Highly Concurrent Algorithms for Solving Linear Systems of Equations

Lennart Johnsson

Computer Science Department
California Institute of Technology

5079:TR:83

CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science

5079:TR:83

HIGHLY CONCURRENT ALGORITHMS FOR SOLVING LINEAR SYSTEMS OF EQUATIONS

by

Lennart Johnsson

Presented at
Monterey Conference on Elliptic Problem Solvers
January 1983

The research in this report was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order 3771, and monitored by the Office of Naval Research under Contract N00014-79C-0597

and in part by the System Development Foundation

California Institute of Technology

HIGHLY CONCURRENT ALGORITHMS FOR SOLVING LINEAR SYSTEMS OF EQUATIONS

Lennart Johnsson

Computer Science
California Institute of Technology
Pasadena, CA 91125

I. INTRODUCTION

There is a growing concern over the reduced rate at which computers of increased performance are being developed for large scale scientific computation, [Lax 82]. Buzbee, [Buzbee 81], presents statistics showing a decrease in the rate of increase in performance from more than two orders of magnitude per decade around 1960 to presently less than one order of magnitude per decade. To significantly change the trend concurrency is necessary. Here we will discuss algorithms for computers built out of several processors of approximately equal complexity. The number of processors may be very large. A taxonomy of ensemble architectures is given by Seitz, [Seitz 82].

The machines envisioned are made up of a thousand to tens of thousands of processors with their own local storage, sparsely interconnected and possibly also connected to some form of auxiliary storage. Two construction projects are currently under way at Caltech based on this metaphor, the Tree Machine project, [Browning 80], [Browning, Seitz 81], [Li 81], and the Homogeneous Machine project, [DeBenedictis, Seitz 82], [Fox, Seitz 83]. These two projects are architectural experiments with a significant difference in processor architecture, size, machine topology, and programming model. Both machines use message passing to synchronize computations in neighboring processors.

The Tree Machine node processors are intentionally kept very simple, the emphasis being on efficient communication and a large number of processors. In our first design a 16-bit processor is used with floating point in software and 1-2k words of local storage. Even though such a processor is very limited, a 1000 chip machine would have 8000 processors and 6M 16-bit words of storage in 2 micron technology. Such a machine would have an estimated capacity of about 0.5 Gflops.

The Homogeneous Machine node is currently based on standard parts, an Intel 8086 processor with an 8087 floating point processor, with additional chips for handling the communication. The node that today requires a small PC board is expected to fit on a single chip at 1/2 micron technology. At 5 MHz clock rate each node has been benchmarked as having 1/6 of the capacity of a VAX 11/780 for certain PDE calculations.

Node processors of a significantly smaller size than those used in the Tree Machine are being studied for special functions, such as those typical in computational linear algebra. Programming consists of changing the data flow within the network of processing elements. Instructions as well as data can be pipelined through the network.

To be able to use the processing power of machines such as the Tree Machine, the Homogeneous Machine, or any other machine with several processors, e.g. the CHiP machine [Snyder 82], or the Ultra Computer, [Schwartz 80], [Gottlieb, Schwartz 82], a good match is necessary between the communication requirements of the algorithms and the topology and bandwidth of communication paths of the machine. The quality of an algorithm can no longer be judged by the number of arithmetic operations.

Much of the analysis of concurrent algorithms has been based on the assumption of unrestricted communication. Excellent surveys are given by Sameh, [Sameh 77], and Heller, [Heller 78]. Results derived under such assumptions can serve as lower bounds against which the merits of various interconnection networks in multiprocessor systems can be judged.

Recent work has focused on devising algorithms with a high degree of concurrency under the restriction of limited communication, [Kung, Leiserson 80], [Kung 80], [Johnsson 81], [Heller, Ipsen 82], [Schreiber 82], etc. Indeed the topology has typically been restricted to regular graphs with interior nodes of degree 4 or 6 (hexagonal arrays). Algorithms for binary tree topologies have been devised by, e.g., Bentley and Kung, [Bentley, Kung 79], Browning, [Browning 80], and Presnell and Pargas, [Presnell, Pargas 81].

Systolic algorithms, like many tree machine algorithms, are very communication intensive. The goal is to maximize concurrency. One communication action per channel per multiply/add operation is often the case. A few registers are typically sufficient for data storage in each node. A few examples are given below. Such algorithms are efficient whenever the time for a communication action is comparable to a local storage reference.

For large problems the assumption of unbounded concurrency, for example a processor per matrix element, is often unrealistic. Computations must be sequenced through a smaller number

of processors. Storage can either be external to the network of processors, each with very limited capacity for data storage, or distributed among the processing nodes. There are advantages and disadvantages to each approach. Obviously the allocation of storage and its organization has algorithmic consequences, [Johnsson 81], [Johnsson 82a], [Heller 82], [Schreiber, Keukes 82].

In the following we will focus on some concurrent algorithms for the conjugate gradient method. The effect of preconditioning on concurrency is discussed and a few algorithms presented. First, a brief, by no means complete, survey of some recent work on concurrent algorithms for the solution of linear systems of equations through factorization of the system matrix will be given. We write a linear system of equations as $Ax = y$ where A is a N by N nonsingular matrix and x and y vectors of size N. Algorithms for LU- as well as QR-decomposition are discussed. Solution methods based on fast Fourier Transforms or cyclic reduction are not included. In the following logx denotes $\log_2 x$.

Detailed analysis of direct hardware (VLSI) implementations of FFT algorithms have been carried out in [Despain 79] and [Johnsson, Cohen 82]. The latter reference contains an algebraic treatment of FFT algorithms partially implemented in space, partially in time. Stone, [Stone 71], has described an O(logN) time FFT algorithm for shuffle networks, as well as O(logN) time algorithms for tridiagonal equations, [Stone 73], [Stone 75]. An $O(\log^2 N)$ cyclic reduction algorithm on a tree is given in [Presnell, Pargas 81].

II. LU-DECOMPOSITION

Concurrent algorithms for Gaussian elimination on unbounded hexagonally interconnected arrays have been presented by Kung, [Kung, Leiserson 80] and others, e.g., [Kung 80]. These algorithms have the characteristic that one communication action per channel is required for every multiply/add operation. A few data registers are required for each arithmetic unit. All interior nodes compute the same function. The boundary nodes compute different functions, but sets of them compute the same function. The algorithms in [Kung, Leiserson 80] are given for the band matrix case. With u nonzero subdiagonals, v nonzero superdiagonals and bandwidth w=u+v+1, uv+w processors are needed to compute L and U in 3N+min(u,v) time. Algorithms that fit arrays of degree 6 arise naturally when the band structure of banded matrices is exploited, [Johnsson 81]. For banded matrices all computations except those in a window of size uv+w are trivial. It has previously been observed for vector machines that computations can advantageously be organized with diagonals as a data abstraction, instead of rows or columns, [Madsen, Rodrigue, Karushi 76].

For a full matrix N^2 processors are required. In that case computations can be organized so that a network with interior processors of degree 4 offers sufficient communication capabilities. Several such algorithms can be devised. An algorithm with A, L, and U resident in a mesh of interconnected processors is implicit in the inversion algorithm described in [Flanders et al 77]. An algorithm computing the factorization as the matrix enters a mesh is easily devised for a data flow similar to that employed by Gentleman and Kung for Given's rotations, [Gentleman, Kung 81]. Algorithms that match a mesh of degree 4 are easily mapped to execute efficiently on Boolean n-cubes. It is also possible to devise band matrix algorithms for a mesh with interior nodes of degree 4, but explicit control is typically required. Partial products are accumulated in place.

The methods of Cholesky, Crout, and Doolitle are closely related to Gaussian elimination. Concurrent algorithms for these methods have been devised with the same communication topology as concurrent algorithms for Gaussian elimination. An algorithm for Cholesky's method based on matrix splitting, called hyperbolic Cholesky, is presented in [Delosme, Morf 81], [Ahmed, Delosme, Morf 82]. A direct, concurrent, factorization is described in [Johnsson 82b], that also contains algorithms for the methods of Crout and Doolittle. The node complexities are comparable. A graphical illustration of the function of an interior node in an array for Gaussian elimination with features for partial pivoting and matrix-matrix multiplication is given in Figure 1, [Johnsson 81]. Figure 2, [Johnsson 82b], shows an interior cell for Cholesky's method.

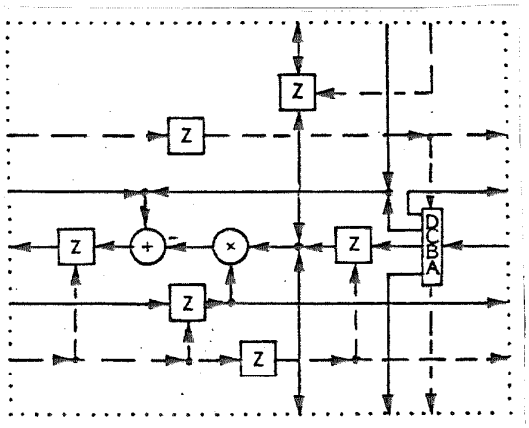

Figure 1: An interior cell for Gaussian elimination

All the above algorithms can be considered as vectorized versions of common sequential algorithms. The speed-up is proportional to the number of processors, but the concurrent algorithms do not necessarily compute a solution in the minimum time possible. For instance, for banded matrices the

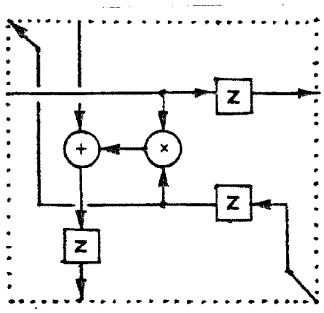

Figure 2: An interior cell for Cholesky's method

above algorithms require O(N) time irrespective of the bandwidth. A uniprocessor can also compute the solution in O(N) time, see, e.g., [Stone 75]. However, algorithms for solving tridiagonal equations in O(logN) time on N processors interconnected in a shuffle network are known, [Stone 73], [Stone 75], as well as $O(\log^2 N)$ algorithms for 2N binary tree interconnected processors, [Presnell, Pargas 81].

The concurrent algorithms above are usually described in their fully pipelined versions. The factorization operations use rows as characteristic entities. Computations on successive row elements are pipelined, as are the row operations for elimination of successive column entries below the diagonal in one column. In addition, the elimination operations for successive columns are pipelined. Pivoting on the diagonal results in the elements of columns and rows moving in arrowhead like formation through the array, if the hexagonally interconnected array is drawn as a rectangular array, Figure 3 [Kung 80], [Johnsson 81]. Pipelining is used to avoid broadcasting, which in general is undesirable in VLSI. Without pipelining in the row and column directions the number of time steps for the factorization is reduced to N+min(u,v) instead of 3N+min(u,v). However, the duration of such a step will be significantly longer, except for very small arrays, [Johnsson 81].

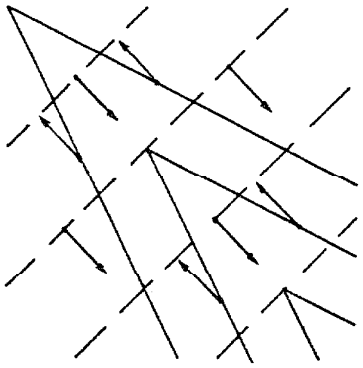

Figure 3: Wave fronts in a fully pipelined array performing Gaussian elimination

The pipelined mesh does not perform well if partial pivoting is needed. The elements in a column are computed at successive instances of time and for a full matrix the time to find the next pivot element and restart the computations goes from O(N) for the first column to O(1) for the last. For a band matrix the time for pivoting is O(u). Hence, a linear array will perform as well as a two-dimensional mesh connected array, if partial pivoting is used. Interprocessor communication limits the performance. Adding a binary tree interconnection within columns of the array would allow Gaussian elimination with partial pivoting to be performed in O(NlogN) time on $O(N^2)$ processors.

If the problem to be solved is of such a size that it is impossible to associate one processor with each element within the active window different algorithms are required. The total amount of storage is determined by the problem size. The storage can either be distributed among the network of processing elements or be external to it to keep the processing nodes small. Block algorithms using external storage are described in [Johnsson 81]. These block algorithms perform a factorization of a diagonal block followed by the proper matrix-matrix multiplications on the block row in which the diagonal element was just factorized. The blocks below the diagonal block are then eliminated and the remaining submatrix finally updated. The computations for different steps of the block algorithm can be pipelined, rendering a third level of pipelining.

The report, [Johnsson 81], also describes an algorithm employing distributed storage. This algorithm can be considered as a "sweeping" algorithm in that a sweep is made over successive columns and/or rows. Both the block and sweeping algorithms have the same numerical properties as Gaussian elimination with pivoting on the diagonal elements. Subdiagonal stripping algorithms as proposed for Given's rotations by [Heller, Ipsen 82] can be devised for Gaussian elimination as well, but will have entirely different numerical properties irrespective of which row is used for pivoting.

An algorithm for block Gaussian elimination, with and without partial pivoting, on a torus is described by Fox, [Fox 82]. This algorithm communicates a vector in each communication action. It is based on the assumption that the bandwidth for interprocessor communication is significantly less than the bandwidth to local storage. The time complexity given by Fox for Gaussian elimination with partial pivoting on a torus can be improved by a factor of $O(2^n/n)$ using the richer interconnect of a Boolean n-cube.

III. QR-DECOMPOSITION

Where the numerical properties of the problem are such that partial pivoting is required in Gaussian elimination, stable methods such as Given's rotations and Householder reflections

are often preferred. Concurrent algorithms requiring one communication action per multiply/add operation for Given's method have been proposed by [Gentleman, Kung 81], [Delosme, Morf 81], [Heller, Ipsen 82] and [Johnsson 82c]. See also [Ahmed, Delosme, Morf 82]. In the algorithm described by Gentleman and Kung the upper triangular matrix R resides in the array after the computations. All rotations required to eliminate the first column are handled by the first (top) row of the array. Similarly, the second column is handled by the second row, etc. The array size is N(N+1)/2. A processor on the matrix input side is associated with a column of the matrix.

Delosme and Morf associate a boundary processor with each row of the matrix. Heller and Ipsen employ a subdiagonal stripping strategy requiring w processors for each subdiagonal of a banded matrix, and hence uw processors for the complete factorization. Johnsson uses the column elimination strategy frequently used in Gaussian elimination. The number of processors used is uv+w. The latter two approaches both have an input processor for each diagonal. The computation time is 3N for a single matrix factorization according to [Gentleman, Kung 81] and [Delosme, Morf 81], 3N+min(u,v) for [Johnsson 82c], and 2(N+u+v) for [Heller, Ipsen 82]. By pipelining several factorizations the factor 3 in the time complexity can be reduced for the algorithms in [Gentleman, Kung 81] and [Delosme, Morf 81].

Given's method is used by Schreiber, [Schreiber 82], [Schreiber, Keukes 82], and [Kung, Gal-Ezer 82], in the QR algorithm for eigenvalue computation. Methods of partitioning large problems to fit on a small array are discussed in [Heller 82], and [Schreiber, Keukes 82].

Given's rotations are based on local operations. The amount of interconnection present in a mesh (nodes of degree 4 or 6) suffice to reduce the computation time with a factor proportional to the number of processors. Householder's method, that has numerical properties similar to Given's, and is often preferred over Given's on sequential machines, requires that a vector norm be computed in every step of the algorithm. A vector norm is a global quantity. A time complexity of O(NlogN) is the best that can be expected. An $O(N^2)$ algorithm is presented in [Johnsson 82d] which also contains a discussion of an O(NlogN) algorithm.

IV. CONJUGATE GRADIENT METHODS

The conjugate gradient method, [Hestenes, Stiefel 52], combined with preconditioning is an interesting alternative to the above methods in solving linear systems of equations, in particular sparse systems, see [Meijerink, van der Vorst 77], [Meijerink, van der Vorst 78], [Kershaw 78], [Munksgaard 79], [Jennings, Malik 78], [Manteuffel 80], [Glowinski et. al. 80], [van der Vorst 82].

The conjugate gradient (CG) method, [Hestenes, Stiefel 52], is defined as follows:

$$a_i = (r_i, r_i)/(p_i, Ap_i)$$

$$x_{i+1} = x_i + a_i p_i$$

$$r_{i+1} = r_i - a_i Ap_i$$

$$b_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$$

$$p_{i+1} = r_{i+1} + b_i p_i$$

with initial condition $r_0 = p_0 = y - Ax_0$

and where A is an N by N positive definite matrix, Ax = y, r residuals and p a direction vector. The vectors x, y, p and r have dimension N, and a and b are scalars. The subscript is the iteration index, and $x_N = x$, where Ax = y.

In the following we will only concern ourselves with the iteration phase of the computations.

The scalar a_i has to be computed before any of the components of x_{i+1} and r_{i+1} can be computed. Similarly b_i has to be known before any of the components of p_{i+1} can be computed. The coefficient a_i can be computed in O(logN) time steps using a tree structure, or in O(N) time using a linear array. The variables a_i and b_i are global. They require information from the entire N-space for their computation. Every step in every direction is dependent on information from every dimension of the space.

For a band matrix of bandwidth w, N binary trees of 2w-1 processors each can be used to compute the product Ap_i in logw steps. Each tree contains a row of A. The quadratic form (p_i, Ap_i) can be computed in another logN steps by making the N trees into subtrees of a tree of 2Nw-1 processors. The leaf nodes each contain one element of A. The products of Ap_i are formed in the leafs. Hence, each subtree contains w components of p_i, and each component of p_i is needed in w subtrees. Also, one component of the p_i is needed in the root of each subtree. The products of (r_i, r_i) can be computed at distance logN from the root one time step prior to the computation of the products of (p_i, Ap_i) and the inner product obtained by performing the summations in passing partial sums towards the root. The value of a_i can be sent down the tree logN levels and x_{i+1} and r_{i+1} computed at that level. For the next iteration it is also necessary to update the direction vector p_i, which requires the computation of b_i. If b_i were calculated directly from the equation above (r_{i+1}, r_{i+1}) would be needed, i.e., another logN steps. These steps can be avoided by rewriting the equation for b_i.

Substituting the equation for r_{i+1} into the expression for b_i and observing the property that $(r_i,r_j) = 0$, $i \neq j$, and $(r_i,Ap_i) = (p_i,Ap_i)$ since $(p_i,Ap_j) = 0$, $i \neq j$, the following equation can be derived

$$b_i = -1 + a_i(Ap_i,Ap_i)/(p_i,Ap_i)$$

The values of a_i and (p_i,Ap_i) are already available in the root of the tree. The value of (Ap_i,Ap_i) can be available in the root one time step after (p_i,Ap_i) by computing the products of this additional inner product after those of (p_i,Ap_i) and passing the terms towards the root. Hence, b_i can be available in the root two steps after a_i, sent down to level logN, and p_{i+1} computed at that level immediately after x_{i+1} and r_{i+1}. Finally, the values of p_{i+1} in the leafs need to be updated.

Each subtree contains w consecutive components of p_i. Each component of p_{i+1} is computed from corresponding components of r_{i+1} and p_i. The proper components of p_i are aleady available in the subtree, but those of r_{i+1} are not. They are available at the roots of adjacent subtrees. Computing w components of r_{i+1} in each subtree from r_i offers no easy escape from the communication problem, since w components of Ap_i are needed. Those components are only available in the same nodes that hold r_{i+1}. A left shift of information by one node at level logN can be accomplished in time 2logN, in a way similar to the rotate operator of Presnell and Pargas, [Presnell, Pargas 81]. Hence, w left shifts can be accomplished in 2wlogN time steps. After an additional logw steps the leafs have received all needed values of r_{i+1}, and p_{i+1} can be computed. The tree is ready for the next iteration. The total time for an iteration of the CG method is 2wlogN+2log(Nw)+2.

The dominating term in this estimate can be reduced to w if a path at level logN from the root is available. The left shifts could then be accomplished in w time steps. With a path through the leafs the time would be reduced to 2wlogw. In both these cases the time complexity is reduced by providing increased communication capability, but without any additional processors. Providing a path through the leafs do not require any processor modification, since spare ports can be used. A path at any other level would require additional processor ports.

If A is the result of a finite difference approximation of differential operators, its nonzero elements will with typical node orderings fall on diagonals. In such a case it is possible to replace w in the previous complexity estimates with the number of nonzero diagonals of A. The trees can be reduced correspondingly.

The processor utilization of this algorithm is fairly low. Increased processor utilization at the expense of a slightly

increased time complexity is obtained by performing the computations for a row of A in one processor. The N subtrees of 2w-1 processors are reduced to a single node. The resulting algorithm will require 2(w+1)logN+w+4 time on a tree of 2N-1 processors. The time complexity can be reduced to 2logN+w+5 by providing a path through the leafs, [Johnsson 82a]. Notice, this algorithm has a lower time complexity than the 2Nw-1 processor algorithm.

If the Tree Machine with logN levels were to be used for these calculations the following sample figures apply. Assume A is of size 1000x1000 with a bandwidth of 30 and that the designs were fabricated in 2 micron technology. Then a floating point multiply/add operation can be estimated to about 20 microseconds. Hence, it would take a 256 chip machine about 3 ms to complete one iteration of the CG method, and ideally compute the solution in 3 seconds. A path through the leafs would reduce the time needed to approximately one third. A 256 chip machine is indeed a physically small machine. In 1 micron technology only 64 chips would be needed for the same machine that would solve the problem in approximately half the time. It would easily fit on a small PC board.

The tree algorithms described above can be mapped onto Boolean n-cubes with a number of processors equal to the number of leaf processors in the tree. It should be noticed that the tree nodes participating in the summation required in computing inner products, only perform additions during three cycles of every CG iteration. Three inner products are computed. Three wave fronts are going up the tree. Two values are broadcasted from the root towards the leafs. These operations are easily mapped onto a Boolean n-cube without loss of performance. A Boolean n-cube is a Hamiltonian graph. Hence, a cyclic shift of length w at any level of the binary tree described above is easily accomplished in time w in an n-cube. An n-cube with N processors can perform the computations of one iteration of the CG method in time 2logN+w+const, i.e., the same time complexity as the algorithm for 2N-1 tree interconnected processors with a path through the leafs.

If there are fewer than N processors available a different form of mapping the problem onto the network of processors is required. If the matrix A is of band type a reduction in the number of required processors is easily obtained by exploiting the structure. A processor is associated with each non-zero diagonal, (instead of rows or columns), [Madsen, Rodrigue, Karushi 76], [Kung, Leiserson 80], [Kung 80], [Johnsson 82a], [Johnsson 82c]. An algorithm for the computation of matrix-vector products in 2N+w steps using w processors is given in [Kung, Leiserson 80]. Another algorithm requiring N+u steps using w processors is described in [Johnsson 82a]. With a few additional processors for the computation of inner products and the variables a_i and b_i it is possible to complete the computations for a step of the CG method in N+u+4 steps, using

w+6 processors, [Johnsson 82a]. Of the w+6 processors, the w that are linearly interconnected corresponds to the non-zero diagonals of A, while the remaining 6 processors are sparsely interconnected and perform the other computations of the CG method. Storage of O(N) is associated with w+3 processors.

Algorithms for different partitionings of the computations needed when the number of processing elements does not exactly match any of the characteristic dimensions of the problem can be found in [Johnsson 82a].

V. PRECONDITIONED CONJUGATE GRADIENT METHODS

The preconditioned conjugate gradient method can be defined as:

$$x_{i+1} = x_i + a_i p_i$$

$$r_{i+1} = r_i - a_i A p_i$$

$$a_i = (r_i, z_i)/(p_i, Ap_i)$$

$$Mz_{i+1} = r_{i+1}$$

$$b_i = (r_{i+1}, z_{i+1})/(r_i, z_i)$$

$$p_{i+1} = z_{i+1} + b_i p_i$$

with initial conditions $r_0 = p_0 = y - Ax_0$

Preconditioning implies that a linear system of equations, $Mz_i = r_i$, must be solved in each step of the conjugate gradient method. With proper preconditioning the number of steps to reach an approximate solution is reduced to the extent that the total time to compute the approximate solution decreases. Most recent work on preconditioning has been concerned with the effect various choices of M have on convergence rate, storage requirement and solution time on seqential machines. Obtaining M through incomplete Cholesky factorization of A has been studied by, [Meijerink, van der Vorst 77], [Manteuffel 80], [Kershaw 78], and [van der Vorst 82], incomplete Crout factorization by [Meijerink, van der Vorst 77], SSOR preconditioning by [Jennings, Malik 78] and [Axelsson 76], and partial elimination depending on size of the elements of A by [Jennings, Malik 78] or the factor L, ($A = LL^T$), by [Glowinski et al 80]. Recently, the feasibility of different preconditioners from a computational point of view has been studied also for vector and multiprocessor machines, [van der Vorst 82], [Jordan 83], and [Rodrigue 83], [Sameh 83].

In the context of the machine topologies presented here, a diagonal preconditioning matrix obviously only demands one extra time step per iteration in the tree algorithms and one extra processor in the solution based on O(w) processors. In

the following it is assumed that $M=LL^T$. The best known algorithm for solving dense triangular linear systems requires $3+(1/2)(3+\log N)\log N$ steps and at most $(15/1024)N^3+O(N^2)$ processors, [Sameh, Brent 77]. For the banded case the results are: $3+(2+\log u)\log N-1/2(\log u+1)\log u$ steps using $(1/2)u^2N+O(uN)$ processors. These results are based on a model where any processor can access any information at any time without delay. The tree, the n-cube and the O(w) processor solution do neither provide this number of processors nor unrestricted communication. Hence, preconditioning on a tree must be expected to have a time complexity that at best is comparable to that of one iteration of the CG method.

The system $Mz=r$ is for $M=LL^T$ solved as

$$Lq = r$$

$$L^Tz = q$$

A column sweep algorithm for the forward and backward substitution requires $N^2/4$ data items to pass through the root of a tree in each substitution phase, with the data structures used above, and if L were dense. For the banded case the lower bound is easily seen to be $u(u+1)/2+\log N$, and if only nonzero elements in L are considered is (the number of nonzero elements in the lower lefthand N/2 by N/2 submatrix of L)+logN. The logN term is due to the allocation of the vector r at that distance from the root.

An O(N) tree algorithm is as follows. Compute q_i in the processor that stores r_i. Column i of L is assumed to be stored in the same processor. To compute q_i the sum of products of q_j and L_{ij}, $j=1,\ldots,i-1$ is needed. A leaf node receives this sum from its parent. Having computed its own component of q, q_i, the node changes mode and computes the products of column i of L and q. Product sums of the product of L and q are computed at successive levels of the tree, i.e., columns properly multiplied by q are added pairwise in processors at distance one from the leafs, such sums added again pairwise at distance two, etc. The computations start from the left leaf cell and terminates at the right leaf cell. Nodes are activated in inorder tree traversal. An analysis of the time complexity of this algorithm yields $2(2N-1-\log N)$ steps. This time is independent of the bandwidth of L. The time equals the path length of inorder tree traversal. This algorithm offers a speed up of O(N) using O(N) processors.

If there is a path of length N through the nodes at distance logN from the root, then the algorithms used for forward and backward substitution on linear arrays, [Kung, Leiserson 80], [Kung 80], [Johnsson 81], can be employed and the number of time steps equals 2N, independent of the bandwidth.

The time complexity of the above two preconditioning algorithms is the same even if L is sparse. Exploiting sparsity leads to reduced storage needs. Reduced time complexity can be obtained if different data structures are employed. For instance, if L is of bordered block diagonal form with block size m a tree algorithm can be devised that concurrently computes all but the last m components of q in 2(2m-1-logm) time steps. To compute the last m components of q an additional 2logN+2(2m-1-logm) steps are needed. These complexity estimates illustrate the importance of the data structures used. This point is further underscored by the $O(\log^2 N)$ cyclic reduction tree algorithm of Presnell and Pargas, [Presnell, Pargas 81].

VI. CONCLUSIONS

Several concurrent algorithms for the solution of linear systems of equations were surveyed. They are all feasible for machines with a large number of sparsely interconnected processors with local storage only. The communication topology of the algorithms have been restricted to meshes with internal nodes of degree 4 or 6 (hexagonal arrays), trees and Boolean n-cubes.

A machine may have fewer processors than that corresponding to some characteristic dimension of the problem used in a natural mapping of the problem onto the machine. Mapping large problems onto few processors requires additional sequencing of the computations and storage management, compared to algorithms fully instantiated in space. Block algorithms and sweeping algorithms are two possibilities that were briefly discussed.

New algorithms for the conjugate gradient method on binary tree interconnected processors as well as Boolean n-cubes have been presented. The algorithms range in time complexity from O(logN+w) to O(N) with the number of processors ranging from 2Nw-1 to O(w). It was shown that a path through the leafs reduces the time complexity significantly. Indeed, a tree of 2N-1 processors with the leafs interconnected to a ring performs the computations of one iteration of the CG method in less time than a pure binary tree with 2Nw-1 processors. The time complexity of concurrent CG algorithms on a Boolean n-cube of N processors is the same as the time complexity of algorithms on a tree of 2N-1 processors with a path through the leafs.

Preconditioned conjugate gradient methods includes the solution of a linear system of equations in each step of the conjugate gradient algorithm. An O(N) algorithm for the forward and backward substitution was presented for a binary tree. This algorithm makes no particular assumption of the structure of the factors of the preconditioning matrix, but assumes the components of the given vector and the solution

vector to be located in successive cells at distance logN from the root. An O(logN+m) algorithm was also presented for the case where the factors of the preconditioning matrix are of bordered block diagonal form with block size m.

ACKNOWLEDGMENT

The author wishes to thank Carver A. Mead and Pey-yun Peggy Li for their careful reading of the manuscript, and many valuable suggestions.

This work was in part supported by the Defense Advanced Project Agency (DARPA) under Contract N00014-79-C-0597 with the California Institute of Technology. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

Partial support was also provided by the System Development Foundation.

REFERENCES

[Ahmed, Delosme, Morf 82]. Ahmed H.M., Delsome J.-M., Morf M. "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing." IEEE Computer (1):65-82, January, 1982.

[Axelsson 76]. Axelsson, O. "A Class of Iterative Methods for Finite Element Equations." Computer Methods in Applied Mechanics and Engineering 9:123-137, 1976.

[Bentley, Kung 79]. Bentley, J.L., Kung, H.T. "A Tree Machine for Sorting Problems." In Proc. Int'l. Conf. on Parallel Processing, pp. 257-266. IEEE Computer Society, 1979.

[Browning, Seitz 81]. Browning, S.A., Seitz, C.L. "Communication in a Tree Machine." Proc., Second Caltech Conference on VLSI, pp. 509-526. Computer Science, California Institute of Technology, 1981.

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

[Buzbee 81]. Buzbee B.L. "Implementing Techniques for Elliptic Problem on Vector Processors." In Elliptic Problem Solvers, ed. M. Schultz, Academic Press, 1981, pp. 85-98.

[DeBenedictis, Seitz 82]. DeBenedictis, E., Seitz, C.L. "Homogeneous Machine - Technical Plan." Technical Report 4705:DF:82, California Institute of Technology, 1982.

[Delosme, Morf 81]. Delosme, J.-M., Morf, M. "Scattering Arrays for Matrix Computations." In Real-Time Signal Processing IV, pp. 74-83. SPIE, August, 1981.

[Despain 79]. Despain A.M. "Very Fast Fourier Transform Algorithms Hardware for Implementation." IEEE Transaction on Computers C-28(5):333-341, May, 1979.

[Flanders et al 77]. Flanders, P.M., Hunt, D.J., Reddaway, S.F., Parkinson, D. "Efficient High Speed Computation with the Distributed Array Processor." In High Speed Computer and Algorithm Organization, pp. 113-128. Academic Press,1977.

[Fox, Seitz 83]. Fox, G.C., Seitz, C.L. "Concurrent Processing and the Decomposition of Problems." Technical Report, California Institute of Technology, 1983.

[Fox 82]. Fox, G.C. "Matrix Operations on the Homogeneous Machine." Technical Report CALT-68-939, California Institute of Technology, August, 1982.

[Gentleman, Kung 81]. Gentleman, S. Morven, Kung H.T. "Matrix Triangulation by Systolic Arrays." In Real-Time Signal Processing, pp. 19-26. SPIE, August, 1981.

[Glowinski et al 80]. Glowinski, R., Lions, Mantel, B., Periaux, J., Pironneau, O., Poirier, G. "An Efficient Preconditionned Conjugate Gradient Method Applied to Nonlinear Problems in Fluid Dynamics via Least Square Formulations." In Computing Methods in Applied Sciences and Engineering, eds. R. Glowinski and J.L. Lions, North-Holland, 1980, pp. 445-487.

[Gottlieb, Schwartz 82]. Gottlieb, A., Schwartz, J.T. "Networks and Algorithms for Very-Large-Scale Parallel Computation." IEEE Computer 15(1):27-36, January, 1982.

[Heller, Ipsen 82]. Heller, D.E., Ipsen, I.P. "Systolic Networks for Orthogonal Equivalence Transformations and Their Applications," pp. 113-122, Proc. Conf. on Advanced Research in VLSI, ed. P. Penfield, Jr., Artech House, Jan. 1982.

[Heller 78]. Heller, D.E. "A Survey of Parallel Algorithms in Numerical Linear Algebra." SIAM Review 20:740-777, 1978.

[Heller 82]. Heller, D.E. "Partitioning Big Matrices for Small Systolic Arrays." In VLSI & Modern Signal Processing, p.109. Computer Science, Univ. of Southern California, November, 1982, also report CS-83-02, Computer Science Dept., Pennsylvania State Univ., University Park, PA.

[Hestenes, Stiefel 52]. Hestenes, M.R., Stiefel E. "Methods of Conjugate Gradient for Solution of Linear Systems." J. Res. Nat. Bur. Standards 49:409-436, 1952.

[Jennings, Malik 78]. Jennings, A., Malik, G.M. "The Solution of Sparse Linear Equations by the Conjugate Gradient Method." Int. J. Numer. Methods 12:141-158, 1978.

[Johnsson, Cohen 82]. Johnsson, S.L., Cohen, D. "A Formal Description of Array Implementations of FFT Algorithms." In VLSI & Modern Signal Processing, pp. 53-63. Computer Science, Univ. of Southern California, 1982.

[Johnsson 81]. Johnsson, S.L. "Computational Arrays for Band Matrix Equations." Technical Report 4287:TR:81, Computer Science, California Institute of Technology, May, 1981.

[Johnsson 82a]. Johnsson, S.L. "Concurrent Algorithms for the Conjugate Gradient Method." Technical Report 5040:TR:82, Computer Science, California Institute of Technology, September, 1982.

[Johnsson 82b]. Johnsson S.L. "VLSI Algorithms for Doolittle's, Crout's and Cholesky's Methods." In Int'l. Conf. on Circuits and Computers 1982, ICCC82, pp. 372-377. IEEE Computer Society, September, 1982.

[Johnsson 82c]. Johnsson S.L. "Pipelined Linear Equation Solvers and VLSI." In Microelectronics '82, pp. 42-46. Institution of Electrical Engineers, Australia, May,1982.

[Johnsson 82d]. Johnsson S.L. "A Computational Array for the QR-method." Proc. Conf. on Advanced Research in VLSI, ed. Paul Penfield, Jr., pp. 123-129. Artech House, January, 1982.

[Jordan 83]. Jordan, T.L. "Conjugate Gradient Preconditioners for Vector and Parallel Processors." In Elliptic Problem Solvers, II, Academic Press, 1983.

[Kershaw 78]. Kershaw, D.S. "The Incomplete Cholesky Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations." J. Comput. Phys. 26:43-65, 1978.

[Kung, Gal-Ezer 82]. Kung, S.Y., Gal-Ezer, R.J. "Linear or Square Array for Eigenvalue and Singular Value Decomposition." In VLSI & Modern Signal Processing, pp. 89-98. Computer Science, U. of Southern California, Nov. 1982.

[Kung, Leiserson 80]. Kung, H.T. and Leiserson, C.E. "Algorithms for VLSI Processor Arrays." In Introduction to VLSI Systems, pp. 271-294. Addison-Wesley, 1980. Mead, Carver A. and Conway, Lynn A.

[Kung 80]. Kung, S.Y. "VLSI Matrix Computation Array Processor." In Proc. MIT Conf. on Advanced Research in Integrated Circuits. MIT, February, 1980.

[Lax 82]. Lax, P.D. "Report on the Panel on Large Scale Computing in Science and Engineering." Technical Report, National Science Foundation, December, 1982.

[Li 81]. Li, P. "The Tree Machine Operating System." Technical Report 4618:TM:81, Computer Science, California Institute of Technology, July, 1981.

[Madsen, Rodrigue, Karushi 76]. Madsen, K., Rodrigue, G.H., Karushi, J.I. "Matrix Multiplication by Diagonals on a Vector/Parallel Processor." Information Processing Letters 5(2):41-45, June, 1976.

[Manteuffel 80]. Manteuffel, T.A. "Solving Structures Problems Iteratively with a Shifted Incomplete Cholesky Preconditioning." In Computing Methods in Applied Science and Engineering, eds. R. Glowinski and J.L. Lions, North-Holland, 1980, pp. 427-444.

[Meijerink, van der Vorst 77]. Meijerink, J.A., van der Vorst. An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix. Math. Comp. 31:148-162, 1977.

[Meijerink, van der Vorst 78]. Meijerink, J.A., van der Vorst. "Guidelines for the usage of incomplete decompositions in solving sets of linear equations that occur in practical problems." Technical Report TR-9, ACCU, Utrecht, 1978.

[Munksgaard 79]. Munksgaard, N. "Solving Sparse Symmetric Sets of Linear Equations by Preconditioned Conjugate Gradients." Technical Report CSS 67, Harwell, 1979.

[Presnell, Pargas 81]. Presnell, H.A., Pargas R.P. "Communication Along Shortest Paths in a Tree Machine." In Functional Programming Languages and Computer Architecture. pp. 107-114. ACM, 1981.

[Rodrigue 83]. Rodrigue, G.H. "Incomplete Factorization and Parallel Computations." In Elliptic Problem Solvers, II, Academic Press, 1983.

[Sameh, Brent 77]. Sameh, A.H., Brent, R.P. "Solving Triangular Systems." SIAM J. Numer. Anal. 14(6):1101-1113, December, 1977.

[Sameh 77]. Sameh, A.H. "Numerical Parallel Algorithms - A Survey." In High Speed Computer and Algorithm Organization. pp. 207-228. Academic Press, 1977.

[Sameh 83]. Sameh, A.H. "Parallel Algorithms for Elliptic Problems." In Elliptic Problem Solvers,II, Academic Press'83

[Schreiber, Keukes 82]. Schreiber, R., Keukes, P. "Systolic Linear Algebra Machines in Digital Signal Processing." In VLSI & Modern Signal Processing. p. 201. Computer Science, Univ. of Southern California, November, 1982.

[Schreiber 82]. Schreiber, R. "Systolic Arrays for Eigenvalue Computation." In Real-Time Signal Processing, pp. 27-34, Vol. 341. SPIE, May, 1982.

[Schwartz 80]. Schwartz, J.C. "Ultracomputers." ACM Trans. on Programming Languages and Systems 2(4):484-521, October, 1980.

[Seitz 82]. Seitz, C.L. "Ensemble Architectures for VLSI - A Survey and Taxonomy." Proc., Conf. on Advanced Research in VLSI, ed. P. Penfield Jr., pp. 130-135. Artech House, 1982.

[Snyder 82]. Snyder, L. "Introduction to the Configurable, Highly Parallel Computer." IEEE Computer 15(1):47-56, January, 1982.

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

[Stone 73]. Stone, H.S. "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations." J. ACM 20(1):27-38, January, 1973.

[Stone 75]. Stone, H.S. "Parallel Tridiagonal Equation Solvers." ACM Trans. on Mathematical Software 1(4):289-307, December, 1975.

[van der Vorst 82]. van der Vorst, H.A. "A Vectorizable Variant of Some ICCG Methods." SIAM J. Scientific and Statistical Computing 3(3):350-357, 1982.