Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution of Tridiagonal Systems of Equations

Lennart Johnsson

YALEU/CSD/RR-339
October 1984
Office of Naval Research contract N00014-84-K-0043

Table of Contents

List of Figures

List of Tables

Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution of Tridiagonal Systems of Equations

S. Lennart Johnsson
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract

The solution of tridiagonal systems by cyclic reduction and a combination of Gaussian elimination and cyclic reduction, (GECR), on a few ensemble architectures are devised and analyzed. The ensembles have no global storage and no global control. Synchronization is accomplished via message passing to neighboring processors. The processors of the ensembles are interconnected to form one of the following configurations; binary tree, shuffle-exchange network, perfect shuffle, boolean k-cube, or 2-dimensional mesh. Algorithms are first devised for one equation per processor, then generalized to multiple equations per processor. It is shown that the benefits of truncating the reduction process in a highly concurrent system is much greater than on a uniprocessor.

Domain decomposition is used to map N>K equations on to K processors, and is compared with cyclic partitioning, and for a binary tree, mapping by contraction. The parallel arithmetic complexity of cyclic reduction and GECR is $O(N/K+\log_2 K)$. The maximum number of communications in sequence for a partition is $O(\log_2 N)$ for cyclic reduction and $O(\log_2 K)$ for GECR. The speed-up increases monotonically with the number of processing elements for the tree, and the shuffle-exchange network. For the perfect shuffle and the k-cube there is an optimum speed-up at $K \approx N/(1+\alpha)$, where $\alpha \approx$ (time to communicate one floating-point number)/(time for a floating point arithmetic operation). For the 2-dimensional mesh the maximum speed-up is attained at $K \approx ((N/2\alpha)^2)^{1/3}$, and for the linear array it is attained at $K \approx (N/\alpha)^{1/2}$. The minimum time complexity is of order $O(\log_2 N)$ for the tree, the shuffle-exchange, the perfect shuffle and the boolean k-cube, of order $O(N^{1/3})$ for the mesh, and $O(N^{1/2})$ for the linear array.

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.

The binary tree, the shuffle-exchange, the perfect shuffle, and the k-cube ensembles consisting of processors with local storage allow for more efficient implementations than an ensemble consisting of processors connected to storage modules via an Ω-network. The difference is a factor of $O(\log_2 K)$ in the communication complexity.

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 fit on a single chip in state-of-the-art technology, [Seitz et.al. 84]. 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 ensemble architectures consisting of a large number of identical processing elements sparsely and regularly interconnected a viable alternative to large mainframe computers. A high nominal performance is attained in an ensemble architecture of the kind we envisage by using a large number of processing elements built in standard technology.

In a single processor mainframe computer, high performance is achieved by using advanced, expensive technology. On the order of 50 - 100 processing elements (or more) in standard technology may be required to match the nominal performance of a single processor in advanced technology. The cost of a processing element produced in large numbers in a mature technology is several orders of magnitude less than the cost of a processor produced in small numbers in a technology pushing state-of-the-art. The same nominal performance for a multiprocessor system designed in standard technology can be obtained at a cost one to two orders of magnitude less than in a single processor using advanced technology.

In the ensemble architectures we consider, the processing elements with their storage are interconnected as a binary tree, a shuffle-exchange network, a perfect shuffle, a boolean k-cube, or a 2-dimensional mesh. (A linear array algorithm is used as part of the algorithm for the 2-dimensional mesh). These interconnection schemes have different characteristics such as wiring complexity, extensibility, communication capabilities between arbitrary pairs of nodes, programming, and program loading. For a feasibility evaluation of these interconnection schemes, it is also 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 odd-even cyclic reduction on these ensemble architectures. Before presenting these algorithms we briefly discuss some of the hardware aspects of the different interconnection schemes.

Ensembles of processors interconnected as binary trees scale particularly well with the technology. The structure is recursive. Furthermore, arbitrarily sized trees can be constructed out of chips containing subtrees of a given size with a fixed interchip connection requirement. Four off chip interconnections are required [Leiserson 82], regardless of what size subtree fits on a chip, and what size tree is being constructed. For shuffle-exchange networks and 2-dimensional meshs the number of interconnections per processor (3 and 4 respectively) is independent of the network size, whereas in the boolean k-cube the number of interconnections is equal to the dimension of the cube. For a K processor ensemble there is a total of K-1 interconnections in a binary tree, $\approx$1.5K in a shuffle-exchange network, $\approx$2K in a perfect shuffle network, 2K in a 2-dimensional mesh, and $(K/2)\log_2 K$ in a boolean k-cube.

In extending a network with additional processors no rewiring of the existing network is necessary for a 2-dimensional mesh and a boolean k-cube. However, in the latter, additional interconnections per processor are needed. With K processors fitting in a planar region (which we refer to as a chip) the number of off chip interconnections is proportional to $\sqrt{K}$ for a 2-dimensional mesh, and grows at that rate as the feature sites of the technology is scaled down, and more processors fit on a chip. For a boolean k-cube the number of off chip connections is proportional to the number of processors per chip and to the number of dimensions implemented off chip. Hence, binary trees have features that scale well with the technology.

High nominal performance as well as high storage bandwidth is relatively easily accomplished in an ensemble architecture. The storage can be viewed as being highly partitioned. The number of partitions equals the number of processors in the ensemble, with each processor having its own storage. This number is considerably higher than the number of partitions in a high performance mainframe computer, given that the ensemble and mainframe are of comparable nominal performance. Whereas in the mainframe any partition is accessible in the same number of cycles, this is not the case in an ensemble architecture of the kind considered here. In our computational model we assume that the access time is proportional to the number of interconnections between a processor requesting data and the processor storing the requested data. This additional structure coupled with the limited communication bandwidth of processor interconnections make the efficient utilization of an ensemble architecture often considerably more difficult than the efficient utilization of a von Neumann architecture.

In the ensemble architectures considered here, each processor has its own program store. The architectures are of the MIMD type in Flynn's classification [Flynn 66]. Synchronization is obtained via message passing. However, even if each processor executes its own code, it is often the case that only a few different types of code are used. There is a high degree of uniformity in many MIMD algorithms. The uniformity often 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. For binary trees recursive program loading can be efficiently employed, if there is a high degree of program uniformity across the tree [Li,Johnsson 83]. 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 significant if the ensemble serves as an attached processor and only a few problems are solved for each loading of the program.

Global communication requires a time proportional to the diameter of the network of interconnected processors. This time defines a lower bound for any computation requiring global communication. Another bound is defined by the time required for arithmetic operations. For example, any computation requiring global communication will, on a mesh, have a term in its time complexity proportional to the circumference of the mesh. Even if the diameter of the network is of the same order as the number of sequential steps of an algorithm on an ideal parallel machine with arbitrarily high storage bandwidth and no storage conflicts, it is not guaranteed that there exists an algorithm of comparable complexity on the ensemble.

In summary, we use a computational model in which processors with local storage are interconnected in one of several network forms. Each processor executes its own program, and synchronizes computations with neighboring processors via message passing. There is no global storage, no global controller. Global communication is accomplished as a sequence of interprocessor communications. Furthermore, we assume that computations and communication can take place concurrently, but that data can first be used in a computation or another communication during the first time step succeeding the one during which it was received. A processor is also assumed to be capable of serving all its ports during one time step. If only one port can be served during one time step, then the complexity estimates that are given below will at most increase by a constant factor. This is true also for our k-cube algorithms, since they only use two ports per time step.

The time for one interprocessor communication is in the following denoted t_c, the time for arithmetic operations t_a. Those time notions are used somewhat loosely. The main purpose is to present algorithms of a minimum order of complexity, and which, after a detailed optimization, should yield a very good performance. Such an optimization should take advantage of particular features of the node architecture. Hence, t_c may in some instances denote the time to communicate 2 values, in others, 8. Similarly t_a in some instances denotes the time to perform a reduction operation on one equation, in others, a backsubstitution operation. The number of data items communicated within the time t_c, and the number of arithmetic operations performed within the time t_a, are <u>independent of the problem size and the ensemble size</u>. We also assume that communication between any pair of directly interconnected processors can take place within time t_c. We ignore the fact that most of the ensembles we consider have some long interconnections when implemented in 2- or 3-dimensional space, whereas the 2-dimensional mesh ideally could be constructed with uniformly short wires.

Odd-even cyclic reduction [Buzbee,Golub,Nielson 70] for N equations proceeds in $\log_2 N$ steps. In an ideal parallel machine model of computation where all storage locations are equally distant, bandwidth between processors and storage is infinite, and access conflicts ignored, it is clear that the computations can be carried out in $O(\log_2 N)$ time with a sufficient number of processing elements.

For a full rank, irreducible, tridiagonal system every element of the solution vector depends on every element of the right hand side. Global communication is needed. Hence, for K processors interconnected as a binary tree, a shuffle-exchange network, a perfect shuffle network, and a boolean k-cube there is a lower bound of $O(\log_2 K)$ for the time complexity, implied by the communication, [Gentleman 78]. For K=N this bound is of the same order as the number of arithmetic steps in the ideal model. For the mesh of size $\sqrt{N} \times \sqrt{N}$ the corresponding bound is $O(\sqrt{N})$. However, even with networks of a diameter of $O(\log_2 N)$ it is not guaranteed that computations with complexity $O(\log_2 N)$ on an ideal parallel machine can be carried out in $\log_2 N$ time. Indeed, in the binary tree algorithm for odd-even cyclic reduction presented in [Presnell, Pargas 81] each step of the algorithm demands a communication time of $O(\log_2 N)$. Their algorithm is of complexity $O(\log_2^2 N)$.

In this report we first give a short description of the computational aspects of odd-even cyclic reduction. Then, we present a binary tree algorithm that is of complexity $O(\log_2 N)$. 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 k-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. An algorithm for a 2-dimensional mesh is presented last. An algorithm for linear arrays is included in the algorithm for the 2-dimensional mesh. All algorithms have entirely distributed control.

We give performance estimates for algorithms with one equation per processor, and for algorithms with several equations per processor. We also analyze the situation where multiple independent problems are to be solved, and the potential benefits from using parallel cyclic reduction, [Hockney, Jesshope 81]. The performance gain possible through truncated cyclic reduction is estimated. $N=2^n-1$ 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, and t_a the time for local data fetching and arithmetic per logic entity, t_c the time for interprocessor communication. The meaning of a logic entity will become clear when the cyclic reduction computation is analyzed below.

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 one equation per processor. The programs also illustrate the degree of program uniformity for cyclic reduction on the various ensembles.

2. Odd-Even Cyclic Reduction

A system of linear equations Ax = y, where A is tridiagonal with main diagonal elements b_i, nonzero subdiagonal elements a_i, and superdiagonal elements c_i can be presented in matrix vector form as:

$$\begin{array}{ccccccc} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot \\ & & & & & a_N & b_N \end{array} \quad \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \\ \\ \\ x_N \end{array} = \begin{array}{c} y_1 \\ y_2 \\ y_3 \\ \\ \\ \\ y_N \end{array}$$

Assuming A is of full rank and of dimension $N = 2^n-1$, odd-even cyclic reduction proceeds in a reduction phase succeeded by a backsubstitution phase. Using subscripts for equation number and superscripts to denote reduction and backsubstitution steps, cyclic reduction is defined by the following set of equations ([Hockney, Jesshope 81]).

Reduction

$$a_i^j = e_i a_{i-2^{j-1}}^{j-1}$$

$$c_i^j = f_i c_{i+2^{j-1}}^{j-1}$$

$$b_i^j = b_i^{j-1} + e_i c_{i-2^{j-1}}^{j-1} + f_i a_{i+2^{j-1}}^{j-1}$$

$$y_i^j = y_i^{j-1} + e_i y_{i-2^{j-1}}^{j-1} + f_i y_{i+2^{j-1}}^{j-1}$$

$$e_i = - a_i^{j-1}/b_{i-2^{j-1}}^{j-1}$$

$$f_i = - c_i^{j-1}/b_{i+2^{j-1}}^{j-1}$$

where $i = \{2^j, 2*2^j, 3*2^j, ..., 2^n-2^j\}$, for reduction steps $j = \{1,2,..., n-1\}$

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

$$x_{2^{n-1}} = y^{n-1}_{2^{n-1}}/b^{n-1}_{2^{n-1}}$$

$$x_i = (y^{j-1}_i - a^{j-1}_i x_{i-2^{j-1}} - b^{j-1}_i x_{i+2^{j-1}})/b^{j-1}_i$$

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

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 $j=m$, $|a_i^j|<\epsilon$ and $|c_i^j|<\epsilon$, where ϵ is an acceptable error bound (or the machine precision), 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 on the order of 10 - 20, independent of the size of the matrix.

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 [Hockney, Jesshope 81]. The tridiagonal system is extended with 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. A Computation Graph for Cyclic Reduction

In studying the mapping of computations on to a network of processing elements it is often useful to represent the data structure and the computations performed on it by a graph. Nodes correspond to data and computations, and directed edges to communication of data. Edges are directed from the source towards the sink. Describing the computational network as a graph with nodes corresponding to processing elements with local storage, and edges to interconnections, the problem of mapping a computation on to the network of processing elements becomes one of embedding one graph in another. Due to the dynamic character of the computation graph, the embedding problem is also dynamic in nature. A static map of nodes in the computation graph to processors in an ensemble is sometimes as efficient as a dynamic map, but this is not always true, as will be proved for the tree.

We represent the coefficients of one equation, the corresponding right hand side, and the unknown corresponding to the column of the diagonal element with one node in the computation graph. The nodes are labeled with the equation number, starting from 0. In the first reduction step every odd equation is modified using the immediately preceeding and succeeding even equations. In the second reduction step, every other odd equation is modified using the immediately preceeding and succeeding odd equations. The computation graph of odd-even cyclic reduction is shown in Figure 3-1. The edges are labeled with the reduction phase in which the information exchange occurs.

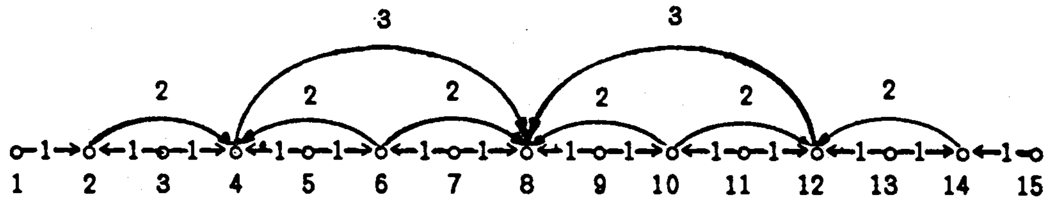


Figure 3-1: A graph representation of odd-even cyclic reduction

The focus in our derivations below is on the communication requirements. The objective is to present mappings that offer a communication complexity that is of the same order as the complexity of cyclic reduction on an ideal parallel machine with unlimited resources. A detailed optimization of operations should take advantage of particular architectural features of the processors in the ensemble. For our complexity estimates we assume that

- a processor can perform one communication on all of its ports concurrently with arithmetic operations

If only one port can be serviced at a time, then the order of the complexity would still be the same for ensembles with a fixed number of ports per processor regardless of the size of the ensemble. For cyclic reduction the same property is also true for our boolean k-cube algorithm. We first study mappings of nodes in the computation graph on to distinct processors in the ensembles. Such maps correspond to a highly concurrent system with one equation per processor. We also study mappings of sets of nodes in the computation graph on to processors in the ensembles. The granularity of computations in a node is increased, as well as the time to carry out the computations.

The fact that the manner in which a data structure is traversed by an algorithm is of great importance in finding efficient mappings is easily illustrated for cyclic reduction. A natural data structure for the tridiagonal matrix, the vector of unknowns, and the right hand side, is a Nx5 array in which successive equations are stored in successive rows of the array. Conceptually this array can be viewed as a one dimensional array. Mapping of the array to a computational network preserving proximity does not necessarily render algorithms of minimum complexity. For instance, a linear array can be mapped onto a binary tree such that the maximum distance between successive nodes of the linear array when embedded in the tree is 3 [Sekanina 60], [Rosenberg, Snyder 78] Figure 3-2. However, this embedding results in an algorithm of complexity $O(\log_2^2 N)$.

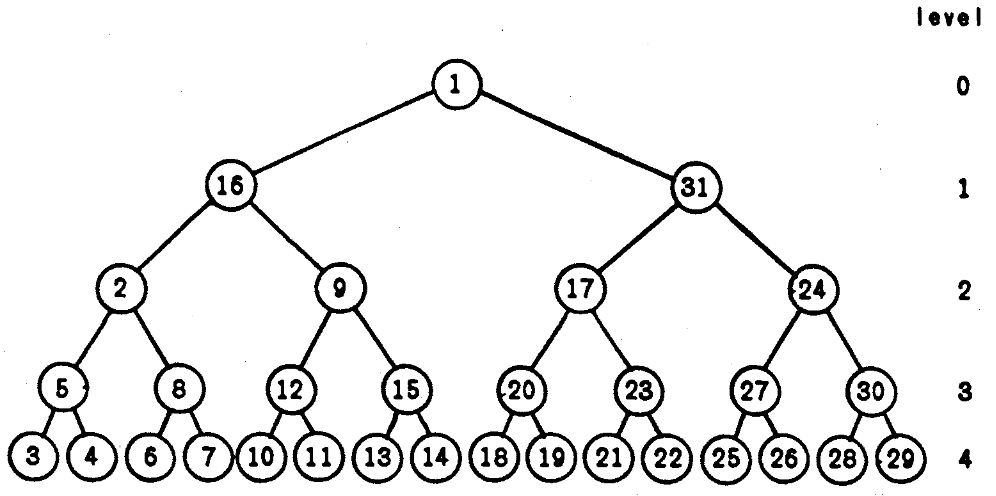


Figure 3-2: Loop embedding in a binary tree [Sekanina 60], [Rosenberg, Snyder 78]

4. Multiple equations per node

If the number of equations is greater than the number of processors, but the processors have sufficient storage to hold several equations, then sets of equations can be mapped into a single processor. Several such mappings are possible. In order to balance the computational load it is desirable to make the number of equations per node as equal as possible. However, such a partitioning is not sufficient to guarantee a uniform distribution of computations among partitions, as will be shown. The set of equation indices, $\{0,1,2,...,2^n-2\}$, is partitioned into K partitions, such that a partition is assigned either $\lfloor N/K \rfloor$ or $\lceil N/K \rceil$ equations. A processor in the ensemble architectures analyzed here is assigned one partition. The assignment may be static or dynamic, but we allow at most one partition per processor. The set of partitions is denoted $P=\{P_0, P_1, ..., P_{K-1}\}$. 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 $R=\{R_i\}$. Clearly, $P=Q \cup R$, and $Q \cap R=\emptyset$. The set of equations participating in reduction step j has indices in the set $M^j=\{(m_j+1)2^j-1\}$, $m_j=\{0,1,2,...,2^{n-j}-2\}$, $j=\{0,1,2,...,n-1\}$. $|M^j|=2^k-1$ after n-k steps.

We will now analyze two different partitionings of the equation index set: **consecutive** and **cyclic**. In the consecutive partitioning, partition P_i is assigned indices $\{p\}$, $iN/K \leq p < (i+1)N/K$. The case where N/K is not an integer is treated in some detail later. In cyclic partitioning P_i is assigned equations $i=p \bmod K$, $p=\{0,1,2,...,2^n-2\}$. Consecutive partitioning corresponds to the notion of *domain decomposition*, frequently used in the solution of partial differential equations. Because of this correspondence we will refer to the consecutive partitioning as domain decomposition.

We restrict the partitioning to be fixed during the course of computations. One of the questions that has to be answered in the partitioning process is whether or not it is always desirable to use as fine a partitioning as is possible with respect to the number of processors available. The answer to this question depends on the partitioning method, even for an ensemble in which the communication cost is negligable. We will show that the (parallel) arithmetic complexity can increase considerably with a small change in the number of partitions.

Two essential characteristics of the ensemble architectures are the limited number of processors with which a given processor can communicate, and the nonzero cost associated with each communication action. Figure 3-1 displays the communication requirements for N=K, i.e., one equation per partition. The efficient use of ensemble architectures requires a good match between the communication needs of the algorithms and the topology of the ensemble. The optimum size of the ensemble for a given problem depend on the topology as well as the relative costs of communication and arithmetic.

We will now analyze the parallel arithmetic complexity and interpartition communication needs for cyclic reduction with the set of equations partitioned cyclically and by domain decomposition. The communication complexity cannot be determined until a mapping on to a specific ensemble is made. For an optimal map it is necessary to properly match the interpartition communication needs with the communication capabilities of the ensemble. The arithmetic complexity is determined only by the partitioning and the assumption of one processing unit per partition. We state some of the properties of cyclic reduction on a partitioned equation set in terms of theorems for easy reference in subsequent sections.

4.1. Domain decomposition

There are several issues that need to be addressed. For instance, is it of any significance with respect to the total time for arithmetic and communication which partitions are assigned $\lfloor N/K \rfloor$ equations and which are assigned $\lceil N/K \rceil$ equations. Does the first n-k reduction steps yield one equation per partition regardless of which partitions belong to the sets Q and R? What is the required number of equation transfers between partitions in the first n-k reduction steps, and the last k-1 reduction steps? Is the communication unidirectional? If the direction of communication alternates then pipelining cannot be used to reduce the effect of communicating partitions being mapped to processors at more than unit distance apart.

The number of partitions assigned $\lceil N/K \rceil$ indices is clearly NmodK. 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 binary trees it is reasonable to assume that they are balanced, i.e., $K=2^k-1$ for some integer k. For shuffle networks and boolean cubes $K=2^k$.

Lemma 1. For $K=2^k-1$, $|R|$ is $2^{n\bmod k}-1$, and $|Q_i|=(\Sigma_{m=1}^{\lfloor n/k\rfloor} 2^{n-mk})$

Proof. $N=2^n-1=(\Sigma_{m=1}^{\lfloor n/k\rfloor} 2^{n-mk})(2^k-1)+2^{n\bmod k}-1$. Q.E.D.

Hence, for $n\bmod k=0$ all partitions are of equal size, and all partitions are assigned an odd number of indices. The first reduction step will change this perfect balance in the number of equations per partition that participate in the reduction. Let $E^j=\{E_i^j\}$, $i=0,1,2,\dots$ be the set of partitions in step j that have an odd number of equations participating in the reduction computations. Let $F^j=\{F_i^j\}$, $i=1,2,\dots$ be the partitions with an even number of equations participating in reduction step j.

Theorem 1. For $K=2^k-1$, $|E^j|=2^{k-(j-n)\bmod k}-1$, $j=0,1,\dots,n-1$, and $|E_i^j| - |E_i^{j+1}|\leq 1$.

Proof. The proof is by induction. Consider $j=0$. If $n\bmod k=0$ then $|R|=0$ and $|P_i|$ is the same for all i. $|P_i|$ is odd, since N is odd and K is odd. Hence, $|E^0|=|P|=K=2^k-1$ and the theorem holds.

If $n\bmod k\neq 0$, $j=0$, then by lemma 1 $|R|$ is odd and $|Q|$ is even. Furthermore, $|R_i|$ is odd and $|Q_i|$ is even, since N is odd. It follows that $E^0=R$ and $|E^0|=|R|=2^{n\bmod k}-1=|E^j|=2^{k-(j-n)\bmod k}-1$. Also $F^0=Q$ and $|F_i^j|=|E_i^j|-1$. Hence, the theorem is true for $j=0$.

Assume the theorem holds for $j>0$. Then there are two cases: $F^j=\emptyset$, or $F^j\neq\emptyset$. If $F^j=\emptyset$, then $|E^j|=K$, and every partition has an odd number of equations participating in step $j+1$. The participating equations have indices $M^j=\{(m_j+1)2^j-1\}$, $m_j=\{0,1,2,\dots,2^{n-j}-2\}$. The index of each participating equation is derived from the set m_j, and each partition has an equal number of indices from the set m_j. The number is odd. In step $j+1$ reduction computations are performed on equations corresponding to the odd indices in the set m_j, and it follows that $|E_{2l}^j|=a$, $l=0,1,\dots,(K-1)/2$, a odd, is reduced to $(a-1)/2$, and that $|E_{2l+1}^j|=a$, $l=0,1,\dots,(K-1)/2-1$ is reduced to $(a-1)/2+1$. Hence, an even number of partitions perform $(a-1)/2$ reductions, and an odd number of partitions $(a-1)/2+1$ reductions. It follows that $(a-1)/2$ is even since $|m_{j+1}|$ is odd, and $F^{j+1}=\{E_{2l}^j\}$, $l=0,1,\dots,(K-1)/2$, $E^{j+1}=\{E_{2l+1}^j\}$, $l=0,1,\dots,(K-1)/2-1$. Finally, $|E^{j+1}|=(|E^j|-1)/2=(2^{k-(j-n)\bmod k}-2)/2=2^{k-(j+1-n)\bmod k}-1$.

If $F^j\neq\emptyset$, then the partitions with indices in the set F^j have an even number of indices from the set m_j, and precisely half of the equations in those partitions are subject to reduction computations. Of the partitions in E^j, the first and every other one have subsets of indices from m_j starting and ending with even indices. Hence, for those partitions reduction computations is in step $j+1$ performed on $(a-1)/2$ equations where a is the cardinality of E^j. The number of such partitions is even, since $|E^j|$ is odd, and the first and every other partition perform the same number of reduction computations. Hence, $|E^{j+1}|=(|E^j|-1)/2$. Q.E.D.

Theorem 1 states that the difference between the number of equations subject to reduction operations is at most 1 during the reduction phase. The same condition is true for the backsubstitution phase, since in step j of that phase partition P_i computes $\{|E_i^j|$ or $|F_i^j|\}$- $\{|E_i^{j+1}|$ or $|F_i^{j+1}|\}$, j=n-1,n-2,...,0.

Theorem 1 also states that the number of partitions having an odd number of indices from the set m_j equals the number of nodes in a binary tree of height k-(j-n)modk-1.

Corollary 1. The initial partitioning of P into Q and R is of no consequence for the computational balance.

Corollary 2. If r=nmodk and the partitions numbered $2^{k-r}\{0,1,2,..., 2^{k-(-n)\mathrm{mod}k}-2\}$ initially are assigned $\lceil N/K\rceil$ equations, then in reduction step j the set of partitions E^j have indices $2^{(j-n)\mathrm{mod}k}\{0,1,2,...,2^{k-(j-n)\mathrm{mod}k}-2\}$.

The indices of the partitions in corollary 2 correspond to the labels of the nodes at levels 0 through k-(j-n)modk-1 of a binary tree of height k-1 labeled in inorder (see e.g. [Aho,Hopcroft,Ullman 74]), Figure 5-1.

For shuffle networks and boolean cubes the number of nodes is of the form 2^k. We will state the results corresponding to lemma 1 and 2 for $K=2^k$ before investigating the communication requirements between partitions.

Lemma 2. Partitioning P into $K=2^k$ sets of as equal cardinality as possible yields |R|=K-1, $|R_i|=2^{n-k}$, i={0,1,2,...,K-2}, and $|Q_1|=2^{n-k}-1$.

Theorem 2. For $K=2^k$ $|F^j|=K-1$ for j=0,1,2,..., n-k, $|F_i^j|=2^{n-k-j}$, i=0,1,...,K-2, and $|E_1^j|=2^{n-k-j}-1$.

Proof. The proof follows that of lemma 2.

Corollary 3. At reduction step n-k one partition perform no reduction computation, and K-1 partitions perform 1 such computation.

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

Theorem 3. In the first n-k reduction steps partition P_i only interact with partition P_{i+1} for i={0,1,2,...,K-2} and partition P_{i-1} for i={1,2,...,K-1}.

Proof. The number of indices assigned to a partition is at least $\lfloor N/K\rfloor$. The index difference between equations is increasing monotonically during the reduction process, and is 2^{n-k-1} for step n-k. Since the index set in each partition is consecutively ordered and $2^{n-k-1} \leq \lfloor N/K\rfloor$ the theorem follows. Q.E.D.

We will now consider the interpartition communication need for $K=2^k-1$. By theorem 1 it is only necessary to consider adjacent partitions during the first n-k steps. The direction of equation exchange between adjacent partitions changes during the reduction process. The number of changes is different for different partitions.

Theorem 4. In the set of partitions $P_s=2^{k-s-1}\{1,3,5,...,(2^{s+1}-1)\}$, $s=\{0,1,...,k-1\}$, $|P|=2^k-1$ there exist at least one partition which changes direction of exchange with one of its adjacent partitions s+1 times for every k reduction steps.

Proof. A set $P_i \in E^j \cap E^{j+1}$ requires one equation from P_{i-1} and P_{i+1} during reduction step j, since at step j partition P_i has a subset of indices of m_j that starts and ends with odd indices. A set $P_i \in E^j - E^{j+1}$ has equations needed by the adjacent partitions since such a partition has a subset of indices of m_j that starts and ends with even indices. Hence, a partition needs equations from its adjacent partitions in every reduction step for which it remains in the sequence of sets E^j, $j=0,1,2,...$ During the step at which a partition leaves this sequence of sets the direction of exchange is reversed.

A partition in the sequence of sets F^j exhibits several changes in the direction of exchange. Consecutive partitions preceeding a partition P_i such that $P_i \in E^j \cap E^{j+1}$ have a direction of exchange during step j from a partition to its succeeding partition. Similarly, consecutive partitions succeeding a partition $P_i \in E^j \cap E^{j+1}$ have a direction of exchange from a partition to its preceeding partition. The converse is true for partitions $P_t \in E^j - E^{j+1}$. They form pivoting points for the direction of communication. Let the set of consecutive partitions preceeding P_t be $P_{\{q\}}$, $P_{\{q\}} \subset F^j$, and the set of consecutive partitions succeeding P_t be $P_{\{r\}}$, $P_{\{r\}} \subset F^j$. In step j+1 either $P_{\{q\}}$ or $P_{\{r\}}$ has its direction of exchange reversed since the direction of exchange is the same for $P_{\{q\}} \cup P_t \cup P_{\{r\}}$ for steps $j+1, j+2, ..., j \bmod k=0$.

It remains to be shown that in the set P_s there exist at least one partition changing direction of exchange s+1 times. It is true for s=0 by lemma 2. The remainder of the theorem can be shown by induction. Q.E.D.

Corollary 4. The communication for successive reduction steps can only be partially pipelined.

Theorem 5. For $K=2^k$ the direction of exchange is constant during the first n-k steps.

Proof. The proof follows directly from theorem 2, since $F^j=R=K-1$ for $j=0,1,2,..., n-k$.

Corollary 5. If for $K=2^k-1$ each partition is assigned 2^{n-k+1} (or 2^{n-k}) consecutive indices, then the direction of communication is constant for the first n-k+1 (n-k) reduction steps. Each partition requires a total of $2^{n-k+1}-1$ (or $2^{n-k}-1$) reduction computations during these steps.

The benefit of modifying the number of equations 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.

4.2. Cyclic storage

Theorem 6. With K odd, $K \neq 1$, then only every Kth element of the sequence of sets M^j, $j=\{0,1,2,...,n+1-\log_2(K+1)\}$ is located in the same partition, and the balance of computation is as even as possible.

The proof is by contradiction.

Theorem 7. If $K=2^k$ then after k reduction steps, all indices in the sequence of sets M^j, $j=\{k+1,...,n-1\}$ are in the same partition

Proof. Immediate since the index difference in reduction computations is 2^j, $j=\{0,1,2,...,n-2\}$.

Corollary 6. The communication required in each reduction step is between partitions differing in index by $2^j \bmod K$. The number of such communications from each partition is $\lceil N/(2^jK)\rceil$ for $j=\{0,1,2,...,\log_2(N/K)\}$ if K is odd. For $K=2^k$ the communication per partition is $\lceil N/K \rceil$ for the first k steps, then all equations are in the same partition.

4.3. Comparison of domain decomposition and cyclic partitioning

The parallel arithmetic complexity of domain decomposition is $\Sigma_{j=1}^{n-1}|E^j|$ for the reduction phase and $|E^0|+k-1$ for the backsubstitution phase. Hence, the total arithmetic complexity is of the form $O(N/K+\log_2K)$ (approximately $(8+9NR)N/K+(5+8NR)\log_2Kt_a$) where NR is the number of right hand sides). The arithmetic complexity is the same for cyclic partitioning with K odd. For $K=2^k$ and cyclic partitioning the parallel arithmetic complexity is $O((N/K)\log_2K)$. i.e., considerably higher than for the other partitioning schemes.

The communication requirements for domain decomposition is $2(\log_2N-\log_2K)$ communications with adjacent partitions plus a total of $2(\log_2K-1)$ communications between partitions differing in indices by $1,2,...,2^{k-2}$. Using the cyclic partitioning with K odd yields 4N/K communications between adjacent partitions for the first n-k reduction steps and last backsubstitution steps. For the last k-1 reduction steps and first backsubstitution steps the communication need is the same as for domain decomposition. For cyclic partitioning and $K=2^k$ the total number of communications is $(N/K)\log_2K$ (N/K in each of $\log_2K$ steps) between partitions that differ in index by 1,2,...,K/2. l Clearly, domain decomposition is superior.

For cyclic reduction truncated after $m \leq n-k$ steps the parallel arithmetic complexity is reduced by $O(N/(2^m K))$ for domain decomposition and the communication complexity reduced to m sequential communications with adjacent partitions.

4.4. Domain decomposition with reduced communication complexity

The arithmetic complexity of domain decomposition as well as cyclic partitioning is of the desired order, i.e. $O(N/K+\log_2 K)$, which is the same as the lower bound for the solution of an irreducible linear system of equations. The communication complexity is $O(\log_2 N)$. By abandoning cyclic reduction for the first n-k reduction steps (and last n-k backsubstitution steps) it is possible to achieve a lower communication complexity.

The following algorithm obtained by combining Gaussian elimination and cyclic reduction, GECR, requires $2\log_2 K$ sequential communication actions between partitions with indices differing by 1,2,4,...,K/4. The algorithm proceeds in 4 phases. Phase 1, 4 are entirely local to a partition, and partition 2 requires one communication action between adjacent partitions. Steps 1 and 2 are based on Gaussian elimination, step 3 on cyclic reduction, and step 4 is a backsubstitution phase. Steps 1, 2, and 4 are the same as in the algorithm by Wang, [Wang 81]. Similar ideas are used in [Sameh 84].

The algorithm is:

- Phase 1: Eliminate the subdiagonals by Gaussian elimination, concurrently in all partitions.
- Phase 2: Eliminate the superdiagonal by Gaussian elimination, concurrently in all partitions, starting from the second last equation in each partition, and terminating with the last equation in the preceeding partition.
- Phase 3: Solve a tridiagonal system of K equations. The system is made up of the the last equation from each partition.
- Phase 4: Solve for the remaining variables concurrently in each partition.

In phase 1 fill-in occurs in column $i*N/K$ of partition i $i=\{1,2,...,K-1\}$. In the second phase fill-in is created in column $(i+1)N/K$ of partition i, $i=\{0,1,...,K-1\}$. The last equation of each partition in this system of equations form a tridiagonal system in the variables corresponding to the columns $(i+1)N/K$, $i=\{0,1,2,...,K-1\}$.

The appearance of the matrix after phase 2 is shown in Figure 4-1.

The arithmetic complexity of GECR is almost identical to that of cyclic reduction, i.e. 8 arithmetic operations for reduction on the matrix, 4 for each right hand side in the reduction process, and 5 per right hand side in the backsubstitution phase. The communication complexity of phases 1,2, and 4 is reduced from $2(\log_2 N-\log_2 K)$ to 1. The number of communication actions for phase 3 is $2(\log_2 K-1)$. For sufficiently fast convergence truncated cyclic reduction based on domain decomposition may be competitive with GECR.

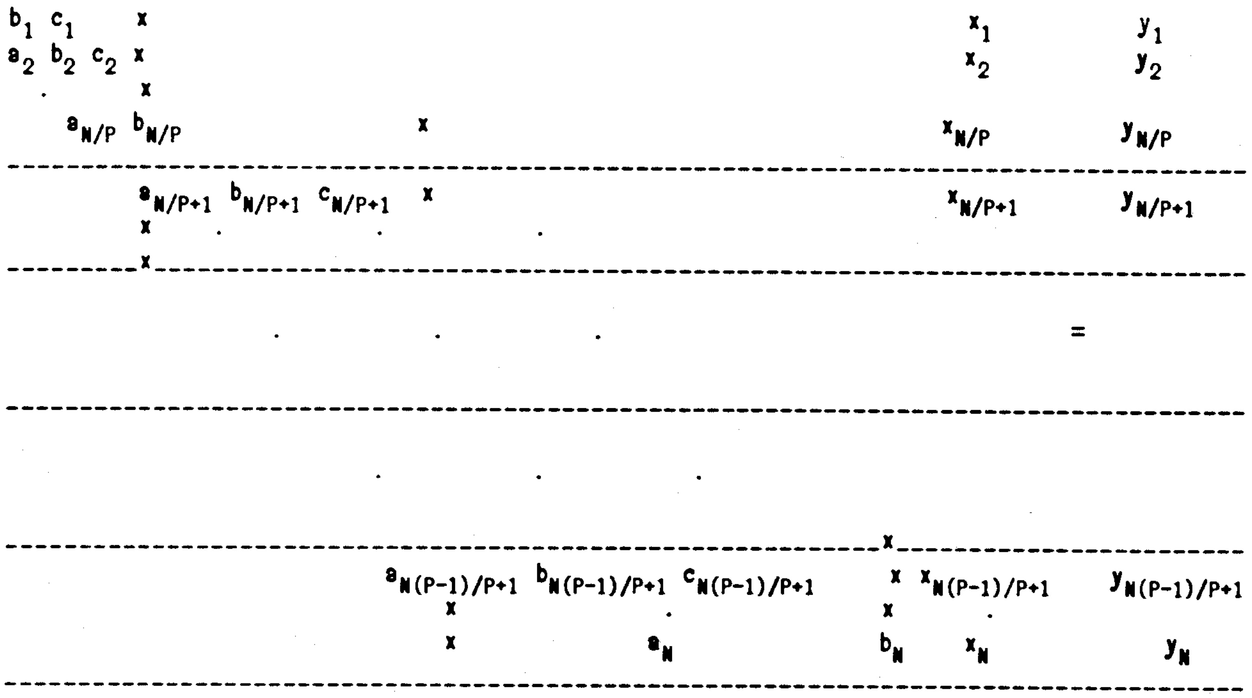


Figure 4-1: The matrix after phase 2 of GECR

4.5. A comparison of cyclic reduction and Gaussian elimination

The parallel arithmetic complexity of 2-way Gaussian elimination (performing elimination from both ends) is (3+5NR)(K-3)/2+5+6NR, and that of cyclic reduction is (5+8NR)(K$\log_2$-2)+5+6NR where NR is the number of right hand sides. In this complexity estimate for cyclic reduction two communication actions per reduction step are required. If the overhead for communication is large, then it might be beneficial to accept an arithmetic complexity of (8+9NR)($\lceil$K$\rceil$-2)+5+6NR, which can be achieved with one communication action per reduction step. The parallel arithmetic complexity of cyclic reduction is always lower than that of Gaussian elimination. The communication complexity of 2-way Gaussian elimination is (1+NR)(K-1). The communication complexity of cyclic reduction depends on the communication capabilities of the ensemble. For a linear array it is (1+NR)(5/4K-7/4), and for a shuffle network it is (1+NR)(2+3($\log_2$K-2)), $K \geq 7$. On a shuffle network the communication complexity of cyclic reduction is always less than that of Gaussian elimination. For K<7 the communication complexity as well as the arithmetic complexity of cyclic reduction and 2-way Gaussian elimination are the same.

For a linear array the number of processors for which cyclic reduction becomes competitive with Gaussian elimination depends on the cost of communication relative to arithmetic. For $t_c=t_a$ cyclic reduction is always preferable regardless of the number of right hand sides. For $t_c=10t_a$ Gaussian elimination is always more efficient on a linear array, regardless of the number of right hand sides. For $t_c=5t_a$ Gaussian elimination has a lower total complexity for $K\leq15$, independent of the number of right hand sides. For $K\geq63$ cyclic reduction is always of lower complexity, regardless of the number of right hand sides. The choice of method for a linear array is critically dependent upon the relative communication cost.

Gaussian elimination becomes increasingly competitive if multiple independent problems are to be solved. In such a case the computations for independent problems are easily pipelined without conflicts using Gaussian elimination. Full advantage can be taken of its lower arithmetic complexity per step, 3+2NR in the elimination phase and 2NR in the backsubstitution phase, compared to 5+8NR n the reduction phase and 5NR in the backsubstitution phase for cyclic reduction. Furthermore, in the algorithm using cyclic reduction one processor is used $\log_2K$ cycles. Hence, if the problems are mapped on to the linear array in the same way, then the solution of a new problem can be initiated only every $\log_2K$ cycles. Even if the linear array has an end-around connection, i.e., it is a ring, and different problems are mapped on to the ring such that interleaving of computations is successful, Gaussian elimination would still be more effective for a sufficiently large number of problems due to its lower arithmetic complexity.

4.6. Multiple independent problems

With multiple independent problems the solution to successive problems can either be pipelined, or the partitions be made larger so that the total number of partitions for all problems remain constant. This strategy corresponds to partitioning the ensemble into subsets of processing elements, each subset treating a separate problem.

Theorem 8. Pipelining the computations of multiple independent problems mapped to processors statically in the same way yields a higher parallel arithmetic complexity than partitioning the ensemble.

Proof. Since all partitions need to perform computations until the problem is reduced to K equations it suffices to consider the case N=K. Partition 2^{k-1} is subject to reduction computations during $\log_2K$ steps. Hence, if the map is identical for all problems there is a separation of $\log_2K$ steps between problems if they are pipelined. The total complexity for P problems is $O(P\log_2K)$. The arithmetic complexity for partitioning the ensemble is $O(\log_2K+P-\log_2P)$, P=1,2,...,K. Q.E.D.

The effect on the communication complexity is not as easy to establish. Clearly, for pipelining statically and identically mapped problems the communication complexity grows in proportion to the number of problems solved. Partitioning results in multiple equations per processor, and a different map may be preferable. In such a case, the topology of the ensemble will affect the way in which the communication complexity changes. However, the following holds:

Theorem 9. Let the communication complexity for partitioning the ensemble of K processors for P problems be $CC_{pa}(K,P)$, and the complexity for pipelining the problems be $CC_{pi}(K,P)$. Then, $CC_{pa}(K,P) \rightarrow 0$ $P \rightarrow K$, and $CC_{pi}(K,P) = P\alpha \log_2 K + \beta$.

From theorems 8 and 9 it follows that for $P<K$, and P sufficiently large partitioning the ensemble is always more effective than pipelining. The particular value of P for which the potential equality holds varies with the topology of the ensemble and the mapping used.

5. Cyclic reduction on a binary tree of processors

Binary trees are often used in the analysis of algorithms and have good properties from a construction point of view. Several ensemble architectures with tree interconnected processors have been proposed, and are in various stages of design and construction. The Caltech tree machine [Browning 80] [Browning,Seitz 80] [Seitz et.al. 84], is a MIMD architecture with each node planned as a 16-bit microprocessor. The NON-VON machine of Columbia University [Shaw 82], is an ensemble of 8-bit processing elements operating on instructions broadcast from a control processor attached to the root of the tree, i.e., it is an SIMD machine. NON-VON-4 is perceived as a MIMD architecture [Shaw 84]. Mago has investigated a reduction machine using tree interconnected processors [Mago 79]. The TRAC (Texas Reconfigurable Array Processor) [Sejnowski et.al. 80], is a MIMD ensemble architecture, but of a different kind than the ones investigated here in that the processors have insignificant local storage, and processors and storage are at opposite sides of a switch network.

5.1. One equation per node

The data structure is mapped only on to the leaf nodes in many tree algorithms. Intermediate nodes perform communication and computations of partial and/or final results. This form of mapping is employed in [Presnell, Pargas 81] yielding a cyclic reduction algorithm of complexity $O(\log_2^2 K)$ $(N=K)$.

From the computation graph of cyclic reduction it is clear that 1 node communicates with $2(\log_2 K-1)$ distinct nodes, 2 nodes communicate with $2(\log_2 K - 2)$ nodes, etc. No mapping of the graph in Figure 3-1 on to a binary tree exists such that proximity is preserved.

Theorem 1 and corollary 2 suggest an inorder map of the equations on to the nodes of the binary tree, see Figure 5-1. Indeed, an inorder map allows cyclic reduction to be performed in $O(\log_2 K)$ time. In the inorder map we identify equation index i with node index i+1.

Corollary 7. The effective tree height is reduced by 1 per reduction step.

Corollary 7 follows from theorem 1 and corollary 2.

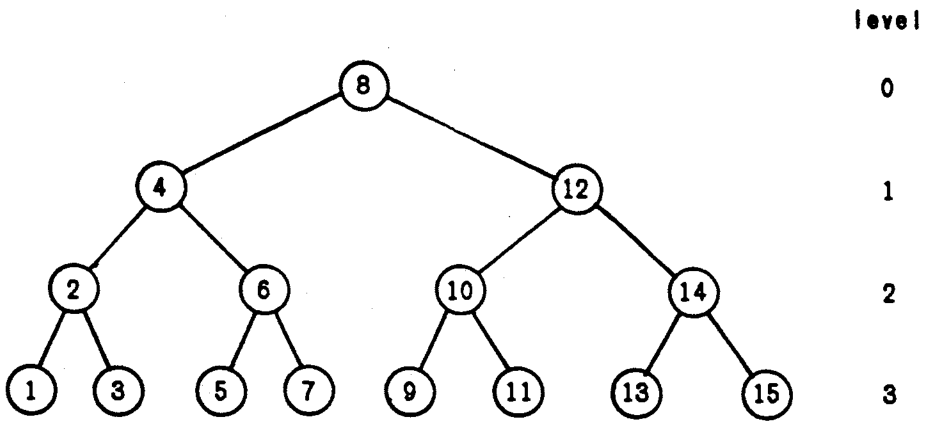


Figure 5-1: Inorder labeling of a binary tree

The first step of the cyclic reduction algorithm requires k time steps. In an inorder labeling of a binary tree, node 2^k is at distance k from the nodes $2^k \pm 1$. The required communication paths for reduction step 1 are indicated in Figure 5-2. The tree edges forming the path from the leftmost and rightmost leaf nodes to the root need only transmit one equation. Edges connecting leaf nodes with their respective parent node also need only transmit one equation. All other edges have to transmit two equations in the first reduction step. The reduction computations are carried out in the nodes that store the equations subject to modification. Each subsequent reduction step requires only one additional time step, since the inorder mapping makes possible pipelining of successive reduction steps. The total number of time steps for the reduction phase is 2k. The back substitution also requires 2k time steps.

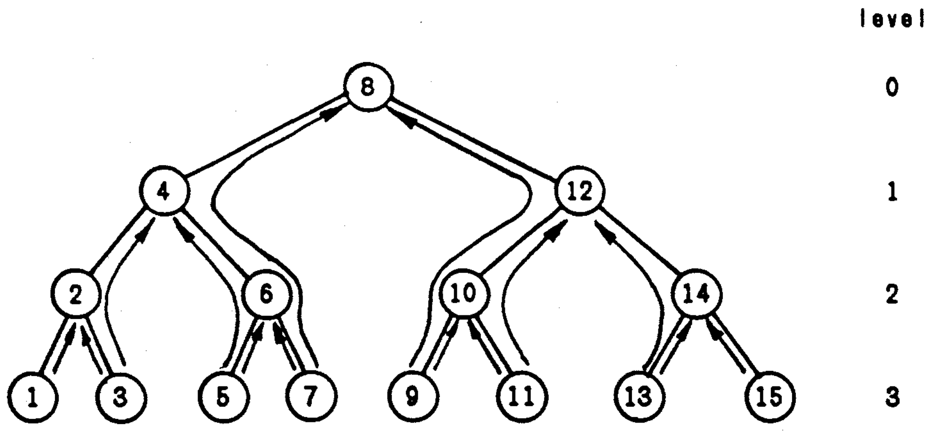


Figure 5-2: Communication for reduction step 1

In order to make the behavior of the algorithm clear we will now describe a version that distinguishes only between the root, the intermediate level nodes, and the leaf nodes. The nodes on the path from node 1 and node N to the root are executing the same code as other intermediate level nodes for reasons of simplicity and clarity. No loss of performance occurs.

The time for the additional communication is masked by the time required for communication in other parts of the tree. A concise description in the form of pseudo code is contained in the Appendix.

The algorithm starts and terminates in the leaf processors. They send their equations to their respective parent processors. These nodes (at level k-2 from the root) send the two equations they receive to their parents, and perform the computations for reduction step 1 on the equation they store. The processors at level k-3 receive two equations from each of their two children processors. One of the equations a processor receives from each of its children processors is to be forwarded to its own parent processor, one is used in reduction step 1 for the equation the processor stores. Hence, processors at level k-3 send two equations to their parents, precisely as nodes at level k-2. After k-1 steps equations labeled $2^{k-1}\pm1$ have propagated to the root. Assuming that a data item can first be used during the time step following the one during which it was received one additional step is needed to complete reduction step 1. The leaf nodes do not participate in reduction step 2. The effective tree height is reduced by 1 per reduction step.

By pipelining the reduction steps a new reduction step can be initiated every other time step. Associating a wave front with each reduction step, wave fronts are spaced one level apart. There are a maximum of $\lfloor k/2 \rfloor$ wave fronts in the tree during the reduction phase. Pipelining occurs naturally in a system in which synchronization is accomplished via message passing. As soon as a processor has finished the communication actions, and the computations associated with a reduction step, it proceeds to the next reduction step. If the partner processor in a communication is not ready to participate, the requesting processor has to wait until the partner is ready.

On completion of the last reduction step the root processor computes variable $x_{2^{k-1}}$ and sends it to its two children processors. Those compute the variables $x_{2^{k-2}}$ and $x_{3*2^{k-2}}$. For the sake of program uniformity we also have the root processor send x_0 to the left child and x_{K+1} to the right child. By so doing each node will receive two x-values from its parent in the backsubstitution phase. There is one wave front propagating from the root towards the leaves in the backsubstitution phase. There are K computations and K-1 communication steps in sequence.

A concise description of odd-even cyclic reduction on a binary tree is in the Appendix given in the form of pseudo code. The progression of the computations is for K=15 illustrated in Figure 5-3. Superscripts refer to reduction step. The numbers on edges and nodes denote equation numbers. A few time steps of the backsubstitution computations are shown in Figure 5-4.

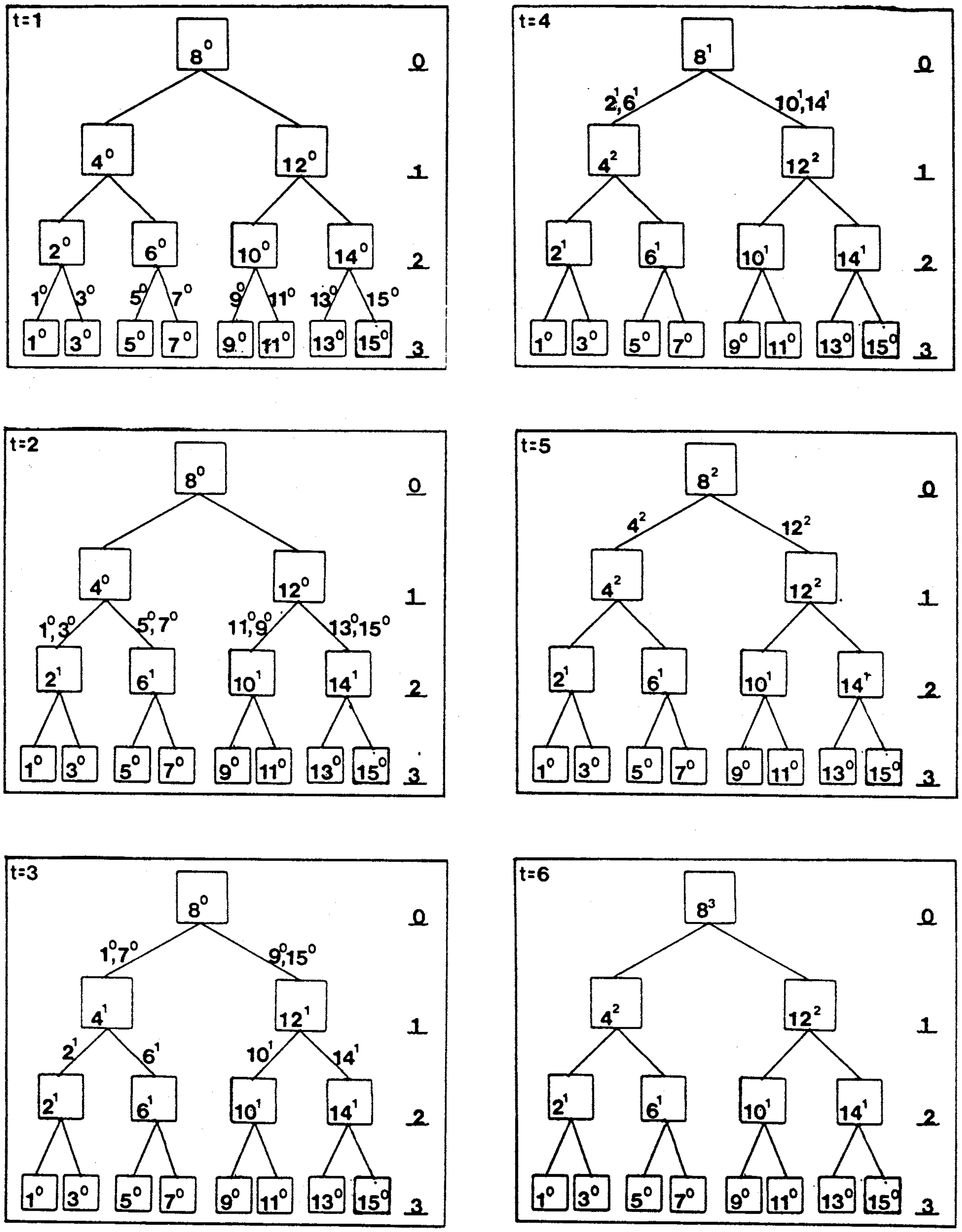


Figure 5-3: Cycles 1 - 6 of the reduction phase

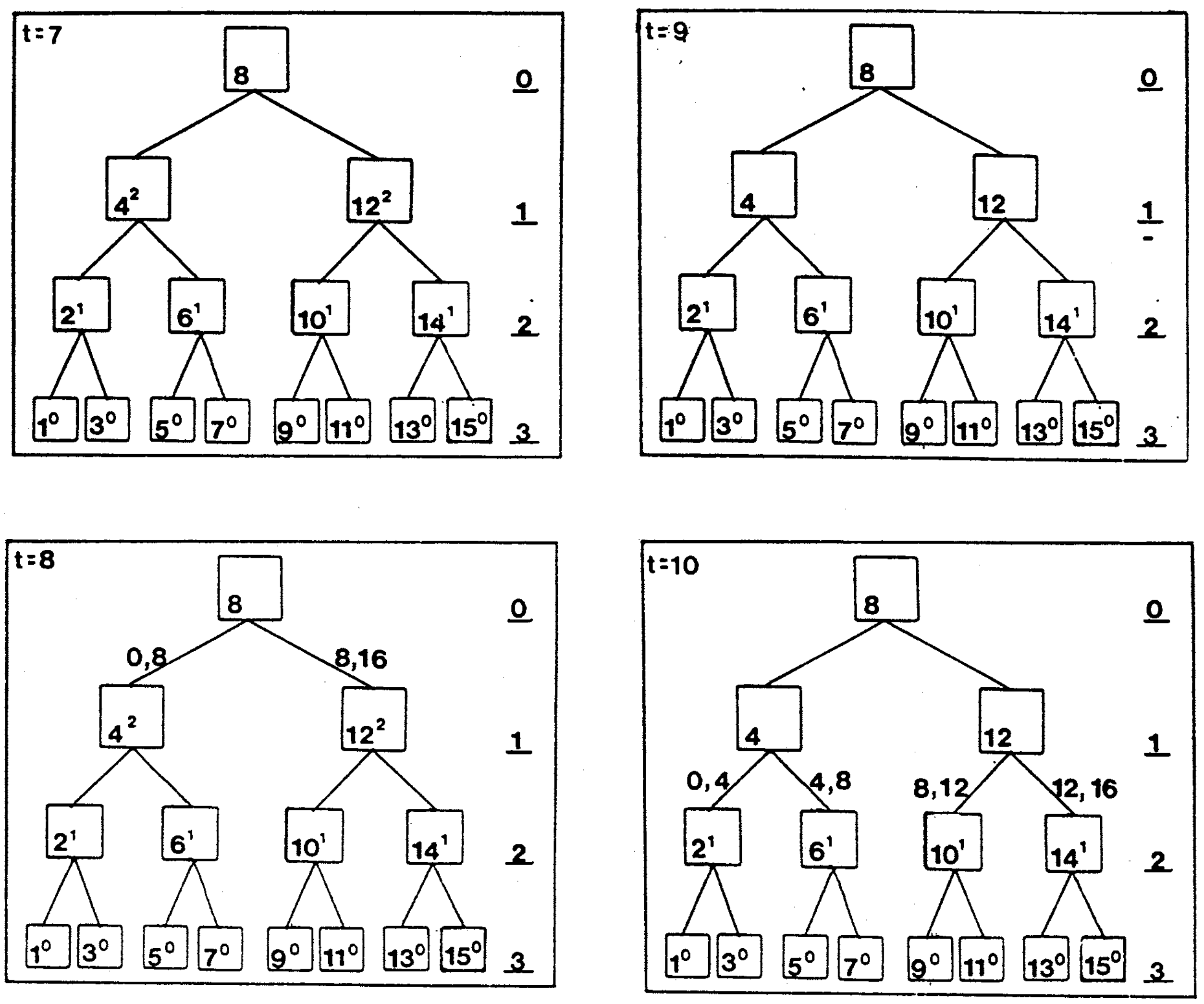


Figure 5-4: Cycle 1 - 4 of the backsubstitution phase

The computational complexity is:

Reduction.	$(k-2)(\max(t_a,t_c)+t_c)+t_a+t_c,$	$k>1$
Backsubstitution.	$(k-1)t_c+kt_a$	
Total.	$(k-2)\max(t_a,t_c)+2(k-1)t_c+(k+1)t_a,$	$k>1$

In the estimates t_c is the maximum time required for a communication, and t_a is the maximum time required for arithmetic operations on one equation. For each reduction step, except the last, the required time is $t_c+\max(t_a,t_c)$. If $t_a \gg t_c$, then the total time for cyclic reduction with one equation per node is $(2k-1)t_a$. With $t_c \gg t_a$ the time is $(3k-4)t_c$.

The estimates can be refined somewhat with the following assumptions;

- the time for communication is proportional to the number of values communicated,
- the time for arithmetic is proportional to the number of arithmetic operations,
- communication and computation can overlap in a pipelined fashion

In the reduction phase 2 equations, i.e., 8 variables are communicated between processor pairs. In the backsubstitution phase 2 variables are communicated. In the reduction phase 2 multiplications, 4 multiply-and-add operations, and 2 divisions are covered by t_a. In the backsubstitution phase 2 multiply-and add operations, and 1 division are performed.

If t_c^1 denotes the time for communication of 1 variable, and t_a^1 denotes the time for a multiply-and-add operation, assuming that division can be accomplished in this time as well, the following alternate estimates can be derived.

Reduction.

$$(k-3)(t_c^1 + 8\max(t_a^1,t_c^1)) + 2t_c^1 + 4\max(2t_a^1,t_c^1) + 3\max(2t_a^1,t_c^1) + 2t_a^1, \qquad k \geq 3$$

$$2t_c^1 + 4\max(2t_a^1,t_c^1) + 3\max(2t_a^1,t_c^1) + 2t_a^1, \qquad k=2$$

Backsubstitution.

$$(k-1)(2t_a^1 + t_c^1) + \max(t_a^1, t_c^1), \qquad k \geq 1$$

Total.

$$2(k-3)(t_a^1 + t_c^1 + 4\max(t_a^1,t_c^1)) + 7\max(2t_a^1,t_c^1) + \max(t_a^1,t_c^1) + 2(3t_a^1 + 2t_c^1), \qquad k \geq 3$$

In these estimates the fact that the leaves in the reduction phase have to send only 4 values is used. The processors along the paths from the leftmost and rightmost leaves to the root also only have to send 4 values to their respective parent processor in the reduction phase. This property results in special treatment of trees with $k \leq 3$.

5.2. Truncated cyclic reduction

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 n-1 communication steps to complete the first reduction step. However, fewer time steps are required subsequent to this propagation phase. The backsubstitution phase still needs k-1 communication steps, since $x_{K/2}$ has to propagate to the leaf level. The number of arithmetic steps prior and subsequent to solving the diagonal system is m. The complexity estimates for truncated cyclic reduction are

Reduction.	$(k-1)t_c + (m-1)\max(t_a,t_c) + t_a$
Backsubstitution.	$(k-1)t_c + (m+1)t_a$
Total.	$2(k-1)t_c + (m-1)\max(t_a,t_c) + (m+2)t_a$

If $t_c \ll t_a$, then the estimated time is reduced in proportion to $(k-m)/k$, if $t_c \approx t_a$ then the reduction is proportional to (k-m)/2k, and if $t_c \gg t_a$ the reduction in the estimated time for the reduction phase is proportional to (k-m)/3k.

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 1/3. The reduction in time for the parallel version of 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. Multiple equations per node

With N>K cyclic reduction based on domain decomposition and GECR are two candidate algorithms. Their mapping on to a binary tree is discussed next. For the binary tree mapping by **contraction** is another possibility, and we will show that it is inferior to the other two.

5.3.1. Domain decomposition

An inorder mapping of partitions to processors in a binary tree has the advantage that after the first n-k reduction steps there is one equation per processor ordered such that the algorithm for one equation per processor can be used. The final k-1 steps of the reduction process are completed in time proportional to k, i.e., the minimum order possible. However, proximity is not preserved between partitions having consecutive indices. Consecutive partitions may be located at a distance of up to k-1 nodes apart. For partitions mapped to nodes in inorder we have:

Theorem 10. With partitions mapped to processors in inorder, every k reduction steps require approximately $(k/2)^2+k$ communication steps in a tree of $K=2^k-1$ processors, assuming the reduced system has at least K equations.

Proof. Theorem 4 together with the inorder map of partitions states that at least one partition at level s exhibits s+1 changes in the direction of information exchange with one of its adjacent partitions for every k reduction steps. The minimum number of communication steps between a partition at level s and either of its adjacent partitions is k-1-s for an inorder map. Hence, for every k reduction steps there exists at least one partition at level s requiring (s+2)(k-1-s)+(k-s-2) communication steps. In this expression it is assumed that successive communication requests in the same direction are pipelined. The maximum total communication cost occurs at level $\lfloor (k-1)/2 \rfloor$ and the theorem follows. Q.E.D.

An alternative to mapping partitions in inorder to processors is to use a proximity preserving map, such as the one devised by Sekanina. [Sekanina 60]With such a map the communication time per reduction step during the first n-k steps is independent of k. However, the final k-1 steps require a time of $O(k^2)$. By changing the proximity preserving map to the inorder map after the first n-k steps the time complexity of the last k-1 steps is proportional to k. We will show that the change of map requires time $O(k)$. Hence, the communication complexity is of $O(n-k)$ for the first n-k steps of cyclic reduction, and $O(k)$ for the last k reduction steps.

Since we only need to embed a path, not a loop, as described in [Rosenberg, Snyder 78], and the embedding is changed to an inorder embedding during the execution of the algorithm, we modify the embedding algorithm by Rosenberg and Snyder (R&S) to generate a path embedding as in Figure 5-5.

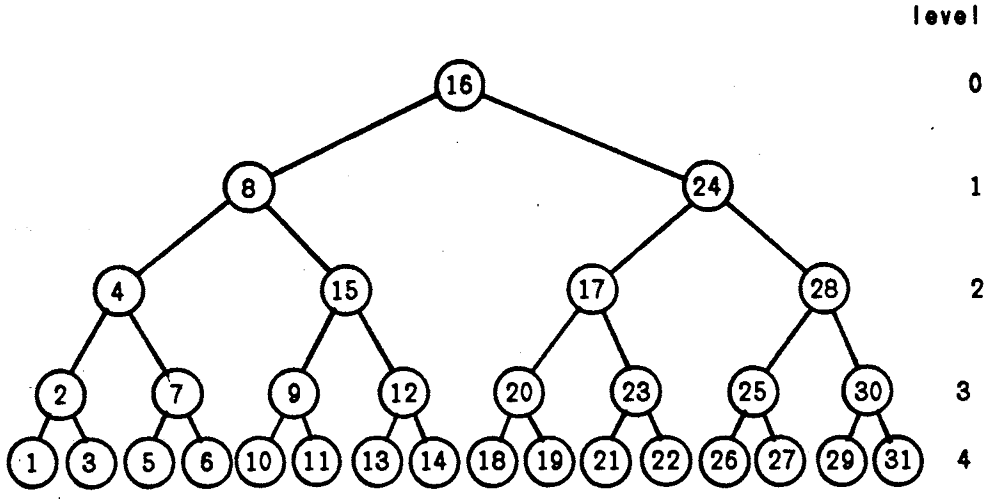


Figure 5-5: Mapping of a path to a binary tree preserving proximity

The modifications of the R&S algorithm are as follows:

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. In combination with modifications 2) and 3) the node labels in the left subtree of the root decrease progressing towards the leftmost leaf node in Figure 5-5, and conversely increase towards the rightmost leaf node in the right subtree of the root.

The second modification is made only to increase the formal similarity with a tree labeled in inorder. The two labelings generated by the R&S algorithm and an algorithm with only the second modification are clearly 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.

The algorithm is of the depth-first type and labels every other node on descent, and labels nodes in postorder on ascent, except nodes on the path from the root to the leftmost and rightmost child which are labeled in inorder.

Theorem 11. The maximum distance between adjacent nodes in the path is 3 when embedded by the path embedding algorithm.

Proof. It suffices to prove that the maximum distance between consecutively labeled nodes is bounded by the maximum distance generated by the R&S algorithm. On descent, the path embedding algorithm is identical to the R&S algorithm, except for the order in which left and right subtrees are visited. This difference only affects the symmetry. Hence, on descent the maximum distance between consecutively labeled nodes is 2. On ascent nodes are labeled when last encountered in the path embedding algorithm as well as the R&S algorithm. However, the modified algorithm differs in when a node is last encountered. By always labeling nodes at every other level in postorder in the R&S algorithm a return path with maximum distance of 2 between consecutive nodes is guaranteed. The path embedding algorithm uses the same strategy, except for nodes between the leftmost and rightmost leaf nodes and the root. This strategy can only decrease the maximum distance between successively ordered nodes, except for the distance between the node labeled last in the left subtree of the root and the node labeled first in the right subtree. However, this distance is of no relevance for the path embedding. Q.E.D.

Theorem 12. 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 $2^{k-1}-(2^{k-2}-1)-1=2^{k-2}$. 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 $2^{n-1}-1$, 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 suffice 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 $3*2^{k-3}-1$. 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 $3*2^{k-3}$, 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. Q.E.D.

Theorem 13. Cyclic reduction based on domain decomposition can on a binary tree be performed with a communication complexity of $O(\log_2(N/K)+\log_2 K)=O(\log_2 N)$.

Theorem 13 follows from theorems 11 and 12.

Some of the nodes adjacent in the path are at a distance of 1 in the embedding generated by the path embedding algorithm. Others are at a distance of 2 or 3. The number of directions of exchange between adjacent partitions vary from 1 to k for every k reduction steps according to theorem 4. Hence, pipelining can not be used at all between at least one pair of partitions. In a worst case mapping such partitions would be at a distance of 3. Furthermore, communication paths are shared between two different pairs of nodes adjacent in the path. With this assumption the complexity for each of the first n-k reduction steps is $\max((\lceil(2^{n-j}-1)/K\rceil-2)t_a,6t_c) + 2t_a)$, $1\leq j\leq n-k$. The arithmetic complexity is the same as for the algorithm entirely based on an inorder mapping. The time for changing the map of the path embedding algorithm to an inorder map is $(k-3)t_c$. For n=k+1 it does not pay to use the proximity preserving map. The estimated complexities are:

Reduction.
$$(k-2)(\max(t_a,t_c) + t_c) + t_a + (k-3)t_c + 6t_c + t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,6t_c) + 2t_a), \quad n>k,\ k>1$$

Backsubstitution.
$$(k-1)t_c + kt_a + (k-3)t_c + \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,6t_c) + 2t_a)$$

Total (CR).
$$(k-2)\max(t_a,t_c) + (4k-3)t_c + (k+2)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,6t_c) + 2t_a) +$$
$$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil-2)t_a,6t_c)+2t_a)$$

The total complexity is $O(N/K+\log_2 N)$. Using a proximity preserving map for the first n-k reduction steps always yields a lower total complexity for n>k+1 than a constant inorder map.

5.3.2. Gaussian Elimination - Cyclic Reduction

Using the inorder map for GECR requires k-1 communication steps in phase 2 of the algorithm. The remaining communications are as in cyclic reduction for one equation per processor. The estimated complexity is

Total (GECR). $(k-2)\max(t_a,t_c) + 3(k-1)t_c + (k+1)t_a + 2(\lceil N/K\rceil-1)2t_a$

5.3.3. Contraction

This mapping allows successive reduction steps to be pipelined in the binary tree, in a way similar to the case with one equation per processor. Conceptually, the contraction map can be obtained by a two stage procedure. First, the equations are mapped in inorder to nodes in a logical tree with N nodes. Then, levels 0 through n-k of this logical tree are mapped to the root processor of the processor tree. Level n-k+j of the logical tree is mapped to level j of the processor tree. The root of the processor tree stores equations $\{2^{k-1}, 2*2^{k-1}, ..., (2^{n-k+1}-1)2^{k-1}\}$, and processor p at level r of the processor tree stores $\{p2^{n-k}\pm2^{k-1-r}, p2^{n-k}\pm3*2^{k-1-r}, ..., p2^{n-k}\pm(2^{n-k}-1)2^{k-1-r}\}$. The root processor receives $2^{n-k+1}-1$ equations, i.e., essentially twice the number of equations of the other processors. Hence, the contraction map is poorly balanced. Its advantage is the possibility of pipelining successive reduction steps. Figure 5-6 shows this mapping for n=5, k=3.

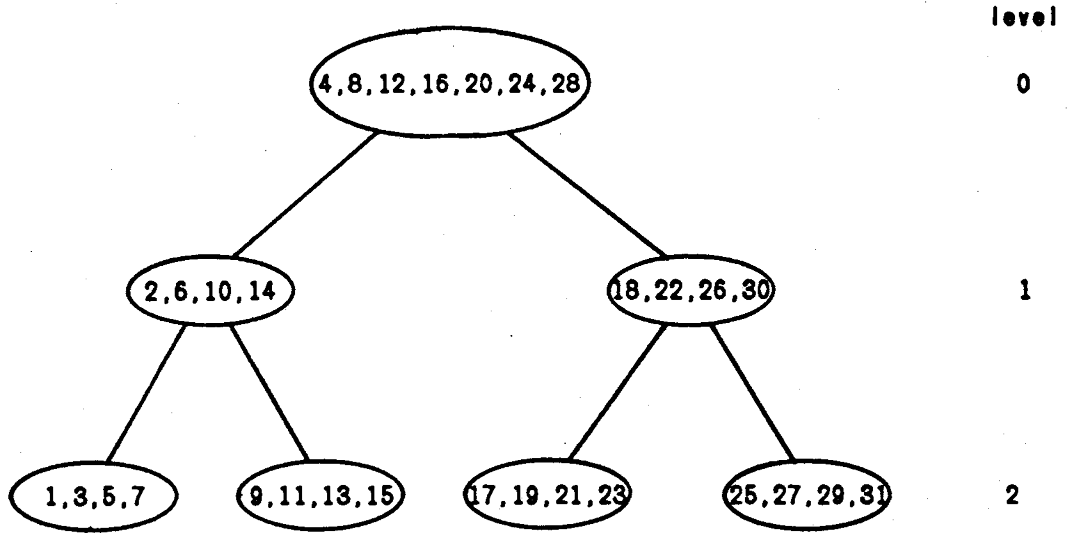


Figure 5-6: Mapping N equations to K processors by contraction

With this mapping the first k-1 reduction steps are performed in a pipelined manner on the tree, and the last n-k reduction steps are performed in the root processor. The converse is true for backsubstitution. The processors on the path from the leftmost and rightmost leaf processors to the root need to communicate $2^{n-k+1}-1$ equations to their respective parent processor (instead of 1 for n=k). The leaf processors need to communicate 2^{n-k} equations, and the remaining processors 2^{n-k+1} equations to their parent processor in the reduction phase. For the backsubstitution phase the number of values per communication is $2^{n-k}+1$ instead of 2.

The complexity estimates below make use of the assumption that communication and computation can overlap in a pipelined manner, i.e., reduction on one equation is performed while the necessary communication is carried out for the reduction of another equation in the same processor. In the backsubstitution x-values are sent from the root as they are computed.

Therefore, computations of x-values can start in several processors before all values are computed in the root.

Reduction.
$(k-2)(2^{n-k+1}-1)\max(t_a,t_c) + (k-1)t_c + \max((2^{n-k+1}-1)t_a,(2^{n-k}-2)t_c) + (2^{n-k+1}-(n-k+1))t_a$

Backsubstitution.
$\max((2^{n-k+1}-1)t_a+t_c,(2^{n-k}+1)t_c) + (k-2)(t_a+t_c) + t_a$

The arithmetic complexity for mapping by contraction and domain decomposition compares as $3N/K$ and $N/K+\log_2 K$. The communication complexities compare as $(N/K)\log_2 K$ and $\log_2 N$. Domain decomposition is clearly preferable, in that mapping all processors participate in the first n-k reduction steps. The last k-1 steps are performed with a decreasing number of processors engaged. In mapping by contraction the converse is true. The first k-1 steps are performed with a decreasing number of processors engaged, and the final n-k steps are performed in the root processor. Not only is the balance of computations better for domain decomposition, but the communication is also less.

5.4. 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 can not 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 preceeding problem is complete. Hence, except for the first problem, the propagation time from the leaves to the root is masked by computations. A reduction phase can be initiated at the rate of $(k-1)\max(t_a,t_c)$. A backsubstitution can be initiated at the rate $\max(t_a,t_c)$. For P problems a term $(P-1)k\max(t_a,t_c)$ is added to the complexity estimates.

Partitioning the tree such that there are equally many partitions as problems for $P \leq K$ results in approximately P equations per processor. The first partitioning increases the computational complexity with a term $2kt_c$+const due to the communication from root to leaves. For $P>2$ a communication cost proportional to $k-\log_2 P$ is instead incurred by the rearrangement of the proximity preserving map to an inorder map and back to the proximity preserving map. The arithmetic complexity increases as $P-\log_2 P$. The time to solve P problems by partitioning the tree increases from $O(\log_2 K)$ for $P=1$ to $O(K)$ for $P=K$.

A careful analysis yields the following:

Theorem 14. Partitioning is a more effective strategy than pipelining in solving multiple independent problems.

6. Shuffle-exchange networks

A shuffle-exchange network can be defined in terms of the binary encoding of the integers $\{0,1,2,...,2^k-1\}$. Let $(p_{k-1}p_{k-2}...p_1p_0)$, $p_i = \{0,1\}$, $i = \{0,1, ...,k-1\}$ be node addresses. Then, node $(p_{k-1}p_{k-2}...p_1p_0)$ is connected to node $(p_{k-2}p_{k-3}...p_0p_{k-1})$ (obtained by a cyclic shift). Furthermore node $(p_{k-1}p_{k-2}...p_10)$ is connected to $(p_{k-1}p_{k-2}...p_11)$. The edges obtained by cyclic shifts are called shuffle edges, and the even-to-odd edges exchange edges, [Leighton 83]. Most nodes in a shuffle-exchange network are connected to three nodes. Nodes (0...0) and (1...1) are only connected to one other node, since the cyclic shifts do not yield a different number. Nodes with addresses obtained by cyclic shifts of each other 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 $O(K/\log_2K)$. An 8-node shuffle exchange network is shown in Figure 6-1, and a 16-node network in Figure 6-2.

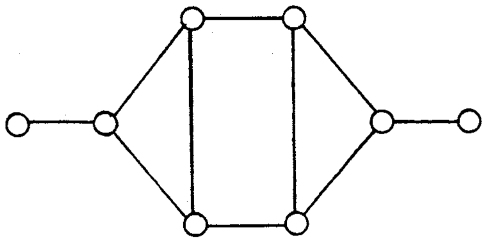


Figure 6-1: An 8-node shuffle-exchange network

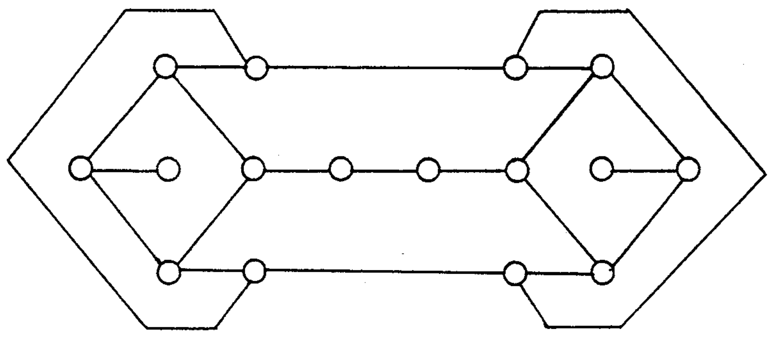


Figure 6-2: A 16-node shuffle-exchange network

The 8- and 16-node shuffle-exchange networks are planar, but arbitrary shuffle-exchange networks are in general not planar.

Proposition 1. The diameter of a shuffle-exchange graph is 2k-1.

Proposition 2. The maximum distance between consecutively labeled nodes is 2(k-2) and occurs between nodes (100..0) and (011..1).

Proposition 1 states that the diameter of a 2^k node shuffle-exchange graph is almost the same as the diameter of a 2^k-1 node binary tree.

Proposition 2 is an important difference compared to the perfect shuffle network treated in the next section. Cyclic reduction requires communication between consecutively numbered equations. Identifying an equation with a node having the same number in the shuffle-exchange graph does not yield an efficient algorithm. The maximum distance between consecutively ordered nodes has to occur between an odd node and its succeeding even node. It is also easily seen that the distance is at most twice the number of bit reversals.

Proposition 3. 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.

One such mapping is obtained by labeling the nodes in the tree in breadth-first order, starting at the root and assigning to it the number 1. With this labeling of the tree, the number assigned to the left child of a node is twice the number assigned to the parent node. The number assigned to 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 node in the tree 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 at distance 2.

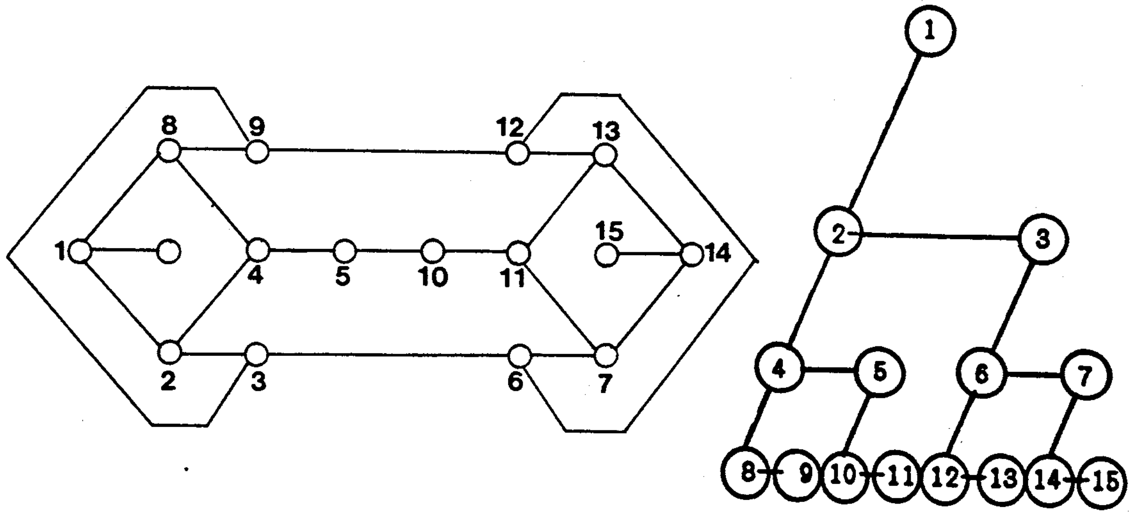


Figure 6-3: A binary tree embedding in a 16-node shuffle-exchange network

The equations are mapped to the processors in inorder, and the algorithms described for the binary tree can be used with minor modifications. Pseudo code for cyclic reduction on a shuffle-exchange network is contained in the Appendix. The algorithm is derived from the binary tree algorithm with one equation per processor.

The following complexity estimates apply to the shuffle-exchange algorithm.

Reduction.	$2(k-1)t_c + (k-2)\max(t_a,2t_c) + t_a$
Backsubstitution.	$2(k-1)t_c + kt_a$
Total.	$4(k-1)t_c + (k-2)\max(t_a,2t_c) + (k+1)t_a$

With this map of the binary tree on to the shuffle-exchange network, equation $2^{k-1}-1$ is stored at a distance of 2k-3 from equation 2^{k-1}. This distance is approximately twice that in the binary tree, and is reflected in the propagation time required for the first reduction step. If the communication time dominates the time for arithmetic, then the shuffle-exchange algorithm based on the tree algorithm requires twice the time of the tree algorithm.

The above shuffle-exchange algorithm inherits the properties of the tree algorithm. Hence, for truncated cyclic reduction there is still a propagation time of $O(k)$.

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.

7. Perfect shuffle networks

A perfect shuffle network is closely related to the shuffle-exchange network. In addition to the connections of that network, the perfect shuffle also has connections between every odd processor and its succeeding even processor, modulo the size of the network. Hence, in the perfect shuffle network processors having successive addresses are linearly interconnected. Stone has described mappings of bitonic sort, and the FFT, on to a perfect shuffle network [Stone 71].

7.1. One equation per processor

Odd-even cyclic reduction is easily mapped on to a perfect shuffle network. Let a processor initially store an equation with the same number as its address. Then, the first reduction step can be carried out in the time needed for one communication and the time of one reduction computation. 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 complexity estimates for the perfect shuffle algorithm are:

Reduction.	$(2k-3)t_c + (k-1)t_a$
Backsubstitution.	$(2k-3)t_c + kt_a$
Total.	$2(2k-3)t_c + (2k-1)t_a$

7.2. Truncated reduction, multiple equations per node

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 time complexity is $2(2m-1)t_c+(2m+1)t_a$.

For N>K, partitioning can be used to define a mapping of equations on to processors. Partition i can be identified with processor i. Processor i communicates one equation with processors i+1 ($i \neq 2^k-2$) and i-1 ($i \neq 0$) during the first n-k reduction steps. The communication is local and no shuffle operations are needed. The complexity estimates are:

Reduction.	$2(k-1)t_c + kt_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,t_c) + 2t_a)$
Backsubstitution.	$2(k-1)t_c + (k+2)t_a + \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,t_c) + 2t_a)$
Total (CR).	$4(k-1)t_c + 2(k+1)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,t_c) + t_a) +$
	$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,t_c) + 2t_a)$
Total (GECR).	$2(2k-3)t_c + (2k-1)t_a + t_c + (\lceil N/K\rceil-1)2t_a$

Parallel cyclic reduction can be implemented on the perfect shuffle with no principal difficulty. The performance is the same as for two independent problems solved concurrently. Two problems can be solved concurrently by mapping one problem into processors 0 through 2^k-2, the other into processors 1 through 2^k-1. Then, in the first reduction step one set of reduced equations are computed in even processors, the other in odd processors. After the first shuffle operation one problem is contained in the processors with addresses in the lower half of the address space, and the other problem in the processors with addresses in the upper half of the address space.

Theorem 15. Partitioning is more effective than pipelining in solving multiple independent problems on a perfect shuffle network.

Proof. By theorem 8 it is only necessary to consider the communication complexity. One processor is used in all 2(2k-3) communication steps. The communication complexity for partitioning is $2(2k-3-\lceil \log_2 P \rceil)$ and the theorem follows. Q.E.D.

8. Boolean k-cube

Boolean k-cube configured ensembles of the kind considered here have been built at Caltech, [Seitz 84]. A 6-cube has been in operation since the fall of 1983 and a 10-cube is being planned. The Connection Machine conceived at MIT is another k-cube configured ensemble with processors of a finer grain size. The Connection Machine is anticipated to be constructed eventually as a 14 to 15 dimensional cube. The NYU Ultracomputer [Schwartz 80] [Gottlieb et.al. 83] uses a switch network of k stages with K/2 switches per stage. Processors are connected to one side of the switch, storage modules to the other. Hence, the Ultracomputer does not fit the computational model used here, but there is no principal difficulty in adapting our results to the Ultracomputer model.

8.1. One equation per processor

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 Figure 8-1.

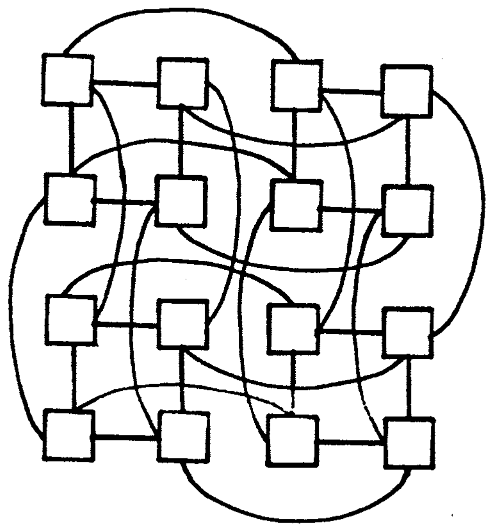


Figure 8-1: A Boolean 4-cube

Boolean k-cube algorithms can be obtained by embedding the binary tree in the cube and adapting the tree algorithm to the cube in a way similar to what was done for the shuffle-exchange network. 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 $(00...0p_rp_{r-1}...p_0)$, its left child is assigned to processor $(00...1p_rp_{r-1}...p_0)$, and its right child to processor $(00...1p_r'p_{r-1}...p_0)$, where p_r' is the complement of p_r. This mapping inherits the characteristics of the tree algorithm.

We will now briefly describe two k-cube algorithms that have the same properties as the perfect shuffle algorithm. The first is referred to as an *in-place* algorithm, the second as 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 pseudo code 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 Figure 3-1. For convenience we number the equations starting from 0. The equations are initially mapped to the processors in the cube using a Gray code. Gray codes for successive integers differ by only 1 bit. This property guarantees that the first reduction step only involves nearest neighbor communication. There exist several different Gray codes. We choose to use a binary-reflected Gray code. This code has properties that are useful in subsequent reduction steps.

Let G_i be the Gray code of i, and $G(k)=(G_0,G_1,...,G_{2^k-1})$ be a k-bit code. Then, following [Reingold,Nievergelt,Deo 77], a binary-reflected Gray code is defined by

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

or

$$G(k+1) = (G_00,G_01,G_11,G_10,G_20,G_21,...,G_{2^k-1}1,G_{2^k-1}0)$$

The binary-reflected 4-bit Gray code G(4) is (0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000). Let $i=(r_kr_{k-1}...r_0)$ and $G_i=(g_kg_{k-1}...g_1)$.

Proposition 4. $g_j=(r_j+r_{j-1})\text{mod}2$ (encoding) and $r_j=(\Sigma_{i=j+1}^{k}g_i)\text{mod}2$ (decoding) (note $r_k=0$).

For proof see [Reingold,Nievergelt,Deo 77].

Proposition 5. 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.

The proof is immediate from Proposition 4.

Proposition 5 determines the minimal communication complexity for the in-place algorithm based on embeddings according to the binary-reflected Gray code.

Theorem 16. The number of communication steps per reduction step is 2 for the in-place algorithm, except for the first for which it is one. The communication paths can be made disjoint.

Proof. The communication distance for the first step is 1 from the definition of the code, and 2 for the remaining reduction steps by Proposition 5. 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.

Q.E.D.

The computational complexity is

Reduction.	$(2k-3)t_c + (k-1)t_a$
Backsubstitution.	$(2k-3)t_c + kt_a$
Total.	$2(2k-3)t_c + (2k-1)t_a$

These estimated computational complexities are identical to the complexities for the perfect shuffle network.

The folding algorithm is conceptually simple, and we find it also easier to program. In the in-place algorithm it is necessary to compute the addresses of processors with which communication is to take place in the various steps of the algorithm, even when the processor no longer performs any reduction computations.

In reduction step j equations with indices $M^j=\{(m_j+1)2^j-1\}$, $m_j=\{0,1,2,..., 2^{k-j}-2\}$ participate, and reduction is performed on equations with indices 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 proposition 5 consecutively ordered nodes in M^{j+1} are not.

Proposition 6. By performing one exchange operation on selected pairs of adjacent nodes between successive reduction steps there exist a path of length $|M^j|$ through the nodes storing the equations with indices in M^j for $j=0,1,2,3,...,k-1$.

Proof. It is true for $j=0$ by construction. From the definition of the binary-reflected Gray code, it follows that $G_{2q+1}(k\text{-}p) = (G_q(k\text{-}p\text{-}1)X)$, where $X=\{0,1\}$ and $p=0,1,...,k\text{-}2$ and $q=0,1,...2^{k-p-1}\text{-}1$. The equations in M^j on which reduction is performed are the ones with $m_j=\{1,3,...,2^{k-j}\text{-}3\}$, i.e., the indices are of the form $2i+1$, $i=0,1,..,2^{k-j}\text{-}2$. Let $p=j\text{-}1$. Then $G_{2q+1}(k\text{-}(j\text{-}1)) = (G_q(k\text{-}j)X)$, $q=0,1,...2^{k-j}\text{-}1$. Hence, an equation with index in the set $\{(2i+2)2^j\text{-}1\}$, $i=0,1,..,2^{k-j}\text{-}2$ is either in a node with $X=1$ or can be moved to that node by an exchange operation between nodes $(G_q(k\text{-}j)0)$ and $(G_q(k\text{-}j)1)$. The theorem follows by induction. Q.E.D.

From proposition 4 and the proof of proposition 6 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_i1), $i=K/2\text{-}1$. The exchange-reduction steps are repeated k-3 additional times on successively smaller subcubes. In the jth step, exchange is performed on bit j from the right, 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...11) computes $x_{2^{k-1}-1}$. Backsubstitution is then carried out in reversed 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 computational complexity is the same for both algorithms based on the binary-reflected Gray code.

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

8.2. Truncated reduction, multiple equations per node

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.

Using domain decomposition for N>K and identifying partition i with processor i yields an algorithm in which the first n-k reduction steps require only nearest neighbor communication. The computational complexity is the same as for the perfect shuffle algorithm.

With multiple independent problems to be solved partitioning and pipelining can be applied. With one equation per processor the number of processors participating in the computations is reduced by a factor of 2 in each reduction step. Also, in the first reduction step half of the processors are performing only communication operations. As for the perfect shuffle, better utilization of the resources is accomplished by solving 2 problems concurrently. After the first reduction step one problem is confined to half of the cube, the other problem to the other half of the cube. The processor utilization is 100% during the first reduction step, but decreases to 2/K for the final reduction step.

Additional gain in utilization is achieved by pipelining independent cyclic reduction computations. Two problems can be initiated at once. Then, one new problem can be initiated every other cycle. This difference compared to the perfect shuffle is due to the complete symmetry of the k-cube. The programming becomes nontrivial, but it is not necessary to wait for approximately 2k cycles for each initiation of a new problem. Pipelining the solution of independent problems on a cube results in 100% processor utilization, except for the final k-2 reduction steps. The total time for the reduction phase of P problems is proportional to 2(P+k-3). Pipelining of the backsubstitution phase of the different problems can be done in a similar way, with the total time being proportional to P+k-2. Hence, pipelining can improve the speed-up from $O(K/\log_2 K)$ to $O(K)$ for P of at least $O(\log_2 K)$.

Partitioning of the cube into subcubes increases the processor utilization to 100% as P approaches N. With pipelining of independent problems there is always a phase with less than 100% processor utilization, regardless of the time for communication relative to the time for arithmetic. The time for solving P independent problems increases linearly with P if the problems are pipelined. With partitioning there is one logarithmically decreasing term, and one linearly increasing term. Hence, the following theorem holds

Theorem 17. Partitioning is a more efficient way than pipelining to solve multiple independent problems on a cube. The order of complexity for partitioning increases from $O(\log_2 K)$ to $O(P)$ as P increases from 1 to K. For pipelining the complexity instead increases to $O(P+\log_2 K)$.

Note that theorem number 17 holds also if partitions for different problems are mapped to processors such that a processor receives the same partition for every K problems. The difference between partitioning and pipelining is considerably smaller than with identical maps for all problems. The separation between different problems is 4 communication steps and 2 arithmetic steps.

9. Two-dimensional mesh

Two-dimensional processor arrays have been proposed and built in various forms, e.g., the ILLIAC IV, the ICL DAP, and the MPP, to mention a few. Some have end-around connections effectively making them into cylinders, or toruses, possibly twisted. There exist several possible mappings of cyclic reduction on to mesh configured arrays that have the same computational complexity, but that differ in the communication complexity. We will show a couple of different mappings of equations to processors for which the number of communication steps is proportional to $\sqrt{K}$, but which have slightly different constants of proportionality. The mesh algorithm makes use of a linear array algorithm which we describe first. For simplicity $k/2$ is assumed to be an integer.

9.1. Linear array

Mapping equations on to processors by identifying equation indices with processor indices implies, with processors numbered consecutively, that the communication in the first reduction step is with neighbor processors. In the second reduction step the communication is between even numbered processors, etc. The estimated time complexity is:

Reduction.	$(k-1)t_a + (K/2-1)t_c$
Backsubstitution.	$kt_a + (K/2-1)t_c$
Total.	$(2k-1)t_a + (K-2)t_c$

The algorithm briefly outlined above is an *in-place* algorithm. As an alternative a shuffle operation can be implemented between each step of the cyclic reduction algorithm. The order of the communication time is the same as for the in-place algorithm. Figure 9-1 illustrates shuffle operations on a linear array.

```
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8

1 - 3 - 5 - 7 - 2 - 4 - 6 - 8

1 - 5 - 3 - 7 - 2 - 6 - 4 - 8
```

Figure 9-1: Implementing shuffles of decreasing sizes on a linear array

The number of communication steps required for the explicit implementation of shuffle operations is K-3 for each of the reduction and backsubstitution phases.

Proposition 7. In-place algorithms 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 explicit implementation of shuffle operations.

Depending on the relative communication cost Gaussian elimination may be of a lower total complexity on the linear array than cyclic reduction, see the discussion in section 4.5, page 15.

For $N>K$ and domain decomposition the first n-k steps require only local communication. There is one communication action per reduction step. The time complexity is:

Reduction. $(k-1)t_a + (K/2-1)t_c + t_c + t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,t_c) + 2t_a)$

Backsubstitution. $kt_a+(K/2-1)t_c+t_c+2t_a+\Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil-\lceil(2^{j-1}-1)/K\rceil-2)t_a,t_c)+2t_a)$

Total (CR). $2(k+1)t_a + Kt_c + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil - 2)t_a,t_c) + 2t_a)$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,t_c) + 2t_a)$

Total (GECR). $2k-1)t_a + (2^k-1)t_c + (\lceil N/K\rceil-1)2t_a$

For truncated cyclic reduction the in-place algorithm is even more effective than the algorithm using an explicit implementation of the shuffle operation. For an in-place algorithm the communication cost increases as a function of the reduction step index, whereas the highest communication cost (half of the total) is incurred in the first step of the shuffle based algorithm.

Proposition 8. Partitioning is more efficient than pipelining in treating multiple independent problems on a linear array.

One processor is used in $\log_2K$ steps, and the proposition follows.

9.2. Two-dimensional mesh

We first give an algorithm for a mesh without end-around connections. Equations are mapped to processors row by row and for each row alternately in right-to-left and left-to right order. The embedding is of "serpentine" type, see Figure 9-2.

As for the linear array we consider an *in-place* algorithm, and an algorithm with explicit implementation of shuffle operations. The first reduction step requires one communication between neighboring processors, followed by the reduction computation. The second reduction step requires communication between processors at a distance of 2 apart, etc. After k/2 steps there is one equation per row to be considered in further reduction steps. These equations are at opposite ends of successive rows. 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 time complexity is

Reduction. $3(\sqrt{K}-1)t_c + (k-1)t_a$

Backsubstitution. $3(\sqrt{K}-1)t_c + kt_a$

Total. $6(\sqrt{K}-1)t_c + (2k-1)t_a$

Proposition 9. Cyclic reduction on a 2-dimensional mesh can be performed in time $(3\sqrt{K}+\log K-4)t_c + (2k-1)t_a$ by explicitly implementing shuffle operations.

Proposition 8 states that explicitly implementing the shuffle operation yields a more efficient algorithm with respect to communication than the in-place algorithm outlined above. The shuffle operations in the algorithm outlined below are implemented only in the first k/2 steps, i.e., before the reduction computations are limited to one column. 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. Let rows and columns be numbered from 0 to $\sqrt{K}$-1.

Outline of a 2-dimensional mesh algorithm for the reduction phase:

```
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 the linear array algorithm
end
```

Figure 9-2 shows some reduction steps.

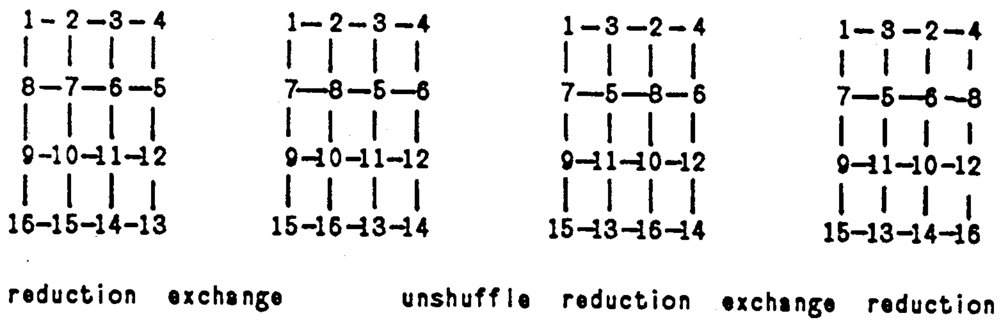


Figure 9-2: Odd-even cyclic reduction on a mesh. The reduction phase

The number of arithmetic steps is the same as in the in-place algorithm. The total time complexity is:

Total. $(3\sqrt{K}+\log K-4)t_c + (2k-1)t_a$.

Hence, we conclude that for the mesh it is beneficial to implement the shuffle operations in one dimension, followed by an in-place algorithm in the other.

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 cylinder, then embedding the equations from top to bottom in consecutive order yields an in-place algorithm with a total communication complexity of $(3\sqrt{K}-4)t_c$. Even without end-around connections there exist embeddings that yield lower communication complexity than the one used above, e.g., the one in Figure 9-3. Its communication complexity is $2(2\sqrt{K}-3)$.

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

Figure 9-3: An embedding with communication time $(2\sqrt{K}-3)t_c$

The communication complexity is $O(\sqrt{K})$ for the mesh. This order can not be reduced since the diameter of the mesh is $O(\sqrt{K})$. It is the highest order term of the computational complexity.

Truncated cyclic reduction reduces the communication time, but not in proportion to the number of reduction steps avoided because of the nonuniform communication cost. The in-place algorithm benefits more from truncating the reduction process in that the communication cost is highest towards the end of of the first $k/2$ steps, and last $k/2-1$ steps, respectively. With explicit implementation of the shuffle operations the communication cost is reversed, i.e., highest for the first step in each half of the reduction process. It might be preferable in the case of truncated cyclic reduction to use an in-place algorithm also for the reduction computations in the first dimension of the mesh. Parallel cyclic reduction can be implemented on the mesh to execute in time $O(\log_2 K)$. All processors are used throughout the $O(\log_2 K)$ steps.

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 Figure 9-2).

Domain decomposition can be used to map N>K equations on to the processor array.

Proposition 10. Partitioning is a more efficient strategy than pipelining in handling multiple independent problems on a mesh.

Using pipelining, the communication time increases by at least 3/2PlogK. Using partitioning the communication time decreases. From theorem 8 the arithmetic complexity is also favorable for partitioning. Exploiting the symmetry, 4 different problems could be embedded with a moderate degree of conflict. Neglecting those conflicts, the arithmetic complexity still grows faster for pipelining than partitioning.

10. Summary and Discussion

10.1. One equation per processor

For networks of diameter $O(\log_2 K)$, the concurrent algorithms for one equation per processor have a communication complexity that is of the same order as the arithmetic complexity of algorithms for a multiprocessor machine with no communication constraints and a sufficient number of processors. For the 2-dimensional mesh the communication complexity is proportional to $\sqrt{K}$, and for the linear array proportional to K, the time for global communication.

The binary tree algorithm for N=K is the most efficient. With the computational model used, this result also means that for N=K the binary tree interconnection is superior to the other forms of investigated interconnect. For $t_a \gg t_c$ the difference in estimated complexities of the algorithms for the different forms of interconnect is insignificant, for $t_a \approx t_c$ the time complexity of the binary tree algorithm is $\approx 2/3$ of that of the perfect shuffle and boolean k-cube algorithms, and for $t_a \ll t_c$ the tree algorithm requires $\approx 3/4$ of the time for the perfect shuffle and k-cube algorithms. The shuffle-exchange algorithm requires $\approx 7/4$th the time of the tree algorithm for $t_a \approx t_c$, and twice the time of the tree algorithm for $t_a \ll t_c$. The communication time of the algorithm for the 2-dimensional mesh is proportional to $\sqrt{K}$, and for a linear array proportional to K. The arithmetic complexity is the same for all algorithms. The estimated complexities are summarized in Table 10-1.

10.2. Truncated cyclic reduction

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 10-2 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

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

Table 10-1: Estimated complexities for some ensembles, one equation per processor

binary tree, the shuffle-exchange, the perfect shuffle and the k-cube is significantly better than on a uniprocessor. Most operations are performed in the first few reduction steps, and the last few backsubstitution steps.

Configuration	Total
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 10-2: Estimated complexities for truncated, m-step, cyclic reduction

10.3. Multiple equations per processor

For N>K domain decomposition is used to allocate equations to processors. For trees it was shown that mapping by contraction is inferior to domain decomposition, due to excessive communication and poor balancing of the computational load. Table 10-3 summarizes the complexity estimates for multiple equations per node.

With a large number of equations per node the difference between different interconnections is insignificant. The speed-up is approximately linear in the number of processors. The range for linear speed-up depends on the type of interconnection, and the ratio $\alpha=t_c/t_a$. For the binary tree (and the shuffle-exchange network) the speed-up increases monotonically for the mappings used. The minimal computational time is $O(\log_2 N)$. The speed-up has a maximum for $K\approx N/(1+\alpha)$ for the perfect shuffle and the k-cube. The minimal time is$O(\log_2 N)$ as for the binary tree. The maximum speed-up for a 2-dimensional mesh is obtained for $K\approx((N/2\alpha)^2)^{1/3}$, and for the linear array for $K\approx(N/\alpha)^{1/2}$. The minimal computational times are $O(N^{1/3})$ and

Binary tree

Total (CR). $(k-2)\max(t_a,t_c) + (4k-3)t_c + (k+2)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil - 2)t_a,6t_c) + 2t_a) +$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,6t_c) + 2t_a)$

Total (GECR). $(k-2)\max(t_a,t_c) + 3(k-1)t_c + (k+1)t_a + 2(\lceil N/K\rceil-1)2t_a$

Shuffle-exchange

Total (CR). $(k-2)\max(t_a,2t_c) + (4k-3)2t_c + (k+2)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,12t_c)+2t_a) +$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,12t_c) + 2t_a)$

Total (GECR). $(k-2)\max(t_a,2t_c) + 6(k-1)t_c + (k+1)t_a + 2(\lceil N/K\rceil-1)2t_a$

Perfect shuffle and boolean k-cube

Total (CR). $4(k-1)t_c + 2(k+1)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil - 2)t_a,t_c) + 2t_a) +$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,t_c) + 2t_a)$

Total (GECR). $(4k-5)t_c + (2k-1)t_a + (\lceil N/K\rceil-1)2t_a$

2-dimensional mesh

Total (CR). $(3\sqrt{K}+\log K-2)t_c + 2(k+1)t_a + \Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil - 2)t_a,t_c) + 2t_a) +$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil - 2)t_a,t_c) + 2t_a)$

Total (GECR). $(3\sqrt{K}+\log K-3)t_c + (2k-1)t_a + (\lceil N/K\rceil-1)2t_a$

Linear array

Total (CR). $2(k+1)t_a+Kt_c+\Sigma_{j=k+1}^{n-1}(\max((\lceil(2^j-1)/K\rceil-2)t_a,t_c) + 2t_a) +$

$+ \Sigma_{j=k+1}^{n}(\max((\lceil(2^j-1)/K\rceil-\lceil(2^{j-1}-1)/K\rceil-2)t_a,t_c) + 2t_a)$

Total (GECR). $(2k-1)t_a + (K-1)t_c + (\lceil N/K\rceil-1)2t_a$

Table 10-3: Estimated complexities for some ensembles, multiple equations per processor, k>1

$O(N^{1/2})$ respectively. The decrease in speed-up is quite dramatic for a linear array with slow communication as can be seen in Figure 10-1. The Figure displays graphically the complexity estimates in Table 10-3. 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 Figure 10-2, and the dependence on the relative communication cost in Figure 10-3. Figure 10-4 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.

We also evaluate the complexity estimates for GECR for comparison. The communication complexity for GECR is lower, and the arithmetic complexity approximately the same as for cyclic reduction. The complexity estimates are approximate and the curves should be interpreted accordingly. The qualitative behavior should be correct, though. The advantage of GECR over cyclic reduction is increasing as a function of the ratio t_c/t_a, and is the greatest for intermediate values of the size of the ensemble relative to the problem size. For large ensembles the communication in GECR is approaching that of cyclic reduction. Figure 10-5 shows the difference for binary trees, perfect shuffle networks and boolean cubes. Figure 10-6 shows the estimated difference for a 2-dimensional mesh and a linear array. The speed-up for GECR on a binary tree and a perfect shuffle network should be approximately equal, see Figure 10-7.

10.4. Multiple independent problems

With multiple independent problems to be solved, pipelining or partitioning of the processors into subnetworks can be employed as techniques to make efficient use of the processors. Partitioning was shown to make the most efficient use of the processors for all investigated topologies. For P=K, partitioning results in one problem per processor, and the performance of all interconnection schemes is the same. No use of the interconnections is made. With multiple independent problems it will at some point become more effective to use only Gaussian elimination, even in the case of truncated cyclic reduction.

10.5. Programming issues

All algorithms 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 [Li,Johnsson 83]. For a few equations per node the ratio of total program store to data store is significant.

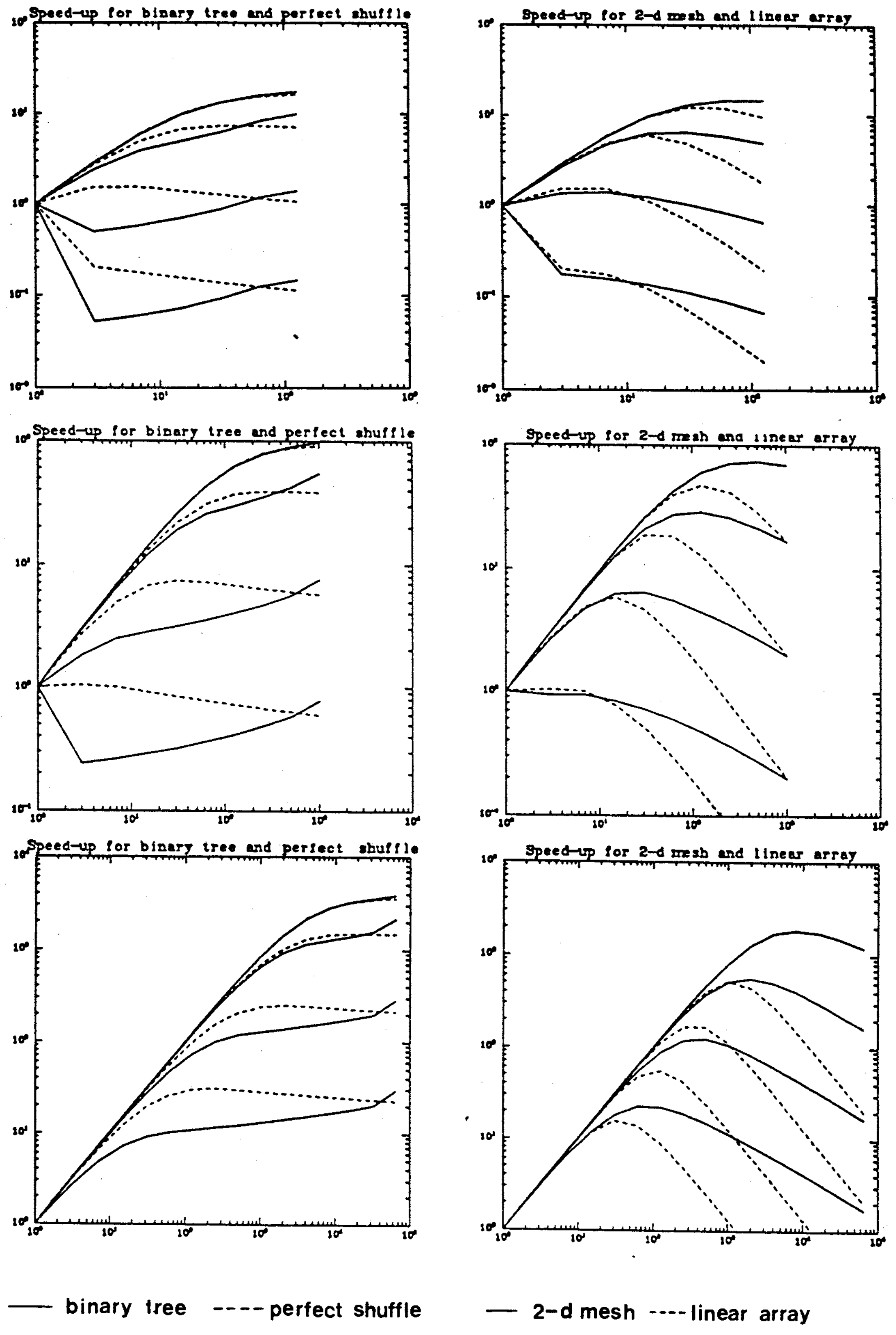


Figure 10-1: Speed-up for $N=2^7-1$, $N=2^{10}-1$, and $N=2^{16}-1$. $\alpha=0.1$, 1, 10, and 100

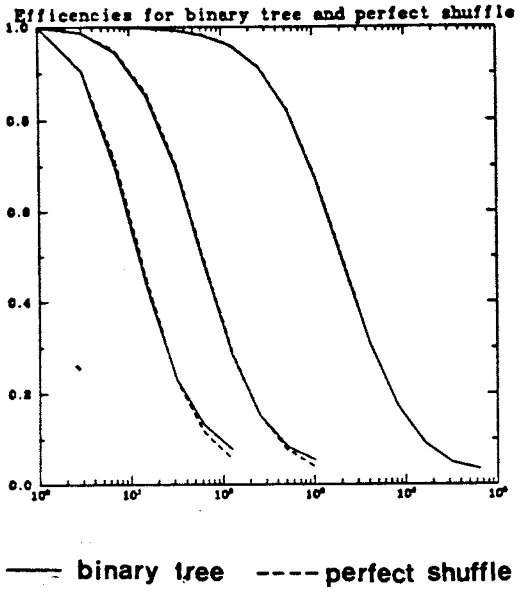


Figure 10-2: The efficiency for a binary tree, perfect shuffle and k-cube, α=1 N=2^7-1, 2^{10}-1, and 2^{16}-1

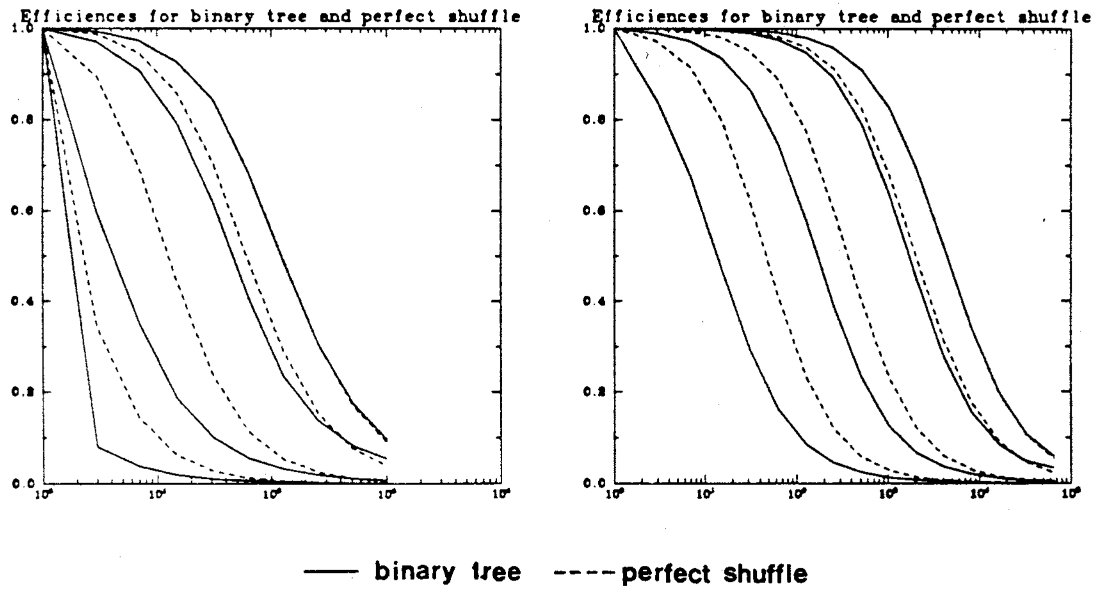


Figure 10-3: The efficiency for a binary tree, perfect shuffle and k-cube, N=2^{10}-1, 2^{16}-1 α=0.1,1,10,100

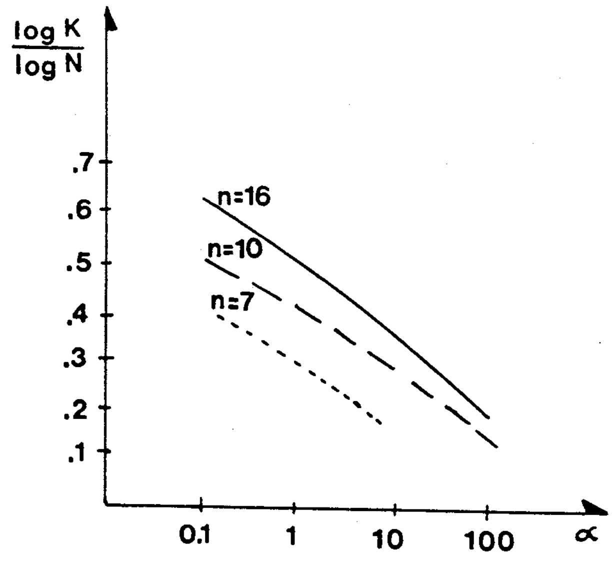


Figure 10-4: Relative number of processors for 80% efficiency (tree, perfect shuffle, cube)

10.6. Generalizations

We have in our computational model assumed that each processor has its own local storage. In architectures such as the Ultracomputer and the TRAC, processors and storage units are on opposite sides of a switching network. Our k-cube algorithm can be adopted to the Ultracomputer with no principal difficulty. However, the complexity estimates increase since the time for messages to pass through the switch is proportional to k. By pipelining the switch its bandwidth can be increased from $O(K/\log_2 K)$ to $O(K)$, but the sequential dependencies in the cyclic reduction algorithm are such that full advantage cannot be taken of such a feature. Interprocessor communication is needed between each reduction step, and the result of one step is needed in the next step. The computational complexity is:

Reduction.
$(k-1)(t_a + (2k+1)t_c) + \Sigma_{j=k}^{n-1}(\max(\lceil(2^j-1)/K\rceil \max(t_a,t_c), (2k+1)t_c + 2t_a)$

Backsubstitution.
$(k-1)(t_a + (2k+1)t_c) + 2kt_c + t_a + \Sigma_{j=k}^{n}(\max((\lceil(2^j-1)/K\rceil-\lceil(2^{j-1}-1)/K\rceil)\max(t_a,t_c),(2k+1)t_c+2t_a)$

Total (CR).
$(2k-1)t_a + 2(2k(k-1)-1)t_c + \Sigma_{j=k}^{n-1}(\max(\lceil(2^j-1)/K\rceil\max(t_a,t_c),(2k+1)t_c+2t_a) +$

$+ \Sigma_{j=k}^{n}(\max((\lceil(2^j-1)/K\rceil - \lceil(2^{j-1}-1)/K\rceil)\max(t_a,t_c),(2k+1)t_c+2t_a)$

Total (GECR).
$(2k-1)t_a + 2(2k(k-1)-1)t_c + \max((\lceil N/K\rceil-1)2t_a, 2kt_c) + kt_c$

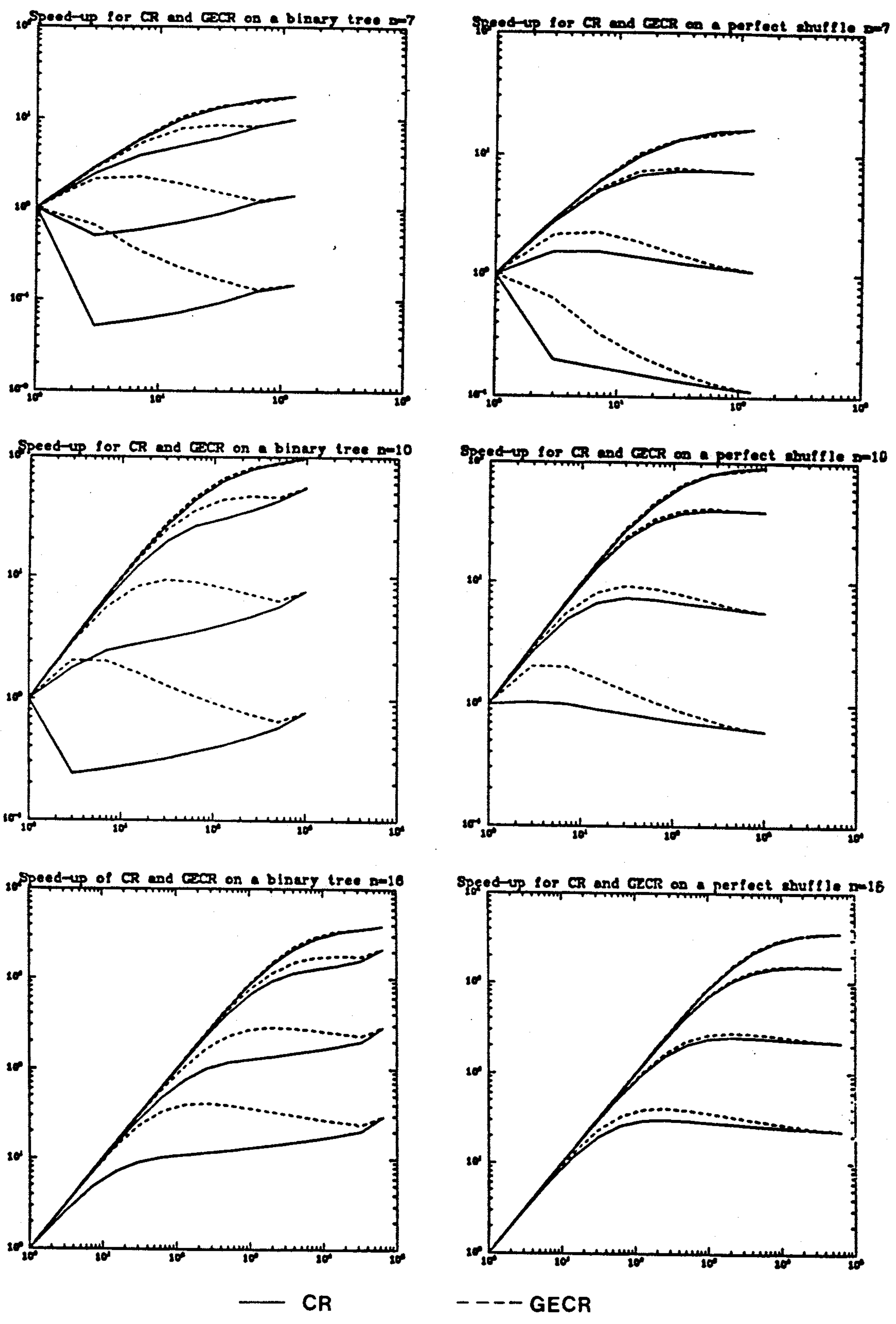


Figure 10-5: Speed-up for CR and GECR on a binary tree and perfect shuffle, $N=2^7$-1, 2^{10}-1, 2^{16}-1 α=0.1,1,10,100

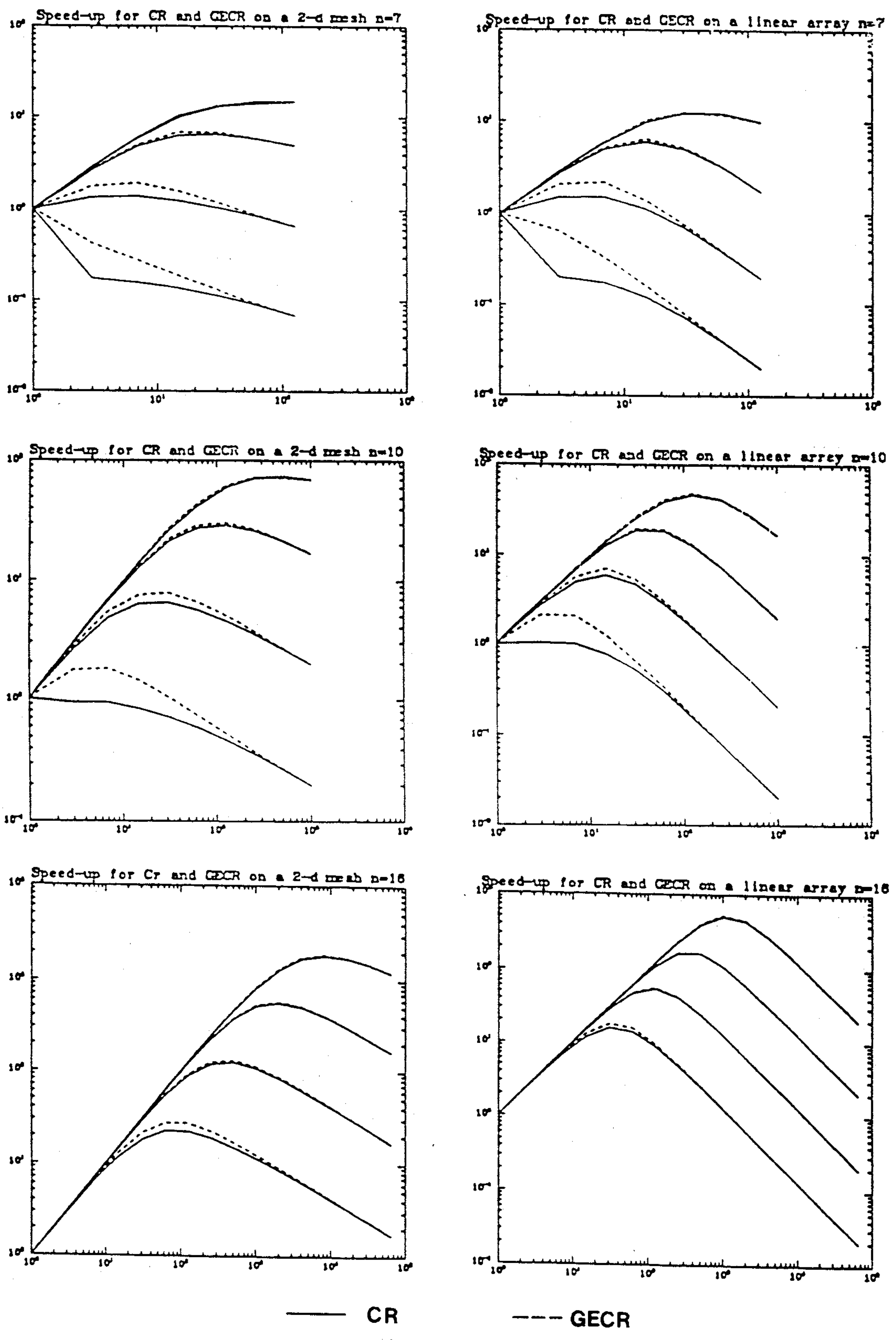


Figure 10-6: Speed-up for CR and GECR on a 2-d mesh and a linear array, $N=2^7-1,\ 2^{10}-1,\ 2^{16}-1$

$\alpha=0.1$, 1, 10, and 100

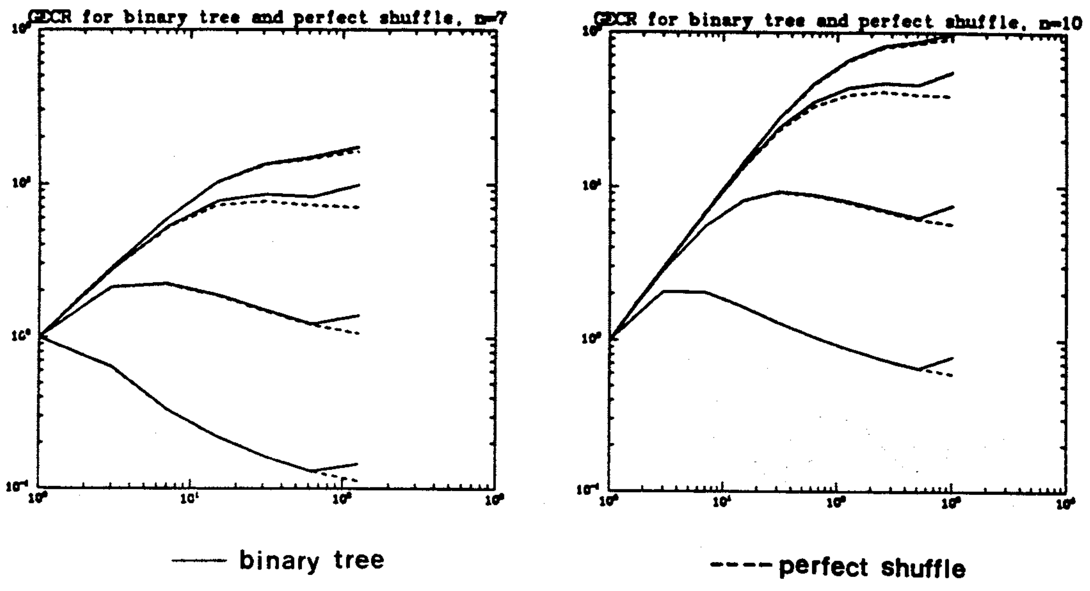


Figure 10-7: Speed-up for GECR on a binary tree and perfect shuffle, N=2^7-1, 2^{10}-1 α=0.1, 1, 10, and 100

This complexity is higher than for the binary tree, the shuffle exchange, the perfect shuffle, and the k-cube by a factor of $\log_2 K$. Figure 10-8 shows the speed-up for N=127 and N=1023 as a function of the number of processors in the ensemble. The speed-up for the perfect shuffle (dashed lines) is included for comparison. Figure 10-9 shows the relative merit of GECR.

Improved performance can be obtained by furnishing the processors with local storage to reduce the need to send data through the switch. The time to solve a tridiagonal system can also be reduced by choosing a different method that allows effective use of pipelining, such as Gaussian elimination, or a combination of different methods.

11. Conclusions

A $O(\log_2 N)$ algorithm is presented for odd-even cyclic reduction on N equations on four processor networks of diameter $\log_2 N$: a binary tree, a shuffle-exchange network, a perfect shuffle network, and a boolean k-cube. The binary tree algorithm is of a slightly lower complexity than the algorithms for the other forms of interconnect with one equation per processor. But the communication complexity makes it inferior for multiple equations per processor. The algorithms for the 2-dimensional mesh and the linear array have an identical arithmetic complexity, but the communication complexity is higher due to the larger diameter of the ensemble.

Truncation of the reduction process offers a greater advantage for a highly parallel architecture

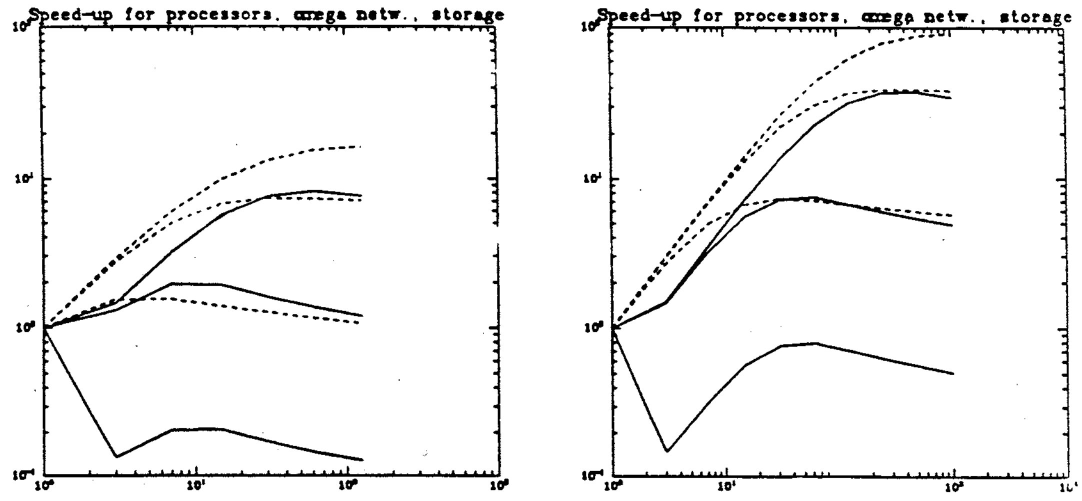
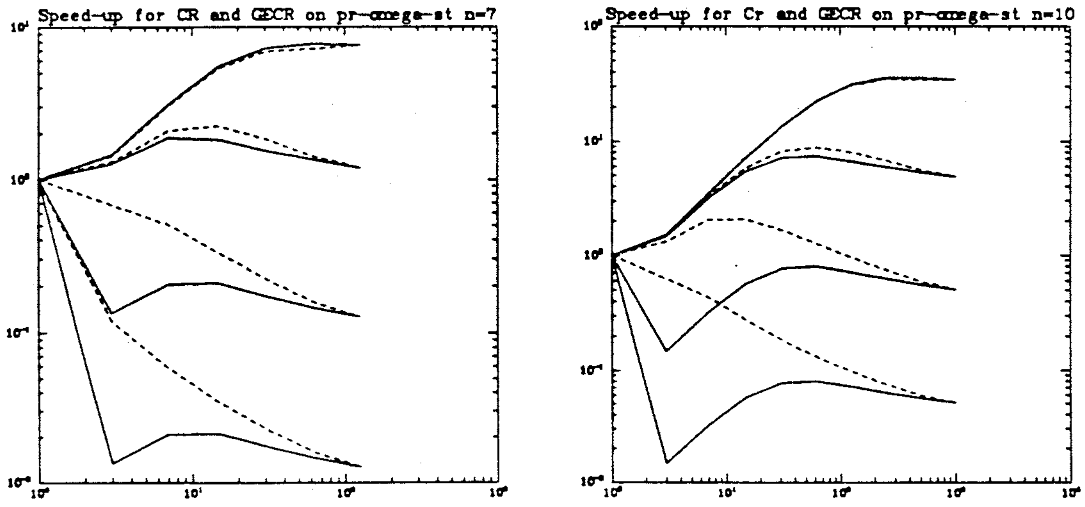


Figure 10-8: Speed-up on processor-Ω-network-storage config. for N=127 and N=1023 α=0.1, 1, 10



α=0.1, 1, 10, and 100

Figure 10-9: Speed-up for CR and GECR on proc.-Ω-network-stor., N=127, and N=1023

than on a uniprocessor. The complexity of the tree algorithm is not reduced to the same extent as the complexity of the algorithms for the perfect shuffle and k-cube interconnect if truncation of the reduction is possible. These other forms of interconnect can be more effective for truncated cyclic reduction.

Domain decomposition is superior to cyclic partitioning. If the number of partitions is odd, then the parallel arithmetic complexity is the same for cyclic partitioning and domain decomposition, but the communication complexity is higher. Cyclic partitioning into 2^k partitions is particularly bad, yielding considerably higher communication complexity as well as poor distribution of computations, with substantially increased time for arithmetic as a consequence. With multiple equations per processor the tree is inferior to the perfect shuffle and k-cube interconnect. The difference in performance between the algorithms for the tree, the perfect shuffle and the k-cube depend on the cost of communication relative to the cost of arithmetic, the problem size, and the number of processors. The difference may be significant for a large relative communication cost.

The tree algorithm for cyclic reduction employs a proximity preserving map of equations to processors for $N>K$. When the reduction process has progressed to the point where there is one equation per processor, then a switch to an inorder map is made. A static map yields the complexity on the other topologies. The GECR method is efficient even with a static map of partitions to the binary tree. Using GECR, the tree offers the same performance as the perfect shuffle and k-cube for $N>K$.

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 communication is an order of magnitude slower than arithmetic then Gaussian elimination is of a lower time complexity. Hence, on a linear array it may be preferable to use Gaussian elimination in parallel in each processor, followed by Gaussian elimination across the linear array.

For a large problem the speed-up is approximately linear in the number of processors for any form of interconnect. The difference between different forms of interconnect become significant if the number of processors exceeds a number that depends on relative communication cost as well as problem size. For a linear array the maximum speed-up is obtained for $K \sim \sqrt{(N/\alpha)}$, for a 2-dimensional mesh for $K \approx ((N/2\alpha)^2)^{1/3}$, and for the perfect shuffle and k-cube for $K \approx N/(1+\alpha)$, where $\alpha = t_c/t_a$. The minimum time is $O(\log_2 N)$ for the topologies of diameter $\log_2 N$, $O(N^{1/3})$ for the 2-dimensional mesh, and $O(\sqrt{N})$ for the linear array. The minimum time complexity is of the same order for both cyclic reduction and GECR. The communication complexity is $O(\log_2 N)$ for cyclic reduction and $O(\log_2 K)$ for GECR. The arithmetic complexity is approximately the same.

Partitioning of the ensemble into subsystems is shown to be more efficient than pipelining of computations, if a number of independent problems should be solved.

Though the algorithms make use of the MIMD feature of the architecture, only a few different codes are used. In the 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.

12. Acknowledgement

This report has benefited from stimulating discussions with Stanley C. Eisenstat, Michael J. Fischer (who suggested the embedding of the binary tree in the shuffle-exchange network), Abhiram Ranade, and Yousef Saad. Thanks is also due to Andrea Pappas for her careful reading of the manuscript.

The author also gratefully acknowledges the generous support of the Office of Naval Research under contract N00014-84-K-0043.

I. Appendix

Algorithms

A 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):

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

# Backsubstitution
```

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

A shuffle-exchange network algorithm

Processor(1):

```
# Root of the binary tree algorithm
# i is the equation number assigned to the processor
# $a_i, b_i, c_i, y_i$ are the coefficients and right hand side assigned to the processor

    $x_0 := 0$; $x_{N+1} := 0$
    $k := 1$
    $m := i/(2k)$

# Reduction computations

    while m is even do

# receive data fom the left child, left cyclic shift is down the tree
        receive (al, bl, cl, yl, $a_{i-k}$, $b_{i-k}$, $c_{i-k}$, $y_{i-k}$) from processor (00..10)
# receive data from the right child
        receive ($a_{i+k}$, $b_{i+k}$, $c_{i+k}$, $y_{i+k}$, ar, br, cr, yr) from processor (00...10)

# reduction
        $e_i := -a_i/b_{i-k}$
        $f_i := -c_i/b_{i+k}$
        $a_i := e_i a_{i-k}$
        $c_i := f_i c_{i+k}$
        $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
        $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
        $k := 2k$
        $m := m/2$
    enddo

# the last reduction step

    for m odd do

# receive data from the left child, left cyclic shift is down the tree
        receive ($a_{i-k}$, $b_{i-k}$, $c_{i-k}$, $y_{i-k}$) from processor (00...10)
# receive equations from the right child
        receive ($a_{i+k}$, $b_{i+k}$, $c_{i+k}$, $y_{i+k}$) from processor (00...10)
# reduction
        $e_i := -a_i/b_{i-k}$
        $f_i := -c_i/b_{i+k}$
        $a_i := e_i a_{i-k}$
        $c_i := f_i c_{i+k}$
        $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
        $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$

    enddo
```

```
# backsubstitution

        $x_i := y_i/b_i$

        send $(x_{i-2k}, x_i, x_{i+2k})$ to processor (00...10)
# left cyclic shift is down the tree
```

Processors($0p_{k-2}...p_1p_0$):

```
# except processors 0 and 1
# i is the equation number assigned to the processor
# $a_i, b_i, c_i, y_i$ are the coefficients and the right hand  side assigned to the processor

        k := 1
        m := i/(2k)

# Reduction phase

        while m is even do
# receive data from the left child, left cyclic shift is down in the tree
                receive (al, bl, cl, yl, $a_{i-k}$, $b_{i-k}$, $c_{i-k}$, $y_{i-k}$) from processor ($p_{k-2}p_{k-3}...p_0 0$)
# receive data from the right child
                receive ($a_{i+k}$, $b_{i+k}$, $c_{i+k}$, $y_{i+k}$, ar, br, cr, yr) from processor ($p_{k-2}p_{k-3}...p_0 0$)

# forward equations received from children to parent, via the corresponding even processor
                if $p_0$=1 then
                    send (al, bl, cl, yl, ar, br, cr, yr) to processor ($0p_{k-2}...p_1 0$)
                endif

# forward equations received from children to parent via the corresponding even processor
# forward equations from right child to parent, right cyclic shift is up the tree

                if $p_0$=0 then
                    send (al, bl, cl, yl, ar, br, cr, yr) to processor ($00p_{k-2}...p_1$)
                    receive (al, bl, cl, yl, ar, br, cr, yr) from processor ($0p_{k-2}...p_1 1$)
                    send (al, bl, cl, yl, ar, br, cr, yr) to processor ($00p_{k-2}...p_1$)
                endif

# reduction
                $e_i := -a_i/b_{i-k}$
                $f_i := -c_i/b_{i+k}$
                $a_i := e_i a_{i-k}$
                $c_i := f_i c_{i+k}$
                $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
                $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
                k := 2k
                m := m/2
        enddo
```

```
# last reduction step in which the processor participates

      for m odd do

# receive data from the left child, left cyclic shift is down the tree
            receive ($a_{i-k}$, $b_{i-k}$, $c_{i-k}$, $y_{i-k}$) from processor ($p_{k-2}...p_0 0$)
# receive data from the right child
            receive ($a_{i+k}$, $b_{i+k}$, $c_{i+k}$, $y_{i+k}$) from processor ($p_{k-2}...p_0 0$)

# forward equations received from children to parent via the corresponding even processor
            if $p_0=1$ then
            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 processor ($0p_{k-2}...p_1 0$)
            endif

# forward equations received from children to parent via the corresponding even processor
# forward equations from right child to parent, right cyclic shift is up the tree

            if $p_0=0$ then
            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 processor ($00p_{k-2}...p_1$)
            receive ($a_{i+3k}$, $b_{i+3k}$, $c_{i+3k}$, $y_{i+3k}$, $a_{i+5k}$, $b_{i+5k}$, $c_{i+5k}$, $y_{i+5k}$) from proc. ($0p_{k-2}...p_1 1$)
            send ($a_{i+3k}$, $b_{i+3k}$, $c_{i+3k}$, $y_{i+3k}$, $a_{i+5k}$, $b_{i+5k}$, $c_{i+5k}$, $y_{i+5k}$) to processor ($00p_{k-2}...p_1$)
            endif

# reduction
            $e_i := -a_i/b_{i-k}$
            $f_i := -c_i/b_{i+k}$
            $a_i := e_i a_{i-k}$
            $c_i := f_i c_{i+k}$
            $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
            $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$

# send the reduced equation computed locally to the parent via the corresponding even processor
            if $p_0=1$ then
                  send ($a_i$,$b_i$,$c_i$,$y_i$) to processor ($0p_{k-2}...p_1 0$)
            endif

# send the reduced equation computed locally to the parent
# forward the reduced equation computed by the right child to the parent
# right cyclic shift is up the tree
            if $p_0=0$ then
                  send ($a_i$,$b_i$,$c_i$,$y_i$) to processor ($00p_{k-2}...p_1$)
                  receive ($a_{i+2k}$,$b_{i+2k}$,$c_{i+2k}$,$y_{i+2k}$) from processor ($0p_{k-2}...p_1 1$)
                  send ($a_{i+2k}$,$b_{i+2k}$,$c_{i+2k}$,$y_{i+2k}$) to processor ($00p_{k-2}...p_1$)
            endif
      enddo

# backsubstitution

# receive the x-values required for both the left and right subtrees of the parent
# forward the x-values needed by the right subtree to the corresponding odd processor
    if $p_0$=0 then
        receive ($x_{i-2k}$, $x_{i+2k}$, $x_{i+6k}$) from processor ($00p_{k-2}...p_1$)
        send ($x_{i+2k}$, $x_{i+6k}$) to processor ($0p_{k-2}...p_1 1$)
    endif

# receive the x-values for the right subtree, forwarded by the corresponding even processor
    if $p_1$=1 then
        receive ($x_{i-2k}$, $x_{i+2k}$) from processor ($0p_{k-2}...p_1 0$)
    endif

    $x_i := (y_i - a_i x_{i-2k} - c_i x_{i+2k})/b_i$

# send necessary x-values down the tree
    send ($x_{i-2k}$, $x_i$,$x_{i+2k}$) to processor ($p_{k-2}p_{k-3}...p_0 0$)
```

Processors($1p_{k-2}...p_1p_0$):

```
# The leaf nodes are all mapped on to processors with bit $p_{k-1}$=1.
# send the equation stored locally to the parent via the corresponding even processor
# wait to receive x-values from parent via the corresponding even processor
    if $p_0$=1 then
        send ($a_i$, $b_i$, $c_i$, $y_i$) to processor ($1p_{k-2}...p_1 0$)
        receive ($x_{i-1}$, $x_{i+1}$) from processor ($1p_{k-2}...p_1 0$)
    endif

# send the equation stored locally to the parent
# forward the equation from the right child to the parent
# wait to recieve x-values for itself and the corresponding right child
# forward necessary x-values to the right child
    if $p_0$=0 then
        send ($a_i$, $b_i$, $c_i$, $y_i$) to processor ($01p_{k-2}...p_1$)
        receive ($a_{i+2}$, $b_{i+2}$, $c_{i+2}$, $y_{i+2}$) from processor ($1p_{k-2}...p_1 1$)
        send ($a_{i+2}$, $b_{i+2}$, $c_{i+2}$, $y_{i+2}$) to processor ($01p_{k-2}...p_1$)
        receive ($x_{i-1}$, $x_{i+1}$, $x_{i+3}$) from processor ($01p_{k-2}...p_1$)
        send ($x_{i+1}$, $x_{i+3}$) to processor ($1p_{k-2}...p_1 1$)

    endif

    $x_i := (y_i - a_i x_{i-1} - c_i x_{i+1})/b_i$
```

A boolean k-cube algorithm

```
# Let $(g_k g_{k-1} \ldots g_1)$ 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
# 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

# determine the integer m that the k-j highest order bits encode, $(G_m(k-j)111)$
        $m := dec(g_k g_{k-1} \ldots g_{j+1})$

# determine the address of the processors holding preceeding and succeeding
# integers in the k-j dimensional cube

    $G_{m+1}(k-j) := enc(m+1)$
    $G_{m-1}(k-j) := enc(m-1)$

# even processors in the k-j subcube send their equations to the processors
# holding preceeding and succeeding odd integers, except the processor holding
# m=0 only communicates with the processor holding m=1, and the processor holding
# $m = 2^{k-j}-2$ only communicates with the processor holding m-1
# processors holding odd integers communicates and performs reduction computaions
        if m even and m>0 then
            send $(a_i, b_i, c_i, y_i)$ to the processor $(G_{m-1}(k-j)11 \ldots 1)$
    if m even and $m < 2^{k-j}-2$ then
            send $(a_i, b_i, c_i, y_i)$ to the processor $(G_{m+1}(k-j)11 \ldots 1)$
    if 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 $(G_{m-1}(k-j)11 \ldots 1)$
            receive $(a_{i+2^j}, b_{i+2^j}, c_{i+2^j}, y_{i+2^j})$ from processor $(G_{m+1}(k-j)11 \ldots 1)$

# reduction computation in processors with m odd

            $e_i := -a_i/b_{i-k}$
            $f_i := -c_i/b_{i+k}$
            $a_i := e_i a_{i-k}$
            $c_i := f_i c_{i+k}$
            $b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
            $y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
    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 ($g_k g_{k-1} .. g_{j+2} 111...1$)
                receive($a_i,b_i,c_i,y_i$) from the processor ($g_k g_{k-1} .. g_{j+2} 111...1$)
            endif

            if m is even and $g_{j+1}=1$ then
                send($a_i,b_i,c_i,y_i$) to the processor ($g_k g_{k-1} .. g_{j+2} 011...1$)
                receive($a_i,b_i,c_i,y_i$) from the processor ($g_k g_{k-1} .. g_{j+2} 011...1$)
            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...1) computing $x_{2^{k-1}-1}$
# processors (0011...1) and (1111...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 increase
# the dimensions of the cube, and make the proper exchanges (reversing the
# exchanges in the reduction phase.

    while $j \geq 0$ 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...1)

            if $m=1$ then
                $x_i := y_i/b_i$
                send($x_{-1},x_i$) to processor ($G_{m-1}11...1$)
                send($x_i,x_K$) to processor ($G_{m+1}11...1$)
            endif

# solve for $x_{2^{k-2}}$

            if $m=0$ then
                receive($x_{i-2^{k-2}},x_{i+2^{k-2}}$) from processor ($G_{m+1}11...1$)
                $x_i := (y_i - a_i x_{i-2^{k-2}} - c_i x_{i+2^{k-2}})/b_i$
            endif

# solve for $x_{3*2^{k-2}}$
```

```
          if m=2 then
                receive($x_{i-2^{k-2}}$,$x_{i+2^{k-2}}$) from processor ($G_{m-1}$11...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 occured during the reduction phase

     j := j-1
     m := dec($g_k g_{k-1} \ldots g_{k-j}$)
     $G_{m-1}$ :=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} .. g_{j+2}$111...1)
          receive($a_i$,$b_i$,$c_i$,$y_i$,$x_i$) from the processor ($g_k g_{k-1} .. g_{j+2}$111...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} .. g_{j+2}$011...1)
          receive($a_i$,$b_i$,$c_i$,$y_i$) from the processor ($g_k g_{k-1} .. g_{j+2}$011...1)
     endif

# processors with m even receives $x_{i-2^{j+1}}$ from processor ($G_{m-1}$(k-j)11...1) except if m=0
# processors with m even receives $x_{i+2^{j+1}}$ from processor ($G_{m+1}$(k-j)11...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}$11...1)
          send($x_i$) to processor ($G_{m+1}$11...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...1)
          if $m<2^{k-j}-2$ then
                receive($x_{i+2^{j+1}}$) from processor ($G_{m+1}$(k-j)11...1)
          $x_i := (y_i - a_i x_{i-2^{j+1}} - c_i x_{i+2^{j+1}})/b_i$
     endif
  enddo
```

References

[Aho,Hopcroft,Ullman 74]
Aho A.V., Hopcroft J.E., Ullman J.D.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.

[Browning 80] Browning S.A.
The Tree Machine: A Highly Concurrent Computing Environment.
Technical Report 1980:TR:3760, Computer Science, California Institute of Technology, January, 1980.

[Browning,Seitz 80]
Browning S.A., Seitz C.L.
Communication in a tree machine.
In *Proceedings, Second Conference on Very Large Scale Integration*, pages 509-526. Computer Science, California Institute of Technology, 1980.

[Buzbee,Golub,Nielson 70]
Buzbee,B.L., Golub, G.H., Nielson, C.W.
On Direct Methods for Solving Poissons's Equations.
SIAM J. Numer Anal 7(4):627-656, December, 1970.

[Flynn 66] Flynn M.J.
Very High-Speed Computing Systems.
Proc. of the IEEE 12:1901-1909, 1966.

[Gentleman 78] Gentleman W.M.
Some Complexity Results for Matrix Computations on Parallel Processors.
J. ACM 25(1):112-115, January, 1978.

[Gottlieb et.al. 83]
Gottlieb A., Grishman R., Kruskal C.P., McAuliffe K.P., Rudolph L., Snir M.
The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer.
IEEE Trans. Computers C-32(2):175-189, 1983.

[Hockney, Jesshope 81]
Hockney R.W., Jesshope C.R.
Parallel Computers.
Adam Hilger, 1981.

[Leighton 83] Leighton F.T.
Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.
MIT Press, 1983.

[Leiserson 82] Leiserson, C.E.
Area-Efficient VLSI Computation.
MIT Press, 1982.

[Li,Johnsson 83] Li, P., Johnsson, L.
The Tree Machine: An evaluation of program loading strategies.
In *1983 International Conference on Parallel Processing*, pages 202 - 205. IEEE Computer Society, August, 1983.

[Mago 79] Mago, G.A.
A Network of Microprocessors to Execute rreduction Languages.
J. of Computer and System Sciences 8(6):435-471, 1979.

[Presnell, Pargas 81]
Presnell, H.A., Pargas, R.P.
Communication Along Shortest Paths in a Tree Machine.
In *Proc. of the 1981 Conference on Functional Programmming Languages and Computer Architecture*, pages 107-114. ACM, 1981.

[Reingold,Nievergelt,Deo 77]
Reingold E.M., Nievergelt J., Deo N.
Combinatorial Algorithms.
Prentice Hall, 1977.

[Rosenberg, Snyder 78]
Rosenberg A.L., Snyder L.
Bounds on the Costs of Data Encodings.
Mathematical Systems Theory 12:9-39, 1978.

[Sameh 84] Lawrie D.H.,Sameh A.H.
The Computational Complexity and Communication Complexity of a Parallel Banded System Solver.
ACM Trans. Math. Software 10(2):185-195, June, 1984.

[Schwartz 80] Schwartz J.T.
Ultracomputers.
ACM Trans. on Programming Languages and Systems 2:484-521, 1980.

[Seitz 84] Seitz C.L.
The Cosmic Cube.
CACM : , 1984.

[Seitz et.al. 84] Lutz C., Rabin S., Seitz C.L., Speck D.
Design of the Mosaic Element.
In *Proceedings, Conf. on Advanced research in VLSI*, pages 1-10. Artech House, 1984.

[Sejnowski et.al. 80]
Sejnowski M.C., Upchurch E.T., Kapur R.N., Charlu D.P.S., Lipovski G.J.
An Overview of the Texas Reconfigurable Array Computer.
In *Proceedings, National Computer Conference*, pages 631-641. IEEE, 1980.

[Sekanina 60] Sekanina,M.
On an ordering of the set of Vertices of a Connected Graph.
Publ. of the Faculty of Science of the University of Brno (142):137-142", 1960.

[Shaw 82] Shaw D.
The NON-VON Supercomputer.
Technical Report, Dept. of Computer Science, Columbia University, August, 1982.

[Shaw 84] Shaw D.
SIMD and MSIMD Variants of the NON-VON Supercomputer.
Technical Report, Dept. of Computer Science, Columbia University, 1984.

[Stone 71] Stone H.S.
Parallel Processing with the Perfect Shuffle.
IEEE Trans. on Computers C-20(2):153-161, February, 1971.

[Wang 81] Wang H.H.
A Parallel Method for Tridiagonal Equations.
ACM Trans. Math. Software 7(2):170-183, June, 1981.