Yale University
Department of Computer Science

Highly Parallel Banded Systems Solvers

S. Lennart Johnsson

YALEU/DCS/TR-581
August 1987

Highly Parallel Banded Systems Solvers[1]

S.Lennart Johnsson[2]
Departments of Computer Science
and Electrical Engineering
Yale University
New Haven, CT 06520

Abstract

We present algorithms for the solution of banded systems of equations on parallel architectures, in particular ensemble architectures, i.e., architectures that have a large number of processing elements. Each processor has its own local storage. The band is considered dense. Concurrent elimination of a single variable yields a linear speed-up for ensembles configured as tori, or Boolean cubes, if $N \gg m$, with a maximum ensemble size of $m(m+R)$ (or $2m(m+R)$) processors for a banded system of N equations, bandwidth $2m+1$ and R right hand sides. The minimum attainable computational complexity is of order $O(N)$. Concurrent elimination of multiple variables as well as concurrent elimination of each such variable yields a minimum complexity of $O(m + m\log_2 \frac{N}{m})$ for a total of $(2m+R)N$ ensemble nodes. To attain this complexity the ensemble should be configured as clusters, each in the form of a torus of dimension m by $2m+R$, or a Boolean cube of appropriate dimension. Furthermore, corresponding processors in different clusters are assumed to be interconnected to form a binary tree, shuffle-exchange, perfect shuffle, or Boolean cube network. The number of clusters should be of order $O(\frac{N}{m})$ for minimum computational complexity.

1 Introduction

Banded systems of equations occur in a variety of applications, in particular where the finite element technique is used. Both direct and iterative methods are used for the solution of such systems. With the emergence of parallel architectures it is important to find algorithms that can make effective use of such architectures. It is also important that they are portable in the sense that good performance is obtained on both coarse and fine grain architectures; as well as shared memory, and distributed memory architectures. We have devised concurrent algorithms for banded systems of equations feasible for a wide range of architectures. In this paper, we present the algorithms and

[1]Proceedings of Symposium on Parallel Computations and Their Impact on Mechanics, ASME, December 1987.

[2]Currently on leave at Thinking Machines Corp.

an analysis of their computational complexity. A report on the implementation of one of the algorithms on coarse grain, shared memory architectures like the Alliant FX/8 and the Sequent Balance 21000, is given in [3]. Experiences from the implementation on a data parallel computer such as the Connection Machine will be reported elsewhere.

Due to the characteristics of silicon technologies and Very Large Scale Integration (VLSI) systems of medium scale parallelism are common, and data parallel architectures are emerging. The data movement capability is the most important factor in designing a high performance architecture. A high storage bandwidth is accomplished by multiported (interleaved) storage. A high total data bandwidth is accomplished by interconnecting the processors in a network. The network load can be reduced by dividing the storage among the processing nodes, and exploiting the locality of data interaction present in most problems.

The focus in this paper is on the data and control structures that yield a good load balance, and allow for the exploitation of locality. Complexity estimates are derived for network topologies in the form of tori. Such networks can be embedded in Boolean cubes, which also have additional communication capabilitites. We comment on the consequences of this capability. We also consider systems in which clusters in the form of tori and Boolean cubes are interconnected to form binary trees, shuffle-exchange, perfect shuffle, or Boolean cube networks.

Algorithms presented here are elimination methods. We present three classes of concurrent algorithms for the solution of banded systems of linear equations. One class exploits for concurrency, the independence of the data set for the elimination of a single variable from the system of equations. Another exploits the independence of the data sets for the elimination of different variables; and the third is a combination of the two.

Independence of the data sets for the elimination of a single variable as well as different variables is apparent in the graph model of elimination [20]. An $N \times N$ matrix can be represented by a graph of N vertices with a directed edge between a pair of vertices for each nonzero matrix element. The graph can be considered as undirected for a symmetric matrix. In this case, the elimination of a variable from the set of equations corresponds to the elimination of a vertex from the graph. In the asymmetric case, the factorization transforms the original graph to an acyclic graph. Because of this analogy we refer to the elimination of a single variable from the set of equations as *vertex elimination.*

Systolic algorithms for banded systems [19,10,17] maximally exploit the independence of operations in the elimination of a single vertex. Such algorithms are derived for mesh configured architectures. Data movement in these and similar algorithms for Householder reflections [9], and Given's rotations [6,1,7,15] is not minimal. Algorithms with considerably reduced data movement, but the same degree of concurrency and

processor utilization are described below. The algorithms cover different combinations of the number of processors and problem sizes. Algorithms for coarse grained architectures can be obtained by the folding of computations from space into time. This folding can be automated, which gives rise to an opportunity to optimize the granularity of operations, pipelining and vectorization.

Algorithms derived from systolic algorithms have a speed-up that is linear in the number of nodes, including communication. For a system of dimension N, bandwidth $2m+1$, and R right hand sides, the maximum number of processors is $m(m+R)$, or $2m(m+R)$ with the elimination proceeding concurrently from the first and last rows towards the middle [4]. The third class of algorithms yields algorithms of minimum complexity $O(m+m\log_2\frac{N}{m})$ for an $N \times N$ system of bandwidth $2m+1$. The maximum number of nodes for the second class of algorithms is $(2m+R)N$.

Algorithms are analyzed with respect to communication and arithmetic complexities. Since all complexity terms tend to be of the same order in a highly concurrent (data parallel) architecture, the complexity expressions are quite detailed.

1.1 Preliminaries

The system of equations is $AX = Y$, where A is an $N \times N$, symmetric, irreducible matrix such that $|a_{ij}| = 0$ for $|i-j| \geq m$. Y is an $N \times R$ matrix. It is convenient to store the matrix and the right hand side in an array of $m+1+R$ columns and N rows. The dimensionality of the array data structure and the topology of the architecture is the same. In the case of a Boolean cube configured architecture a two-dimensional array can be embedded by the use of a two-dimensional *binary-reflected* Gray code [21,8]. An array of size $2^k \times 2^l$ can be embedded in a $(k+l)$-cube preserving adjacency by separately encoding the row and column indices in k and l bits, respectively.

When the matrix size exceeds the number of nodes in the ensemble, several matrix elements must be identified with the same node. Two natural schemes for the identification of elements are *consecutive*, and *cyclic* identification [8]. In *consecutive* storage, block matrices are assigned to nodes. For a symmetric banded system stored in an array of N rows and $m+1+R$ columns, array element (i,j) is stored in node (p,q), where $p = \lfloor \frac{i}{\lceil \frac{m+1}{2^k} \rceil} \rfloor mod 2^k$ and $q = \lfloor \frac{j}{\lceil \frac{m+1+R}{2^l} \rceil} \rfloor$ for algorithms exploiting the independence of operations for the elimination of a single variable. In *cyclic* storage, array elements (i,j) are identified with node (p,q), if $p = imod2^k$, and $q = jmod2^l$. The number of processors in the column direction is $2^l \leq m+1+R$ and in the row direction $2^k \leq m+1$. The number of clusters is limited to $P \leq \lceil \frac{N}{m} \rceil$. The two storage schemes are illustrated in Figure 1. For the concurrent elimination of multiple vertices we assume that there are $K_c = 2^{k_c}$ nodes in each cluster and that there are P clusters. The rows are divided

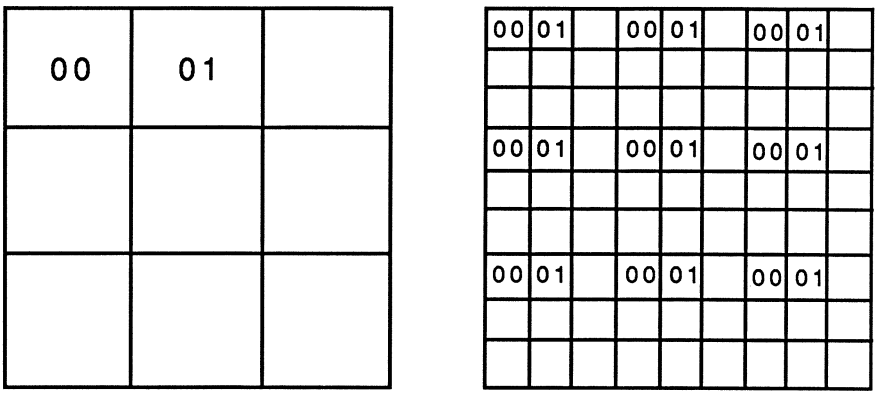

Figure 1: Consecutive and cyclic storage of a banded matrix.

into sets of $\frac{N}{P}$ equations.

There is an apparent difference in granularity suggested by the two storage forms. However, optimum use of the architecture may require that in the consecutive storage scheme parts of block matrices be communicated in the same communication action instead of the whole block matrix (in order to increase concurrency). Conversely, in the cyclic storage scheme it may be desirable to combine several matrix elements for each communication action in order to reduce the adverse effect of communications overhead. If there is an overhead in performing arithmetic operations, the same arguments apply. Optimization yields the same result in computations in which all elements participate in the same manner in all the steps of the computation, i.e., in *uniform* computations like matrix multiplication.

In describing and analyzing equation solvers based on direct methods, it is convenient to introduce the concepts of *operational windows* and *computational windows* [10]. We define an *operational window* to be the set of operations associated with the elimination of a single vertex. A *computational window* is defined by the set of operations performed concurrently in the case of zero communication time. Operations defined by an operational window are divided into disjoint sets by the computational windows (which completely cover it), Figure 2. The Figure shows 2 successive operational windows for pivoting on the digonal.

Computational windows for a given operational window are processed one at a time by a *processing plane* (or *cluster*) in the case of zero communication time, Figure 4. Computational and operational windows are *not* fixed with respect to processor or storage addresses, contrary to the familiar systolic algorithms. Data and control structures for a few mappings of computational windows into time for a single cluster are analyzed in [10]. The mapping of the operational windows into time is equivalent to establishing

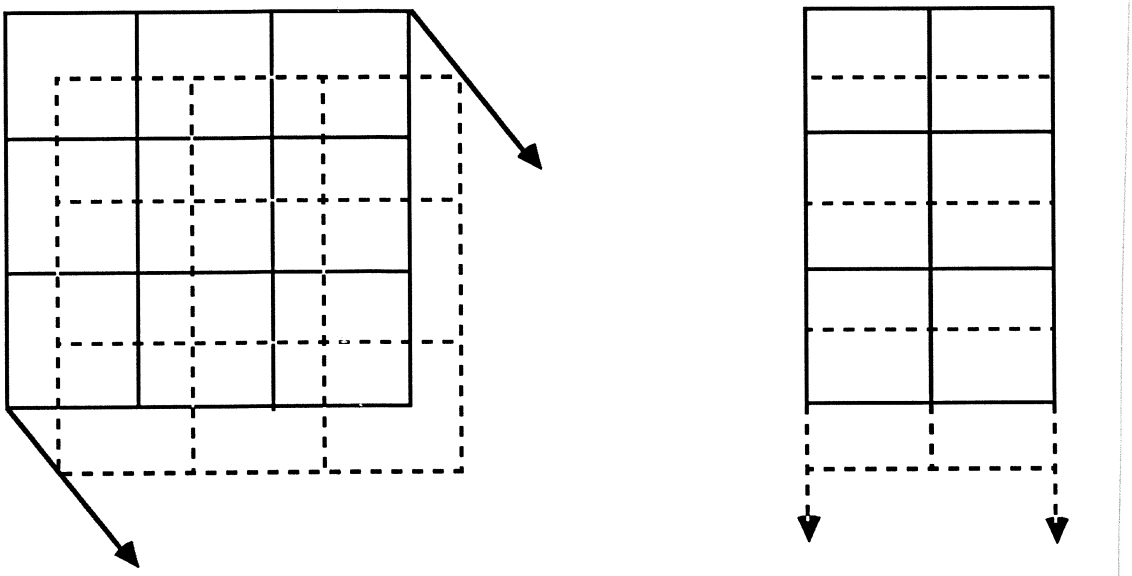

Figure 2: An operational window and corresponding computational windows for the elimination of a vertex by Gaussian elimination and with pivoting on the diagonal.

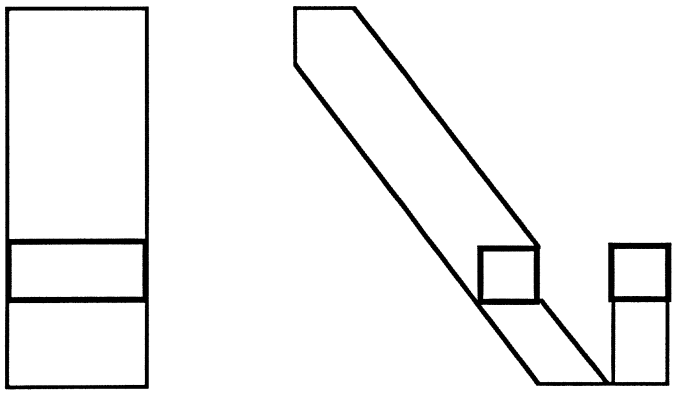

Figure 3: Operational windows for the concurrent elimination of multiple vertices.

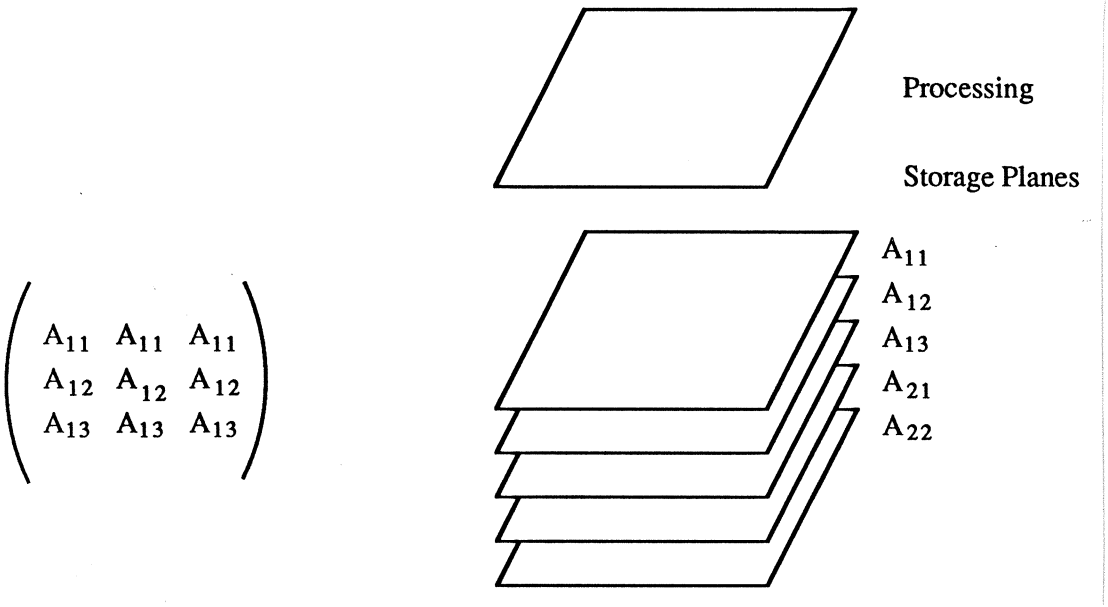

Figure 4: A processing plane and computational windows.

the pivoting order.

Arithmetic complexity is easily estimated by counting the number of sequential arithmetic operations per computational window, the number of computational windows per operational window, and the maximum number of operational windows processed in sequence by any cluster for the entire computation. In general, the number of arithmetic and communication operations per computational window will vary from window to window. Also, the number of computational windows per operational window will vary.

Complete processing of one computational window before the processing of another window is initiated, corresponds to zero communication time, or no computations in any processor if there is communication somewhere in the system. In a message passing system such a state can only be attained through special efforts. In a MIMD (Multiple Instruction Multiple Data) type architecture [5] computations proceed at the rate data is available, much as in a dataflow system. As a processor has performed the computations for one computational window it moves to the next at a rate determined by its processing capability, and the availability of data needed for the computations. In a SIMD (Single Instruction Multiple Data) type architecture different processors perform the same action on the same or different computational windows.

In the following, K_c denotes the number of processors in each of P clusters, $PK_c = P_e$, t_a the time to perform any one of the arithmetic operations addition, subtraction or multiplication and t_d the time to perform a division. The time to communicate an operand between a pair of processors is denoted t_c. Overhead in communication is ignored here. This assumption is approximately true for some architectures, such as the Connection Machine, but not for the current breed of hypercubes.

2 Concurrent Elimination of a Single Vertex

Algorithms described in this section yield a linear speed-up for up to $m(m+R)$ processors, or $2m(m+R)$ processors for 2-way Gaussian elimination, [4]. An analogous algorithm for Cholesky's method is described in [12]. Similar algorithms can be devised for Householder's reflections [9], or Given's rotations [6,1,7,15].

2.1 Gaussian Elimination on a Torus

The problem of solving a banded system of equations on a data parallel architecture configured as a torus, or a Boolean cube, is similar to the solution of band matrix problems by systolic algorithms [19,6,1,7,9,15,17]. The exception is that the data for the entire problem, or a large portion of it, is stored in the primary storage. Optimization of the granularity of operations to reduce overhead in arithmetic and communication, possibly at the expense of reduced concurrency, is omitted here. We carry out the discussion and analysis of the algorithms in the context of cyclic storage, since it tends to enforce a greater insight into the control and data structures of the algorithms.

A systolic algorithm for cyclicly stored matrices is given in [10]. In systolic algorithms, the entire concatenated matrix AY is shifted one step for each operational window during factorization as well as backsubstitution. During factorization and forward elimination, outer products of a row vector of $m+R+1$ elements and column vectors of m elements are formed. Regardless of the value of N, $m(m+R+1)$ elements are updated for each operational window. Updating the proper elements of $A = LU$ *in-place* results in reduced data movement. *In-place* algorithms are *dual* to common systolic algorithms in that the operational window is moving over the data (the processor addresses) instead of the operational window being fixed (with respect to processor addresses), and the data passing through it.

A backsubstitution algorithm dual to the one in [19] can only use an $m \times R$ torus, which for $R \ll \sqrt{K_c}$ results in poor processor utilization. For small values of R, inner-products are accumulated *in-space* instead of in-place. The speed-up is of order $O(mR)$ for $mR < \gamma K_c$, $\gamma \in [1/2, 1]$, and of order $O(K_c)$ for $mR \geq \gamma K_c$. Hence, for $R = 1$ and $m < K_c$ the speed-up is proportional to m.

In the *in-place* algorithm for factorization and forward elimination, the pivot row is broadcasted to the other m rows of the operational window; and the pivot column broadcasted to the other $m+R$ columns. Optimal algorithms for this operation on Boolean cubes are described in [18]. Some of them are identical to mesh algorithms. With pivoting on the diagonal the pipelining of communications and arithmetic operations is straightforward. Partial pivoting considerably reduces concurrency. Essentially

one dimension of a 2-dimensional array is lost from a concurrency point of view [10]. The running time increases by a factor of order 2^{k_c}.

For an in-place algorithm, the number of elements that need to be transferred between a given pair of processors for each of the first $N - m$ operational windows is the same for the cyclic and consecutive storage schemes (and equal to $\lceil \frac{m+R+1}{\sqrt{K_c}} \rceil$). For each of the first $N - m$ operational windows the maximum time for arithmetic for any processor is the same for both consecutive and cyclic storage. The same property also holds for backsubstitution. We formulate this observation as a lemma.

Lemma 1 *The maximum number of arithmetic operations performed by any processor, and the maximum number of data elements that need to be communicated between a processor and its neighboring processors for the factorization, forward elimination, and backsubstitution of the first $N - m$ equations are the same for consecutive and cyclic storage.*

The preservation of computational and communication balance is illustrated in Figure 5. As the operational window is moved one row and column in the forward phase, the first matrix row of each of the top computational windows is no longer actively participating in the computations. However, a new matrix row at the bottom of the new operational window is included, as is a new column at its right edge, except for the last m operational windows. In the cyclic storage scheme this matrix row is stored either in the processor row storing the most recently used pivot row (if $m + 1 mod \sqrt{K_c} = 0$), or stored in a processor row previously inactive during the execution of the last row of computational windows of the preceding operational window. The situation for the new column is similar. In the consecutive storage scheme the new row is either stored in the same processor row as the last block row of the previous operational window or in the processor row storing the pivot row of that operational window.

The solution of a banded system of equations is divided into two major parts:

1. The factorization, forward elimination, and backsubstitution on the first $N - m$ equations; and
2. The solution of a dense system of m equations.

The algorithm proposed for the first part yields 100% processor utilization, an initiation phase excepted. However, it is not as efficient for the second part as the proposed dense matrix algorithms. For a matrix of narrow bandwidth, $m \ll N$, this difference is of no major concern. However, it is the dominating complexity term for $m \approx N$.

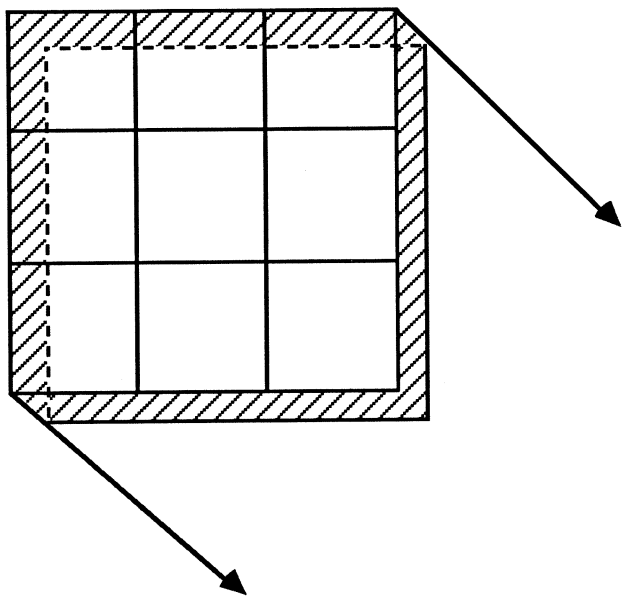

Figure 5: The effect of shifting the operational window 1 step on the balance of computations.

The first	Computation	Complexity
$N-m$ equations	Factorization and Forward solve	$(N-m)\{2t_c+t_d+(\lceil\frac{m+1}{\sqrt{K_c}}\rceil-1)max(2t_a,t_c,t_d)+$ $+((\lceil\frac{m+1+R}{\sqrt{K_c}}\rceil-1)\lceil\frac{m+1}{\sqrt{K_c}}\rceil+1)max(2t_a,t_c)\}-$ $-max(2t_a,t_c)+2(\sqrt{K_c}-2)t_c+t_d+2t_a$
	Backward solve, many R.H.S. few R.H.S.	$(N-m)\alpha+\beta\sqrt{K_c}+const$ for $\lceil\frac{R}{\sqrt{K_c}}\rceil=\lceil\frac{m}{\sqrt{K_c}}\rceil=1$ $(N-m)(\alpha\frac{m}{\sqrt{K_c}}+\beta)+\gamma\sqrt{K_c}+const$ for $\lceil\frac{R}{\sqrt{K_c}}\rceil=1$, $\lceil\frac{m}{\sqrt{K_c}}\rceil\approx\frac{m}{\sqrt{K_c}}$ $(N-m)(\alpha\frac{mR}{K_c}+\frac{\beta m+\gamma R}{\sqrt{K_c}})+\delta\sqrt{K_c}+const$ for $\lceil\frac{R}{\sqrt{K_c}}\rceil\approx\frac{R}{\sqrt{K_c}}$, $\lceil\frac{m}{\sqrt{K_c}}\rceil\approx\frac{m}{\sqrt{K_c}}$.
	Backward solve	$\alpha N+\beta R+const, \quad \lceil\frac{m}{\sqrt{K_c}}\rceil R\le\gamma\sqrt{K_c}$ $\alpha\frac{NmR}{K_c}, \quad mR\ge K_c$
The last m equations	Factorization and forward solve	$m(2t_c+t_d)+\sqrt{K_c}\lceil\frac{m}{\sqrt{K_c}}\rceil(\lceil\frac{m}{\sqrt{K_c}}\rceil+1)max(2t_a,t_c)/2$ $+\sqrt{K_c}\lceil\frac{m}{\sqrt{K_c}}\rceil(\lceil\frac{R}{\sqrt{K_c}}\rceil+\lceil\frac{m}{\sqrt{K_c}}\rceil-1)/2)$ $max(2t_a,t_c,t_d)+\sqrt{K_c}\lceil\frac{m}{\sqrt{K_c}}\rceil(\lceil\frac{m}{\sqrt{K_c}}\rceil-1)(3\lceil\frac{R}{\sqrt{K_c}}\rceil+2\lceil\frac{m}{\sqrt{K_c}}\rceil-1)t_a/3$
	Backward solve many R.H.S.	$\lceil\frac{m}{\sqrt{K_c}}\rceil(\lceil\frac{m}{\sqrt{K_c}}\rceil-1)\lceil\frac{R}{\sqrt{K_c}}\rceil(\frac{1}{2}\sqrt{K_c}-1)max(2t_a,t_c)$ $+2(\sqrt{K_c}-1)t_c+2t_a$
	Backward solve few R.H.S.	$(\lceil\frac{m}{\sqrt{K_c}}\rceil R+\frac{1}{2}\sqrt{K_c}-R)t_c$ $+\sum_{j=1}^{\lceil\frac{m}{\sqrt{K_c}}\rceil} max(t_a+\sqrt{K_c}(t_a+t_c),(\lceil\frac{m}{\sqrt{K_c}}\rceil-j)R(t_a+max(t_a,t_c)))$

Table 1: The complexity of solving a banded system of bandwidth $2m+1$ and order N.

2.1.1 The Complexity of Solving the First N-m Equations

For the following analysis we assume the cyclic storage scheme, that communication and computation can take place concurrently for each node; and that the ensemble topology is that of a torus, formed by a $\sqrt{K_c}\times\sqrt{K_c}$ mesh with end-around connections. We consider Gaussian elimination without partial pivoting, and the computation of the product form of the inverse of the lower triangular matrix L ($A=LU$, $L^{-1}=L_NL_{N-1}\dots L_1$) *in-place*. $U=L_NL_{N-1}\dots L_1A$, where only column j of L_j has nonzero elements, except for the diagonal elements. The nonzero off diagonal elements are obtained from the elements of column j of $L_{j-1}\dots L_1A$. The matrix $L^{-1}Y$ is also computed in-place, i.e., it replaces Y.

For the factorization broadcasting of successive pivot rows and columns can be pipelined, if partial pivoting is unnecessary. Alternatively, each broadcasting can be completed before any other operation. The first alternative is natural for systems synchronized by message passing. The other is natural for data parallel architectures. We summarize the complexity estimates in Table 1.

For factorization and forward elimination there are a total of $\lceil\frac{m+1}{\sqrt{K_c}}\rceil\lceil\frac{m+1+R}{\sqrt{K_c}}\rceil$ computational windows for each operational window. The time for processor (1,1) to com-

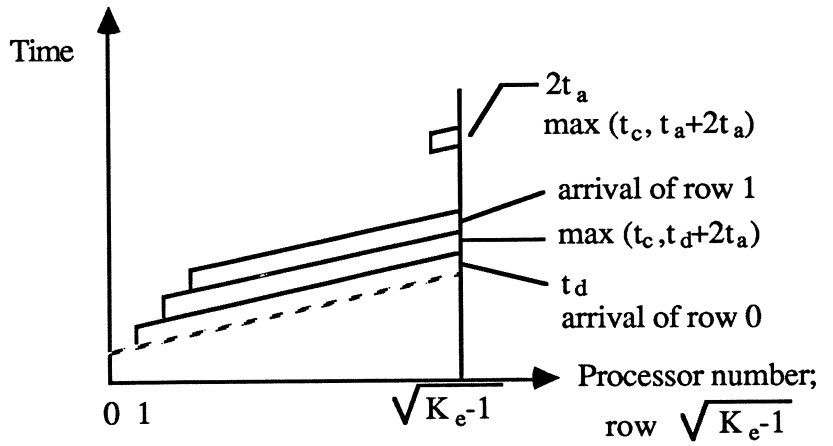

Figure 6: The utilization of the processors in row 0 during the first $\sqrt{K_c}$ operational windows.

plete the computations on the first column of computational windows is $(\lceil\frac{m+1}{\sqrt{K_c}}\rceil - 1)max(2t_a, t_c, t_d) + 2t_c + t_d + 2t_a$; and the time for the completion of the computations on the first operational window is $(\lceil\frac{m+1}{\sqrt{K_c}}\rceil - 1)max(2t_a, t_c, t_d) + (\lceil\frac{m+R+1}{\sqrt{K_c}}\rceil - 1)\lceil\frac{m+1}{\sqrt{K_c}}\rceil max(2t_a, t_c) + 2t_c + t_d + 2t_a$. The last term can be overlapped with communication for the next operational window. For the computations on an operational window to be complete an additional propagation time of $2(\sqrt{K_c} - 2)t_c$ is needed.

Figure 6 shows the active periods of processors in row 0 of the torus during the elimination of the subdiagonal elements in columns $0 - (\sqrt{K_c} - 1)$. Figure 7 shows in a stylized form, the interleaving of operations on 3 stripes of width $\sqrt{K_c}$ (the maximum degree of interleaving). Interleaving of the forward elimination on the right hand sides can be performed similarly.

The application of L_j is illustrated in Figure 8.

Remark 1. If $N - m \ll m$, and $R \ll m$ then there exist an optimum size of the mesh that is less than the matrix bandwidth, $\sqrt{K_{c_{opt}}} = O(m^{2/3})$.

Remark 2. The arithmetic complexity of a square torus for factorization and forward elimination is identical to the complexity of a torus based on a mesh congruent to the operational window, if $(m+1)mod\sqrt{K_c} = (m+1+R)mod\sqrt{K_c} = 0$. However, a square mesh minimizes the propagation time.

Remark 3. Any matrix multiplication algorithm can be used for rank-1 updates. We choose an algorithm that has the same data flow as the algorithm for factoring the

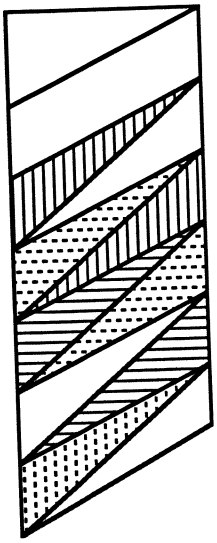

Figure 7: Interleaving of the elimination on 3 stripes of width $\sqrt{K_c}$.

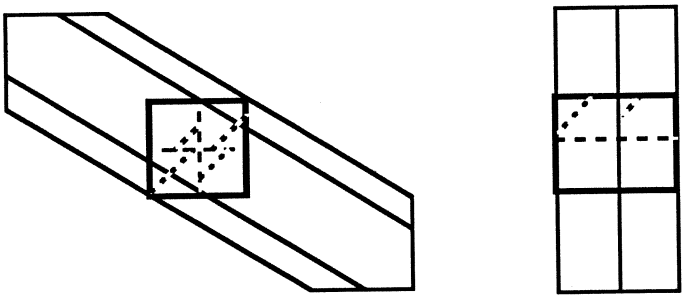

Figure 8: Storage and application of L_j.

Complexity	Speed-up	Comment
Fact. and forw. elim. first $N-m$ eqns.	$O(K_c)$	$\lceil \frac{m+1}{\sqrt{K_c}} \rceil$
Backsubstitution	$O(mR)$	$\lceil \frac{R}{\sqrt{K_c}} \rceil = \lceil \frac{m}{\sqrt{K_c}} \rceil = 1$
first $N-m$ eqns.	$O(R\sqrt{K_c})$	$\lceil \frac{R}{\sqrt{K_c}} \rceil = 1$, $\lceil \frac{m}{\sqrt{K_c}} \rceil \approx \frac{m}{\sqrt{K_c}}$
many RHS	$O(K_c)$	$\lceil \frac{R}{\sqrt{K_c}} \rceil \approx \frac{R}{\sqrt{K_c}}, \lceil \frac{m}{\sqrt{K_c}} \rceil \approx \frac{m}{\sqrt{K_c}}$
few RHS	$O(mR)$	$m \leq \gamma \frac{K_c}{R}$
	$O(K_c)$	$m > \gamma \frac{K_c}{R}$, $\gamma \in [1/2, 1]$
Fact. and forw. elim. last m eqns.	$O(K_c)$	$\sqrt{K_c} \ll m, R$
Backsubstitution	$O(R\sqrt{K_c})$	$\lceil \frac{R}{\sqrt{K_c}} \rceil = 1$
many RHS	$O(K_c)$	$\lceil \frac{R}{\sqrt{K_c}} \rceil \approx \frac{R}{\sqrt{K_c}}$
few RHS	$O(K_c)$	$m > \frac{K_c}{R}$

Table 2: Speed-up for the solution of a banded system of bandwidth $2m+1$ and order N.

diagonal block, for ease of pipelining. The matrix multiplication algorithm is described in some detail in [8] (algorithm MMT3).

Backsubstitution consists of the computation $Y \leftarrow Y - U(*,q)X(q,*)$. For the first $N-m$ equations, $U(*,q)$ is effectively an $m \times 1$ vector, $X(q,*)$ a $1 \times R$ vector, and Y an $m \times R$ matrix. Recurrence can be based on outer products, or R inner products of a $1 \times m$ vector $U(q,*)$ and $m \times 1$ subvectors $X(q,*) - X(q+m,*)$.

In dual systolic algorithms, the matrix Y is stationary and U and X are communicated. Inner products are accumulated *in-place*. Elements of U are communicated along processor rows, and x-values along columns. The data flow can be made identical to the flow in the matrix multiplication algorithm by Cannon [2]. However, it is also possible to send the columns of U to the first column of Y, and then broadcast them from there. The elements of U and X are multiplied on their path towards the last processor column and final row respectively, and added to the local value of Y. Computations of successive rows and columns are skewed with respect to each other. The algorithm yields a speed-up proportional to the number of nodes in the ensemble for $R \gg \sqrt{K_c}$, i.e., for a large number of right hand sides.

For few right hand sides a skewing operation is performed on the right hand sides such that row i is rotated q processor columns in the direction of increasing column index, $q = i mod \sqrt{K_c}$. The computation of a new set of x-values is accomplished by

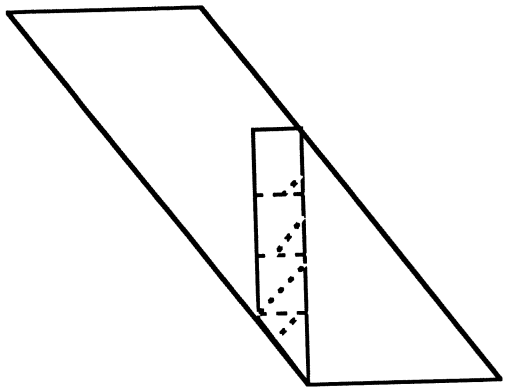

Figure 9: Wavefronts for backsubstitution with accumulation of inner products in-space.

distributing known x-values along processor column, and accumulating inner products along processor rows *in-space* [11]. The matrices U and Y are stationary. The distribution of x-values can be pipelined. For each right hand side one inner product is computed for each x-component, regardless of m. The computation of inner products for different right hand sides can be pipelined. The algorithm yields a speed-up of order m for $m \leq \gamma K_c/R$, and of order K_c for $m > \gamma K_c/R$, $\gamma \in [1/2, 1]$.

Figure 9 illustrates the wavefronts associated with a single right hand side. The wavefronts in different block matrices are delayed with respect to each other. For $\frac{m}{\sqrt{K_c}} \leq \sqrt{K_c}$ all wavefronts for a column of block matrices are issued from the processor in the lower right hand corner before a new set is to be issued for the next x-value computed by the processor.

For the solution of the last m dense matrix equation the algorithm described in [11] can be used. This factorization algorithm has a data movement similar to the algorithm described above for the elimination of the first $N-m$ vertices. The factors are computed in-place. Gauss-Jordan elimination is performed on the diagonal blocks.

2.2 Summary

Exploiting the independence of operations in the elimination of a single vertex yields a linear speed-up that is proportional to the number of processors. For a symmetric matrix Cholesky's method can be used instead of Gaussian elimination, with the same savings in storage as in the sequential case; and with a reduction in time complexity that also is comparable to the savings in the sequential case for $\lceil \frac{m}{\sqrt{K_c}} \rceil \gg 1$ [12]. For $\lceil \frac{m}{\sqrt{K_c}} \rceil = 1$ there is no reduction in time.

Linear speed-up is limited to ensembles of at most $m(m+R)$ nodes for Gaussian elimination ($2m(m+R)$ for 2-way elimination) and $m(m/2+R)$ nodes for Cholesky's

method. This constraint is fundamental to the method. For m of order $O(N)$ the speed-up may be satisfactory, however, for $m \ll N$ the degree of concurrency is low. In this case, exploiting the independence in the elimination of different vertices is the main source of concurrency.

3 Concurrent Elimination of Multiple Vertices

3.1 Preliminaries

For systems with $\frac{m}{N} \ll 1$, (i.e., matrices with narrow bands) the main source of concurrency is the independence of operations for the elimination of different vertices. Whereas, for $\frac{m}{N} \approx 1$ the main source of concurrency is in the operations required for the elimination of a single vertex. The order of the minimum complexity obtained through combining the two forms of concurrent elimination is $O(m + m \log_2 \frac{N}{m})$. This complexity is an improvement by a factor of m^2 compared to exploiting only the independence of operations in the elimination of different vertices, and an improvement by a factor of $\frac{N}{m+m \log_2(N/m)}$ compared to exploiting only the independence of the operations in the elimination of a single vertex. The latter improvement is of order $\frac{N}{\log_2 N}$ for $\frac{m}{N} \ll 1$, and of order $O(1)$ for $\frac{m}{N} \approx 1$.

The system of equations is partitioned into a block tridiagonal system of P partitions, $P \leq \lceil \frac{N}{m} \rceil$. A *cluster* of K_c processors is assigned to each partition. Corresponding processors of different clusters are connected to form binary trees, perfect shuffle networks, or Boolean cubes. Each processor is assumed to have sufficient storage $(O(N \frac{(2m+R)}{PK_c}))$.

The solution of the banded system is carried out in three phases [16]. In phase 1, the system is transformed from a banded system into a system that can be solved in two phases; the first of which (Phase 2) is the solution of a block tridiagonal system of order mP. The third phase is backsubstitution on the remaining equations. In phases 1 and 3, a cluster of processors is employed for each partition, whereas, the number of clusters used in phase 2 depends on the method choosen for the solution of the block tridiagonal system. Each partition in phases 1 and 3 concurrently performs the computations associated with the elimination of a single vertex. Computations in different partitions take place concurrently. The number of vertices being eliminated concurrently is equal to the number of clusters.

Algorithms that concurrently eliminate multiple vertices do not use a perfect elimination order. They require more arithmetic operations than the elimination of vertices in the normal order because of fill-in, Figure 3. Arithmetic and communication complexities for the degenerate case with clusters of a single processor is analyzed in [16].

Here we treat the general case, and discuss the partitioning of a given set of processors into clusters with the goal of minimizing the estimated complexity. Note, that if $P = \frac{N}{m}$, then phases 1 and 3 vanish.

Phase 1 of the algorithm [16] consists of 3 steps.

$$
\begin{array}{cccccccccccccc}
A_0 & B_0 & & & & & & & & & X_0 & & Y_0 \\
C_1 & A_1 & B_1 & & & & & & & & X_1 & & Y_1 \\
 & C_2 & A_2 & B_2 & & & & & & & X_2 & & Y_2 \\
 & & \cdot & \cdot & \cdot & & & & & & \cdot & & \cdot \\
 & & & \cdot & \cdot & \cdot & & & & & \cdot & & \cdot \\
 & & & & C_i & A_i & B_i & & & & X_i & = & Y_i \\
 & & & & & \cdot & \cdot & \cdot & & & \cdot & & \cdot \\
 & & & & & & \cdot & \cdot & \cdot & & \cdot & & \cdot \\
 & & & & & & & \cdot & \cdot & \cdot & \cdot & & \cdot \\
 & & & & & & & & \cdot & \cdot & \cdot & & \cdot \\
 & & & & & & & & C_{P-1} & A_{P-1} & X_{P-1} & & Y_{P-1}
\end{array}
$$

The diagonal blocks are matrices of bandwidth m. The blocks are of the form

$$
A_i = \begin{pmatrix} & A_{i_{11}} & & E_{i_1} \\ & & & E_{i_2} \\ & & & E_{i_3} \\ 0 & 0 & A_{i_{21}} & E_{i_4} \end{pmatrix} B_i = \begin{pmatrix} 0 & 0 & F_{i_1} \\ 0 & 0 & F_{i_2} \\ 0 & 0 & F_{i_3} \\ B_{i_{21}} & 0 & F_{i_4} \end{pmatrix} C_i = \begin{pmatrix} 0 & G_{i_1} \\ 0 & G_{i_2} \\ 0 & G_{i_3} \\ 0 & G_{i_4} \end{pmatrix} Y_i = \begin{pmatrix} H_{i_1} \\ H_{i_2} \\ H_{i_3} \\ H_{i_4} \end{pmatrix}
$$

Let $q = \frac{N}{P}$. The matrices $E_{i_j}, F_{i_j}, G_{i_j}, j = \{1,3,4\}, i = \{0,1,...,P-1\}$ are $m \times m$ matrices and $E_{i_2}, F_{i_2}, G_{i_2}, i = \{0,1,...,P-1\}$ are $q - 3m \times m$ matrices, $q \geq 3m$. For the right hand side we introduce the matrices H_{i_j}, which for $j = \{1,3,4\}, i = \{0,1,...,P-1\}$ are $m \times R$ matrices, and $H_{i_2}, i = \{0,1,...,P-1\}$ are $q - 3m \times R$ matrices.

Initially $E_{i_1} = E_{i_2} = 0$, $E_{i_3} = A_{i_{12}}$, $E_{i_4} = A_{i_{22}}$, $F_{i_j} = 0$, $j = \{1,2,3,4\}$, $G_{i_1} = C_{i_{12}}$, $G_{i_j} = 0$, $j = \{2,3,4\}$, and $H_{i_1} = \{Y_{(i-1)q+1}, ..., Y_{(i-1)q+m}\}^T$, $H_{i_2} = \{Y_{(i-1)q+m+1}, ..., Y_{(i-1)q+q-2m}\}^T$, $H_{i_3} = \{Y_{(i-1)q+q-2m+1}, ..., Y_{(i-1)q+q-m}\}^T$, $H_{i_4} = \{Y_{(i-1)q+q-m+1}, ..., Y_{iq}\}^T$, $i = \{0,1,...,P-1\}$.

Step 1

Diagonalize the blocks $A_{i_{11}}$ by premultiplying each partition by

$$
\begin{pmatrix} A_{i_{11}}^{-1} & 0 \\ 0 & I \end{pmatrix}
$$

The computations are

$$\begin{pmatrix} E_{i_1} \\ E_{i_2} \\ E_{i_3} \end{pmatrix} \leftarrow A_{i_{11}}^{-1} \begin{pmatrix} E_{i_1} \\ E_{i_2} \\ E_{i_3} \end{pmatrix} \begin{pmatrix} G_{i_1} \\ G_{i_2} \\ G_{i_3} \end{pmatrix} \leftarrow A_{i_{11}}^{-1} \begin{pmatrix} G_{i_1} \\ G_{i_2} \\ G_{i_3} \end{pmatrix} \begin{pmatrix} H_{i_1} \\ H_{i_2} \\ H_{i_3} \end{pmatrix} \leftarrow A_{i_{11}}^{-1} \begin{pmatrix} H_{i_1} \\ H_{i_2} \\ H_{i_3} \end{pmatrix}$$

The inverse of $A_{i_{11}}^{-1}$ is, in general, not computed explicitly. Instead, the factored form $A_{i_{11}} = L_{i_{11}} U_{i_{11}}$ or $A_{i_{11}} = L_{i_{11}} L_{i_{11}}^T$ is used.

Step 2

Eliminate the upper triangular matrix $A_{i_{21}}$ by premultiplying each partition by

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -A_{i_{21}} & I_m \end{pmatrix}$$

The computations are: $E_{i_4} \leftarrow -A_{i_{21}} E_{i_3} + E_{i_4}$, $G_{i_4} \leftarrow -A_{i_{21}} G_{i_3}$, and $H_{i_4} \leftarrow -A_{i_{21}} H_{i_3} + H_{i_4}$, $i = \{0, 1, .., P-1\}$

Step 3

Eliminate the lower triangular matrices $B_{i_{12}}$ by multiplying the first m rows of partition $i+1$ by $-B_{i_{21}}$ and adding the product to the last m rows of partition i, $i = \{0, 1, .., P-2\}$. This step requires interpartition communication. A block of q rows of the premultiplying matrix is of the form

$$\begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & I & -B_{i_{21}} & 0 & 0_m \end{pmatrix}$$

The computations are: $E_{i_4} \leftarrow -B_{i_{21}} G_{(i+1)_1}$, $F_{i_4} \leftarrow -B_{i_{21}} F_{(i+1)_1}$, and $H_{i_4} \leftarrow -B_{i_{21}} H_{(i+1)_1}$, $i = \{0, 1, ..., P-2\}$

At this point the form of partition i and $i+1$ of the banded matrix is

$$\begin{pmatrix} G_{i_1} & & E_{i_1} & 0 & 0 & & \\ G_{i_2} & I & E_{i_2} & 0 & 0 & & \\ G_{i_3} & & E_{i_3} & 0 & 0 & & \\ G_{i_4} & & E_{i_4} & 0 & F_{i_4} & & \\ & 0 & G_{(i+1)_1} & & E_{(i+1)_1} & 0 & 0 \\ & 0 & G_{(i+1)_2} & I & E_{(i+1)_2} & 0 & 0 \\ & 0 & G_{(i+1)_3} & & E_{(i+1)_3} & 0 & 0 \\ & 0 & G_{(i+1)_4} & & E_{(i+1)_4} & 0 & F_{(i+1)_4} \end{pmatrix}$$

The last m equations from each partition together form a block tridiagonal system of equations of order mP.

Phase 2, the solution of the block tridiagonal system of order mP requires intercluster communication. The major operations are:

- Diagonalization of the diagonal blocks of the reduced block tridiagonal system;
- Matrix-matrix multiplication and addition; and
- Matrix-vector (or matrix-matrix) multiplication (backsubstitution).

Phase 3 is local to a cluster and consists of matrix-vector (or matrix-matrix) multiplication and addition (backsubstitution).

In exploiting the symmetry of the banded system, Cholesky's method can be used to factor $A_{i_{11}}^{-1}$ and the computations proceed according to steps 1 - 3. In using Gauss-Jordan elimination of the computation and application of $L_{i_{11}}^{-1}$ and the elimination of $A_{i_{21}}$ is performed in a forward elimination step; and the application of $U_{i_{11}}^{-1}$ and the elimination of $B_{i_{12}}$ in a backward elimination step. Computations in these steps are local to a cluster, except for the elimination of $B_{i_{21}}$. Each cluster is assigned a banded system of q equations, each with R right hand sides. For a nonsymmetric matrix the number of nonzero diagonals is $2m+1$. For a symmetric matrix it suffices to store $m+1$ nonzero diagonals. The matrix is of order $q \times q + 2m$ in the nonsymmetric case. We first consider Gauss-Jordan elimination.

3.2 Phase 1. Transformation of the Banded System into a "Separable" System

3.2.1 Concurrent Gauss-Jordan Elimination in Clusters

Transformation of the banded system to a system that is solvable in two additional phases can be performed by Gauss-Jordan elimination, concurrently performed in all clusters. We briefly outline the computations and give a complexity estimate. The rows and columns are labeled $iq + j$ with $i = \{0, 1, ..., P-1\}$, and $j = \{0, 1, ..., q-1\}$. The elimination of subdiagonal elements in columns $iq + j, i = \{0, 1, ..., P-1\}$ and $j = \{0, 1, ..., q-m-1\}$ is performed in a forward elimination phase, concurrently for all i, and for each i sequentially for j (pivoting on the diagonal in each partition). Rows with the same value of i are assigned to the same cluster. In the elimination of the subdiagonal elements in column $iq + j$, row $iq + j$ is communicated to the processors storing rows $iq + j + k, k = \{1, 2, ..., m\}$. No interpartition communication is required

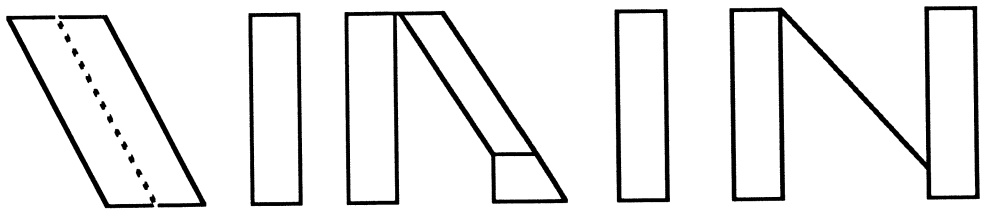

Figure 10: The locations of nonzero elements in a partition during steps 1 - 3.

since $j+k \leq q-1$. The elimination of the superdiagonal elements in columns $iq+j, i = \{0,1,...,P-1\}$ and $j = \{0,1,...,q-m-1\}$ is performed in order of decreasing values of j, concurrently for all i. For this elimination row $iq+j$ is communicated to the processors storing rows $iq+j-k, k = \{1,2,...,m\}$. Hence, communication between adjacent partitions is required for the elimination operations on columns with $j < m$. The elimination of $B_{i_{21}}$ requires communication of $m(2m+R)$ elements, if the elimination of $B_{i_{21}}$ is performed in cluster i. By performing the elimination operations in cluster $i+1, m(3/2m+R)$ elements need to be communicated.

After the forward elimination phase, the blocks $((i+1)q+k, iq+j), k,j \in \{0,1,..,q-1\} \times \{q-m,...,q-1\}$ are filled-in for all $i = \{0,1,..,P-2\}$. Similarly, after the backward elimination phase the blocks $(iq+k, iq+j), k,j \in \{0,1,..,q-1\} \times \{q-m,...,q-1\}$ are filled-in for all $i = \{0,1,..,P-1\}$, and so are the blocks $(iq+k, iq+j)$, $k \in \{-m,-m+1,...,-1\}$, $j \in \{q-m,...,q-1\}$ for $i = \{1,2,...,P-1\}$ (caused by the elimination of $B_{i_{21}}$). The locations of the nonzero elements in partition i, initially, after the elimination of the subdiagonal elements, and after the elimination of superdiagonal elements are shown in Figure 10.

For each cluster we assume that the rows and columns are stored cyclicly, and employ the algorithms for the elimination of a single vertex. Computation and application of the factors $L^{-1}_{iq+j}, j = \{0,1,..,q-m-1\}$ in partition i for $P > 1$ differ from the case $P = 1$ only in that there is fill-in, and there are additional computational windows for each operational window. The complexity increases accordingly. The backward elimination is carried out similarly to the forward elimination, except that the direction of communication for rows and columns may be reversed (but need not be) compared to the forward elimination step. Also, in the last m steps of the backward phase processor row 0 of cluster i communicates the pivot row to processor row $\sqrt{K_c}-1$ of cluster $i-1$ instead of the row $\sqrt{K_c}-1$ of the same cluster.

The time for diagonalizing the diagonal blocks $A_{i_{11}}$ and eliminating $A_{i_{21}}$ and $B_{i_{21}}$ through Gauss-Jordan elimination, requires a time of approximately $2(\frac{N}{P}-m)\{2t_c+t_d+(\lceil\frac{m+1}{\sqrt{K_c}}\rceil-1)max(2t_a,t_c,t_d)+((\lceil\frac{2m+1+R}{\sqrt{K_c}}\rceil-1)\lceil\frac{m+1}{\sqrt{K_c}}\rceil+1)max(2t_a,t_c)\}-2max(2t_a,t_c)+$

$4(\sqrt{K_c}-2)t_c + 2(t_d + 2t_a)$ on P clusters, each in the form of a torus of $\sqrt{K_c} \times \sqrt{K_c}$ processors.

The communication bandwidth for intercluster and intracluster communication is assumed to be the same. In the above estimate, no pipelining between the forward and backward elimination phases is assumed. This assumption only affects the propagation term $(4(\sqrt{K_c}-2))$, which is of lower order for few partitions, narrow bands, and small tori. However, for a maximally concurrent implementation the propagation time may be the dominating term. For $\lceil \frac{m+1+R}{\sqrt{K_c}} \rceil = 1$ the complexity estimate becomes $2(\frac{N}{P} - m)(2t_c + t_d + max(2t_a, t_c)) + 4\sqrt{K_c} + const$, which is 0 for $P = \frac{N}{m}$ (the maximum number of partitions). For $\lceil \frac{m}{\sqrt{K_c}} \rceil \approx \frac{m}{\sqrt{K_c}}$ the high order terms are $\alpha(\frac{N}{P} - m)m\frac{2m+1+R}{K_c} + \beta\sqrt{K_c}$. The speed-up is of order $O(K_c)$.

3.3 Phase 2. Solution of the (Reduced) Block Tridiagonal System

Two candidate methods for the solution of the reduced block tridiagonal system are: two-way block Gaussian elimination (2BGE), and block cyclic reduction (BCR). 2BGE is only of interest on intercluster connections forming linear arrays; while BCR is particularly interesting on intercluster connections forming binary trees, shuffle-exchange or perfect shuffle networks, and Boolean cubes. However, as shown in [16] BCR may be of a lower total complexity than 2BGE even on a linear array.

The solution of the block tridiagonal system proceeds in three steps:

- Diagonalization of the diagonal blocks (multiplication of a block row by the inverse of the diagonal block);
- Elimination of off diagonal blocks (one block per block row for 2BGE, 2 per block row for BCR); and
- Computations of unknowns through backsubstitution.

We discuss each of these three steps separately. For simplicity, we assume that $P = 2^p - 1$. 2BGE proceeds in $\lfloor \frac{P}{2} \rfloor$ elimination and backsubstitution steps, BCR in $p-1$ elimination and backsubstitution steps.

3.3.1 Multiplication of a Block Row by the Inverse of the Diagonal Block

In both 2BGE and BCR the elimination of an off-diagonal block is accomplished through a linear combination of two rows. This elimination can be carried out by first multiplying

Configuration	Algorithm		Ensemble size
	BCR	2BGE	
Linear array	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	$m=\sqrt{K_c}$
Perfect shuffle	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	
Boolean cube	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	$m(\alpha+\beta\lceil\frac{R}{2\sqrt{K_c}}\rceil)$	
Binary tree	$m(\alpha+1+\beta\lceil\frac{R}{\sqrt{K_c}}\rceil)$		
Linear array	$m(\frac{m(16m/3+2R)}{2K_c}+\frac{m+R/2}{\sqrt{K_c}}+\gamma)+const$	$m(\frac{m(10m/3+2R)}{2K_c}+\frac{m+R}{2\sqrt{K_c}}+\gamma)+const$	
Perfect shuffle	$m(\frac{m(16m/3+2R)}{2K_c}+\frac{m+R/2}{\sqrt{K_c}}+\gamma)+const$	$m(\frac{m(10m/3+2R)}{2K_c}+\frac{m+R}{2\sqrt{K_c}}+\gamma)+const$	$\lceil\frac{m}{\sqrt{K_c}}\rceil\approx\frac{m}{\sqrt{K_c}}$
Boolean cube	$m(\frac{m(16m/3+2R)}{2K_c}+\frac{m+R/2}{\sqrt{K_c}}+\gamma)+const$	$m(\frac{m(10m/3+2R)}{2K_c}+\frac{m+R}{2\sqrt{K_c}}+\gamma)+const$	$\lceil\frac{R}{\sqrt{K_c}}\rceil\approx\frac{R}{\sqrt{K_c}}$
Binary tree	$m(\frac{m(14m/3+2R)}{K_c}+\frac{2m+R}{\sqrt{K_c}}+\gamma)+const$		

Table 3: The complexity of multiplying a block row by the inverse of the diagonal block.

the block row used to eliminate an off-diagonal block by the inverse of its diagonal block, $E_{i_4}^{-1}$. Then, the resulting row is multiplied by the block matrix to be eliminated, and the two rows are added. There is one block row per cluster. In 2BGE the off-diagonal blocks of the pivot row are $(F_{i_4}H_{i_4})$ for the first half of the block rows (and $(G_{i_4}H_{i_4})$ for the second half), and in BCR $(G_{i_4}F_{i_4}H_{i_4})$. Computation of the inverse of the diagonal block and multiplication of the off-diagonal blocks thereby can be accomplished using Gauss-Jordan elimination. The speed-up is proportional to the number of processing elements in a cluster, $m \geq \sqrt{K_c}$ [11]. Computations for a block row can be shared between two adjacent clusters at the extra expense of communicating the diagonal block between the two clusters. Then, the number of columns of the off diagonal blocks treated by a cluster is $\frac{1}{2}(m+R)$ for 2BGE and $(m+\frac{1}{2}R)$ for BCR. Intracluster communication can take place concurrently with the computations. From [11], we derive the complexity estimates in Table 3 for 2BGE and BCR:

An algorithm for the shuffle-exchange network can be obtained from the binary tree algorithm by embedding the tree in the shuffle-exchange network (see e.g. [14]).

3.3.2 Block Forward Elimination/Reduction

The elimination of an off diagonal block requires the multiplication of a block row by a matrix, followed by the addition of two block rows. The multiplication can take place in the cluster that stores the block row; the cluster that stores the block to be eliminated; or it may be distributed among several clusters. For 2BGE we choose to perform the multiplication in the two adjacent clusters sharing the computations of step 1 of phase 2. For BCR a block row is multiplied by 2 $m \times m$ matrices, one for each off diagonal element to be eliminated. We choose to perform the multiplication of one complete block row by an $m \times m$ matrix concurrently in distinct clusters with intercluster connections forming

perfect shuffle networks or Boolean cubes. For BCR on a binary tree of clusters we instead choose to perform the multiplication in the processor that stores the elements to be eliminated, in order to allow for effective pipelining. Communication complexity would otherwise be $O(\log_2^2 P)$ [14]. For BCR on a linear array we choose to perform the multiplication in two clusters; one storing the block row in which elimination is to take place, and one cluster adjacent to that cluster. Hence, the number of columns of the multiplier is $(m+R)/2$ for 2BGE and $(2m+R)$ for BCR on a linear array, and perfect shuffle and Boolean cube networks. For the binary tree the number of columns of the multiplier is the same, but the multiplicand has $2m$ columns instead of m columns.

Communication of a block row prior to the multiplication is between clusters. For 2BGE the communication is between clusters storing partitions i and $i+1$ in step i of the elimination process. In BCR, the communication is between clusters assigned partitions that differ in index by 2^i, $i = \{0, 1, ..., \log_2 P - 2\}$. The amount of data transferred between corresponding processors in clusters is $\lceil \frac{m}{\sqrt{K_c}} \rceil^2$ prior to multiplication and $\lceil \frac{m}{\sqrt{K_c}} \rceil \lceil \frac{m+R}{2\sqrt{K_c}} \rceil$ after multiplication using 2BGE. In BCR on a perfect shuffle or Boolean cube network $\lceil \frac{m}{\sqrt{K_c}} \rceil \lceil \frac{m+R/2}{\sqrt{K_c}} \rceil$ elements are communicated prior to multiplication and $\lceil \frac{m}{\sqrt{K_c}} \rceil \lceil \frac{2m+R}{\sqrt{K_c}} \rceil$ afterwards. For BCR on a binary tree $\lceil \frac{m}{\sqrt{K_c}} \rceil \lceil \frac{2m+R}{\sqrt{K_c}} \rceil$ elements are communicated prior to multiplication.

For matrix multiplication we employ any one of the algorithms described in [11]. The speed-up is proportional to the number of processors in a cluster. For BCR on ensembles with intercluster connections as binary trees, the matrix multiplication algorithms yield the result directly. For the other cases it is necessary to perform an additional matrix addition, for 2BGE of $\lceil \frac{m}{\sqrt{K_c}} \rceil \times \lceil \frac{m+R}{2\sqrt{K_c}} \rceil$ matrices, and for BCR on perfect shuffle and Boolean cube networks of $\lceil \frac{m}{\sqrt{K_c}} \rceil \times \lceil \frac{m+R}{\sqrt{K_c}} \rceil$ matrices. Assuming that intercluster communication occurs concurrently with computations the complexity estimates are as follows:

For $\lceil \frac{m}{\sqrt{K_c}} \rceil = 1$ the highest order term is $\lceil \frac{m+R}{2\sqrt{K_c}} \rceil \sqrt{K_c}$ for 2BGE, and $\lceil \frac{2m+R}{\sqrt{K_c}} \rceil \sqrt{K_c}$ for BCR on a linear array, perfect shuffle and Boolean cube networks, and $2\lceil \frac{2m+R}{\sqrt{K_c}} \rceil \sqrt{K_c}$ for BCR on a binary tree. For $\lceil \frac{m}{\sqrt{K_c}} \rceil \approx \frac{m}{\sqrt{K_c}}$ and $\lceil \frac{R}{\sqrt{K_c}} \rceil \approx \frac{R}{\sqrt{K_c}}$ the highest order terms become $m^2 \frac{m+R}{2K_c}$ for 2BGE, $m^2 \frac{2m+R}{K_c}$ for BCR on a linear array, perfect shuffle and Boolean cube networks, and $2m^2 \frac{2m+R}{K_c}$ for BCR on a binary tree of clusters.

3.3.3 Backsubstitution

Backsubstitution using 2BGE requires the communication of $\lceil \frac{m}{2\sqrt{K_c}} \rceil \lceil \frac{R}{\sqrt{K_c}} \rceil$ x-values between a pair of processors in different clusters prior to the computation of a matrix-vector product; and the communication of $\lceil \frac{R}{\sqrt{K_c}} \rceil$ matrix-vector products to complete

the computation of a set of m new x-values. For BCR $\lceil \frac{m}{\sqrt{K_c}} \rceil \lceil \frac{R}{\sqrt{K_c}} \rceil$ x-values and $\lceil \frac{R}{\sqrt{K_c}} \rceil$ matrix-vector products are communicated for the computation of each set of m x-values.

Matrix-vector products can be computed by accumulating inner products *in-place*, or *in-space*. Such matrix multiplication algorithms are described in [11]. In-place algorithms are feasible for $\lceil \frac{R}{\sqrt{K_c}} \rceil \approx \frac{R}{\sqrt{K_c}}$, and in-space algorithms are suitable for $R < \sqrt{K_c}$. Using any of the matrix multiplication algorithms in [16], the following complexity estimates can be derived for the high order terms:

For $\lceil \frac{m}{\sqrt{K_c}} \rceil = 1$ the estimate is $(\lceil \frac{R}{\sqrt{K_c}} \rceil \sqrt{K_c})$ for 2BGE and BCR on a linear array, perfect shuffle and Boolean cube networks, and $2(\lceil \frac{R}{\sqrt{K_c}} \rceil \sqrt{K_c})$ for BCR on a binary tree. For $\lceil \frac{m}{\sqrt{K_c}} \rceil \approx \frac{m}{\sqrt{K_c}}$ and $\lceil \frac{R}{\sqrt{K_c}} \rceil \approx \frac{R}{\sqrt{K_c}}$ the estimates are $\frac{m^2}{4K_c} \lceil \frac{R}{\sqrt{K_c}} \rceil \sqrt{K_c}$ for 2BGE, $\frac{m^2}{K_c} \lceil \frac{R}{\sqrt{K_c}} \rceil \sqrt{K_c}$ for BCR on a linear array, perfect shuffle and Boolean cube networks, and $\frac{2m^2}{K_c} \lceil \frac{R}{\sqrt{K_c}} \rceil \sqrt{K_c}$ for BCR on intercluster connections forming binary trees.

3.3.4 Summary of Complexity Estimates for Phase 2.

Adding the complexity estimates above we arrive at the estimates:

The solution of a block tridiagonal system with blocks of size $m \times m$, R right hand sides, and P block rows can be solved on P clusters with intercluster connections forming linear arrays, binary trees, perfect shuffle and Boolean cube networks and intracluster connections forming tori or Boolean cubes in a time that is approximately

2BGE on a linear array:

$$(m(m\frac{13m/3+7R/2}{2K_c} + \frac{m+R}{2\sqrt{K_c}} + \gamma) + c_1)(P-1)/2 + m(m\frac{2m/3+R/2}{K_c} + \frac{m+R/2}{\sqrt{K_c}}) + c_2$$

BCR on a linear array:

$$(m(m\frac{14m/3+3R}{K_c} + \frac{m+R/2}{\sqrt{K_c}} + \gamma) + c_1)(\log_2 P - 1) + m\frac{m+R}{K_c}(P-1) + m(m\frac{2m/3+R/2}{K_c} + \frac{m+R/2}{\sqrt{K_c}}) + c_2$$

BCR on perfect shuffle or Boolean cube networks:

$$(m(m\frac{14m/3+3R}{K_c} + \frac{m+R/2}{\sqrt{K_c}} + \gamma) + c_1)(\log_2 P - 1) + m(m\frac{2m/3+R/2}{K_c} + \frac{m+R/2}{\sqrt{K_c}}) + c_2$$

BCR on a binary tree:

$$(m(m\frac{26m/3+5R}{K_c} + \frac{2m+R}{\sqrt{K_c}} + \gamma) + c_1)(\log_2 P - 1) + m(m\frac{2m/3+R/2}{K_c} + \frac{m+R/2}{\sqrt{K_c}}) + c_2$$

3.4 Phase 3. Solving for the Remaining $(\frac{N}{P} - m)R$ Variables

On completion of phase 2 m x-values are known per cluster. From these x-values the remaining $\frac{N}{P} - m$ x-values within each cluster can be computed. The x-values can be obtained using either an *in-place* or an *in-space* algorithm for the accumulation of inner products. The highest order terms are:

$\frac{(\frac{N}{P}-m)2m}{K_c}\lceil\frac{R}{\sqrt{K_c}}\rceil\sqrt{K_c}$ for 2BGE and BCR.

3.5 The Complexity of Concurrent Elimination of Multiple Vertices

The highest order terms in the complexity estimates for fast banded systems solver for a linear array using 2BGE for the (reduced) block tridiagonal system is $\frac{(\frac{N}{P}-m)m(2m+3R)}{K_c} + m^2\frac{(13m/3+7R/2)(P-1)}{4K_c}$. Clearly, the speed-up is linear in K_c, and the complexity is minimized for $\sqrt{K_c} = m$, the maximum size of a square torus for which the efficiency (speed-up/(number of processor)) is of order $O(1)$ for the operations of highest complexity. If the number of processors in the two dimensions can be chosen independently, then a rectangular torus of size $m \times m + R$ yields the lowest complexity. It is also readily seen that if $PK_c = P_e = const$, the minimum complexity is attained when K_c is maximized. From the expression for the total complexity it is also apparent that there exists an optimum value for P, and that this optimum is of order $O(\sqrt{\frac{N}{m}})$ regardless of K_c (neglecting lower order terms). Indeed, $P_{opt} \approx \alpha\sqrt{\frac{N}{m}}$ where α is a small multiple of the ratio of the arithmetic and communication bandwidths.

If instead of block Gaussian elimination, block cyclic reduction is used to solve the reduced system on a linear array the highest order terms are $(\frac{N}{P} - m)m\frac{(2m+3R)}{K_c} + m^2(\log_2 P - 1)\frac{(14m/3+3R)}{K_c} + m(P-1)\frac{(m+R)}{K_c}$, where the last term is entirely due to communication. As in the previous case the speed-up is linear in K_c, and maximizing the cluster size minimizes complexity. The optimum number of partitions is of order $O(\sqrt{N})$, as in the case of clusters of size 1 [16].

For intercluster connections forming perfect shuffle and Boolean cube networks, the highest order terms in the complexity estimate are $(\frac{N}{P} - m)m\frac{2m+3R}{K_c} + m^2(\log_2 P - 1)\frac{14m/3+3R}{K_c}$, and the speed-up is linear in K_c. This is also true for intercluster connections forming binary trees, since the total complexity in this case is of the form $(\frac{N}{P} - m)m\frac{2m+3R}{K_c} + m^2(\log_2 P - 1)\frac{26m/3+5R}{K_c}$. The number of clusters that minimizes the complexity is of order $O(\frac{N}{m})$, as is the case of clusters of size 1.

We summarize the above results in a few theorems.

Theorem 1 *Exploiting for concurrency the independence of operations (operands) in the elimination of a single vertex yields a speed-up linear in the number of processors for elimination methods on tori and Boolean cubes with a matrix embedding technique preserving proximity, if* $N \gg m$. *Otherwise, there exist an optimum,* $\sqrt{K_{c_{opt}}}$, *of order* $O(m^{2/3})$.

Theorem 2 *Let sets of vertices form complete subgraphs, and let such a subgraph represent a node in a quotient graph. If the nodes in the quotient graph are linearly connected, then exploiting the independence of operations (operands) in the elimination of vertices belonging to different nodes in the quotient graph yields a speed-up sublinear in the number of processors.*

The reason for stating theorem 2 for graphs forming paths, is that such graphs correspond to block tridiagonal matrices. Concurrency in vertex elimination on arbitrary sparse matrices is investigated in [13]. The reduced speed-up in the concurrent elimination of multiple vertices is due to the fact that a progressively decreasing number of vertices (nodes in the quotient graph) remains to be eliminated.

Corollary 1. The minimization of the complexity of the fast banded system solver with respect to cluster size and number of clusters can be made independent of each other for elimination methods on ensembles in the form of sets of clusters with intracluster connections forming a torus or Boolean cube, and a matrix embedding technique preserving proximity. The complexity of solving a banded system of bandwidth $2m+1$ is minimized for clusters of size $m \times m$ for tori with an equal number of processors in both dimensions. With a different number of processors in the two dimensions the complexity is minimized for tori of size $m \times (2m + R)$.

Theorem 3 *The optimum number of clusters depends on the form of intercluster connection and is of order* $O(\sqrt{\frac{N}{m}})$ *if Gaussian elimination is used for the solution of the reduced block tridiagonal system, of order* $O(\sqrt{N})$, *if block cyclic reduction is used for the reduced system on a linear array, and of order* $O(\frac{N}{m})$, *if block cyclic reduction is used on perfect shuffle, Boolean cube, or binary tree networks.*

This result concurs with the result in [16], as should be expected from Corollary 1.

Corollary 2. For a sufficiently large number of clusters block cyclic reduction is of lower computational complexity than Gaussian elimination. The crossover point increases with the cluster size K_c and decreases with the bandwidth.

4 Summary

The algorithms that exploit for concurrency, the independence of operations required for the elimination of a single variable, yield a speed-up proportional to the number of processing elements configured as tori and Boolean cubes. The arithmetic and communication complexity is of order $O(Nm\frac{m+R}{K_c})$ for a matrix of bandwidth $2m+1$ and size N, and the number of processors equal to K_c, $K_c < m(m+R)$.

We have also presented and analyzed a fast banded system solver with a minimum complexity of $O(m + m\log_2(\frac{N}{m}))$. Particularly, this solver is efficient on processor networks consisting of clusters with intracluster connections forming tori or Boolean cubes, and intercluster connections forming binary trees, shuffle-exchange, perfect shuffle or Boolean cube networks. Complexity for P clusters and K_c processors per cluster is $O(Nm\frac{m+R}{PK_c} + m^2\log_2 P\frac{m+R}{K_c})$. For a linear array, there is a term linear in P due to communication. Moreover, if block Gaussian elimination is used instead of block cyclic reduction then the logarithmic term is replaced by a term linear in P. The Boolean cube and perfect shuffle networks offer an improvement over the binary tree by a constant factor; and the binary tree is in turn more efficient than the shuffle-exchange network by a constant factor [14].

Acknowledgement

The support of the Office of Naval Research under contract number N00014-84-K-0043 is gratefully acknowledged.

References

[1] Hassan M. Ahmed, Jean-Marc Delosme, and Martin Morf. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer*, 15:65–82, January 1982.

[2] L.E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm.* PhD thesis, Montana State University, 1969.

[3] Jack Dongarra and S. Lennart Johnsson. Solving banded systems on a parallel processor. *Parallel Computing*, 5(1&2):219–246, 1987. (ANL/MCS-TM-85, November 1986).

[4] D. Evans and M. Hatzopoulus. The solution of certain banded systems of linear equations using the folding algorithm. *Computer Journal*, 19:184–187, 1976.

[5] Michael J. Flynn. Very high-speed computing systems. *Proc. of the IEEE*, 12:1901–1909, 1966.

[6] W. Morven Gentleman and H.T. Kung. Matrix triangularization by systolic arrays. In *Real-Time Signal Processing IV, Proc. of SPIE*, pages 19–26, SPIE, 1981.

[7] Donald E. Heller and Ilse C.P. Ipsen. Systolic networks for orthogonal equivalence transformations and their applications. In P. Penfield Jr, editor, *Proceedings, Advanced Research in VLSI*, pages 113–122, Artech House, 1982.

[8] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, 4(2):133–172, April 1987. (Report YALEU/DCS/RR-361, January 1985).

[9] S. Lennart Johnsson. A computational array for the qr-method. In Jr. P. Penfield, editor, *Proc., Conf. on Advanced Research in VLSI*, pages 123–129, Artech House, January 1982.

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

[11] S. Lennart Johnsson. Dense matrix operations on a torus and a boolean cube. In *The National Computer Conference*, July 1985.

[12] S. Lennart Johnsson. *Fast Banded Systems Solvers for Ensemble Architectures.* Technical Report YALEU/DCS/RR-379, Department of Computer Science, Yale University, March 1985.

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

[14] S. Lennart Johnsson. *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations.* Technical Report YALE/DCS/RR-339, Department of Computer Science, Yale University, October 1984.

[15] S. Lennart Johnsson. Pipelined linear equation solvers and vlsi. In *Microelectronics '82*, pages 42–46, Institution of Electrical Engineers, Australia, May 1982.

[16] S. Lennart Johnsson. *Solving Narrow Banded Systems on Ensemble Architectures.* Technical Report YALEU/DCS/RR-343, Dept. of Computer Science, Yale University, November 1984.

[17] S. Lennart Johnsson. Vlsi algorithms for doolittle's, crout's and cholesky's methods. In *International Conference on Circuits and Computers 1982, ICCC82*, pages 372–377, IEEE, Computer Society, September 1982.

[18] S. Lennart Johnsson and Ching-Tien Ho. *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes.* Technical Report YALEU/DCS/RR-500, Yale University, Dept. of Computer Science, November 1986. To appear in IEEE Trans. Computers.

[19] H.T. Kung and Charles E. Leiserson. *Algorithms for VLSI Processor Arrays*, pages 271–292. Addison-Wesley, 1980.

[20] Seymor Parter. The use of linear graphs in gaussian elimination. *SIAM Review*, 3(2):119–130, 1961.

[21] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms.* Prentice Hall, 1977.