *fill-in* is generated.

The cyclic reduction algorithm defines a partial order of elimination. This partial order is the same as the partial order produced by nested dissection [13], [12], which minimizes the order of the solution time on a PRAM model of computation [25]. Even though nested dissection is asymptotically optimal with respect to the order of the arithmetic operations [39], its arithmetic complexity is approximately twice that of Gaussian elimination for tridiagonal systems ($17N - 18\log_2 N + 2$ compared to $8N - 7$), because the partial elimination order defined by cyclic reduction (nested dissection) yields *fill-in*. The partial order of elimination is often represented as *elimination trees* [6], or *quotient trees* [11]. The elimination trees for Gaussian elimination, 2-way Gaussian elimination, and cyclic reduction are shown in Fig. 1.

In cyclic reduction 2^{n-j} variables are eliminated in step $j = \{1, 2, \cdots, n-1\}$. On a uniprocessor the elimination in each step is performed sequentially. But, in spite of this fact and the higher arithmetic complexity of odd-even cyclic reduction, it is

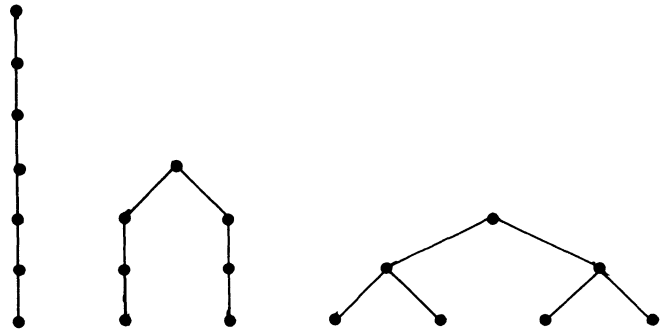

FIG 1. *Elimination trees for Gaussian elimination, 2-way Gaussian elimination, and cyclic reduction.*

preferable on most architectures with pipelined arithmetic units, such as vector architectures. The reason is that in Gaussian elimination only one (or two) variable(s) can be eliminated concurrently, and the number of operations required for each such elimination is very limited in a tridiagonal system (5 in the forward phase, 3 in the backsubstitution). Note however that for a banded system of half bandwidth $\gg 1$ the concurrency in the elimination of a single variable increases, and the number of variables that can be eliminated concurrently decreases [31].

To derive estimates of the solution time on an ensemble architecture it is necessary to incorporate the communication required for the elimination operations. We consider several cases:

- The ensemble is sufficiently large to hold one equation per processing element;
- Multiple equations per processing element;
- Multiple right-hand sides;
- Multiple independent tridiagonal systems.

3.2. Odd–even cyclic reduction. A tridiagonal system of irreducible linear equations $Ax = y$, where A is of dimension $N = 2^n - 1$, can be presented in matrix vector form as:

$$\begin{pmatrix} b_1 & c_1 & & & & x_1 & y_1 \\ a_2 & b_2 & c_2 & & & x_2 & y_2 \\ & a_3 & b_3 & c_3 & & x_3 & y_3 \\ & & \ddots & \ddots & \ddots & = & \\ & & & a_N & b_N & x_N & y_N \end{pmatrix}.$$

Odd-even cyclic reduction proceeds in a reduction phase succeeded by a backsubstitution phase. Using subscripts for equation numbers and superscripts to denote reduction and backsubstitution steps, cyclic reduction is defined by the following set of equations:

Reduction:

$$\begin{aligned} a_i^j &= e_i a_{i-2^{j-1}}^{j-1}, \\ c_i^j &= f_i c_{i+2^{j-2}}^{j-1}, \\ b_i^j &= b_i^{j-1} + e_i c_{i-2^{j-1}}^{j-1} + f_i a_{i+2^{j-1}}^{j-1}, \\ y_i^j &= y_i^{j-1} + e_i y_{i-2^{j-1}}^{j-1} + f_i y_{i+2^{j-1}}^{j-1}, \\ e_i &= -a_i^{j-1} / b_{i-2^{j-1}}^{j-1}, \\ f_i &= -c_i^{j-1} / b_{i+2^{j-1}}^{j-1} \end{aligned}$$

where $i = 2^j, 2\times 2^j, 3\times 2^j, \cdots, 2^n - 2^j$, for reduction steps $j = 1, 2, \cdots, n-1$.

The initial conditions are $a_i^0 = a_i$, $b_i^0 = b_i$, $c_i^0 = c_i$, and $y_i^0 = y_i$.

After $n-1$ reduction steps only one equation of the following form remains:

$$a_{2^{n-1}}^{n-1} x_0 + b_{2^{n-1}}^{n-1} x_{2^{n-1}} + c_{2^{n-1}}^{n-1} x_{2^n} = y_{2^{n-1}}^{n-1}.$$

A correct solution for $x_{2^{n-1}}$ is obtained with $x_0 = x_{N+1} = 0$. Remaining variables are obtained through backsubstitution.

Backsubstitution:

$$x_{2^{n-1}} = y_{2^{n-1}}^{n-1} / b_{2^{n-1}}^{n-1},$$

$$x_i = (y_i^{j-1} - a_i^{j-1} x_{i-2^{j-1}} - b_i^{j-1} x_{i+2^{j-1}}) / b_i^{j-1}$$

where $i = \{2j-1, 3\times 2^{j-1}, 5\times 2^{j-1}, \cdots, 2^n - 2^{j-1}\}$, and $j = \{n-1, n-2, \cdots, 1\}$.

In the above algorithm, 12 arithmetic operations are needed per equation in the reduction computation, and 5 per unknown in the backsubstitution. A careful count gives a total of $17N - 18n + 2$ arithmetic operations, disregarding index computations.

If the matrix A is strongly diagonally dominant, then a_i^j and c_i^j tend to zero with j. The stronger the diagonal dominance the more rapid is the convergence. If for $j = m$, $|a_i^j| \ll \varepsilon$ and $|c_i^j| \ll \varepsilon$, where ε is an acceptable error bound, then the reduced tridiagonal system at step m can be treated numerically as a diagonal system. No further reduction computations are necessary. Instead, a diagonal system of $2^{n-m} - 1$ equations is solved, and the backsubstitution process started. This truncated cyclic reduction method requires $2m$ steps. The reduction in the total number of operations is $16(2^{n-m} - 1) - 18(n-m) + 2$. Solving partial differential equations by difference approximation yields matrices for which acceptable precision renders values of m in the order of $10-20$, independent of the size of the matrix [18].

The reduction phase terminates when one equation with one unknown remains. If several reduction steps terminating in different equations are carried out concurrently, then the backsubstitution phase becomes unnecessary. Hockney describes such a cyclic reduction method and refers to it as Parallel Cyclic Reduction [18]. The tridiagonal system is extended with the equations for $i < 1$ and $i > N$ such that $a_i = c_i = 0$ and $b_i = 1$ for $i > 1$ and $i > N$. The parallel cyclic reduction algorithm is of arithmetic complexity $O(N \log_2 N)$, but needs only half the number of sequential steps ($\log_2 N$ instead of $2 \log_2 N$).

3.3. A computation graph for cyclic reduction. In studying the mapping of computations on to a network of processing elements we represent the variable dependencies and the operations performed upon the variables by a directed graph. Nodes correspond to operations consuming and producing data, and directed edges to communication of data. Edges are directed from the source towards the sink. Similarly, we describe the processing ensemble as a graph with nodes corresponding to processing elements with local storage, and edges to interconnections. The mapping problem becomes a problem of embedding one graph in another. In general, the embedding problem should be viewed as a dynamic problem due to the dynamic character of the computation graph.

We represent the coefficients of one equation, the corresponding right-hand side(s), and the unknown(s) corresponding to the column of the diagonal element with one node in the computation graph. Hence, for one right-hand side 5 storage cells are required in each node. The nodes in the computation graph in Fig. 2 are labeled with the equation number, starting from 0. The superscript denotes the reduction step during which the equation is modified.

In the interest of conserving storage, nodes in the same column of the computation graph in Fig. 2 can be identified. The storage requirement is approximately half of that required with the nodes stored in distinct storage cells. The storage conserving scheme is one source of poor performance of many implementations of cyclic reduction on architectures with primary storage partitioned into banks [34]. The degradation is easily seen from Fig. 3, in which nodes in the same column are identified.

With 2^k storage banks and cyclic storage, as performed by most FORTRAN compilers, all equations j such that $i = j \bmod 2^k$ are assigned to bank i. After k reduction steps all equations are in the same bank. The performance may be degraded further towards the end of the reduction phase if the storage is pipelined. In such a case, and with only few equations participating in each reduction step, Gaussian elimination may be preferable to cyclic reduction with respect to storage bandwidth. (It may also be preferable with respect to the time for arithmetic on a pipelined architecture for few equations.) With an odd number of storage banks no bank conflicts occur in cyclic reduction.

4. Finiteness. If the number of equations is greater than the number of processing elements, then several equations have to be identified with the same element (assuming it has sufficient storage). It is desirable to perform the identification of equations such that the amount of arithmetic is distributed as evenly as possible throughout the computations, without excessive communications. There are two commonly used schemes for identification: *consecutive* and *cyclic*. In the *consecutive* scheme all equations $i = \{0, 1, \cdots, N-1\}$ that satisfy the relation $p = \lfloor iK/N \rfloor$ are identified with partition p, $p = \{0, 1, \cdots, K-1\}$. The number of partitions is K. In *cyclic* storage all equations i that satisfy the relations $p = i \bmod K$ are identified with partition p. The consecutive scheme for identification is also often referred to as *domain decomposition*

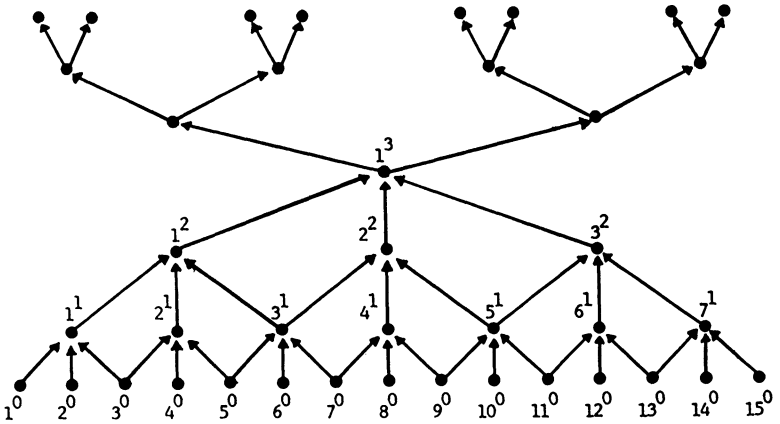

FIG. 2. *A computation graph for odd-even cyclic reduction.*

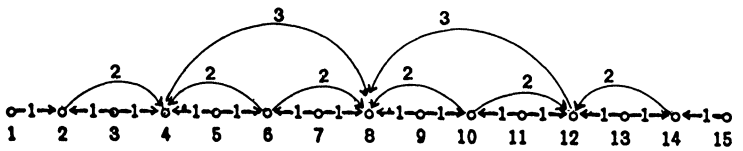

FIG. 3. *A storage conserving computation graph for cyclic reduction.*

or *substructuring*. The cyclic storage scheme yields a higher processor utilization for LU-decomposition on dense matrices [21], [30], [28].

Both the consecutive and the cyclic schemes for identification result in quotient graphs with either $\lfloor N/K \rfloor$ or $\lceil N/K \rceil$ equations per node. Though this is the best possible balance in the distribution of equations, it does not necessarily translate to an even distribution of arithmetic (as already mentioned in the case of banked storage architectures).

We now show that the consecutive scheme for identification is superior for the solution of tridiagonal systems of equations. The analysis will be carried out first for cyclic reduction, then for a combination of Gaussian elimination and cyclic reduction, where the Gaussian elimination is performed within partitions, as in the substructured Gaussian elimination in [58]. Then, we show the relationship between the consecutive scheme, and the algorithms by Sameh et al., and Wang, and (incomplete) nested dissection.

The set of partitions is denoted $P = \{P_0, P_1, \cdots, P_{K-1}\}$. Let the subset of P that is assigned $\lfloor N/K \rfloor$ equations be $Q = \{Q_i\}$, and the subset assigned $\lceil N/K \rceil$ equations be $R = \{R_i\}$, where $|R| = N \bmod K$. For an ensemble of linearly interconnected processors there is no topologically natural choice of K. For a 2-dimensional mesh, K is naturally chosen as the product of two integers, often as a square for reasons of symmetry. For complete binary trees we take $K = 2^k - 1$, implying that we assign one partition to each node of the tree, not only to the leaf nodes. For shuffle networks and boolean cubes, $K = 2^k$.

4.1. Domain decomposition.

4.1.1. Communication aspects of cyclic reduction. For the mapping of partitions onto nodes in a complete binary tree, the following observation is useful. For $K = 2^k - 1$ and $N = 2^n - 1$, the number of partitions with $\lceil N/K \rceil$ equations is equal to the number of nodes in a tree with $n \bmod k$ levels, i.e., $|R| = 2^{n \bmod k} - 1$. If $\lceil N/K \rceil$ equations are assigned to the top $n \bmod k$ levels of a k-level complete binary tree labeled in order [1], then the first $n - k$ reduction steps result in a reduced system that has one equation per tree node. Moreover, the difference in the number of equations subject to elimination operations in any pair of partitions is at most one, throughout the entire reduction phase [26].

For $K = 2^k$, $|R| = K - 1$, i.e., one partition performs elimination operations on one equation less than the other partitions during the first $n - k$ reduction steps.

With respect to interpartition communication needs we first establish the following.

LEMMA 4.1. *In the first $n - k$ reduction steps, partition P_i only communicates with partition P_{i+1} for $i = \{0, 1, 2, \cdots, K-2\}$ and partition P_{i-1} for $i = \{1, 2, \cdots, K-1\}$.*

Proof. The number of indices assigned to a partition is at least $\lfloor N/K \rfloor$. The index difference between equations used in the elimination operations is increasing monotonically during the reduction phase, and is 2^{n-k-1} for step $n - k$. But, the equation index set in each partition is consecutively ordered and $2^{n-k-1} \leqq \lfloor N/K \rfloor$. □

By Lemma 4.1 the interpartition communication during the first $n - k$ reduction steps is entirely between adjacent partitions. However, it is also necessary to establish the direction of communication, particularly if the direction of communication changes. If the direction of communication is changing from one reduction step to the next, pipelining of the communications for successive reduction steps cannot be used to "hide" the latency associated with long routing distances, if adjacent partitions are mapped to processing elements far apart. A proximity preserving path embedding is not necessarily optimal with respect to communication time. For the complete binary

tree a proximity preserving path embedding yields a communication complexity of order $O(\log_2^2 K)$ [26].

THEOREM 4.1. *In the set of partitions* $P_s = 2^{k-s-1}\{1, 3, 5, \cdots, (2^{s+1}-1)\}$, $s = \{0, 1, \cdots, k-1\}$, $|P| = 2^k - 1$ *there exists at least one partition which changes direction of communication with one of its adjacent partitions* $s+1$ *times for every* k *reduction steps.*

We omit the proof of Theorem 4.1 and refer to [26]. The theorem states that among the odd partitions there exists at least one partition that changes the direction of communication with one of its neighboring partitions k times for every k reduction steps. Figure 4 illustrates the interpartition communication.

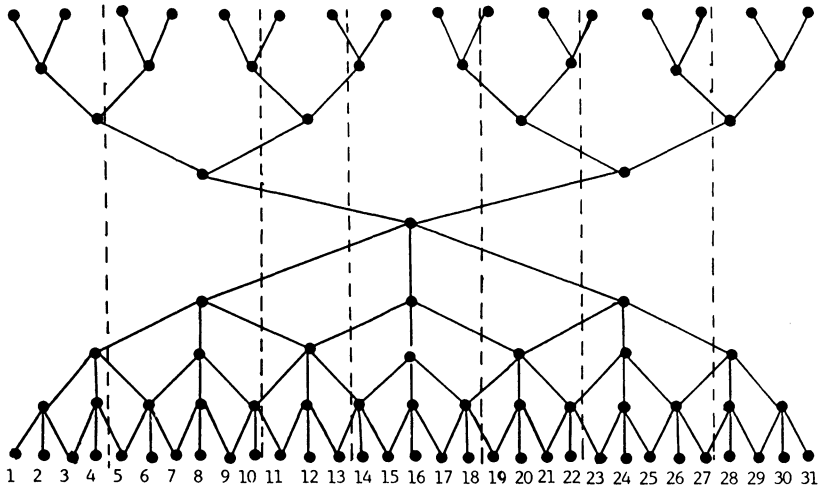

FIG. 4. *Partitioning of the computation graph of cyclic reduction.*

COROLLARY 4.1. *The communication for successive reduction steps can only be partially pipelined.*

COROLLARY 4.2. *If each partition is assigned* 2^{n-k+1} *or* 2^{n-k} *consecutive indices,* $|P| = 2^k - 1$, *then the direction of communication is constant for the first* $n-k+1$ *or* $n-k$ *reduction steps.*

The benefit of modifying the number of equations N to the form $2^m(2^k-1)$ is that the number of indices per partition is even for the first several reduction steps and it is possible to pipeline the communications. This advantage is obtained at the expense of a possible increase in the arithmetic complexity of at most a factor of 2 for the first $n-k+1$ steps. The arithmetic complexity is the same for subsequent steps.

THEOREM 4.2. *For* $K = 2^k$ *the direction of exchange is constant during the first* $n-k$ *steps* [26].

The number of arithmetic operations performed in sequence is approximately $(8+9NR)N/K + (5+6NR)\log_2 K$, where NR is the number of right-hand sides. For this estimate it is assumed that the computations required for the elimination operations on the last K rows are shared between two processing elements: the one holding the row used for the elimination, and the one holding the row subject to elimination. It is also assumed that the reduction and backsubstitution are performed identically on all right-hand sides. We will later present a scheme for improved load balance during the final k reduction steps. The backsubstitution phase can be implemented with communications as in a complete binary tree, in which case most communications include two floating-point numbers, or in a way corresponding to the communication

pattern in the reduction phase. In the latter case each communication only includes one floating-point number per right-hand side. For *NR*-1 an arithmetic complexity of $9 \log_2 K$ for the reduced tridiagonal system can be achieved at the expense of $2(\log_2 K - 1)$ additional communications.

4.1.2. Gaussian elimination locally, cyclic reduction globally. In cyclic reduction the partition assigned equation $2^{n-1} - 1$ requires a total of $n-1$ communications. For the first $n-k$ reduction steps the communication is with adjacent partitions, but during the last k steps with partitions differing in index by 2^j, $j = \{0, 1, \cdots, 2^{k-2}\}$. The lower bound for the communication time is $O(\log_2 K)$ for ensembles with nodes of degree three.

If instead of the partial elimination order defined by cyclic reduction, the elimination order within partitions is taken to be the order of the equation indices (increasing or decreasing order), then the communication requirements are reduced to one communication for the first $n-k$ reduction steps. The reduced system after $n-k$ local eliminations can still be made a tridiagonal system. The fact that local forward and backward elimination and one communication with adjacent partitions yield a reduced tridiagonal system was observed by Wang [58]. The method of Sameh et al. [49], [35] uses no communication in the first $n-k$ reduction steps, but results in a pentadiagonal system. Similar schemes have also been described in [45] and [9]. The advantage of the local elimination order with one communication, as described below, is that it preserves not only diagonal dominance but also symmetry and positive definiteness, also when applied to arbitrary banded matrices [29]. We refer to the algorithm with reduced communication complexity as GECR for Gaussian elimination–cyclic reduction. It requires $\log_2 K$ communications for the reduction phase. Algorithm GECR proceeds in three phases:

- Local forward and backward elimination. The backward elimination requires one communication with an adjacent partition.
- Solve a reduced system of equations with one equation per partition by cyclic reduction.
- Local backsubstitution.

In phase 1 *fill-in* occurs in columns $i \times N/K - 1$ of partition i, $i = \{1, 2, \cdots, K-1\}$, and columns $(i+1)N/K - 1$ of partition i, $i = \{0, 1, \cdots, K-1\}$. The last equation of each partition in this system of equations forms a tridiagonal system in the variables corresponding to the columns $(i+1)N/K - 1$, $i = \{0, 1, 2, \cdots, K-1\}$. The appearance of the matrix after phase 1 is shown in Fig. 5.

The arithmetic complexity of GECR is almost identical to that of cyclic reduction, because of the fill-in in the forward and backward eliminations. Phase 1 of GECR requires 4 arithmetic operations for each elimination operation on the matrix, except in the first and last partitions, which yield a total of 8 arithmetic operations for the forward and back eliminations. For the first and last partitions the forward and backward eliminations require a total of 5 arithmetic operations per equation. There are 2 operations on each right-hand side in each of the forward and backward eliminations, i.e., a total of 4 arithmetic operations per right-hand side in phase 1. There are 5 arithmetic operations per right-hand side in phase 3. The communication complexity of phase 1 is 1. The number of communication actions for phase 2 is $2(\log_2 K - 1)$. With a symmetric allocation of matrix elements as described in § 4.3, the number of arithmetic and communication operations may change slightly.

Note that if a processing element has pipelined arithmetic units and communication is fast, then the fact that cyclic reduction "vectorizes" may make it preferable to GECR

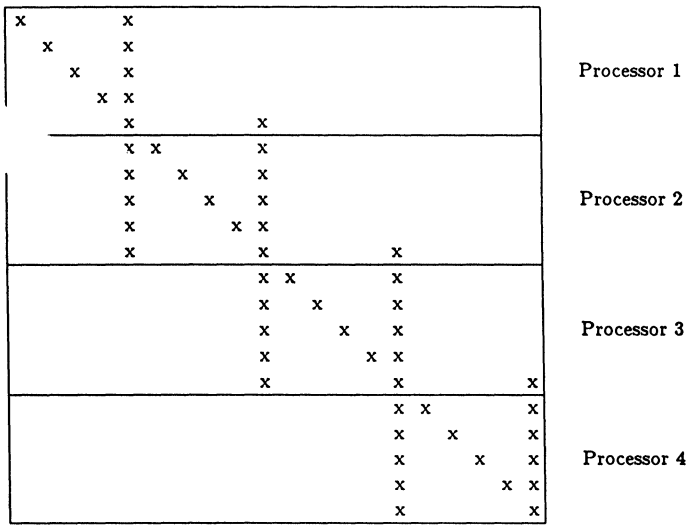

FIG. 5. *The matrix after phase* 1.

with respect to computational complexity. However, if there are many right-hand sides, or if multiple independent problems shall be solved, then "vectorization" across right-hand sides, or across different problems, can be used in conjunction with GECR.

The fact that algorithm GECR preserves diagonal dominance in phase 1 can be exploited for truncation of the reduction process if the diagonal dominance is sufficiently strong [15], [16], [18], [29], [45].

4.2. Cyclic storage. Under the consecutive scheme for identifying equations with partitions, the arithmetic complexity is of order $O(N/K + \log_2 K)$ and the communication complexity for GECR is of order $O(\log_2 K)$. The order of either of these two complexity measures cannot be reduced.

With the cyclic scheme for identifying equations, K must be odd in order to keep the balance of equations subject to elimination operations as even as possible throughout the computations. However, cyclic partitioning is inferior to the consecutive partitioning scheme with respect to communications. This is true for cyclic reduction, and GECR offers no advantage since successive equations are in adjacent partitions. There is no locality in data allocation that can be used advantageously with respect to communication. For a dense system of equations there is no data locality of elimination operations, and the cyclic storage scheme can be used to balance the load [21], [30]. In the solution of tridiagonal systems all elimination operations are local. The cyclic scheme is inherently inferior to the consecutive scheme with respect to computational complexity for tridiagonal (and banded) systems.

4.3. Divide-and-conquer. In this section we relate to each other the methods of incomplete nested dissection [13], Wang [58], and Sameh et al. [49], [35]. Nested dissection on the graph of a tridiagonal matrix is a recursive bisection. The separators are single nodes. The bisection is terminated when there are $K = 2^k$ partitions for K processes. Labeling the bisectors last and in reverse order, and the remaining nodes in the same order as in the initial labeling yields an elimination tree as in Fig. 6. This relabeling corresponds to row and column permutations to yield a system of equations as in Fig. 7. The method by Wang can also be described in terms of a set of separators, and permutation of rows and columns. The elimination tree is labeled K-section in Fig. 6, and the matrix is shown in Fig. 8.

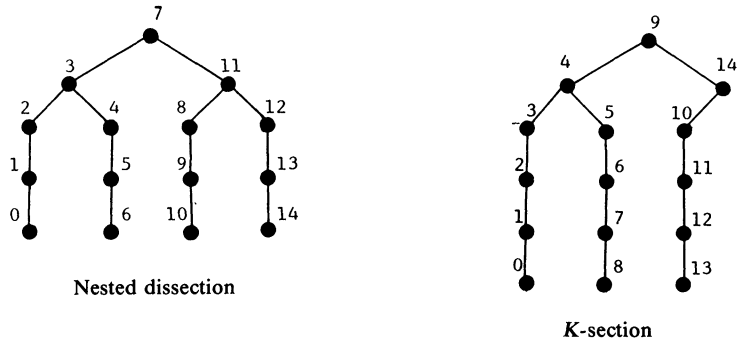

FIG. 6. *Partitioning of tridiagonal systems of equations.*

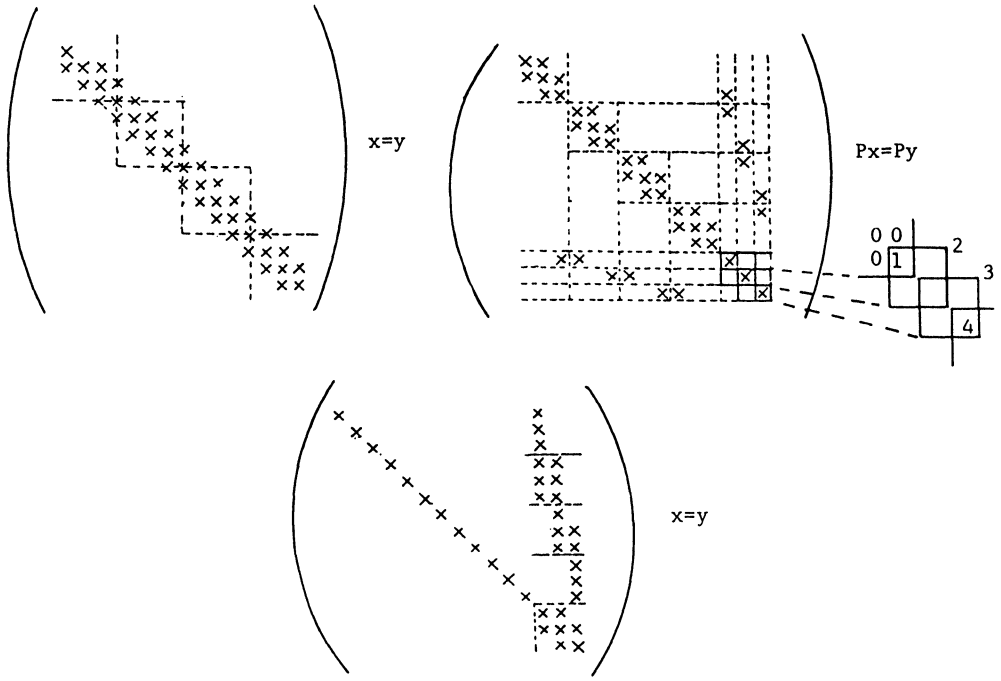

FIG. 7. *Matrix partitioning and permutation for nested dissection.*

Figures 7 and 8 also indicate the association of matrix elements with partitions. If the association of matrix elements with partitions is made the same in the method by Wang as in incomplete nested dissection, then the elimination operations in the first phase are completely local, but an "assembly" of the reduced tridiagonal system requires communication. The matrices to be assembled are 2×2 matrices, except for the first which is a 1×1 matrix. In case of nested dissection the last matrix is also of size 1×1. The matrices to be assembled are shown as squares in Figs. 7 and 8.

If an even number of partitions is desired, as for boolean cube configured ensembles, then incomplete nested dissection is a convenient partitioning strategy. If the number of partitions is of the form 2^k-1, as in complete binary trees, then the K-section strategy is preferable. With the symmetric association of matrix elements

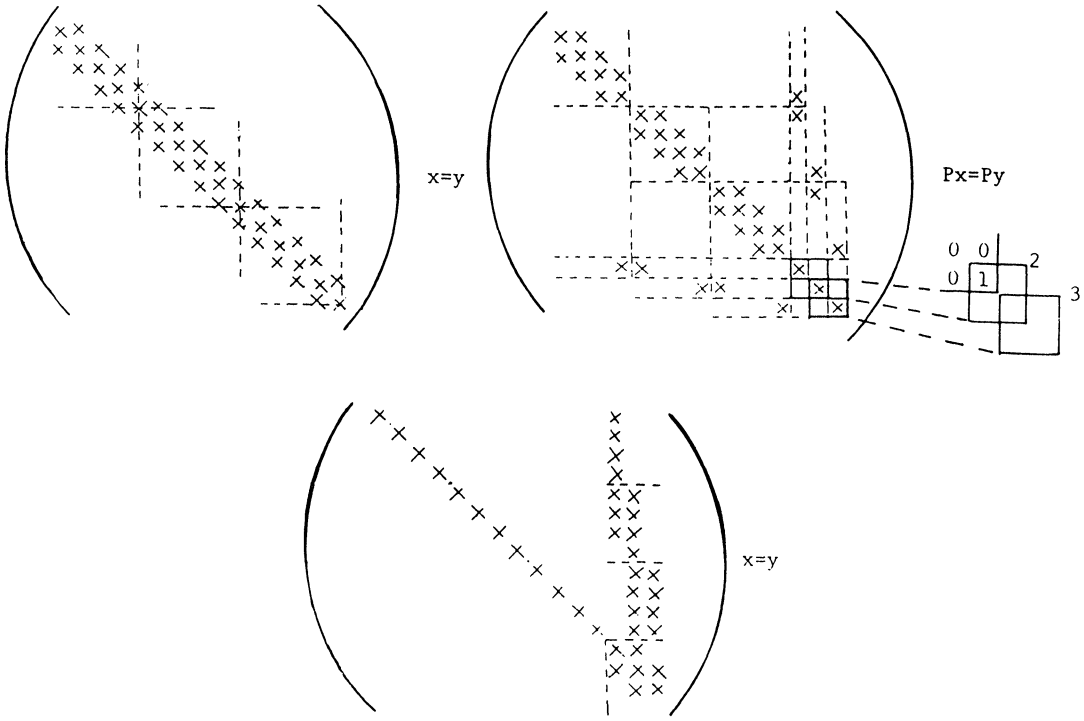

FIG. 8. *Matrix partitioning and permutation for K-section.*

and partitions, the elimination operations on the reduced system can take place as the block matrices are assembled, as is common practice in sparse matrix codes, see e.g. [22], [6]. In the case of nested dissection, elimination is performed on pairs of matrices in the form of a complete binary tree. In the case of K-section one leaf node is missing. The indicated elimination order corresponds to a particular permutation of the separators. After the elimination phase follows a backsubstitution phase that proceeds from the root towards the leaf nodes. Embeddings of the elimination trees in various ensemble architectures will be discussed in § 5.

The algorithm by Sameh et al. cannot be viewed as a symmetric permutation of rows and columns ($PAPP^T x = Py$), a local solve, global solution of a reduced system, and a local backsubstitution. The reduced system is pentadiagonal and of order $2(K-1)$. It can be arranged into a tridiagonal system by pairwise column permutations, but the potential symmetry and diagonal dominance is lost.

4.4. Multiple tridiagonal systems. If there is complete freedom in allocating equations and right-hand sides, then the computations could be localized by solving the system for each right-hand side (or each independent system) in a subensemble of $\lceil K/NR \rceil$ processing elements.

We assume that the system of equations with the multiple right-hand sides is distributed with one equation per partition for all right-hand sides. For multiple independent tridiagonal systems we also assume one equation per partition. Evenly distributed tridiagonal systems occur naturally in the solution of Poisson's problem on ensemble architectures [30] by so-called fast solvers, and in the Alternating Direction Implicit (ADI) method [24]. If the number of equations is greater than the number of partitions, then we assume the consecutive scheme for identifying equations with

partitions, and local forward and backward elimination. We first compare the complexities of solving a system with NR right-hand sides and one equation per partition by either Gaussian elimination or cyclic reduction, then consider the concurrent solution of P independent tridiagonal systems.

The analysis in this section uses an idealized model of computation in which all interpartition communication requires one start-up. The results for Gaussian elimination apply to all ensembles considered in the next section. The arithmetic complexity derived for cyclic reduction also applies for the ensembles studied next, but the communication complexity is higher, in general.

4.4.1. Multiple right-hand sides. For Gaussian elimination the operations on different right-hand sides can be pipelined to keep the processor utilization as high as possible. The parallel arithmetic complexity of 2-way Gaussian elimination for $N = K$ is $(3+5NR)(K-3)/2+5+6NR$ without pipelining of the operations for different right-hand sides. The number of elements communicated in sequence is $(1+NR)\times(K-2)$ and the number of communication actions (start-ups) is $K-2$. In the case of maximum pipelining the arithmetic complexity is $3K+5NR-6$, the number of element transfers in sequence is $K+2NR-1$, and the number of communication actions is also $K+2NR-1$. Optimizing packet sizes, i.e., the amount of data per communication, such that the total time is minimized, yields

$$2\sqrt{(K/2-2)\tau}\left(\sqrt{(NR+2)(t_c+2t_a)}+\sqrt{(NR(t_c+3t_a)}\right) + (5NR-(K/2-1))t_a+2NRt_c+(K-4)\tau. \tag{4.1}$$

The optimum packet size in the forward elimination is

$$\sqrt{\frac{(NR+2)\tau}{(K/2-2)(t_c+2t_a)}}$$

and in the backward elimination

$$\sqrt{\frac{NR\tau}{(K/2-2)(t_c+3t_a)}}.$$

A naive implementation of cyclic reduction in which the computations for all right-hand sides are treated identically yields an arithmetic complexity of $(5+6NR)\times(\log_2 K-2)+5+6NR$ for $N = K$, if the operations required for the elimination of the two off-diagonal elements in a row are distributed over two processing elements. Two communication actions per reduction step are required. The communication complexity is $(3+3NR)(\log_2 K-2)+2(1+NR)$ and the number of start-ups is $3(\log_2 K-1)$, assuming one start-up per interpartition communication. If all operations for the elimination of the off-diagonal elements in a row are performed in one processing element, then the arithmetic complexity is $(8+9NR)(\log_2 K-2)+5+6NR$ and the number of start-ups is reduced to $2(\log_2 K-2)$.

In the naive implementation of cyclic reduction for multiple right-hand sides the processor holding equation $2^{k-1}-1$ is a bottle-neck. It participates in $\log_2 K$ reduction steps for each right-hand side. Balancing the computations such that for $NR < K$ right-hand sides the reduction phase in cyclic reduction converges to distinct processors for different right-hand sides eliminates this problem. The balancing is accomplished by adding "virtual" equations at either end of the system of equations. These virtual equations do not change the solution to the real system, and do not incur any arithmetic or communication operations. The principle in this *balanced* cyclic reduction algorithm is the same as in Hockney's parallel cyclic reduction [18], but we perform concurrent reduction processes on *different* right-hand sides.

To estimate the arithmetic and communication complexities for the balanced cyclic reduction algorithm, one can proceed from the computation graph in Fig. 2. $NR - \lceil NR/K \rceil$ computation graphs of height $\log_2 K$ and $\lceil NR/K \rceil$ of height $\log_2 K - 1$ are superimposed, shifted one step with respect to each other in the horizontal direction. It is easily seen that each processor performs a number of arithmetic operations identical to that of one complete cyclic reduction computation. Hence, the arithmetic complexity is

$$17\left(\log_2 K - 1 + \sum_{i=1}^{\log_2 K - 2} \left\lfloor \frac{NR \bmod K - 1}{2^i} \right\rfloor\right)$$

approximately if $NR \bmod K \leqq (K+1)/2$, else

$$17\left(\frac{K-1}{2} + \sum_{i=1}^{\log_2 K - 2} \left\lfloor \frac{NR \bmod K - (K+1)/2}{2^i} \right\rfloor\right).$$

The speed-up is linear. It is also clear from superimposing the computation graphs that the number of communication actions (start-ups) occurring sequentially is $2 \log_2 K$. However, the amount of data communicated increases, and is approximately $(5K - \log_2 K)\lceil NR/K \rceil$. The complexity of the balanced cyclic reduction algorithm is approximately

$$(4.2) \qquad ((17K - 18 \log_2 K + 2)t_a + (5K - \log_2 K)t_c)\lceil NR/K \rceil + 2\tau \log_2 K.$$

Note that in the balanced cyclic reduction algorithm the operations on the system matrix are repeated for each right-hand side. The parallel arithmetic complexity of cyclic reduction is always lower than that of Gaussian elimination for $NR = 1$, $K > 3$, but for $NR > 1$ pipelined Gaussian elimination may be of a lower complexity. The value of NR at which the cross-over occurs depends on the communication and arithmetic bandwidths, and start-up times. The cross-over point in the idealized architecture is determined by (4.1) and (4.2).

In general, interpartition communication requires more than one communication, as will be shown in subsequent sections. From the complexity expressions for the idealized architecture it is clear that if the start-up time for communication is high, then cyclic reduction is always preferable. On the other hand if the start-up time can be ignored compared to the times for arithmetic or the data transfer time, then Gaussian elimination has an advantage for NR sufficiently large. Both Gaussian elimination and the balanced cyclic reduction algorithm compute the solution *in-place*. Another alternative is to perform a transpose of the data, solve the system locally and transpose back. We will analyze the total complexity of this scheme for the boolean k-cube and compare it with the *in-place* algorithms.

4.4.2. Multiple independent tridiagonal systems. We first establish the following:

THEOREM 4.3. *If P tridiagonal systems shall be solved on a K processor ensemble, $P \leqq K$, and there is no restriction on data allocation, then partitioning of the ensemble into P subensembles yields a lower complexity than sharing the entire ensemble for all P problems.*

Proof. Assume that the solution processes for the P problems on the entire K processor ensemble are pipelined. Let the time be $\alpha(K) + \beta P$. The first term corresponds to the propagation time, and the second to the time for arithmetic in one processor. Then, if the ensemble is partitioned into P subensembles, the time for arithmetic remains constant, since each subensemble must perform P times the work, but for only one problem instead of P problems. The propagation term decreases to $\alpha(K/P)$. □

If each tridiagonal system is distributed evenly over the ensemble, then the same techniques as were discussed for multiple right-hand sides can be applied. For ensembles sufficiently small compared to the number of problems, Gaussian elimination yields a lower complexity than cyclic reduction, if start-ups can be ignored. For sufficiently large P the cross-over point ignoring start-ups is at approximately

$$P = \frac{K(4ta + tc)}{(9ta + 3tc)}.$$

For a 2-dimensional $N \times N$ grid problem P is of the form N/K. Cyclic reduction yields a lower complexity than Gaussian elimination for $N < 85$–128 if $K = 16$, and for $N < 1365$–2048 for $K = 64$ according to the approximate formula, again ignoring start-ups. If start-ups are the determining factor, then cyclic reduction is preferable. An alternative to solving each problem distributed across the ensemble is to perform local forward and backward elimination, then rearrange the data for the global phase such that for $P \geqq K$ problems each tridiagonal system is solved locally and then the solution is distributed to the original data order. We analyze this scheme for the boolean cube.

5. Tridiagonal systems solvers on ensemble architectures. From the analysis above it follows that we only need to consider consecutive partitioning (domain decomposition, substructuring). The computations proceed in three phases of which phase 1 and 3 are local, and phase 2 requires global communication. In the following we focus on phase 2, and assume that there is one equation per processing element. Depending upon the number of systems to be solved, ensemble topology, the arithmetic and communication bandwidths, and start-up times, cyclic reduction or Gaussian elimination may be preferable with respect to the time complexity for phase 2. The mapping of cyclic reduction onto linear arrays, 2-dimensional meshes, complete binary trees, shuffle-exchange and perfect shuffle networks, and boolean cubes is the subject of this section. The arithmetic complexity has been given above, and it suffices to consider the communication complexity for the various mappings.

5.1. Linear arrays. We first assume that partitions are mapped statically to processing elements by identifying partition indices with processor indices. We refer to the algorithm based on the static mapping as an *in-place* algorithm. The communication in reduction step j is with processors at distance 2^j, $j = \{0, 1, \cdots, k-2\}$. Instead of a static map an *unshuffle* operation can be performed between each reduction step. The unshuffle operation moves equations subject to further elimination operations into a subarray, preserving the relative order. We refer to the second algorithm as the *shuffle* algorithm.

It is easily verified that the *in-place* algorithm has $K-2$ start-ups, the same as 2-way Gaussian elimination. The *shuffle* algorithm has $2(K-3)$ start-ups. Unshuffle operations on an array of 8 processing elements are shown in Fig. 9. The number of elements communicated are the same in the *in-place* and *shuffle* algorithms, and it follows that the *in-place* algorithm is superior. The *shuffle* algorithm is useful for 2-dimensional arrays, however.

THEOREM 5.1. *An in-place algorithm for cyclic reduction on a linear array, with equation indices identified with indices of consecutively labeled processors, has a lower communication complexity than algorithms using shuffle operations.*

For truncated cyclic reduction the *in-place* algorithm is even more advantageous than the *shuffle* based algorithm. For an *in-place* algorithm the communication cost

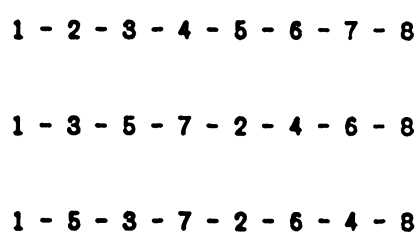

FIG. 9. *Implementing shuffles of decreasing sizes on a linear array.*

doubles for each reduction step, whereas the highest communication cost (half of the total) is incurred in the first step of a *shuffle* based algorithm.

The total complexity for GECR on a linear array is of the form $\alpha N/K+\beta \log_2 K+\gamma K$. It follows that the complexity attains a minimum for K of order $O(\sqrt{N})$. If instead of cyclic reduction Gaussian elimination is used for phase 2, then the total complexity is of the form $\alpha N/K+\gamma K$, and the order of the optimum size array is the same as for GECR.

In comparing the complexity of Gaussian elimination and cyclic reduction for one right-hand side we derive a complexity estimate for 2-way Gaussian elimination of $(4K+2)t_a+(2K-4)t_c+(K-2)\tau$, and an estimate of $(17(\log_2 K-2)+14)t_a+(9K/4-5)t_c+(K-2)\tau$ for cyclic reduction.

THEOREM 5.2. *Cyclic reduction on a linear array always yields a lower complexity than Gaussian elimination for the solution of one tridiagonal system if* $\alpha = t_c/t_a \leqq 1$. *Conversely, Gaussian elimination is of a lower communication complexity if* $\alpha = t_c/t_a \geqq 16$.

The theorem follows directly from the complexity estimates. For $\alpha=4$ cyclic reduction is of a lower complexity for $k\geqq 4$, and for $\alpha\geqq 8$ it is of lower complexity for $k\geqq 5$, assuming $K=2^k-1$.

For multiple right-hand sides or multiple independent problems a similar analysis can be carried out. The complexity of Gaussian elimination for multiple independent problems is approximately $(K+2P-4)(4t_a+2t_c+\tau)$, if the communications are pipelined such that one equation is sent per communication. Without pipelining the complexity is $((4t_a+2t_c)P+\tau)(K-2)$, and with optimized packet sizes for the forward and backward substitution, the complexity becomes approximately

$$\sqrt{2(K-4)P\tau}(\sqrt{3t_c+5t_a}+\sqrt{t_c+3t_a})+8Pt_a+4Pt_c+(K-4)\tau. \tag{5.1}$$

The optimum packet size for the forward phase is

$$\sqrt{\frac{3P\tau}{(K/2-2)(t_c+5/3t_a)}}$$

and for the backward phase

$$\sqrt{\frac{P\tau}{(K/2-2)(t_c+3t_a)}}.$$

The time for the balanced cyclic reduction algorithm is approximately

$$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil+(K-2)\tau. \tag{5.2}$$

The last terms of the complexity estimates (5.1) and (5.2) are approximately equal. Basing the comparison between the two methods on the remaining highest order terms of the two expressions yields a cross-over point of

$$P\approx 0.2\frac{1+\alpha/2+\sqrt{15(1+3\alpha/5)(1+\alpha/3)/16}}{(1+\alpha/9)^2}(K-4)\tau/t_a \quad (\alpha=t_c/t_a).$$

5.2. Two-dimensional meshes. We assume for convenience that the mesh has $2^{k/2}$ processing elements in each dimension. We estimate the communication complexity of *in-place* and *shuffle* algorithms for two "serpentine" embeddings of the partitions.

With a "serpentine" embedding proceeding from the upper left-hand corner to the lower right-hand corner of the mesh, an *in-place* algorithm requires communication between processors at distance 2^{j-1} in reduction step j, $1 \leqq j \leqq k/2$. After $k/2$ steps there is one equation per row that is subject to further elimination operations. The equations are at opposite ends of successive rows, and reduction step $k/2+1$ requires communication between processors at a distance of $2^{k/2}$. After yet another reduction step, processors with equations to be part of subsequent reduction operations are located in every other row of the first column. The linear array algorithm is applied at this point. The communication for reduction steps j, $k/2+2 \leqq j \leqq k-1$ are between processors at distance $2^{j-k/2-1}$. The total number of start-ups for the reduction and backsubstitution is $6(\sqrt{K}-1)$. With the "serpentine" starting and ending close to the center, as in Fig. 10, the number of start-ups is reduced to $2(2\sqrt{K}-3)$, a reduction in start-ups by a factor of $\frac{1}{3}$.

The number of start-ups can be reduced further by using a *shuffle* algorithm for the first $k/2$ reduction steps (and last $k/2$ backsubstitution steps) and an *in-place* algorithm for the last $k/2-1$ steps. The resulting number of start-ups for the "serpentine" embedding proceeding from the upper to the lower left hand corners is $3\sqrt{K}+\log_2 K-4$. After the first reduction computation an exchange operation is performed between distinct pairs of adjacent processors in every other row. This operation brings equations that are involved in the second reduction step into the same set of columns. An unshuffle operation on columns moves the columns with even equations into one half of the array. After $k/2$ communication, reduction, and exchange operations, and $k/2-1$ unshuffle operations of successively decreasing sizes, the equations that are taking part in the last $k/2-1$ reduction steps are within one column. Figure 11 shows some reduction steps.

52	51	50	49	48	47	46	45
53	54	55	56	41	42	43	44
60	59	58	57	40	39	38	37
61	62	63	64	33	34	35	36
4	3	2	1	32	31	30	29
5	6	7	8	25	26	27	28
12	11	10	9	24	23	22	21
13	14	15	16	17	18	19	20

FIG. 10. *An embedding with* $2(2\sqrt{K}-3)$ *start-ups.*

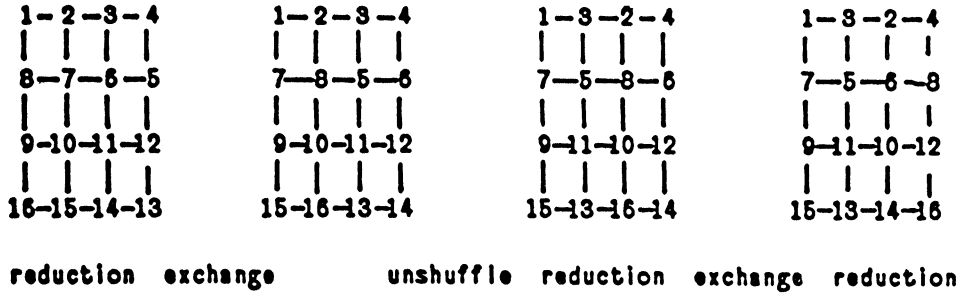

FIG. 11. *Odd-even cyclic reduction on a mesh.*

If the mesh has end-around connections from the end of one row to the beginning of the next row, i.e., the mesh is effectively a twisted cyclinder, then the *in-place* algorithm yields $3\sqrt{K}-4$ start-ups, if the equations are embedded from top to bottom in row major order.

Truncated cyclic reduction reduces the communication time, but not in proportion to the number of reduction steps avoided, because of the nonuniform communication cost.

Parallel cyclic reduction can be implemented on the mesh such that the number of start-ups is approximately $3\sqrt{K}-2$. All processors are used throughout the $O(\log_2 K)$ reduction steps.

The number of start-ups for multiple independent problems, or multiple right-hand sides, is approximately $2(3\sqrt{K}-2)$. The total number of elements transferred between any pair of processors is approximately $5NR$ for NR right-hand sides (or NR problems). Note that two problems can be solved concurrently by mapping one problem to processors 1 through $K-1$, and the other to processors 2 through K. After the first shuffle operation the two problems are confined to half of the array (left and right in Fig. 11).

5.3. Complete binary trees.

5.3.1. Inorder embedding. Presnell and Pargas [43] propose a cyclic reduction algorithm of communication complexity $O(\log_2^2 K)$ for complete binary tree configured ensembles. They map the equations to leaf nodes and use rotation operations for communication. In [27] we present an algorithm for complete binary trees that has a communication complexity of $O(\log_2 K)$.

The partial elimination order of cyclic reduction as defined by the elimination tree suggests an inorder mapping of partitions to processing elements. Such a mapping also allows for conservation of storage, Fig. 3.

With an inorder mapping of partitions to tree nodes, the first step of the cyclic reduction algorithm requires $k-1$ communication start-ups. Partition 2^k-1 is at distance $k-1$ from partitions 2^k-2 and 2^k. Successive reduction steps can be pipelined. Each reduction step following the first only incurs one additional communication action. The number of communications for the reduction phase is $2k-3$. Backsubstitution requires $k-1$ communications, and the total number of start-ups is $3\log_2 K-4$. Figure 12 illustrates the inorder mapping of partitions to tree nodes. The Appendix contains pseudo code for cyclic reduction on a complete binary tree.

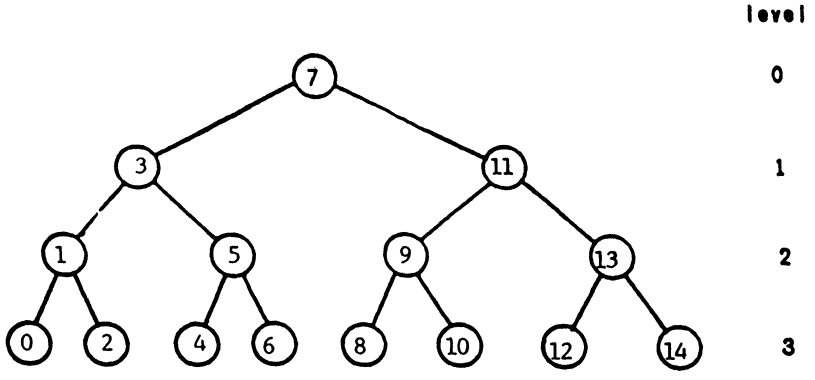

FIG. 12. *Inorder labeling of a complete binary tree.*

In the binary tree algorithm based on an inorder map of equations to nodes of the tree, the order of the time complexity cannot be reduced for truncated cyclic reduction. It takes $k-1$ communications to complete the first reduction step. However, fewer communications are required subsequent to this propagation phase. The backsubstitution phase still needs $k-1$ communication steps, since $x_{K/2-1}$ has to propagate to the leaf level. For m reduction steps the number of start-ups is $2(k-1)+m-1$, and the arithmetic complexity is proportional to $2m+1$.

The reduction in the estimated time complexity is proportional to the reduction in the number of steps in the reduction process. The constant of proportionality varies from 1 to $\frac{1}{3}$. The relative reduction in time for concurrent cyclic reduction is much greater than on a uniprocessor. This property is a consequence of the fact that approximately half of the number of arithmetic operations is performed in the first reduction step and the last backsubstitution step, one quarter in the second step, etc.

5.3.2. A proximity preserving path embedding. The inorder embedding fails to take full advantage of truncating the reduction phase. A proximity preserving path embedding has the potential to take full advantage of an early truncation of the reduction phase, but requires $O(\log_2^2 K)$ communications for the complete cyclic reduction algorithm. We describe a proximity preserving path embedding derived from the loop embedding due to Sekanina [54], [47].

The path embedding in Fig. 13 is generated by an algorithm that is a simple variation of the algorithm described by Rosenberg and Snyder [47]. The labeling proceeds in a depth-first manner. Every other node is labeled on descent. Nodes are labeled in postorder on ascent, except nodes on the path from the root to the leftmost and rightmost child which are labeled in inorder.

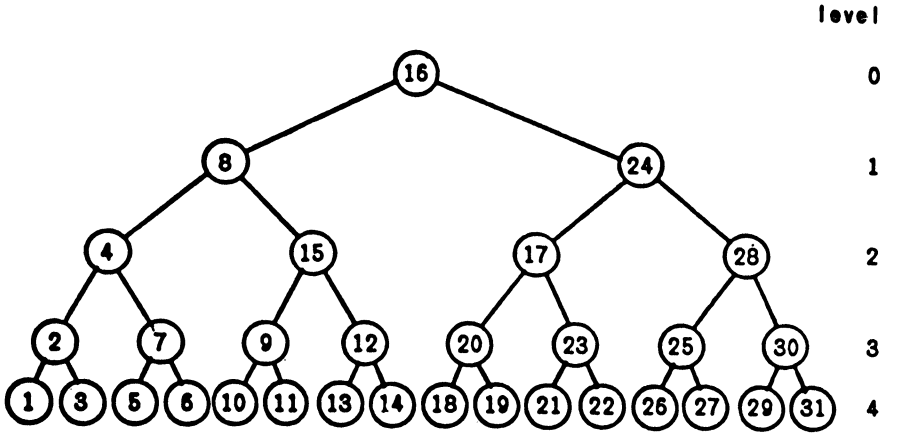

FIG. 13. *Mapping of a path to a binary tree preserving proximity.*

1. The nodes in the left subtree of the root are labeled in descending order, and the nodes in the right subtree in ascending order.
2. After having descended one level the right child of a node is always visited before the left child in the left subtree of the root. The converse is true for the right subtree of the root.
3. The nodes on the paths from the leftmost and rightmost leaf nodes to the root are labeled in inorder.

The first modification only affects the label assigned to a node. The second modification is made only to increase the formal similarity with a tree labeled in

inorder. The two labelings generated by the Rosenberg and Snyder algorithm and an algorithm with only the second modification are isomorphic. The third modification can be made without the maximum distance between adjacent nodes being increased, since we are only embedding a path, not a loop. The distance between the last labeled node in the left subtree and the first labeled node in the right subtree is of no concern in the embedding of a path.

THEOREM 5.3. *The maximum distance between adjacent nodes in the path is* 3 *when embedded by the path embedding algorithm.*

The proof follows that of Rosenberg and Snyder [47]. The modification of the arguments are simple, and can be found in [26].

Though not used for truncated cyclic reduction, we notice the following property.

THEOREM 5.4. *The number of communication steps necessary to change the embedding generated by the path embedding algorithm to an inorder embedding is* $k-3$.

Proof. The proof is in three parts.

(i) The nodes on the paths from the leftmost and rightmost leaf nodes to the root, including the leaf nodes, are labeled in inorder. The root is initially assigned its final inorder label. Its left child is labeled after the right subtree of that node is labeled. Hence, its label is $2k^{-1}-(2k^{-2}-1)-1=2^{k-2}$. By induction it is also true for the remaining nodes along the path. The same property is clearly true for the path to the rightmost leaf node.

(ii) The number of communication steps is at least $k-3$. In the left subtree of the root one node at level 2 is labeled $2^{k-1}-1$, i.e., it is odd and shall be at the leaf level in an inorder labeling.

(iii) It remains to be shown that there is no conflict in data movement. Since the arguments for the right subtree are symmetric, it suffices to prove that there are no conflicts in data movement for the left subtree of the root. The left subtree of the node at level 2 that needs to be relabeled is labeled after its right subtree. The labels start at $2^{k-2}+1$ and end at $3\times 2^{k-3}-1$. Hence, no exchange between the left and right subtrees is necessary. The right child of the node at level 2 being considered has the label $3\times 2^{k-3}$, since it is labeled last in the right subtree. A local exchange gives the node at level 2 the correct label. The theorem follows by induction. □

5.3.3. Parallel cyclic reduction, and multiple independent systems. Inorder mapping of equations to processors is not feasible for Hockney's parallel cyclic reduction. With all odd equations mapped to the leaves, each step requires $O(\log_2 N)$ time. Pipelining cannot be performed for all of the parallel reductions.

The inorder map allows the reduction phases of independent problems to be pipelined. The computation of one problem is initiated before the reduction phase of the preceding problem is complete. Hence, except for the first problem, the propagation time from the leaves to the root is masked by computations.

5.4. Shuffle-exchange networks. A shuffle-exchange network can be defined in terms of the binary encoding of the integers $\{0, 1, 2, \cdots, 2^k-1\}$. Let $(p_{k-1}p_{k-2}\cdots p_1p_0)$, $p_i=\{0,1\}$, $i=\{0,1,\cdots,k-1\}$ be node addresses. Then, node $(p_{k-1}p_{k-2}\cdots p_1p_0)$ is connected to node $(p_{k-2}p_{k-3}\cdots p_0p_{k-1})$ (obtained by a cyclic shift). Furthermore, node $(p_{k-1}p_{k-2}\cdots p_10)$ is connected to $(p_{k-1}p_{k-2}\cdots p_11)$. The edges obtained by cyclic shifts are called *shuffle edges*, and the even-to-odd edges *exchange edges*, [36]. Most nodes in a shuffle-exchange network are connected to three nodes. Nodes $(0\cdots 0)$ and $(1\cdots 1)$ are only connected to one other node, since a cyclic shift yields the same number. Nodes with addresses obtained by cyclic shifts of the same bit string are said to belong to the same necklace. The number of degenerate

necklaces, i.e., necklaces with fewer than k nodes, is of $O(\sqrt{K})$, whereas the number of full necklaces is of $O(K/\log_2 K)$. An 8-node shuffle-exchange network is shown in Fig. 14, and a 16-node network in Fig. 15.

The 8- and 16-node shuffle-exchange networks are planar, but arbitrary shuffle-exchange networks are in general not planar. The diameter of a shuffle-exchange graph is $2k-1$, i.e., almost identical to that of a complete binary tree. The maximum distance between consecutively labeled nodes is $2(k-2)$ and occurs between nodes $(100\cdots0)$ and $(011\cdots1)$. In general, the distance between a pair of nodes is at most twice the number of bit reversals in the addresses.

THEOREM 5.5. *A binary tree of 2^k-1 nodes can be mapped to a shuffle-exchange graph of 2^k nodes such that the distance between nodes adjacent in the tree is at most 2 in the shuffle-exchange graph.*

Proof. Label the nodes in the tree in breadth-first order, starting at the root and assign it the label 1. The label assigned to the left child of node i is $2i$. The label of the right child is $2i+1$. The label of the left child can be obtained by a left cyclic shift of the parent's address. All nodes above the leaf level have the highest order bit 0, and the left cyclic shift always generates an even address. Clearly, if a node of the binary tree is identified with a node in the shuffle-exchange network having the same address, then a parent node and its left child are adjacent. The right child of a tree node is mapped to the odd processor adjacent to the even processor of the left child. Hence, for every node in the tree, its left child is at distance 1, and its right child is at distance 2 (see Fig. 16).

For cyclic reduction on a shuffle-exchange network the partitions are mapped to the processors as described by the tree embedding algorithm, and the inorder tree algorithm used with minor modifications.

Equation $2^{k-1}-1$ is stored at a distance of $2k-3$ from equation 2^{k-1}. The number of communication start-ups is $5k-7$. The above shuffle-exchange algorithm inherits the properties of the tree algorithm. Hence, for truncated cyclic reduction there is still a propagation time of order $O(k)$.

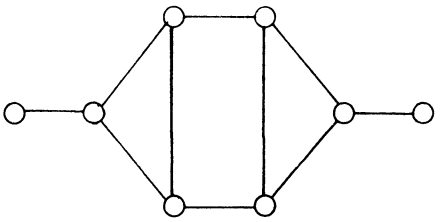

FIG. 14. *An 8-node shuffle-exchange network.*

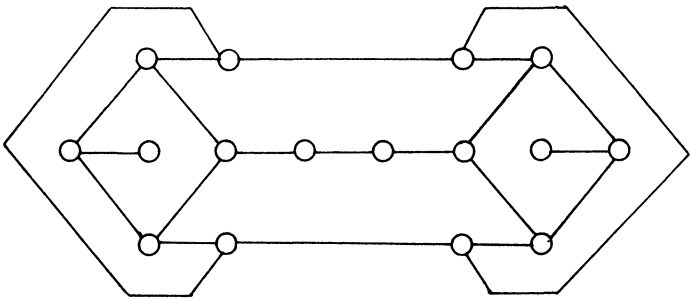

FIG. 15. *A 16-node shuffle-exchange network.*

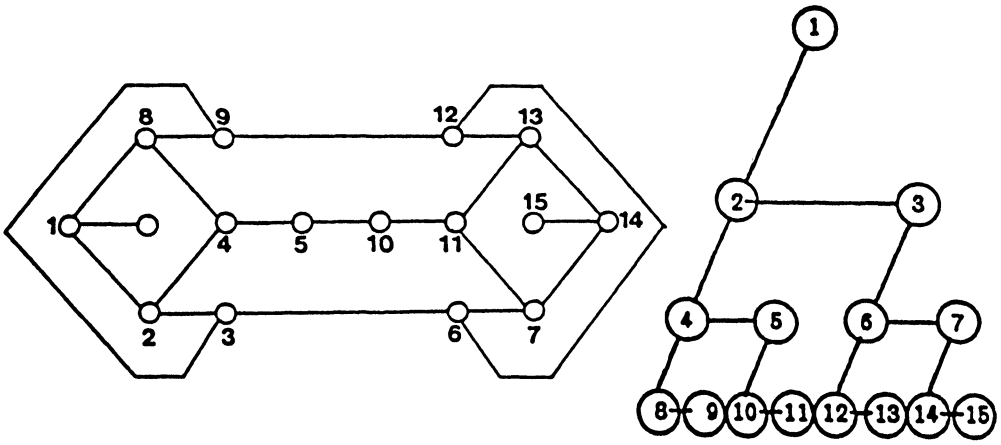

FIG. 16. *A binary tree embedding in a 16-node shuffle-exchange network.*

It is possible to embed two trees in the shuffle-exchange network in such a way that for cyclic reduction the computations for the two trees take place in distinct sets of processors. A processor may be shared for communication purposes. If one tree is mapped to the shuffle-exchange network as described above, then the other tree can be embedded by identifying a tree node with a processor whose address is equal to the bit-wise complementation of the tree node number.

Multiple independent problems can be pipelined in the same way as in the binary tree, in order to reduce the effects of the propagation time on the performance.

5.5. Perfect shuffle networks. A perfect shuffle network differs from a shuffle-exchange network in that an even node is also connected to the preceding odd node [55]. Hence, in the perfect shuffle network processors having successive addresses form a path.

Identify a partition with the processor having the same number. Then, the first reduction step only requires local communication. After the first reduction step is completed, an unshuffle (right cyclic shift) operation is carried out, bringing every even, one step reduced, equation into the processors with addresses in the lower half of the address space. The second reduction step is now carried out in these processors, requiring only nearest neighbor communication. The reduction phase is completed after $k-1$ reduction steps and $k-2$ unshuffle operations. The backsubstitution requires k equation solutions with nearest neighbor communication (in addresses) and $k-2$ shuffle operations (left cyclic shift).

This simple algorithm is not applicable to the shuffle-exchange network, since processors with successive addresses are in general not nearest neighbors. The number of communication start-ups for the perfect shuffle algorithm is $2(2k-3)$.

Truncating the reduction after m steps reduces the number of unshuffle operations to $m-1$. The total number of communication steps in each phase of the cyclic reduction computation is reduced to $2m-1$. The number of start-ups is $2(2m-1)$.

Parallel cyclic reduction can be implemented on the perfect shuffle with no principal difficulty.

Multiple independent problems can be solved concurrently using the balanced cyclic reduction algorithm. The number of start-ups is $2(2k-3)$. After the first shuffle operation half of the problems are contained in the processors with addresses in the lower half of the address space, and the other half of the problems in the processors

with addresses in the upper half of the address space. The total number of elements communicated over any interprocessor communication link is approximately $5NR$.

5.6. Boolean k-cubes. In a boolean k-cube, processors can be assigned addresses so that adjacent processors differ by only 1 bit. Each processor in a k-cube of $K = 2^k$ processors has k neighbors. There is a total of $kK/2$ connections. The diameter of the k-cube is k. A 3-cube is the common 3-dimensional cube. A boolean 4-cube is shown in Fig. 17.

Boolean k-cube algorithms for cyclic reduction can be obtained by embedding a binary tree in the cube, and adapting the tree algorithm to the cube. One tree embedding is obtained by assigning the root of the breadth-first numbered binary tree to processor 0 in the k-cube. Then, for a tree node assigned to processor $(00\cdots 0p_rp_{r-1}\cdots p_0)$, its left child is assigned to processor $(00\cdots 1p_rp_{r-1}\cdots p_0)$, and its right child to processor $(00\cdots 1\bar{p}_rp_{r+1}\cdots p_0)$, where $\bar{p}_r$ is the complement of p_r. The right child of a node is at distance 2 from its parent. Another embedding in which some adjacent tree nodes are at distance 2 when embedded in the cube is obtained by labeling the tree in inorder. Yet another one is used below. It is also possible to embed a 2^k-1 node binary tree in a $2k$-cube such that adjacent tree nodes are at distance 1 in the cube [32].

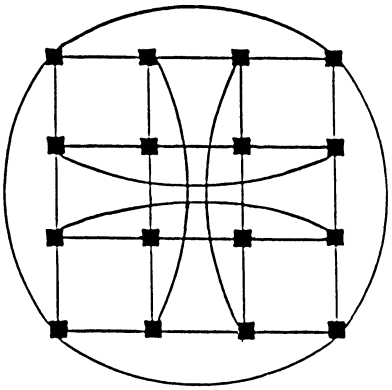

FIG. 17. *A boolean 4-cube.*

We will now briefly describe two k-cube algorithms: an *in-place* algorithm, and a *folding* algorithm. In the first algorithm equations remain in their original location throughout the computations. In the second, computations are performed in cubes of successively lower dimensions by properly moving equations to lower dimensional cubes as the computations progress. The *folding* algorithm is described in detail in terms of pseudocode in the Appendix. Note that in the k-cube, processors with successive addresses are in general not neighbors. For instance, processors $K/2-1$ and $K/2$ are a distance k apart. The objective is to embed the graph of Fig. 2. For convenience we number the equations starting from 0. The equations are initially mapped to the processors in the cube using a *binary-reflected* Gray code [44]. Gray codes for successive integers differ by only 1 bit. This property guarantees that the first reduction step only involves nearest neighbor communication.

Let G_i be the binary-reflected Gray code of i, and $G(k) = (G_0, G_1, \cdots, G_{2^k-1})$ be a k-bit code. Then, a binary-reflected Gray code is defined by

$$G(k+1) = (0G_0, 0G_1, \cdots, 0G_{2^k-1}, 1G_{2^k-1}, \cdots, 1G_1, 1G_0)$$

or

$$G(k+1) = (G_0 0, G_0 1, G_1 1, G_1 0, G_2 0, G_2 1, \cdots, G_{2^k-1} 1, G_{2^k-1} 0).$$

The binary-reflected 4-bit Gray code is

$$G(4) = (0000, 0001, 0011, 0010, 0110, 0111, 0101,$$
$$0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000).$$

Moreover, let $i = (r_k r_{k-1} \cdots r_0)$ and $G_i = (g_k g_{k-1} \cdots g_1)$. Then the encoding and decoding is given by $g_j = (r_j + r_{j-1}) \bmod 2$ (encoding) and $r_j = (\sum_{i=j+1}^{k} g_i) \bmod 2$ (decoding) (note $r_k = 0$) [44].

LEMMA 5.1. *For the binary-reflected Gray code G_i and G_{i+2^j} differs in precisely 2 bits for $j > 0$. Those bits are g_j and g_{s+1}, where s is the bit position in which the carry stops propagating when 2^j is added to i.*

Proof. Let the binary encoding of i be $(i_n i_{n-1} \cdots i_0)$ and that of $r = i + 2^j$ be $(r_n r_{n-1}, \cdots, r_0)$. Furthermore let the Gray code of i be $G(n) = (g_n g_{n-1} \cdots g_1)$ and that of r be $H(n) = (h_n h_{n-1} \cdots h_1)$. Then $i_m = r_m$, $m = \{0, 1, \cdots, j-1\}$ and $r_m = \bar{i}_m$, $m = \{j, j+1, \cdots, s\}$, where $s \geqq j$ is the bit where the carry stops propagating. It follows from the encoding formula that $h_m = g_m$, $m = \{1, 2, \cdots, j-1, j+1, \cdots, s\}$ and $h_j = \bar{g}_j$, $h_{s+1} = \bar{g}_{s+1}$. □

THEOREM 5.6. *The number of communication steps per reduction step is 2 for the in-place algorithm, except for the first step for which it is 1.*

Proof. The communication distance for the first step is 1 from the definition of the code, and 2 for the remaining reduction steps by Lemma 5.1. It remains to be shown that the communication paths can be made disjoint. The following routing guarantees disjoint paths:

- Processor G_i sends data to processor G_{i+2^j} by first sending it to the processor with address obtained by complementing the lowest order bit that differs in the codes G_i and G_{i+2^j}.
- Processor G_{i-2^j} sends data to processor G_i by first sending it to the processor with address obtained by complementing the highest order bit that differs in G_{i-2^j} and G_i. □

In the *in-place* algorithm some processors only serve as routing nodes for all but the first reduction step. It is necessary to compute the addresses of these processors, and to insert the necessary communication code, also for the reduction and backsubstitution steps in which no computations take place. In the *folding* algorithm there is no need for communication code when the processor no longer takes part in the reduction computations.

For the *folding* algorithm we observe that the equations that participate in reduction step j have indices in the set $M^j = \{(m_j + 1)2^j - 1\}$, $m_j = \{0, 1, 2, \cdots, 2^{k-j} - 2\}$. Elimination is performed on equations with indices in the set M^{j+1}. If the equations in M^j are embedded in the cube with one equation per node, and such that consecutive equations are in adjacent nodes, then from Lemma 5.1 consecutively ordered nodes in M^{j+1} are at distance 2.

LEMMA 5.2. *By performing one exchange operation on selected pairs of adjacent nodes between successive reduction steps the equations subject to elimination operations form a path with successively higher ordered equations in adjacent processors.*

Proof. It is true for $j = 0$ by construction. From the definition of the binary-reflected Gray code, it follows that $G_{2q+1}(k-p) = (G_q(k-p-1)X)$, where $X = \{0, 1\}$, $p = 0, 1, \cdots, k-2$ and $q = 0, 1, \cdots, 2^{k-p-1} - 1$. The equations in M^j on which reduction

is performed are the ones with $m_j = \{1, 3, \cdots, 2^{k-j} - 3\}$, i.e., the indices are of the form $2i+1$, $i = 0, 1, \cdots, 2^{k-j} - 2$. Let $p = j - 1$. Then $G_{2q+1}(k-(j-1)) = (G_q(k-j)X)$, $q = 0, 1, \cdots, 2^{k-j} - 1$. Hence, an equation with index in the set $\{(2i+2)2^j - 1\}$, $i = \{0, 1, \cdots, 2^{k-j} - 2\}$ is either in a node with $X = 1$ or can be moved to that node by an exchange operation between nodes $(G_q(k-j)0)$ and $(G_q(k-j)1)$. The theorem follows by induction. □

From the encoding and decoding formulas and the above lemma it follows that each processor has sufficient information locally to determine if an exchange is necessary, and with which processor to exchange data. The necessary information items are: processor address, equation index (which can be computed from the address), and reduction step index (j). The second reduction step is carried out in the subcube $(G_i 1)$, $i = K/2 - 1$. The exchange-reduction steps are repeated $k-3$ additional times on successively smaller subcubes. In the jth step, exchange is performed on the jth lowest order bit, if the leading $k-j$ bits encode an even integer and $g_j = 1$, or the integer is odd and $g_j = 0$.

Processor $(01 \cdots 11)$ computes $x_{2^{k-1}-1}$. Backsubstitution is then carried out in reverse order compared to the reduction phase. The number of processors computing x-values doubles for every step. In the final step half of the processors compute half of the unknowns.

The communication complexity is the same for both the *in-place* and *folding* algorithm. The number of start-ups is $2(2k-3)$.

In the k-cube algorithms based on binary-reflected Gray codes, full advantage can be taken of truncated cyclic reduction. Each reduction step is carried out on all relevant equations during the same time step. Hence, truncating the reduction after m steps reduces the total time proportionally.

Parallel cyclic reduction can be carried out on the k-cube in $O(k)$ time.

Multiple independent problems can be solved by pipelined Gaussian elimination with a complexity of

$$(5.1) \qquad \sqrt{2(K-4)P\tau}(\sqrt{3t_c + 5t_a} + \sqrt{t_c + 3t_a}) + 8Pt_a + 4Pt_c + (K-4)\tau$$

and by the balanced cyclic reduction algorithm with a complexity of

$$(5.3) \qquad ((17K - 18\log_2 K + 2)t_a + (5K - \log_2 K)t_c)\lceil P/K \rceil + 2(2\log_2 K - 1)\tau.$$

An alternative to these two *in-place* algorithms is to transpose the data such that each tridiagonal system is solved locally. If the transpose operation is carried out recursively, then $\log_2 K$ steps are required on the k-cube [32]. The communication complexity for the two transpose operations are $(5P/2t_c + 2\tau)\log_2 K$, $P \geqq K$. Pipelining the communication and optimizing the packet size in order to minimize the communication time yields a total complexity of

$$(5.4) \qquad (8 - 7/K)Pt_a + 2\sqrt{5P(\log_2 K - 1)\tau t_c} + 5P/2t_c + 2(\log_2 K - 1)\tau.$$

In (5.4) the same packet size $(\sqrt{5P\tau/(4(\log_2 K - 1)t_c)})$ is used in both transpose operations. The number of elements communicated is approximately half of that for the balanced cyclic reduction algorithm, and so is the arithmetic complexity, since Gaussian elimination is used locally instead of cyclic reduction. The complexity of the optimally pipelined transpose—local solve—transpose algorithm TGET and that of the balanced cyclic reduction algorithm compares approximately as $a + b + 2\sqrt{ab} + c$ and $2(a+b+c)$ (with $a = 2(\log_2 K - 1)$, $b = 5Pt_c/2$, and $c = 8Pt_a$).

THEOREM 5.7. *An optimally pipelined transpose—local solve—transpose tridiagonal solver is of a lower complexity than a balanced cyclic reduction solver by a small constant factor* (*between* 0.5 *and* 1).

A total optimum may be achieved by combining cyclic reduction with Gaussian elimination for the last several reduction steps [23].

5.7. Shared memory architectures. We have in our computational model assumed that each processor has its own local storage. In architectures such as the Ultracomputer [50], [14] and the TRAC [53], processors and storage units are on opposite sides of a switching network. The communication requirements for parallel cyclic reduction [18] have been analyzed for the TRAC architecture by Kapur and Browne [33], and for the Ultracomputer by Peskin [42]. The communication complexity for the shared memory architectures increases by a factor proportional to the depth of the switching network, compared to the previously described ensemble architectures. The sequential dependencies in the cyclic reduction algorithm are such that communication through the network is required for each reduction step. Pipelining the switch network does not reduce the effect on the switch latency. The communications complexity is proportional to $\log_2^2 K$. This complexity is higher than for the binary tree, the shuffle-exchange, the perfect shuffle, and the k-cube by a factor of $\log_2 K$.

5.8. Programming issues. All algorithms described above have distributed control, and data is distributed throughout the storage of all processors. The binary tree algorithm has three kinds of programs: one for the root, one for intermediate level processors, and one for the leaf processors. The shuffle-exchange network algorithm also uses three different codes. The boolean k-cube algorithm uses the same code in all processors. Hence, even though all algorithms are of the MIMD type, the degree of program uniformity across the ensemble is indeed high. The time for program loading can be made short for the tree by using recursive program loading, and an encoding of the binary tree as described in [38]. For a few equations per node the ratio of total program store to data store is significant.

6. Summary and conclusions. The communication complexity of the tridiagonal system solvers described here is of the same order as the diameter of the ensemble configuration. Hence, for networks of N processing elements and a diameter of order $O(\log_2 N)$, the total solution time is of the same order as the lower bound for the solution of an irreducible system of equations on any circuitry with bounded fan-in.

In the case of one equation per processing element, $N = K$, and assuming that all elimination and backsubstitution operations are performed in the processing element storing the row subject to elimination, the arithmetic complexity of cyclic reduction is the same for all ensembles, namely $17(\log_2 K - 2) + 14$ operations. This complexity can be reduced to $11(\log_2 K - 2) + 11$ operations if the operations are shared between the processing elements storing the pivot rows and the ones storing the rows subject to elimination. However, this requires additional communication per reduction phase. The arithmetic complexity can be further reduced to $9(\log_2 K - 2) + 11$ with two additional communications per step.

In the first case, the number of elements per communication is 4 in the reduction phase and 1 in the backsubstitution phase, except for the binary tree (and the shuffle-exchange) algorithm, for which 2 elements are communicated between a pair of processors in each step. In all but the binary tree (and shuffle-exchange) algorithm the two variables needed in the backsubstitution on an equation can be communicated over different links. If the operations required for the elimination of a single element

is shared between two processing elements, then the maximum number of elements per communication in the reduction phase is reduced to 3.

With all operations for the elimination of the two off-diagonal elements in a row carried out in one processor the number of communication start-ups is $K-2$ for the linear array and an *in-place* algorithm. For the 2-dimensional mesh the number of start-ups is $3\sqrt{K}+\log_2 K-4$ (or $3\sqrt{K}-4$ if the mesh has end-around connections). The number of start-ups for the binary tree is $3\log_2 K-4$, $5\log_2 K-6$ for the shuffle-exchange network, and $2(2\log_2 K-3)$ for the perfect shuffle and boolean cube networks. The number of start-ups increases by 50% for the linear array, if the operations for the elimination of a single element is shared between a pair of processors, and doubles if load sharing is performed as indicated for the lowest arithmetic complexity. For the perfect shuffle and boolean cube algorithms the corresponding numbers are 25% and 50%, respectively. For the binary tree and shuffle-exchange algorithms the load sharing prevents effective pipelining. The communication complexity increases to order $O(\log_2^2 K)$.

We conclude that if $\tau \ll t_a$ then the load sharing can be used effectively on a perfect shuffle and boolean cube. However, if $\tau \geqq t_a$, then load sharing does not pay off (but it may for a banded matrix [29]). The complexity term distinguishing the ensemble configurations is the communication complexity which is summarized in Table 1.

For a linear array cyclic reduction is more efficient than 2-way Gaussian elimination if the communication time is equal to the time for an arithmetic operation. However, if the communication is at least an order of magnitude slower than the arithmetic (a factor of 16 in our estimates), then Gaussian elimination is of a lower time complexity.

TABLE 1
Communication complexities (approximate).

Configuration	Total
Linear array	$(K-2)(5/2t_c+\tau)$
2-dim mesh w/o end-around	$(3\sqrt{K}+\log_2 K-4)(5/2t_c+\tau)$
2-dim mesh w/end-around	$(2k-1)t_a+(3\sqrt{K}-4)(5/2t_c+\tau)$
Binary tree	$10(k-2)t_c+(3k-4)\tau$
Shuffle-exchange	$2(6k-11)t_c+(5k-4)\tau$
Perfect shuffle	$(2k-3)(5t_c+2\tau)$
Boolean k-cube	$(2k-3)(5t_c+2\tau)$

With a large number of equations per node we use the consecutive scheme for identifying nodes with processing elements, i.e., substructuring. Moreover, we perform local forward and backward elimination such that a tridiagonal system with one equation per processing element results, and the previous analysis applies. The number of local arithmetic operations is approximately $17N/K$. The total complexity for GECR is of order $O(N/K+K)$ for the linear array, $O(N/K+\sqrt{K})$ for the 2-dimensional mesh, and $O(N/K+\log_2 K)$ for the complete binary tree, the shuffle-exchange and perfect shuffle networks, and the boolean k-cube. The difference in complexity between

different ensemble configurations is insignificant for $N \gg K$. The speed-up is at first linear in K. The range for linear speed-up depends on the type of interconnection, and the ratio $\alpha = t_c/t_a$.

The minimal computational time is $O(\log_2 N)$ for the complete binary tree (and the shuffle-exchange network), the perfect shuffle and the boolean k-cube. The speed-up has a maximum for $K \approx N/(1+\alpha)$. The maximum speed-up for a 2-dimensional mesh is obtained for $K \approx (N/2\alpha)^{2/3}$, and for the linear array for $K \approx (N/\alpha)^{1/2}$. The minimal computational times are $O(N^{1/3})$ and $O(N^{1/2})$, respectively. Table 2 summarizes the optimum ensemble sizes and the corresponding time complexities.

TABLE 2
Optimum ensemble sizes and time complexities (approximate).

Configuration	Optimum Size	Minimum Time
Linear array	$O(\sqrt{N/\alpha})$	$O(\sqrt{N})$
2-dim Mesh	$O(N/2\alpha)^{2/3}$	$O(N^{1/3})$
Binary tree	$N/(1+\alpha)$	$O(\log_2 N)$
Shuffle-exchange	$N/(1+\alpha)$	$O(\log_2 N)$
Perfect shuffle	$N/(1+\alpha)$	$O(\log_2 N)$
Boolean k-cube	$N/(1+\alpha)$	$O(\log_2 N)$

The decrease in speed-up is quite dramatic for a linear array with slow communication as can be seen in Fig. 18. The figure displays graphically the complexity estimates in Table 4. The tree algorithm is superior to the perfect shuffle and k-cube algorithms if there is only one equation per processor, but inferior if there are multiple equations per processor. The difference is small for fast communication, but significant for slow communication. The shuffle-exchange algorithm is inferior by a factor of 2, at most. The efficiencies, i.e., the (speed-up)/(number of processors), decrease fairly rapidly if the number of processors is increased beyond a certain number that depends on problem size and the relative cost of communication. The dependence on the problem size for the binary tree, the perfect shuffle, and the k-cube, is illustrated in Fig. 19, and the dependence on the relative communication cost in Fig. 20. Figure 21 shows how the ratio of processors to problem size for a given efficiency decreases as a function of increased relative communication cost. The ratio increases with the problem size.

Truncated cyclic reduction reduces the total time in proportion to the number of steps avoided for the binary tree, the shuffle-exchange, perfect shuffle, and boolean k-cube algorithms. The relative reduction for the binary tree ranges from $(k-m)/k$ for $t_c \ll t_a$ to $(k-m)/3k$ for $t_c > t_a$. The relative reduction for the perfect shuffle and the k-cube is $(k-m)/k$. For the mesh the reduction is not linear due to the shuffle operations. Since the tree algorithm benefits from truncated reduction to a lesser extent than the perfect shuffle and k-cube algorithms, it may lose its edge over those algorithms for truncated reduction. Table 3 gives the complexity estimates for truncated cyclic reduction on a binary tree, perfect shuffle, and k-cube.

The proportional reduction in computational time offered by truncated cyclic reduction on the binary tree, the shuffle-exchange, the perfect shuffle and the k-cube

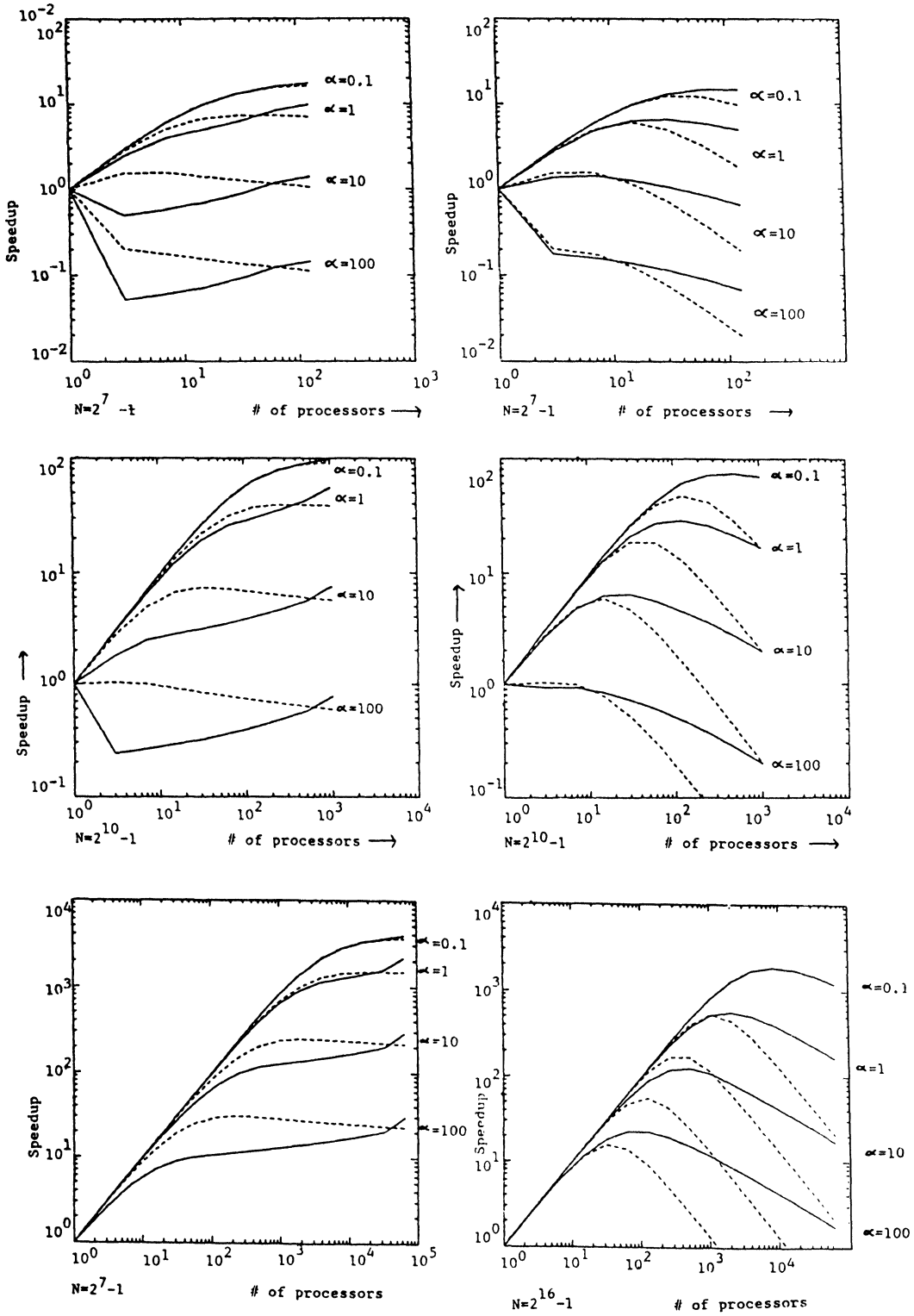

FIG. 18. *Speed-up for* $N = 2^7 - 1$, $N = 2^{10} - 1$, *and* $N = 2^{16} - 1$. $\alpha = 0.1$, 1, 10, *and* 100.

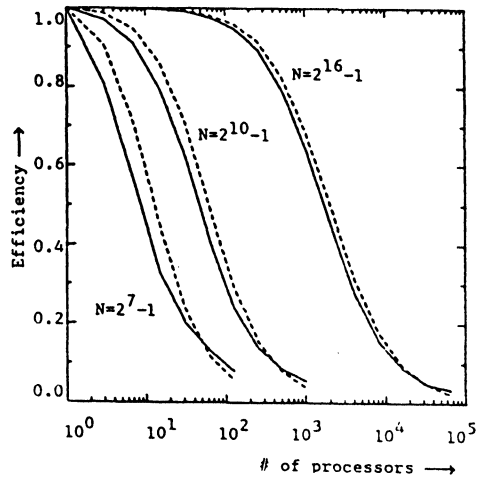

FIG. 19. *The efficiency for a binary tree, perfect shuffle and k-cube,* $\alpha = 1$ $N = 2^7 - 1, 2^{10} - 1,$ *and* $2^{16} - 1$.

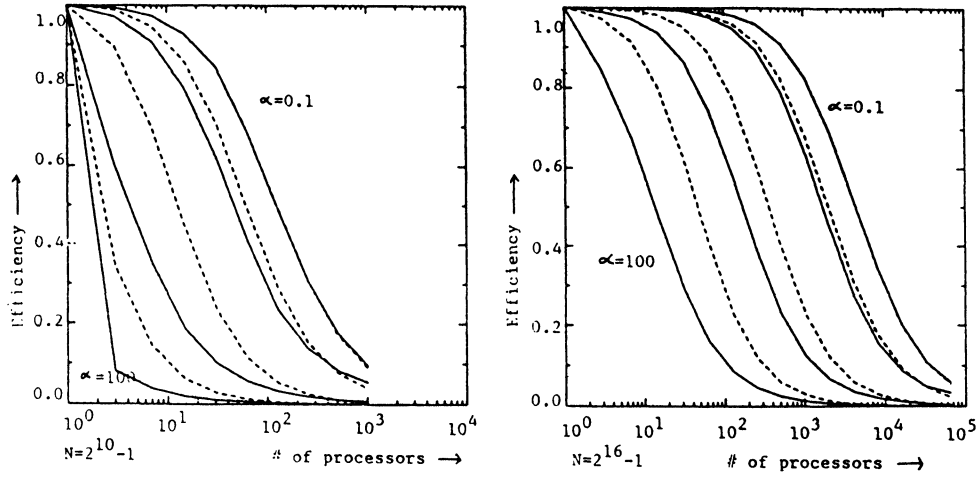

FIG. 20. *The efficiency for a binary tree, perfect shuffle and k-cube,* $N = 2^{10} - 1, 2^{16} - 1$. $\alpha = 0.1, 1, 10, 100$.

is significantly better than on a uniprocessor. Most operations are performed in the first few reduction steps, and the last few backsubstitution steps.

For multiple independent problems partitioning makes the most efficient use of the processors for all investigated topologies. If the data for each problem is uniformly distributed across the ensemble by domain decomposition (substructuring), then the balanced cyclic reduction algorithm yields high processor utilization without an increase in the number of start-ups. The parallel arithmetic complexity eventually becomes approximately twice that of pipelined Gaussian elimination. The number of elements communicated is only slightly larger for cyclic reduction. We summarize the complexity estimates for multiple independent problems on a linear array and a boolean k-cube in Table 4. The complexity of Gaussian elimination is the same for both configurations.

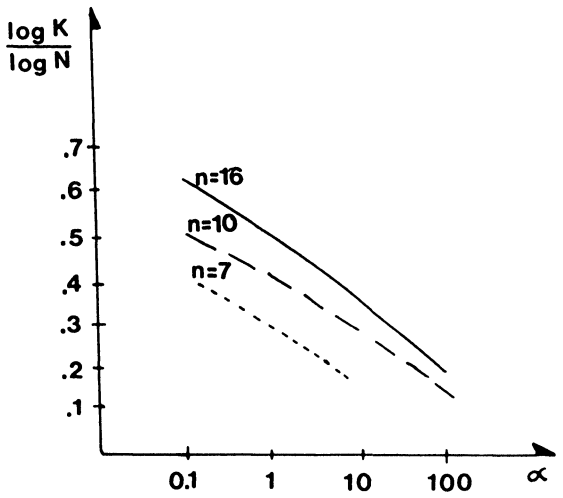

FIG. 21. *Relative number of processors for* 80% *efficiency (tree, perfect shuffle, cube).*

TABLE 3
Estimated complexities for truncated, m-step, cyclic reduction.

Configuration	Complexity
Binary tree	$(m+2)t_a+(m-1)\max(t_a, t_c)+2(k-1)t_c$
Perfect shuffle and k-cube	$(2m+1)t_a+2m(2m-1)t_c$

TABLE 4
Estimated complexities for some ensembles, multiple equations per processor, $k>1$.

Configuration	Complexity
(Linear array)—GE	$\sqrt{2(K-4)P\tau}(\sqrt{3t_c+5t_a}+\sqrt{t_c+3t_a})+8Pt_a+4Pt_c+(K-4)\tau$
Linear array—CR	$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil+(K-2)\tau$
Boolean k-cube—CR	$((17K-18\log_2 K+2)t_a+(5K-\log_2 K)t_c)\lceil P/K\rceil$ $+2(2\log_2 K-1)\tau$
Boolean k-cube—TGET	$(8-7/K)Pt_a+2\sqrt{5P(\log_2 K-1)\tau t_c}+5P/2t_c+2(\log_2 K-1)\tau$

On a linear array Gaussian elimination is of a lower complexity than the balanced cyclic reduction algorithm for

$$P\approx 0.2\frac{1+\alpha/2+\sqrt{15(1+3\alpha/5)(1+\alpha)/16}}{(1+\alpha/9)^2}(K-4)\tau/t_a,$$

approximately. Hence, if the overhead in communication is high relative to the time for a floating-point operation, then the pipelined Gaussian elimination algorithm is preferable only if $P \gg K$. On a boolean cube an optimally pipelined transpose algorithm, local Gaussian elimination, and another transposition is of slightly lower complexity than the balanced cyclic reduction algorithm, which in turn always has fewer start-ups

($O(\log_2 K)$ instead of $O(K)$) than the pipelined Gaussian elimination algorithm. If start-ups are ignored, then the pipelined Gaussian elimination algorithm is of a lower complexity than the balanced cyclic reduction if $P > K(4ta + tc)/(9ta + 3tc)$. Algorithm TGET is always of a lower complexity than pipelined Gaussian elimination.

Finally, we note that all algorithms have distributed control. Moreover, even though the algorithms make use of the MIMD feature of the architecture, only a few different codes are used. In the binary tree there are three kinds: one for the root, one for the intermediate level processors, and one for the leaves. For the k-cube all processors execute the same code. The code for these architectures can be loaded recursively.

7. Appendix.

7.1. A mesh algorithm. Let rows and columns be numbered from 0 to $\sqrt{K} - 1$. Then, the algorithm is:

```
begin
  for i := 1 to k/2 − 1 do
    begin
      Communicate with adjacent processors
      Perform a reduction computation
      Odd rows execute exchange operations between even columns and the
          succeeding odd column
      Perform a shuffle operation of size 2^(k/2−(i−1))
    end
  end
  Communicate with adjacent processor
  Perform a reduction computation
  Odd rows execute exchange operations between the even column and the
      succeeding odd column
  invoke a linear array algorithm
end
```

7.2. A complete binary tree algorithm.

Root processor(i):

$x_0 := 0$; $x_{N+1} := 0$

$k := 1$

$m := i/(2k)$

Reduction computations

while m is even do

receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child

receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$ from the right child

$e_i := -a_i / b_{i-k}$

$f_i := -c_i / b_{i+k}$

$a_i := e_i a_{i-k}$

$c_i := f_i c_{i+k}$

$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$

$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$

$k := 2k$

$m := m/2$

enddo

The last reduction step

for m odd do
receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ from the right child
$e_i := -a_i / b_{i-k}$
$f_i := -c_i / b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
enddo

Backsubstitution

$x_i := y_i / b_i$
send (x_{i-2k}, x_i) to the left child
send (x_i, x_{i+2k}) to the right child
Intermediate level processor(i):
$k := 1$
$m := i/(2k)$

Reduction computations

while m is even do
receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$ from the right child
send $(al, bl, cl, yl, ar, br, cr, yr)$ to the parent
$e_i := -a_i / b_{i-k}$
$f_i := -c_i / b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
$k := 2k$
$m := m/2$
enddo

The last reduction step for node i

for m odd do
receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ from the right child
send $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k}, a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ to the parent
$e_i := -a_i / b_{i-k}$
$f_i := -c_i / b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
send (a_i, b_i, c_i, y_i) to the parent
enddo

Backsubstitution

receive (x_{i-2k}, x_{i+2k}) from the parent
$x_i := (y_i - a_i x_{i-2k} - c_i x_{i+2k})/b_i$
send (x_{i-2k}, x_i) to the left child
send (x_i, x_{i+2k}) to the right child

Leaf processor(i):

Reduction computations

send (a_i, b_i, c_i, y_i) to the parent

Backsubstitution

receive (x_{i-1}, x_{i+1}) from the parent
$x_i := (y_i - a_i x_{i-1} - c_i x_{i+1})/b_i$

7.3. A boolean *k*-cube algorithm. {Let $(g_k g_{k-1} \cdots g_1)$ be the address of the processor in a k-cube, let $G_m(k-j)$ be the $(k-j)$-bit binary-reflected Gray code of m, and let $j+1$ denote reduction step.}

$j := 0$
$g_0 := 1$
$x_{-1} := 0$
$x_K := 0$
while $g_j = 1$ and $j < k-1$ do
  $m := dec(g_k g_{k-1} \cdots g_{j+1})$ (m is the integer encoded by $k-j$ highest order bits $(G_m(k-j)111)$)
  $G_{m+1}(k-j) := enc(m+1)$ (address of the processor holding the succeeding integer in the $(k\text{-}j)$-cube)
  $G_{m-1}(k-j) := enc(m-1)$ (address of the processor holding the preceding integer in the $(k\text{-}j)$-cube)

{even processors in the $(k-j)$-subcube send their equations to the processors holding preceding and succeeding odd integers, except that the processor holding row $m=0$ only communicates with the processor holding the row $m=1$, and the processor holding the row $m = 2+k-j-2$ only communicates with the processor holding row $m-1$}
{processors holding odd integers communicate and perform reduction computations}

if m even and $m > 0$ then
  send (a_i, b_i, c_i, y_i) to the processor $(G_{m-1}(k-j)11 \cdots 1)$
elseif m even and $m < 2^{k-j} - 2$ then
  send (a_i, b_i, c_i, y_i) to the processor $(G_{m+1}(k-j)11 \cdots 1)$
elseif m odd and $m \neq 2^{k-j} - 1$ then
  receive $(a_{i-2^j}, b_{i-2^j}, c_{i-2^j}, y_{i-2^j})$ from processor $(G_{m-1}(k-j)11 \cdots 1)$
  receive $(a_{i+2^j}, b_{i+2^j}, c_{i+2^j}, y_{i+2^j})$ from processor $(G_{m+1}(k-j)11 \cdots 1)$
  $e_i := -a_i/b_{i-k}$
  $f_i := -c_i/b_{i+k}$
  $a_i := e_i a_{i-k}$
  $c_i := f_i c_{i+k}$
  $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
  $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
endif

{exchange data so that equations needed for the next reduction step are in the next lower dimensional subcube (with yet another address bit 1). No exchange is needed

when the cube is reduced to a 2-cube}
    if $j < k-2$ then
      if m is odd and $g_{j+1} = 0$ then
        send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 111 \cdots 1)$
        receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 111 \cdots 1)$
      endif
      if m is even and $g_{j+1} = 1$ then
        send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
        receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
      endif
    endif
    $j := j+1$
  enddo
{the reduction is now completed for equation i. j is the index of the last reduction step in which the processor participated. The reduction phase is complete when $j = n-1$ for a subcube of dimension 2.}

{The backsubstitution starts with processor $(0111 \cdots 1)$ computing $x_{2^{k-1}}$. Processors $(0011 \cdots 1)$ and $(1111 \cdots 1)$ then compute $x_{2^{k-2}-1}$ and $x_{3*2^{k-2}-1}$, respectively. For computation of additional unknowns it is necessary successively to increase the dimensions of the cube, and make the proper exchanges reversing the exchanges in the reduction phase.}
  while $j \geqq 0$ do
{Solve for $x_{2^{k-1}-1}$, $x_{2^{k-2}-1}$ and $x_{3*2^{k-2}-1}$ in the 2-cube of the final step of the reduction phase}
    if $j = k-1$ then
{solve for $x_{2^{k-1}-1}$ in processor $(0111 \cdots 1)$}
      if $m = 1$ then
        $x_i := y_i / b_i$
        send (x_{-1}, x_i) to processor $(G_{m-1} 11 \cdots 1)$
        send (x_i, x_K) to processor $(G_{m+1} 11 \cdots 1)$
      endif
{Solve for $x_{2^{k-2}-1}$}
      if $m = 0$ then
        receive $(x_{i-2^{k-2}}, x_{i+2^{k-2}})$ from processor $(G_{m+1} 11 \cdots 1)$
        $x_i := (y_i - a_i x_{i-2^{k-2}} - c_i x_{i+2^{k-2}}) / b_i$
      endif
{Solve for $x_{3*2^{k-2}-1}$}
      if $m = 2$ then
        receive $(x_{i-2^{k-2}}, x_{i+2^{k-2}})$ from processor $(G_{m-1} 11 \cdots 1)$
        $x_i := (y_i - a_i x_{i-2^{k-2}} - c_i x_{i+2^{k-2}}) / b_i$
      endif
    $j := j-1$
    endif
{Reverse the exchange of equations that occurred during the reduction phase}
    $j := j-1$
    $m := dec(g_k g_{k-1} \cdots g_{k-j})$
    $G_{m-1} := enc(m-1)$
    $G_{m+1} := enc(m+1)$

if m is odd and $g_{j+1}=0$ then
send (a_i, b_i, c_i, y_i) to the processor $(g_k g_{k-1} \cdots g_{j+2} 11 \cdots 1)$
received $(a_i, b_i, c_i, y_i, x_i)$ from the processor $(g_k g_{k-1} \cdots g_{j^2} 11 \cdots 1)$
endif
if m is even and $g_{j+1}=1$ then
send $(a_i, b_i, c_i, y_i, x_i)$ to the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
receive (a_i, b_i, c_i, y_i) from the processor $(g_k g_{k-1} \cdots g_{j+2} 011 \cdots 1)$
endif
{Processors with m even receives $x_{i-2^{j+1}}$ from processor $(G_{m-1}(k-j)11 \cdots 1)$, except if $m=0$. Processors with m even receives $x_{i^{2^{j+1}}}$ from processor $(G_{m+1}(k-j)11 \cdots 1)$, except if $m=2^{k-j}-2$}
if m is odd and $m<2^{k-j}-1$ then
send (x_i) to processor $(G_{m-1}11 \cdots 1)$
send (x_i) to processor $(G_{m+1}11 \cdots 1)$
endif
if m is even then
if $m>0$ then receive $(x_{i-2^{j+1}})$ from processor $(G_{m-1}(k-j)11 \cdots 1)$
if $m<2^{k-j}-2$ then receive $(x_{i+2^{j+1}})$ from processor $(G_{m+1}(k-j)11 \cdots 1)$
$x_i := (y_i - a_i x_{i-2^{j+1}} - c_i x_{i+2^{j+1}})/b_i$
endif
enddo

Acknowledgments. This report has benefited from many stimulating discussions with and suggestions by Stanley C. Eisenstat. Michael J. Fischer suggested the binary tree embedding in the shuffle-exchange networks. Andrea Pappas and Chris Hatchell helped with the manuscript.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[2] S. N. BHATT AND F. T. LEIGHTON, *A framework for solving* VLSI *graph layout problems*, J. Comput. System Sci., 28 (1984), pp. 300-343.
[3] R. P. BRENT AND H. T. KUNG, *On the area of binary tree layouts*, Inform. Process. Lett., 11 (1980), pp. 44-46.
[4] S. A. BROWNING, *The Tree Machine: A Highly Concurrent Computing Ennvironment*, Technical Report 1980:TR:3760, Computer Science, California Institute of Technology, Pasadena, CA, January 1980.
[5] B. L. BUZBEE, G. H. GOLUB AND C. W. NIELSON, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627-656.
[6] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Applications of an element model for Gaussian elimination*, in Sparse Matrix Computations, Academic Press, New York, 1976, pp. 85-96.
[7] M. J. FLYNN, *Very high-speed computing systems*, Proc. of the IEEE, 12, 1966, pp. 1901-1909.
[8] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th ACM STOC, 1978, pp. 114-118.
[9] D. GANNON AND J. VAN ROSENDALE, *On the impact of communication complexity in the design of parallel numerical algorithms*, IEEE Trans. Computers, C-33/12 December 1984, pp. 1180-1194.
[10] W. M. GENTLEMAN, *Some complexity results for matrix computations on parallel processors*, J. ACM, 25 (1978), pp. 112-115.
[11] A. GEORGE AND W. H. LIU, *Algorithms for matrix partitioning and the numerical solution of finite element systems*, SIAM J. Numer. Anal., 15 (1978), pp. 297-327.
[12] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[13] A. GEORGE, *Nested dissection of a regular finite element grid*, SIAM J. Numer. Anal., 10 (1973), pp. 345-363.

[14] A. GOTTLIEB, R. GRISHMAN, C. P. KRUSKAL, K. P. MCAULIFFE, L. RUDOLPH AND M. SNIR, *The NYU ultracomputer—designing an MIMD shared memory parallel computer*, IEEE Trans. Computers, C-32/2 (1983), pp. 175-189.

[15] D. HELLER, D. K. STEVENSON AND J. F. TRAUB, *Accelerated iterative methods for the solution of tridiagonal linear systems on parallel computers*, J. ACM, 23 (1976), pp. 636-654.

[16] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[17] W. D. HILLIS, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[18] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, 1981.

[19] R. W. HOCKNEY, *A fast direct solution of Poisson's equation using Fourier analysis*, J. ACM, 12 (1965), pp. 95-113.

[20] ———, *The potential calculation and some applications*, Meth. Comput. Phys., 9 (1970), pp. 135-211.

[21] I. C. P. IPSEN, Y. SAD AND M. H. SCHULTZ, *Complexity of Dense Linear System Solution on a Multiprocessor Ring*, Technical Report RR YALEU/DCS/RR-349, Dept. of Computer Science, Yale University, New Haven, CT, January 1985.

[22] B. M. IRONS, *A frontal solution program for finite element analysis*, Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 5-32.

[23] S. L. JOHNSSON, C.-T. HO AND F. SAIED, *Fast linear algebra routines on hypercubes*, in Parallel Processing and Medium Scale Multiprocessors, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[24] S. L. JOHNSSON, Y. SAAD AND M. H. SCHULTZ, *Alternating Direction Methods on Multiprocessors*, Technical Report YALEU/CSD/RR-382, Dept. of Computer Science, Yale University, New Haven, CT, August 1985. Commun. Appl. Numer. Meth., to appear.

[25] S. L. JOHSSON, *Gaussian Elimination on Sparse Matrices and Concurrency*, Technical Report 4087:TR:80, Caltech Computer Science Department, December 1980.

[26] ———, *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations*, Technical Report YALE/CSD/RR-339, Dept. of Computer Science, Yale University, New Haven, CT, October 1984.

[27] ———, *Cyclic reduction on a binary tree*, Comput. Phys. Comm., 37 (1985), pp. 195-203.

[28] ———, *Fast Banded Systems Solvers for Ensemble Architectures*, Technical Report YALEU/CSD/RR-379, Dept. of Computer Science, Yale University, New Haven, CT, March 1985.

[29] ———, *Solving Narrow Banded Systems on Ensemble Architectures*, ACM TOMS, 11/3 November (1985), pp. 271-288. Also available as Report YALEU/CSD/RR-418, November 1984.

[30] ———, *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures*, Technical Report YALEU/CSD/RR-367, Dept. of Computer Science, Yale University, New Haven, CT, February 1985.

[31] ———, *Band matrix systems solvers on ensemble architectures*, in Algorithms, Architecture, and the Future of Scientific Computation, University of Texas Press, 1985. Technical report YALEU/CSD/RR-388, Dept. of Computer Science, Yale University, New Haven, CT.

[32] ———, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures*, Technical Report YALEU/CSD/RR-361, Dept. of Computer Science, Yale University, New Haven, CT, January 1985.

[33] R. KAPUR AND J. C. BROWNE, *Block tridiagonal system solution on reconfigurable array computers*, in International Conference on Parallel Processing, IEEE Computer Society, 1981, pp. 92-99.

[34] D. Kershaw, *Solution of single tridiagonal linear systems and the vectorization of the* ICCG *Algorithm on the* CRAY-1, in Parallel Computations, Academic Press, New York, 1982, pp. 85-92.

[35] D. H. LAWRIE AND A. H. SAMEH, *The computational and communication complexity of a parallel banded system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185-195.

[36] F. T. LEIGHTON, *Complexity Issues in* VLSI: *Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press, Cambridge, MA, 1983.

[37] C. E. LEISERSON, *Area-Efficient* VLSI *Computation*, MIT Press, Cambridge, MA, 1982.

[38] P. LI AND L. JOHNSSON, *The Tree Machine: An evaluation of program loading strategies*, in 1983 International Conference on Parallel Processing, IEEE Computer Society, August 1983, pp. 202-205.

[39] RICHARD J. LIPTON, DONALD J. ROSE AND ROBERT ENDRE TARJAN, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346-358.

[40] S. PARTER, *The use of linear graphs in Gaussian elimination*, SIAM Rev., 3 (1961), pp. 119-130.

[41] M. S. PATERSON, W. L. RUZZO AND L. SNYDER, *Bounds on minimax edge length for complete binary trees*, Proc. of the 19th Annual Symposium on the Theory of Computing, ACM, 1981, pp. 293-299.

[42] C. S. PESKIN, *Ultracomputer Implementation of Odd-Even Cyclic Reduction*, Technical Report Ultracomputer note #19, Dept. of Computer Science, New York University, New York, January 1981.

[43] H. A. PRESNELL AND R. P. PARGAS, *Communication along shortest paths in a tree machine*, Proc. of the 1981 Conference on Functional Programming Languages and Computer Architecture, ACM, 1981, pp. 107-114.

[44] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[45] E. REITER AND G. RODRIGUE, *An incomplete Cholesky factorization by a matrix partitioning algorithm*, in Elliptic Problem Solvers II, Academic Press, New York, 1983, pp. 161-174.

[46] D. J. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computations, Academic Press, New York, 1973, pp. 183-217.

[47] A. L. ROSENBERG AND L. SNYDER, *Bounds on the costs of data encodings*, Math. Syst. Theory, 12 (1978), pp. 9-39.

[48] W. L. RUZZO AND L. SNYDER, *Minimum edge length planar embeddings of trees*, in VLSI Systems and Computations, Computer Sciences Press, 1981, pp. 119-123.

[49] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. ACM, 25 (1978), pp. 81-91.

[50] J. T. SCHWARTZ, *Ultracomputers*, ACM Trans. on Programming Languages and Systems, 2 (1980), pp. 484-521.

[51] C. LUTZ, S. RABIN, C. L. SEITZ AND D. SPECK, *Design of the mosaic element*, Proc., Conference on Advanced Research in VLSI, Artech House, 1984, pp. 1-10.

[52] C. L. SEITZ, *Concurrent* VLSI *Architectures*, IEEE Trans. Comp., C-33/12 (1984), pp. 1247-1265.

[53] M. C. SEJNOWSKI, E. T. UPCHURCH, R. N. KAPUR, D. P. S. CHARLU AND G. J. LIPOVSKI, *An overview of the Texas reconfigurable array computer*, Proc. National Computer Conference, IEEE, 1980, pp. 631-641.

[54] M. SEKANINA, *On an ordering of the set of vertices of a connected graph*, Publication of the Faculty of Science of the University of Brno, 412 (1960), pp. 137-142.

[55] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-20 (1971), pp. 153-161.

[56] C. D. THOMPSON, *Area-time complexity for* VLSI, Proc. of the 11th ACM Symposium on the Theory of Computing, ACM, 1979, pp. 81-88.

[57] ———, *A Complexity Theory for* VSLI, Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[58] H. H. WANG, *A parallel method for tridiagonal equations*, ACM Trans. Math. Software, 7 (1981), pp. 170-183.