CALIFORNIA INSTITUTE OF TECHNOLOGY
Computer Science

4087:TR:80

Gaussian Elimination on Sparse Matrices and Concurrency

by

Lennart Johnsson

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

1 Introduction

The purpose of the work reported here is to gain an understanding of the limitations on concurrency that is inherent in Gaussian elimination on sparse systems of linear equations. Implementation issues and machine architectures are largely disregarded in this document. The purpose of this investigation is however to serve as a basis from which algorithms and architectures can be developed and evaluated.

The rapid evolution of the integrated circuit technology has made the solution of ever larger problems feasible. Many problems within the field of applied mathematics and numerical analysis would however benefit from computers of substantially increased computational power. VLSI holds promise to offer such power. The physical characteristics of the technology is however changing in such a way that the potential that VLSI offers can be explored only to a lesser extent by von Neuman type architecture. To fully explore the potential performance of VLSI a machine shall consist of an ensemble of processors that concurrently carry out a substantial part of the total computation with a limited amount of communication between one another. The communication structures of algorithms and machines are of great concern. Locality is the key to high performance.

Algorithms have with few exceptions been devised for sequential machines. Efficient use of ensembles of processors requires new algorithms to be devised. New algorithms are also required for an evaluation of various interconnection patterns for the processors. Precedence relations can significantly reduce concurrency.

Architectures and algorithms suitable for implementation in VLSI technology have been proposed for instance for typical matrix operations such as matrix vector multiplication, matrix matrix multiplication and LU-decomposition. Browning [1] used a machine model consisting of processors connected as a tree. H T Kung and Leiserson [6] have proposed hexagonal arrays and S Y Kung [7] orthogonal arrays for matrix multiplication and LU-decomposition. Typically the matrices studied are treated as dense or possibly as band matrices. Complexity measures such as area and time for those machines and algorithms are summarized in the table below. The measures applies to the case of performing factorization on a matrix as obtained for instance when five or nine point difference approximations are used for partial derivatives at the nodes of an n*n grid.

	area	time	area*time
Tree Machine	$2n^4$	$2n^4+O(n^2 log_2 n)$	$4n^8+.$
Hexagonal Array	n^4	$4n^2$	$4n^6$
Orthogonal Array	n^4	$3n^2$	$3n^6$
Sequential machine sparsity techniques	$1+abn^2\log_2 n$	cn^3	$\sim dn^5\log_2 n$

In the table above the constant a is intended to reflect the difference in area requirements between a processor and the storage of one word. A rough estimate of a is 10^{-3} - 10^{-4}. The constants b and c have values in the range 8 to 10 respectively, Rose and Whitten [12]. Hence d is of the order 10^{-2}.

From the table it is clear that the area*time estimate using sparsity techniques on sequential machines compares favorably with the estimates for the concurrent computing machines. The time estimates for those are better but at the expense of considerably larger area. The number n may be in the range 10^2 to 10^3 for large problems. Sparse matrix techniques exploit the structure of matrices. Band matrices have a simple and very regular form of sparsity. Some problems can be formulated such that the related matrices have a fairly narrow and dense band. For many problems this is not possible or feasible. This is the general experience for networks, Tinney [16], and finite element problems, George [4].

Sparse matrix techniques have been developed and extensively used during the last decade for sequential machines. Except for band matrices we know of no corresponding development for concurrent computing machines.

In section 2 a brief review of solution of sparse systems of linear equations is given. The relationship between sparse matrices and graphs is illustrated. Section 3 contains suitable graph theoretical definitions. A review of results in computational complexity for sparse matrices is given in section 4. A few algorithms that attempts to explore all possible concurrency in the elimination process is described in section 5. A complexity analysis is carried out for their application to a few regular graphs. A discussion on generalizations can be found in section 6.

2 Solution of sparse systems of linear equations

Systems of linear equations arises frequently in mathematical modeling of problems in various fields. In a matrix formulation the matrix can often be considered as sparse, i.e. only a few elements are non-zero. These elements can occur in a very regular pattern such as is typically the case when finite difference methods are used for the solution of partial differential equations, or somewhat less regularly as is the case in many fifnite element problems or fairly irregularly as in many network applications. The problem of solving a set of linear equations is often written as

$$Ax=b$$

where A is a matrix, b a known vector and x the desired solution. Typically, even if A is a sparse matrix its inverse is not sparse. In the interest of saving storage and operations (multiply and add) to compute x, methods like LU-decomposition are used. Using LU-decomposition the matrix A is factored into a lower triangular matrix L and an upper triangular matrix U. The solution x is obtained by what is usually called forward and

backward substitution as follows

$$A=LU$$
$$Ly=b$$
$$Ux=y$$

There exist several methods for the computation of L and U. One is Gaussian elimination.

In the elimination process new non-zero elements are often created. These elements are called fill-in elements. Consider for example a matrix with the structure below (x denotes any nonzero value).

```
x x x x x
x x 0 0 0
x 0 x 0 0
x 0 0 x 0
x 0 0 0 x
```

The elimination creates two dense triangular factors. By simultaneous interchange (permutation) of rows and columns it is possible to create a matrix with the structure below.

```
x 0 0 0 x
0 x 0 0 x
0 0 x 0 x
0 0 0 x x
x x x x x
```

On this matrix the elimination creates no new elements. Hence, the order in which the elimination proceeds is of importance with respect to the amount of storage and number of operations required to compute the triangular factors. Not all orders will give a correct result. Certain precedence relations are inherent in the method. Graphs can be used to improve the understanding of the nature of LU-decomposition. Graphs are especially useful in investigating operations on sparse matrices. A large body of literature exist on techniques for sequential machines and sparse matrices. A good survey was made by Duff[2].

Techniques for sequential machines exploit in various ways the zero non-zero structure of the matrices. Data structures are very important. The number of operations and storage locations required to, for instance, perform Gaussian elimination depends on the structure of the matrix and the way it is described or organized and accessed.

In this document we will limit the study to Gaussian elimination on symmetric, sparse positive definite matrices. Pivoting is assumed not to be an issue.

A graph G(X,E) can be associated with a matrix A of order n. X is a set of n points and E is a set of pairs (i,j) often called edges or arcs. With each element a_{ij} not equal to zero we associate a pair in E. For a symmetric matrix there is always also an element a_{ji} that is non-zero if a_{ij} is non-zero. Hence if there is a pair (i,j) in E there is also a pair (j,i) and the graph can be regarded as undirected. The graphs corresponding to the two matrices above can be drawn as follows:

Figure 1.

Disregarding the node numbers the graphs are the same. The permutation of rows and columns correspond to relabeling or reordering the graph. Gaussian elimination interpreted in a graph context corresponds to removal of nodes. In the process of removing a node new edges are often generated. Two nodes i and j are said to be adjacent if there is an arc connecting them, i.e. there exist a pair (i,j) in E. In removing a node i all the nodes adjacent to i will become mutually adjacent. Hence if the nodes adjacent to i were not mutually adjacent before the removal of i new arcs will be generated.

A graph theoretical interpretation of Gaussian elimination was first published by Parter[10]. During the last decade extensive research has been carried out on this subject. The research has so far been using a sequential machine model when complexity estimates have been derived. Here a graph theoretical interpretation will be used to explore possible concurrency in Gaussian elimination. We will basically follow Harary [5] and Rose [11] in introducing concepts suitable for the analysis.

3 Graph concepts

Given a matrix A we associate with it an ordered graph G such that vertex x_i corresponds to row i of A and (x_i,x_j) E iff $a_{ij}\neq 0$, $i\neq j$. G(X,E) then corresponds to all matrices PAP^T, P being a permutation matrix. A is assumed to be irreducible. If A is a symmetric matrix the graph can be considered as undirected.

For the study of potential concurrency in Gaussian elimination it is convenient to use the following definitions:

Definition 1. The cardinallity of a set X, denoted |X|, is the number of elements in X.

Definition 2. A graph G is a pair (X,E) where X is a finite set of |X| elements called vertices or nodes and $E \subseteq \{(x,y) | x,y \in X, x \neq y\}$ is a set of |E| vertex pairs called edges or arcs.

Definition 3. The adjacency set of x, adj(x), is the set of nodes y defined as $adj(x)=\{y \mid y \in X, (x,y) \in E\}$. The nodes y are said to be adjacent to x.

Definition 4. A path p of length n is an ordered set of distinct vertices. $p=[x_1,x_2,...,x_{n+1}]$ $x_i \in X$, i=1,2,...,n+1 and $x_{i+1} \in adj(x_i)$

Definition 5. A cycle c of length n is an ordered set of distinct vertices such that $c=[x_1,x_2,...,x_n]$, $x_i \in X$, i=1,2,...,n $x_{i+1} \in adj(x_i)$ i=1,2,...,n-1 and $x_1=adj(x_n)$

Definition 6. A connected graph is a graph such that for any $x,y \in X$ there is at least one path from x to y.

Definition 7. The degree of a node x equals |adj(x)|.

Definition 8. A graph such that all nodes have the same degree r is said to be regular and to be of degree r.

Definition 9. The distance d(x,y) between any two nodes $x,y \in X$ in a connected graph G is the length of the shortest path between x and y.

Definition 10. The diameter of a graph G, D(G), is max d(x,y) over all x and y, $x,y \in X$.

Definition 11. The eccentricity of a node $x \in X$, e(x), is max d(x,y) over all $y \in X$.

Definition 12. The radius of a graph G, r(G), is min e(x) over all $x \in X$.

Remark. The diameter D(G)=max e(x) over all $x \in X$.

Definition 13. A node x of a graph G is peripheral if e(x) = D(G).

Definition 14. A central node of a graph G is a node x such that e(x)=r(G)

Definition 15. The center of a graph G is the set of all central nodes.

Definition 16. Two graphs G and H are isomorphic if there exist a one-to-one correspondence between their node sets which preserve adjacency.

Definition 17. An end node or a leaf is a node of degree 1.

Definition 18. A subgraph of a graph G is a graph with all its nodes and edges in G.

Definition 19. The section graph G(Y) of a graph G is a graph with node set $Y \subset X$ and edge set E(Y) such that $E(Y)=\{(x,y)\in E \mid x,y\in Y\}$, $G(Y)=G(Y,E(Y))$

Definition 20. A spanning subgraph of a graph G(X,E) is a subgraph with node set X.

Definition 21. A graph said to be complete if every node is adjacent to every other node. A complete subgraph is a subgraph which is complete.

Definition 22. A tree is a connected graph without cycles.

Theorem 1. (Harary) Every tree has a center consisting of either one node or two adjacent nodes.

Definition 23. A spannig subgraph which is also a tree is called a spanning tree.

Definition 24. A chord of a spanning tree T of a graph G is an edge of G that is not in T.

Definition 25. A rooted tree has one node, its root, distinguished from the others.

Definition 26. In a tree every pair of nodes are joined by a unique path. A node x being on the path from the root to a node y is called the ancestor of y. The node y is called a descendant of x. If x and y are adjacent x is also called parent of y and y said to be a child of x.

Definition 27. A separator of a graph G is a set $S \subset X$ such that the section graph G(X-S) consists of two or more connected components, $C_i=C_i(V_i,E_i)$, $V_i \subset X$, $E_i \subset E$

Definition 28. Let G be a graph with a separator S and the corresponding components $C_i(V_i,E_i)$. The section graphs $G(SUV_i)$ are the branches of G.

Definition 29. A minimal separator is a separator such that no subset of S is also a separator.

Definition 30. A clique C of a graph is a maximal subset of vertices which all are pairwise adjacent.

Definition 31. A separation clique is a set of vertices which forms a clique and a separator.

Definition 32. Let P be a partition of the node set X, $P=\{Y_1,Y_2,\ldots,Y_p\}$ such that $X=U_{i=1}^{p} Y_i$ and $Y_i \cap Y_j=0$, $i \neq j$. The quotient graph of G with respect to the partition P, G/P is the graph $G/P = (P,E)$, where $(Y_i,Y_j)\in E$ iff $Adj(Y_i)\cap Y_j \neq 0$. $Adj(Y) = \{x\in X\setminus Y \mid y\in Y, (x,y)\in E\}$.

Definition 33. A tree partitioning is a partition P such that G/P is a tree. G/P is called a quotient tree.

Definition 34. A tree partitioning is maximal if there does not exist another partitioning $Q = \{Z_1, Z_2, \ldots, Z_q\}$, q>p and for each i and some j $Z_i \subseteq Y_j$ which yields a quotient tree.

Definition 35. A level structure L is a partitioning of the 10.node set X such that for $L = \{L_1, L_2, \ldots, L_l\}$,

$Adj(L_i) \subseteq L_{i-1} \cup L_{i+1}$, $0<i<l$
$Adj(L_0) \subseteq L_1$, $Adj(L_l) \subseteq L_{l-1}$

Definition 36. The length of the level structure L is l.

Definition 37. The width of the level structure L is defined as $w(L) = \max\{|L_i|, L_i \subseteq L\}$

Definition 38. The rooted level structure at x, denoted $L(x)$, has $L_0 = \{x\}$.

Definition 39. An ordering or labeling l of a graph G is a bijection $l:[1,2,\ldots,n] \leftrightarrow X$

Definition 40. An ordered graph G_l is a graph G(X,E) where X is ordered by l.

Definition 41. Given an ordering l of X, the set of nodes monotonely adjacent to a node $x \in X$, denoted Madj(x), is those nodes $Madj(x)=adj(x) \cap \{z \in X | l^{-1}(z) > l^{-1}(x)\}$

Definition 42. The deficiency set of a vertex x X of an ordered graph G_l is $D(x)=\{(y,z)|$ $y,z \in adj(x)$, $y \neq z$, $y \in adj(z)\}$

Definition 43. The y-elimination graph, G_y of G, $y \in X$ is the graph obtained by

1) deleting y and its incident edges from G.

2) adding edges such that all vertices in adj(y) are pairwise adjacent.
$G_y=(X-\{y\}, E(X-\{y\}) \cup D(y))$

Definition 44. The sequence of elimination graphs $G_1, G_2, \ldots, G_{n-1}$ is defined recursively as $G_i=(G_{i-1})_{xi}$ i=2,...,n-1.

Definition 45. The elimination process P on a graph G(X,E,l) is the ordered set $P(G;l)=[G, G_1, G_2, \ldots, G_{n-1}]$

Definition 46. A perfect elimination process P is an elimination process such that $G_i=G(X-\cup_{j=1}^{i}\{x_j\})$

Definition 47. A monotone transitive ordered graph $G_l=(X,E,l)$ is a graph such that $x \in X$, $y \in Madj(x)$, $z \in Madj(x) \Rightarrow y \in adj(z)$

Definition 48. The monotone transitive extension MTE(G;l) of an ordered graph G_l is $MTE(G;l)=\{X, E \cup_{i=1}^{n-1} t_i\}$, $t_i=D(x_i)$ in G_{i-1}. MTE(G;l) is the graph of L^T. For a matrix A which is a perfect elimination matrix no new edges are added.

Definition 49. A triangulated graph G is a graph such that for every cycle $c=[x_1,x_2,\ldots,x_n,x_1]$ of length > 3, there is an edge of G joining two nonconsecutive vertices of c.

Definition 50. An edge set T(I) such that MTE(G;I)=(X,E∪T(I)) is a triangulation of G.

Definition 51. A minimum triangulation of G is a set T(I) such that |T(I)|=min|T(I)|.

Definition 52. Let A be a symmetric matrix and $f_i=\min\{j|\ a_{ij} \neq 0,\ 0<j<i\}$. Let $b_i = i-f_i$. The bandwidth, *b*, is defined as $\max\{b_i,\ 1<i<N\}$. The envelope of the matrix A, Env(A), is defined as $Env(A) = \{(i,j)|\ 0<i-j<b_i\}$. The profile of A is the number $|Env(A)| = Sum\{b_i,\ 1<i<N\}$.

4 A review of complexity results for elimination on sparse matrices

The examples in section 2 showed that node ordering is of great importance with respect to storage and operations requirements for factorization and forward and backward elimination. The ordering may also affect the numerical accuracy. However we assume here that the numerical accuracy does not have to be considered in the ordering process. There exist several criteria for node ordering. One is to order the nodes in such a way that the number of new edges generated in the elimination process is minimized. This criteria is frequently used where only non-zero elements are stored. Storing only non-zero elements is common practice in network problems. Other storage schemes are based on the envelope of a sparse matrix. In performing Gaussian elimination on a symmetric sparse matrix all new edges correspond to matrix elements within the envelope. Storing only the envelope has gained widespread acceptance in finite element methods. For many problems of this kind the envelope is usually dense when the elimination is completed. A sensible ordering criteria for methods based on the envelope of a matrix is to minimize the profile. Minimizing fill-in and profile usually result in very different orderings. There also exist ordering schemes for bandwidth minimization.

Ordering with respect to concurrency will be studied in the next section.

Ordering with respect to fill-in has been studied by Rose [11] and others. In [11] Rose proved the following results.

Lemma 1. For a symmetric positive definite matrix A with associated unordered graph G=(X,E) the following statements are equivalent

1) A is a perfect elimination matrix

2) there exist an ordering I such that $G_I=(X,E,I)$ is monotone transitive

3) in G_I, MD(x)=0 for all $x \in X$

4) P(G;I) is a perfect elimination process

Lemma 2. Every section graph of a triangulated graph is a triangulated graph.

Proof: Assume G(X) is a triangulated graph. Remove a set A. Consider the section graph G(X-A). Every cycle in G(X-A) is a cycle in G. Hence, there exist a chord since G is triangulated and this chord is also in G(X-A).

Theorem 2. (Rose) For a graph G(X,E) the following statements are equivalent.

1) G is triangulated

2) every minimal a,b separator is a clique

3) there exist an ordering I such that $G_I(X,E,I)$ is monotone transitive

Theorem 3. (Rose) Let G be a triangulated graph with subgraph H(X,F), $F \subseteq E$. Then H is triangulated iff for every edge $(x,y) \in E-F$, there exist an x,y separation clique Se of H.

Corollary 1. Let G(X,E) be a graph with a separation clique S, components C_i and branches L_i. Then any minimum triangulation of G contains only edges $e=(x,y) \in T$ with x and y in the same component C_i or edges (x,y) with $x \in C_i$ and $y \in S$.

Proof. Suppose T contains edges (x,y) $x \in C_i$, $y \in C_j$, $i \neq j$. Then if we delete the edge the new graph is still triangulated according to theorem 2. Hence T can not be minimal which is a contradiction.

According to corollary 1 finding a triangulation for a graph reduces to finding triangulations for its branches.

The graph-theoretic concepts and results mentioned so far have been used by Rose to perform a complexity analysis. Complexity measures used are storage and operations (multiply, divide and add). The criteria for node ordering that are discussed and analysed in [11] are minimum arithmetic, minimum fill-in and minimum bandwidth. In a concurrent computing environment there are also other criteria that are of interest such as the number of steps required to compute for instance L and U, the number of processors that can be used, processor-time-communication trade off etc.

The early results of Rose and others have been considerably extended during the last decade. Tarjan [15] gives an overview of results in complexity theory that applies to Gaussian elimination. Bandwidth elimination requires O(bn) storage and $O(b^2n)$ time. The

bandwidth is a function of the ordering of the nodes. Finding an ordering that minimizes the bandwidth is not easy. A graph for which there exist an ordering such that all elements within the band are nonzero is called a dense bandwidth graph. It can be tested in O(n+m) time (m = number of edges) if a graph is a dense bandwidth graph. The ordering can be found easily.

A graph for which there exist an ordering such that the bandwidth is 1 is called a tridiagonal graph. It can be tested in O(n+m) time if a graph is tridiagonal or not. The test is constructive. Algorithms have also been devised to find bandwidth two orderings in O(n+m) time if such an ordering exist.

The problem of deciding if for a given graph there exist an ordering of bandwidth b or less is NP-complete. This result holds also for graphs in the form of trees.

A generalization of the bandwidth concept is a profile. Profile elimination requires $O(b_1+b_2+...+b_n)$ storage and $O(b_1^2+b_2^2+...+b_n^2)$ time. Finding an ordering that produces a good profile is a hard problem. A graph such that there exist an ordering for which the profile does not contain any zeroes is called a dense profile graph. A graph can be examined in O(n+m) time for the dense profile property, Lueker and Booth [8]. The examination is constructive.

It can be tested in O(nm) time if a graph is a perfect elimination graph. For any ordering the fill-in can be computed in O(nm) time. Finding an ordering that minimizes the fill-in is an NP-complete problem. For a symmetric graph testing if it is a perfect elimination graph can be made in O(n+m) time. The fill-in can also be computed in O(n+m) time.

Since the problem of finding an optimal ordering for many relevant criteria and arbitrary graphs either is proven to be NP-complete or conjectured to be so several heuristic algorithms have been devised and used extensively. The algorithms by Cuthill and McKee and the nested dissection algorithms by George [4] are widely used.

Rose and Whitten [12] have carried out an analysis of dissection methods on graphs forming rectangular meshes. Such graphs occur for instance when five or nine point difference approximations are used for partial derivatives in a two dimensional space. Using a structural recursion formalism they show that dissection strategies on an n*n grid requires $O(n^3)$ time and $O(n^2\log_2 n)$ storage. For an n*m grid the corresponding result is $O(nm^2)$ time and $O(nm\log_2 m)$ storage. Gaussian elimination performed on a matrix originating from using for instance a twentyseven point difference approximation of partial derivatives on an n*n*n grid can be performed in $O(n^6)$ time using $O(n^4)$ storage.

Another interesting result have been derived by Eisenstadt, Schultz and Sherman [3]. They compared bandwidth methods with sparse methods based on dissection like strategies. The following results were established for an n*n grid with nodes of bounded degree.

If the bandwidth is at least O(n)

then a bandwidth method requires $O(n^4)$ time
and $O(n^3)$ storage

and sparse elimination requires $O(n^3)$ time
and $O(n^2)$ storage

For an n*n*n grid and nodes of bounded degree the following results hold

If the bandwidth is at least $O(n^2)$

then a bandwidth method requires $O(n^7)$ time
and $O(n^6)$ storage

and sparse elimination requires $O(n^6)$ time
and $O(n^4)$ storage

Recently the nested dissection strategy has been generalised and an analysis been performed of that strategy on graphs having good separators. The analysis was performed by Lipton, Rose and Tarjan [9]. They showed that a sufficient condition for sparse Gaussian elimination to be efficient is that the graph has good separators. The converse is also true. Lipton, Rose and Tarjan also suggest that a different definition of sparsity be used. They suggest that the property of a graph of having or not having good separators be used. The reason for this is that most sparse graphs according to the classical definition are proved not to have good separators and to fill-in during elimination, that is Gaussian elimination is not efficient.

5 Observations on concurrency

In this section algorithms for the concurrent computation of the LU-decomposition of sparse matrices will be devised and analyzed. There are precedence relations inherent in Gaussian elimination. Pivot elements have to be computed before they are used if a correct result is to be obtained without a certain amount of recomputation. In this report we assume that the diagonal elements are used for pivoting. Some diagonal elements will in general change in the elimination process. Which ones depend on the elimination sequence or the ordering of the nodes of the graph. We define a step to correspond to an interval of time during which computations may be performed concurrently. The order in which operations are carried out in a step is arbitrary. Operations that need to be performed in sequence has to be assigned to different steps to assure a correct result. The length of a time interval corresponding to a step has to be such that all computations to be performed

in a step are completed before the end of the interval. The computations that are assigned to a step is not necessarily equal to all computations that can be performed concurrently. In this investigation we will in a few sample cases determine what is a good strategy.

We will focus our analysis of computational complexity on the number of steps necessary to complete the elimination and the number of processors employed. The graphs corresponding to the matrices studied exhibits a high degree of regularity and can be fully characterized by a few parameters. The intent behind the algorithms devised is that they are of a complexity that either equals that of the lower bound for Gaussian elimination on the particular graph studied or that the complexity differs from the lower bound by not to large a factor. Hence the intent is that the algorithms not only have the same asymptotic behavior but also for any given graph have a complexity close to that of the lower bound.

Elimination of the elements to the right of the diagonal in a row or below the diagonal in a column of a matrix corresponds to the elimination of a vertex in the related graph as defined above. In the process of eliminating a node x all rows and columns corresponding to adj(x) will be affected, but all others unchanged. To assure a correct result it is necessary that the elimination of x does not occur concurrently with the elimination of any node in the set adj(x). However any node y not in adj(x), $y \neq x$ can be eliminated concurrently with x. If adj(x) and adj(y) have nodes in common rows and columns corresponding to those nodes will be updated both due to the elimination of x and y.

The computational model used in deriving the estimates below assumes that the number of processors that can be used by the algorithms is available and that any storage location is accessable within a unit of time.We will distinguish between two forms of concurrency. The model used for strong concurrency assumes that product sums can be evaluated within the time associated with a step of the algorithm. Weak concurrency assumes that only one addition in a product sum can be evaluated within one step. Strong concurrency hence leaves some implementation issues open. The physical time associated with a step depends on for instance how the computation of product sums are implemented. The time to perform an elimination according to the strong concurrency model may equal that of the time required by the weak concurrency model. But it may also be less.

Product sums occur whenever more than one node in adj(x) is eliminated concurrently. This is true for any node x. The number of terms in the product sum equals the number of nodes in adj(x) being eliminated concurrently.

Communication is very important in VLSI technology and most processor-storage interconnection patterns makes different storage locations accessable to different processors within different amounts of time. The model used here can serve as a basis against which the merit of different communication structures can be judged.

Paths

Consider a graph consisting of a single path of length n-1. There exist node orderings such that a graph of this kind can be described by a three-diagonal matrix.

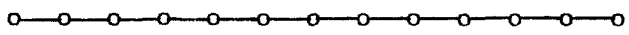

Figure 2.

Three elimination cases can be distinguished.

1) The final node in the elimination process can be any node.

2) The final node has to be one of the end nodes.

3) Both end nodes have to be retained, that is the elimination process is stopped before being completed.

The two latter cases are of interest when considering general graphs parts of which may consist of paths.

A simple algorithm for the concurrent elimination of nodes is to maximize concurrency in each step. In selecting nodes for concurrent elimination it has to be recognized that nodes adjacent to each other can not be eliminated concurrently. An intuitively appealing scheme is to start the selection process with the end nodes, that is nodes of degree 1. Depending on which of the three cases above is investigated and the length of the path both, one or none of the end nodes may be selected. Basically an algorithm assuming strong concurrency will allow for the elimination of half of the remaining nodes in each step. Correspondingly assuming weak concurrency essentially one third of the remaining nodes can be eliminated in each step.

A path will remain a path during elimination. This property is easily proved by contradiction. If elimination of a node creates a loop then it must have had at least three neighbors which contradicts the assumption of the graph being a path.

The following complexity measures for strong concurrency can be derived. Induction can be used to prove the correctness of the estimates.

Case	1	2	3
No. of processors	$pe_i=[m_i/2]+1$	$pe_i=[(m_i-1)/2]$	$pe_i=[m_i/2]$
	$m_{i+1}=m_i-pe_i$	$m_{i+1}=m_i-pe_i$	$m_{i+1}=m_i-pe_i$
	$pe_0=0$, $m_0=n-1$	$pe_0=0$, $m_0=n-1$	$pe_0=0$, $m_0=n-1$
	$m_i>0$	$m_i>0$	$m_i>0$
No. of steps	$[\log_2 n]$	$[\log_2(n-1)]+1$	$[\log_2(n-2)]+1$
New edges	$\Sigma(pe_i-2)$	$\Sigma(pe_i-1)$	Σpe_i

Table 1.

In the table [ ] denotes the largest integer smaller than the number within the brackets.

For weak concurrency similar results can be derived. The number of processors that can be used in each step are $[m_i/3]+1$, $[(m_i-1)/3]+1$ and $[(m_i-2)/3]+1$ respectively. The number of steps for elimination increases approximately with a factor $1/(\log_2 3-1)$.

Theorem 4. The minimum number of steps to perform Gaussian elimination on a sparse matrix with a corresponding graph in the form of a path is as given in table 1.

Proof. We will consider strong concurrency and case one. Due to the precedence relationships not more than $[m_i/2]+1$ nodes can be eliminated in any one step. The algorithm eliminates that number of nodes in every step. After any elimination step the remaining graph is still a path. Decreasing concurrency in any step will obviously leave a larger number of nodes to be eliminated in subsequent steps. At most half of these can be eliminated in the next step. In subsequent steps again at most half of the nodes remaining at the respective steps can be eliminated. Hence for a path maximizing concurrency in every step also minimizes the number of steps to complete the elimination.

Maximizing concurrency in every step does not always minimize the number of steps to complete the elimination. We will show this statement to be true when studying Gaussian elimination on trees next.

The figures in table 1 should be compared with the corresponding figures for sequential algorithms. Such algorithms require n-1 steps for case 1 and no new edges are generated. In case 1 it is possible however to use two processors as long as n>2 without generating new edges (fill-in). The number of steps is [n/2].

Trees

Consider a tree consisting only of one center node of degree n and n leaf nodes. Such a tree can be drawn like a star, Figure 3.

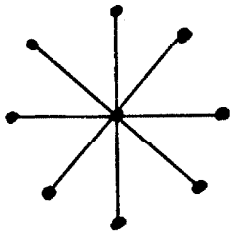

Figure 3.

Considering strong concurrency all the leaf nodes can be eliminated concurrently. Hence with n processors one step only will suffice to complete the elimination process. If we instead consider weak concurrency n steps are required.

Let us now consider a tree with a center node of degree n and all its adjacent nodes having degree n+1. Hence the tree has n^2 leaf nodes, Figure 4.

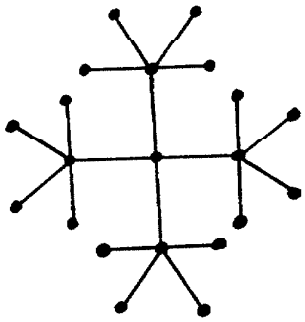

Figure 4.

All n^2 leaf nodes can be eliminated concurrently in the case of strong concurrency. Using n^2 processors this can be done in one step. Having done so we have a tree as in the first example. Assuming weak concurrency n leaf nodes can be eliminated concurrently. In n steps the graph will be reduced to a graph as in the previous example.

The diameter of the graph in the second example is 4. Elimination on this graph can be completed in two steps assuming strong concurrency, in 2n steps assuming weak

concurrency and in n^2+n steps on a sequential machine. No new edges is created in either case.

The elimination strategy adopted in the examples above can be called pruning. Assuming strong concurrency in the second example the center node could have been eliminated concurrently with the leaf nodes in the first step. That would however have made all n nodes adjacent to the center node adjacent to each other. A clique of n nodes would have been created and another n-1 steps required to complete the elimination. This example shows that maximizing concurrency in each step is not equivalent to minimizing the number of steps required to complete the elimination process.

The pruning strategy can be modified to include nodes of degree two to be considered in maximizing concurrency in each step without changing the basic characteristics of the pruning strategy. The graph will remain a tree during the elimination. Elimination of a leaf node does not introduce any new edges. It removes one edge. Elimination of a node x of degree two introduces an edge between the two nodes adjacent to x and removes the edges from x to its adjacent nodes. Hence if there is a cycle in the graph including the nodes adjacent to x, then there was also a cycle in the graph prior to the elimination, which contradicts the assumption.

Assuming strong concurrency the modified strategy will decrease the diameter of the graph with at least two in each step. In the case of weak concurrency the leaf nodes being descendants of one and the same node have to be eliminated sequentially.

Adding chords to a tree increases the adjacency set of the nodes at the ends of the chords. Hence introducing chords can not improve the conditions for concurrency or decrease the number of steps to complete the elimination process. Adding chords need not increase the number of steps though. Hence we just showed the following theorem.

Theorem 5. The minimum number of steps to perform Gaussian elimination on a graph is bounded from below by the minimum number of steps required to perform Gaussian elimination on any spanning tree.

Eliminating a node of degree higher than two generates a clique containing all the nodes adjacent to the node being eliminated. Nodes in a clique can only be eliminated sequentially even if strong concurrency is assumed. We will now investigate if and how the pruning strategy for Gaussian elimination on trees can be modified to reduce the number of steps to complete the elimination. Reducing the number of steps requires that nodes of degree higher than two be eliminated.

Let us consider a node x of degree $k>2$. Assume further that x is not the final node of the elimination. Hence of the k branches at x, k-1 are eliminated one step before the k:th branch. We assume that the branches are ordered with respect to the number of steps required for their elimination. The branch that can be eliminated in the fewest steps is ordered first. Let y be the node in branch k that is adjacent to x. Let the subtrees rooted at the other nodes in the adjacency set of x and not containing x be called T_1, T_2,..., T_{k-1}. Eliminating x we know that at least k-1 additional steps are required to reduce the

clique created by eliminating x to one node which under the assumptions will be y. Let the number of steps to reduce the subtrees T to their respective roots be m_1, m_2,...,m_{k-1}, respectively. If $m_1=m_2=...=m_{k-1}=m$ then m+k-1 steps would be required if x is eliminated concurrently with operations on the subtrees. The modified pruning strategy would give m+2 steps. Hence elimination of x concurrently with elimination in the subtrees is not decreasing the number of steps in this case. For k=3 the same number of steps is required independent of which of the two elimination strategies is used.

The arguments in the previous paragraph proves that the modified pruning strategy applied to balanced trees in general is superior and never inferior to strategies eliminating nodes of degree three or higher.

If we have a strict ordering $m_1<m_2<...<m_{k-1}$ then $\max(k,m_{k-1}+1)$ steps are required if x is eliminated concurrently with operations on the subtrees. The modified pruning strategy requires at most $m_{k-1}+2$ operations. Hence one step can be saved in this case. A step may also be saved even if the ordering is not strict. However if $m_{k-2}<m_{k-1}-1$ the elimination of x concurrently with operations on the subtrees can never reduce the number of steps compared to the modified pruning strategy. Elimination of x concurrently with operations on the subtrees may however require more steps. The following derivation gives the conditions when elimination of x concurrently with operations on the subtrees reduce the number of steps by one.

We have already shown that a necessary condition is

(1) $$m_{k-2}=m_{k-1}-1$$

If x is eliminated concurrently with operations on the subtrees T then elimination of x and the subtrees $T_1,T_2,...,T_{k-3}$ to y requires a number of steps that can be computed as follows.

Let $$S_1=m_1+1$$
and $$S_i=\max(m_i-1,S_{i-1})+1$$

The elimination of x and the subtrees $T_1,T_2,...T_{k-3}$ requires S_{k-3} steps. A necessary and sufficient condition for the elimination of x to be advantageous is that

(2) $$S_{k-3}\leq m_{k-2}=m_{k-1}-1$$

The argument for whether the elimination of a node of degree higher than two is advantageous or not can be applied directly to each tree $T_1,T_2,...,T_{k-1}$ except to their respective roots. The application can be made recursively.

Let us now consider the roots of the trees $T_1,T_2,...,T_{k-1}$. Suppose the degree of the root of tree T_i was k_i prior to the elimination of x. Elimination of the root of T_i after the elimination of x will then generate a clique of $k-1+k_i-1$ nodes. The tree T_i is replaced by k_i-1 trees. At most one step can be saved by this operation since the elimination of the tree T_i and its root which required m_i+1 steps with the modified pruning strategy will now

require at least m_i operations. The situation when a saving actually occurs can be analyzed in the same manner as previously.

Above we assumed that x was not the final node of the elimination. Let us now assume that it would be the final node if the modified pruning strategy is used. The same result as above holds even when x is the final node. Equation (1) holds with k-1 and k-2 replaced by k and k-1 respectively. This modification should be clear since otherwise x would not be the final node using the pruning strategy. Equation (2) also holds with the indices increased by 1.

We have now proved the following theorem.

Theorem 6. The minimum number of steps t to perform Gaussian elimination on a matrix with a corresponding graph in the form of a tree satisfies the inequality

$$[\log_2(D(G)+1)] \leq t \leq r(G)$$

if the final node can be chosen arbitrarily and strong concurrency is assumed.

A path can be viewed as a special kind of tree. Adding nodes to a graph can never reduce the number of steps to perform Gaussian elimination but need not increase the number of steps. Hence the minimum number of steps to complete elimination on a tree is bounded from below by the minimum number of steps required for a path. The upper bound is attained if the tree is balanced and has no nodes of degree two.

If the pure pruning strategy is used then no new edges are created. With the modified strategy a new edge is created every time a node of degree two is eliminated. Eliminating a node of degree k generates k*(k-1)/2 new edges.

Corollary 2. If the final node, x, of the elimination process is specified the number of steps t to complete the elimination satisfies

$$[\log_2 e(x)]+1 \leq t \leq e(x)$$

Cycles

As for paths we can distinguish three cases.

1) The final node can be any node.

2) The final node is specified a priori.

3) Two nodes x and y, $x \neq y$, are to be maintained throughout the elimination.

The third case reduces to two paths with constraints on the elimination as in case 3 for paths. The second case is identical to the first case.

In a cycle all nodes are of degree 2. The algorithm proposed for the cycle maximizes the number of nodes being eliminated concurrently in each step.

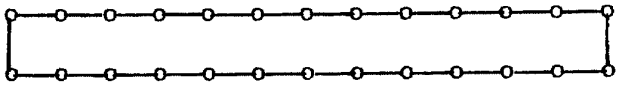

Figure 5.

Theorem 7. The minimum number of steps t to carry out Gaussian elimination on a matrix with a corresponding graph in the form of a cycle containing n nodes is

$$\lceil \log_2(n-1) \rceil + 1$$

assuming strong concurrency.

Proof. Assume that the maximum number of nodes that can be eliminated in a step is eliminated in every step. The degree of all nodes will remain two during the elimination process. The rest of the proof follows that for paths.

The number of nodes pe_i eliminated concurrently in each step satisfies $pe_i=\lceil m_i/2 \rceil$. where $m_{i+1}=m_i-pe_i$, $pe_0=0$, $m_0=n$, $m_i>0$. The number of new edges generated when all concurrency is exploited equals n-3.

A similar analysis can be carried out assuming weak concurrency.

For a purely sequential algorithm the number of steps is n-1 but still n-3 new edges are generated in the elimination process.

Ladder

Cycles of length four can be connected together to form a ladder. A ladder can be formed recursively by adding three edges and two nodes in the following way. Given a ladder or cycle of length four we add one edge to each of two adjacent nodes of degree two and an edge between the two free end nodes of the other two edges. A ladder can be drawn as follows

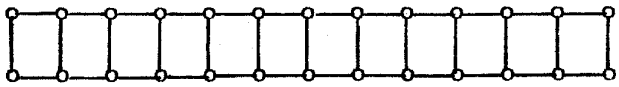

Figure 6.

Let the number of cycles of length four be n-1, that is there are n nodes in the direction of the ladder. For this kind of graph we can construct, assuming strong concurrency, concurrent elimination algorithms for which the number of nodes and steps for elimination are related as follows for i>0

$$3*2^{i} < n < 3*2^{i+1}-1$$
$$t = t_i = 2*i+3$$

$$n = 3*2^{i},\ i = 0$$
$$t = t_i - 1$$

$$n = 3*2^{i+1}-1$$
$$t = t_i + 1$$

The algorithm on which the estimates are based eliminates the maximum number of nodes in each step and assumes strong concurrency. The set of nodes that can be eliminated concurrently is not unique. The set chosen at a particular step has implications on the sets of nodes that can be eliminated concurrently in subsequent steps. The strategy used in deriving the estimates above is that of selecting the set of nodes to be eliminated in each step starting from the "ends" or nodes of lowest degree. Selecting the nodes in this manner does not make the sets of nodes that can be eliminated concurrently unique. If the sets for concurrent elimination consistently were chosen in order from one "end" to the other the number of steps to complete the elimination sometimes would be larger than the numbers stated above. At a particular stage in the elimination process the two strategies may well generate sets for concurrent elimination which contain equally many nodes. The first strategy does however, in some cases, allow a higher degree of concurrency in subsequent stages.

Figure 7 shows the elimination graph after the first step using the algorithm described above.

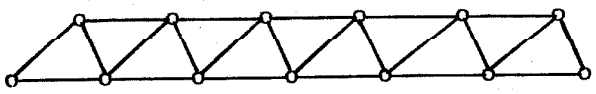

Figure 7.

After the second step of the algorithm the elimination graph can be drawn as in Figure 8.

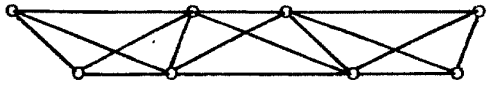

Figure 8.

The graph in Figure 8 consists of three complete graps, each containing four nodes. Hence at most one node at a time can be eliminated from each complete subgraph. Two steps later the elimination graph will be as in Figure 9.

Figure 9.

The elimination can be completed in three steps from Figure 9.

The first strategy of the two mentioned above does not seem to have any adverse effect on concurrency in future stages of elimination. It always generates sets of nodes for concurrent elimination that contains at least as many nodes as the sets generated by the second strategy. The nodes chosen for elimination in the next step are in both strategies selected from the elimination graph of the previous step.

The validity of the estimates for the algorithm can be proved by induction.

Conjecture 1. LU-decomposition on a matrix whose corresponding representation as a graph is a ladder requires at least the number of steps given above, i.e. the minimum number of steps is of the order $2[\log_2(n/3)]$.

The asymptotic behavior of conjecture 1 can not be improved. If the conjecture is false the improvement can not be larger than a factor of two.

Adding edges can never reduce the number of steps to complete the elimination. Hence the minimum number of steps for Gaussian elimination on a ladder is bounded from below by the number of steps required for a cycle of length 2n. This bound equals $[\log_2(2n-1)]+1$.

Spanning trees can also be used to obtain a lower bound. To get as large a lower bound as possible a spanning tree of maximum diameter should be used since the elimination on the ladder requires at least as many steps as any spanning tree (theorem 5). For the ladder there is obviously a spanning tree in the form of a path of length 2n-1. Hence concurrent

elimination will require at least $[\log_2 2n]$ steps.

At this point it should be noticed that for any graph G its diameter is always less or equal to the diameter of any spanning tree. The graph G can be obtained by adding chords to a spanning tree. Adding chords introduces new paths. The shortest path between any two nodes can obviously not increase. Hence from the definition of the diameter the diameter of G has to be less or equal to that of any spanning tree. Adding chords may decrease the diameter and increase the number of steps required for elimination.

Remark. The time complexity of a path, a cycle and a ladder are all of order $\log_2 n$. However for the ladder the logarithm is multiplied by a factor of two, which is not the case for the path and the cycle.

Double ladder

The next natural step towards a general regular mesh is to consider two ladders each containing n cycles of length four joined together along one side. The graph so obtained has 2n cycles of length four and n-1 nodes of degree four, 2n nodes of degree three and four nodes of degree 2. The double ladder can be drawn as follows

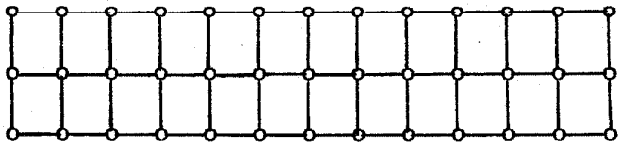

Figure 10.

The same strategy that was adopted for the ladder can be used also in this case. It has to be modified somewhat in certain cases to achieve the following estimates of the number of steps to complete the elimination.

$n = 3 \quad t = 4$
$n = 4,5 \quad t = 5$

$i>0$

$4*2^i+1 < n < 4*2^{i+1}-2$
$t = t_i = 3*i+5$

$n = 4*2^i-2$
$t = t_i-2$

$$4*2^{i}-1 \leq n \leq 4*2^{i}+1$$
$$t = t_i - 1$$

The cases where the strategy used for the simple ladder has to be modified to achieve the estimates above are the ones corresponding to

$$n = 4*2^{i} \text{ and } 4*2^{i}+1$$

In the first case one of the center nodes is retained as long as possible. The same modification as is proposed below for the second case can also be used in the first case to achieve the estimates above.

In the second case maximum concurrency is not used in steps three through five. The maximum concurrency strategy used in the first two steps generates a graph that has two nodes of degree 3, four nodes of degree 5, and two nodes of degree 7, all at the "ends" of the elimination graph. There are also $[n/2]-3$, $n>6$, nodes of degree 6 and $[n/2]-4$ nodes of degree 8. The nodes of degree five and seven are adjacent to the nodes of degree three. During steps three through five the nodes of degree three and five are eliminated. Also $[([n/2]-3)/2]$ nodes of degree 6 that are not mutually adjacent or adjacent to the nodes of degree 5 are eliminated during these steps. The elimination graph after five elimination steps is an array of complete hexagons.

The elimination graph after two steps of the elimination process is shown in Figure 11.

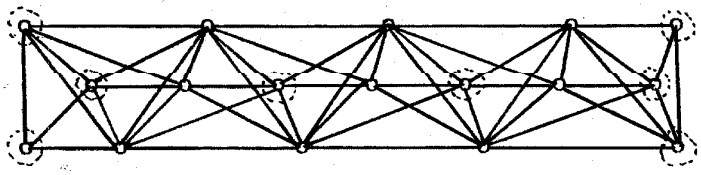

Figure 11.

The nodes eliminated in steps three through five are indicated by circles.

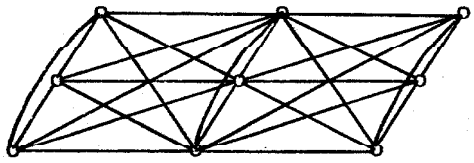

Figure 12.

The graph in Figure 12 illustrates the elimination graph after five steps for n=16.

Induction can be used to prove the correctness of the estimates above.

Conjecture 2. The minimum number of steps to perform Gaussian elimination on a matrix with a corresponding matrix in the form of a double ladder is as given above, i.e. the minimum number of steps is of the order $3[\log_2(n/4)]$.

A lower bound on the number of steps necessary to complete the elimination is $[\log_2(3n-1)]+1$ for the double ladder. This follows from the fact that there exist a cycle of length 3n that contains all nodes of the double ladder. There exists also a spanning tree in the form of a path of length 3n-1.

Remark. The dependency of the number of steps of the algorithm to complete the elimination is again logarithmic in n. It should however be noticed that the constants in the expressions for the double ladder are larger than for the simple one.

Rectangular Meshes

A kind of graph of significant interest is an orthogonal mesh with n nodes in one direction and m nodes in the other. This is a generalization of the ladder and double ladder.

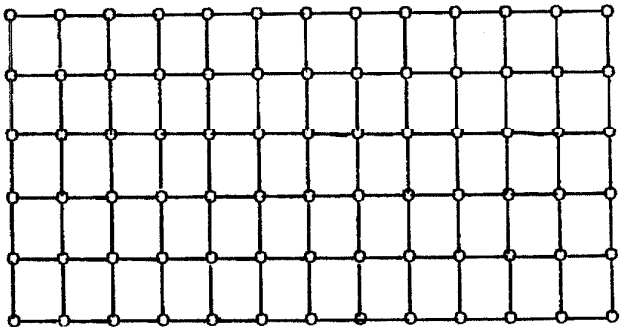

Figure 13.

Graphs of the kind pictured in Figure 13 occurs for instance when five point difference approximations of partial derivatives are used in the numerical solution of partial differential equations by finite difference techniques. If a nine point approximation is used there will also be edges corresponding to the two diagonals in each square.

Conjecture 3. For an orthogonal array with nm nodes, $m<n$, the minimum number of steps to complete the elimination process is of the order $m[\log_2[n/(m+1)]]+2m-1$.

The algorithms discussed for the path, the ladder and the double ladder all obey conjecture 3.

A lower bound on the number of steps is $[\log_2(mn-1)]+1$

An intuitive justification for the bound in conjecture 2 is the following. For $m<<n$ the first few steps of the elimination the graph is transformed into an elimination graph consisting of an array of cliques of 2m nodes. Each clique is obtained from the original graph by the elimination of m^2 nodes. The number of cliques of 2m nodes is of the order $n/(m+1)$. This phase of the elimination can be completed in less than m^2 steps. Two steps were required for the ladder and three steps for the double ladder except for the values of n that needed special consideration. The node set of each clique can be partitioned into two distinct sets X_1 and X_2 each containing m nodes in such a way that except for the cliques at the end of the array X_1 is common to one clique and X_2 to another clique. The quotient graph based on this partitioning of the node set is a path. The length of the path is for most values of n equal to $[n/(m+1)]-1$. For certain values of n it is benefitial to limit the concurrency in the initial phase and extend it a few more steps to create a quotient graph of length one less than given by this expression.

The elimination of a node in the quotient graph requires m steps since the nodes in each partition are fully interconnected. The path models the precedence relations properly since every node in a partition is connected to every node in the two adjacent partitions. The first term in conjecture 2 is now plausible.

When the quotient graph is reduced to one node m-1 additional steps are required to complete the elimination. The other m steps in conjecture 2 stems from the initial phase which for the ladder and double ladder required m steps. For larger values of m we have not discovered a concurrent elimination algorithm that requires m steps for the initial phase. The number of steps has been in the range m to $m\log_2 m$.

Summary of the examples

The potential concurrency in Gaussian elimination is strongly related to the interconnect structure of the graph corresponding to the matrix.

Maximizing concurrency in each step is in general not an optimal strategy with respect to minimizing the number of steps to complete the elimination.

It is difficult if at all possible to create a simple algorithm that also minimizes the number of steps to perform Gaussian elimination on even simple and regular graphs.

In the examples above starting the selection of nodes to be eliminated next from those of lowest degree in the elimination graph from the previous step is consistent with all algorithms above. The selection then proceeds with nodes from the set of nodes with the next higher degree. This selection scheme has to be supplemented with other rules to obtain the results derived above. It seems as if these additional rules can not be based

entirely on the elimination graph from the previous stage in selecting nodes to be eliminated in the next stage. This is in agreement with what was found empirically by Tinney [16] using a sequential machine model. Strategies S2 and S3 in [16] can not be used to generate the optimal selection for the ladders. For the values of n that had to be treated specially, the set of nodes from which a subset to be eliminated shall be chosen all have the same degree and deficiency.

For the examples considered and the algorithms devised for these the time complexity with the computational model assumed is typically logarithmic in the number of nodes and always better than linear.

It is of interest to compare the results derived here with the time complexity estimates derived by H T Kung [6] for hexagonal arrays and S Y Kung [7]for orthogonal arrays. An orthogonal mesh with n nodes in one direction and m nodes in the other direction, $m<n$, can be considered as corresponding to a band matrix with bandwidth 2m-1. The order of the matrix is mn by mn. H T Kung computes a solution using an hexagonal array in 3mn+m steps. The number of processors engaged is m^2. S Y Kung has not to our knowledge published a similiar estimate for LU-decomposition on band matrices. It should however be possible to compute the factors in 3mn steps. To reduce the number of processors to order m^2 other algorithms than those used for the full matrix case has to be devised. The processor requirements can in a fairly straight forward fashion be reduced from m^2n^2 to m^2n.

The basic difference in assumptions made in deriving the results here and those obtained by H T Kung and S Y Kung is that of the communication structure. Here all elements of the matrix or graph have been assumed equally accessable.

6 General graphs

One kind of generalization of the meshes studied above is to include arcs corresponding to the diagonals in the cycles of length four. This type of graph occurs for example when nine point difference approximations of partial derivatives are used in the numeric solution of partial differential equations. The algorithms proposed previously for selecting sets of nodes to be eliminated concurrently can also be used for meshes with diagonal edges included. Since the addition of edges cannot increase possible concurrency the lower bounds given above still apply. The logarithmic term in conjecture 3 still apply as well as the estimate for the final part. The lower bound for the number of steps in the initial part may however be improved. Except for the first few steps the sets of nodes that can be eliminated concurrently should be fairly independant of whether or not the diagonal edges of each cycle of length four are included in the graph.

Conjecture 4. The minimum number of steps to perform Gaussian elimination on a matrix with a corresponding graph in the form of a regular mesh with mn nodes and edges also in the diagonals of every cycle of length four in Figure 13 is of the order $m[\log_2[n/(m+1)]]+2m-1+f$, where f should be no larger than $O(m\log_2 m)$.

Other examples of graphs that may occur as subgraphs of general graphs and are of particular interest in finite element calculations are shown in Figure 14, George [4].

Maximizing concurrency in each step starting the selection of sets of nodes to be eliminated concurrently with the nodes of lowest degree works well on the plain square, the graded L and the squares,i.e. Figures 14 a,b,e,f. This strategy does however not give good results on the plus and H-shaped graphs. They should be considered as special forms of trees. Pairs of adjacent nodes can be considered as composite nodes (recursively) in such a fashion that trees consisting of paths connected together will be the result. The algorithms described for trees can now be applied with the difference that elimination of a composite node essentially takes as many steps as the number of nodes contained in it. This statement is valid for the graphs above and is in general pessimistic.

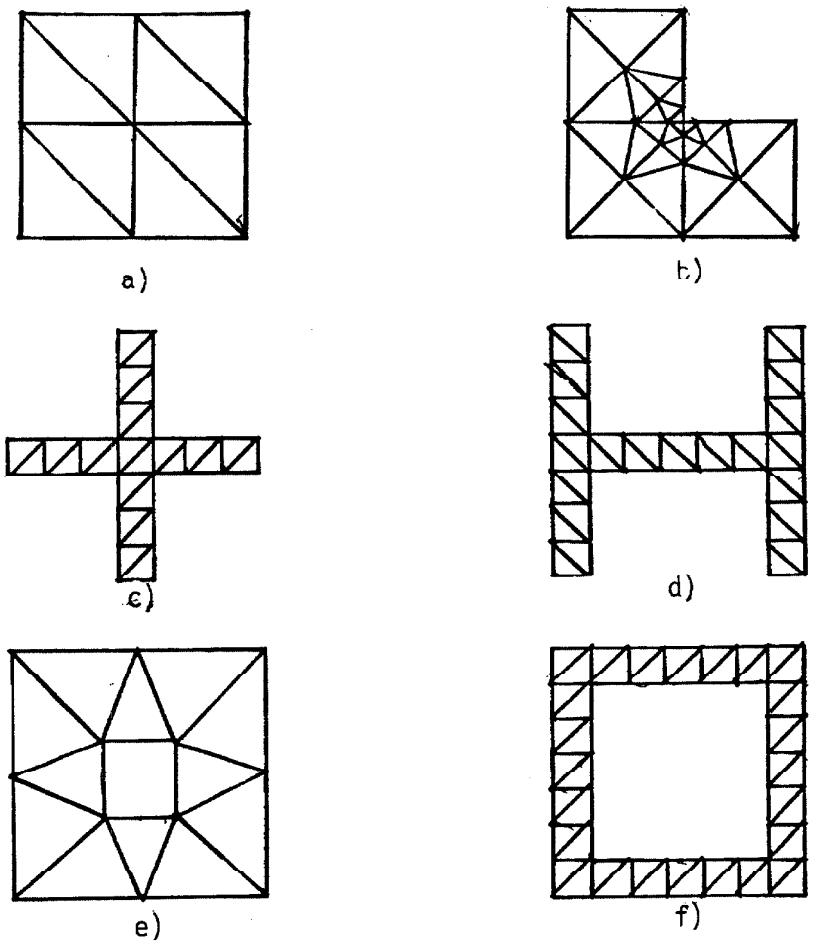

Figure 14. George [4]

The meshes in Figure 14 are often subdivided conformly. Partitioning the node set to find sets for concurrent elimination is still a viable approach. If the node set is partitioned in such a way that the quotient graph is a level structure of width w and length l then the number of steps to complete the elimination is at most $w\log_2 l$. This estimate has the same asymptotic behavior as the bounds derived for meshes.

7 Generalization of algorithms

Devising algorithms that minimize the number of steps to perform Gaussian elimination on a wide range of problems is not an easy task. A number of algorithms have been proposed for sequential machines. Some of those algorithms are both simple to implement and give good results on certain classes of problems such as network problems [16]. George [4] has used nested dissection techniques to find good orderings for finite element methods. Tarjan [14,15] has studied the computational complexity of ordering.

The examples of section 5 leads us to the following general conclusions. If there exist a quotient graph in the form of a path then the number of steps to complete the elimination is at most of the order $w\log_2 l$, where l is the length of the path and w is the maximum cardinallity of the partitions defining the quotient graph. The estimate is in general pessimistic and true when the partitions are of the same cardinallity and the nodes in the partitions interconnected to form complete subgraphs. Since the estimate is linear in w and logarithmic in l longer quotient graphs with partitions of lower cardinallity are better than shorter ones with partitions of higher cardinallity.

A graph in the form of a tree is more favorable than a path with respect to concurrency. A path can always be considered as a tree with a diameter equal to the length of the path. A tree having the same number of nodes as a path has a smaller diameter than a path. A balanced tree containing l nodes and where the center node is of degree k and all other non leaf nodes are of degree k+1 has a radius equal to $\log_k l$. Hence even if a pure pruning strategy is used in the elimination no more than $\log_k l$ steps will be required to complete the elimination. For the path $\log_2 l$ steps are required.

Quotient graphs are more complex but the same properties should hold.

From the above arguments it seems desirable with respect to concurrency to find maximal tree partitionings of the graph. Of two isomorphic quotient trees the tree for which the maximum cardinallity is the smallest is likely to allow for the completion of the elimination in the smallest number of steps. Of maximal tree partitionings the quotient trees of smallest radius are preferable.

George [4], has studied quotient tree and nested dissection algorithms for sequential machine implementation of Gaussian elimination. It should be noticed from the definition of quotient graphs that sets of nodes that are not adjacent in the quotient graph can be

subject to concurrent elimination. Concurrent elimination of nodes in the same set may also be possible, but this is in general not the case.

The quotient tree algorithm does not generate a partitioning which minimize the number of steps to complete the elimination in a concurrent computing environment. Elimination on the plus shaped domain in Figure 14 c can be completed in six steps. The quotient tree algorithm suggest a partitioning of the node set which would require 8 steps. For the double ladder with n = 8 the quotient tree algorithm generates a quotient graph which leads naturally to an elimination scheme requiring 8 steps. The elimination can in this case be completed in 7 steps. For a double ladder with n = 10 the qoutient tree algorithm generates a quotient graph for which the elimination can be performed in 8 steps. The algorithm described in the section concerned with the double ladder also required 8 steps. Hence the quotient tree algorithm performs well for these examples.

The level structure and quotient tree resulting from applying the quotient tree algorithm to the plus shaped graph in Figure 14 c and the double ladder of Figure 10 is shown in Figures 15 and 16 respectively.

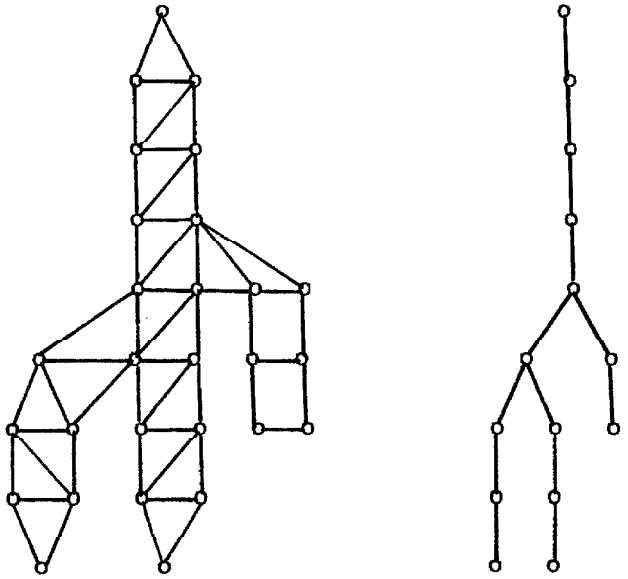

Figure 15.

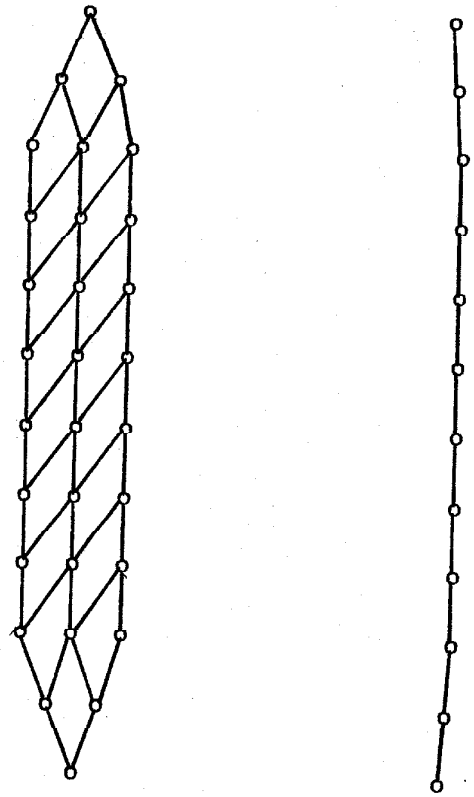

Figure 16.

The initial steps of the nested dissection algorithm determines a minimal separator which is close to the center of the graph. More precisely it determines a minimal separator to the middle level (or one of the middle levels if the number of levels is even) of a level structure rooted at a peripheral node. The nodes of the minimal separator with incident edges are then removed from the graph and the process repeated starting with the generation of new level structures. The process is stopped when no new separators can be found. It should be noticed that except for possibly the first time the graphs considered are disconnected. The separators found by this algorithm are defined to be members of a partitioning of the node set of the graph. The other members are the node sets of each of the subgraphs created by removing all the separators. The quotient graph corresponding to this kind of partitioning is in general not a tree. The partitioning generated by the nested dissection algorithm is not always unique.

For the double ladder for example, the quotient graphs generated by nested dissection may not contain any leaf node at all. Applying the nested dissection algorithm to the plus shaped graph results in a quotient graph with several cycles. However, using these quotient graphs as a basis for concurrent elimination schemes gives good results on the examples studied in this investigation. For the double ladder with n = 8 the number of steps to complete the elimination is 8 and for n = 10 nine steps are required. Hence for

the double ladder one more step is required if the elimination is based on the quotient graph generated by nested dissection than if the algorithms used in section 5 are used.

For the plus shaped graph seven steps are required using nested dissection. The quotient tree algorithm required 8 steps. The number of steps can be reduced to six for the nested dissection algorithm with a modification of the algorithm for finding level structures. The modification that works well for the plus shaped graph reduces the number of levels of the level structure if that can be done without increasing the width as given by the unmodified algorithm.

Both the qoutient tree algorithm and the nested dissection algorithm are intuitively appealing as means of generating quotient graphs that can serve as a basis for concurrent elimination algorithms. In the examples above the two algorithms performed about equally well.

Quotient graphs generated by the nested dissection algorithm for the plus shaped graph of Figure 14 c and the double ladder of Figure 10 are shown in Figures 17 and 18 respectively.

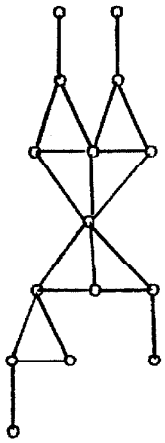

Figure 17.

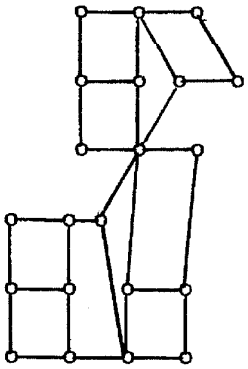

Figure 18.

8 Conclusions

Graphs are a convenient medium for devising algorithms and deriving complexity estimates for Gaussian elimination on sparse matrices. Elimination on sparse matrices using concurrent algorithms has been investigated for a few sample problems. Devised concurrent algorithms required a number of steps to perform the elimination that for trees is logarithmic in the number of nodes in the tree. If the graph corresponding to the matrix can be drawn like a rectangular mesh with m nodes in one direction and n in the other, $m < n$, the minimum number of steps to complete the elimination process is conjectured to be bounded from below by $m\lfloor\log_2\lfloor n/(m+1)\rfloor\rfloor+O(m\log_2 m)$. The concurrent algorithms devised are of a complexity close to this bound for rectangular meshes. For m=1 and m=2 the algorithms agrees with the bound with $O(m\log_2 m)$ equal to 2m-1.

Further investigation is necessary for meshes where m is of the same order as n. Such an investigation should also clarify if the term $O(m\log_2 m)$ in conjecture 4 can be reduced to O(m).

The algorithms devised in this investigation were intended to minimize the number of steps to perform Gaussian elimination on sparse matrices. The utilization of processors decreases as the elimination progresses. Essentially the number of processors active in the first step is of the same order as the number of nodes in the graph. For the path, a regular mesh with nodes of maximum degree four and for a binary tree the number of active processors in the first step is almost precisely equal to half of the number of nodes. For balanced trees of higher degree the number of active processors increases. Towards the end of the elimination process the number of active processors decreases to one.

The area or processor time complexity of the algorithms devised can be summarised as follows assuming strong concurrency.

	area	time	area*time
Path	n/2	$\log_2 n$	$(n/2)\log_2 n$
Tree, degree k	$n-(n-1)/k$	$\log_k(n(k-1)+1)-1$	$O(n\log_k n)$
Regular mesh	nm/2	$m\log_2[n/(m+1)]+O(m\log_2 m)$	$O(nm^2\log_2[n/(m+1)]+nm^2)$

It is of interest to compare these results with the charachteristics of the algorithms and architectures developed for concurrent computation by Kung and Leiserson [6], S Y Kung [7] and Browning [1]. Comparing with algorithms for full matrices is not relevant. Only Kung and Leiserson have to our knowledge published complexity measures for algorithms for LU-decomposition on band matrices. We have not seen any sparse matrix algorithms for the concurrent computation of the triangular factors L and U of a matrix. To make the comparison with algorithms for band matrices fair it is necessary to find orderings that possibly minimize the bandwidth. This is in general a NP-complete problem. For the path and the regular mesh there is a natural ordering which results in bandwidth 1 and m, $m<n$, respectively. Finding an ordering with a small bandwidth for a tree is NP-complete [15]. Ordering the nodes from the leafs towards the center, which in many cases is a good ordering strategy, gives a bandwidth of n/2 for balanced binary trees. For balanced trees of a higher degree the bandwidth is of the order n - n/k. Using these ordering schemes we obtain the following complexity estimates for hexagonal arrays.

	area	time	area*time
Path	4	3n+2	12n+8
Tree, degree k	$(n-n/k+1)^2$	4n-n/k	$O((n-n/k)^3)$
Regular mesh	$(m+1)^2$	3mn+m+1	$3mn(m+1)^2+(m+1)^3$

The algorithms investigated in this report are intended to minimize the number of steps to complete the elimination by exploring the particular structure of a matrix. The time required to compute the L and U factors of a matrix for trees is logarithmic in the number of nodes. The time required by an hexagonal array as given by Kung and Leiserson is proportional to the number of nodes. The minnimal time algorithms deviced here use all processors only in the first step. In the last step only one processor is used. Hence the utility of the processors decrease during the elimination. The area time complexity for a path (three diagonal matrix) is not as good for the minimal time algorithm as it is for the hexagonal solution [6]. However, for a general tree the minimum time algorithms require O.fewer processors and has a complexity measure that is orders of magnitude less than hexagonal arrays based on a bandwidth approach.

For a regular mesh the algorithms proposed in this report are linear or $O(m\log_2 m)$ in the smaller dimension and $O(\log_2 n)$ in the larger dimension. When n>>m the algorithms proposed here are significantly faster than the solutions by Kung and Leiserson [6]. Our algorithms require a larger number of processors but the area*time product is smaller.

The conclusion drawn from this comparison parallells that for the sequential machine case. The more zeroes there are within a band of a matrix the larger is the pay off both in terms of time and number of processors from exploiting the structure of the matrix. Further investigation of how to exploit this potential in algorithms and architectures with a communication structure suitable for VLSI implementation is necessary.

In general, an optimal elimination process in a concurrent computing environment, as in the sequential case, has to take into account possible concurrency in subsequent steps. Maximizing concurrency in each step does not in general minimize the number of steps to complete the elimination. Finding rules that applied to a given graph will determine the set of nodes to be eliminated concurrently in order to minimize the number of steps to perform Gaussian elimination without simulating the entire elimination process is not trivial if at all possible.

General heuristic algorithms that have been used successfully for sequential machine implementations of finite element problems are quotient tree and nested dissection algorithms. These algorithms do not in general provide optimal results neither for sequential machines nor for the computational model used here. Applying these algorithms to a few sample problems to find quotient graphs on which concurrent elimination can be applied rather straightforwardly gave encourageing results. The use of separators to partition and order the node set of a graph for concurrent elimination has also been proposed by Rudin [13].

In this investigation it has been assumed that the order of elimination can be determined entirely from a concurrency viewpoint. If numerical stability is of concern then pivoting may be necessary. Pivoting may increase the number of steps to perform Gaussian elimination but need not do so.

In addition to the concurrent elimination of nodes in a graph corresponding to a symmetric matrix as studied here, there is also the obvious possibility of concurrent computations on the vector level. The elements of a row can always be computed concurrently when proceeding from one step in the elimination sequence to the next. All the rows corresponding to subsequent nodes in the elimination sequence can also be updated concurrently. The vector level of concurrency is the only form of concurrency for full matrices that preserves correctness. Concurrency in vector operations is exploited by H T Kung[6], S Y Kung[7] and Browning[1].

9 References

1 Browning S. *The Tree Machine: A Highly Concurrent Computing Environment*. Ph.D. thesis, Technical Report 3760, Computer Science, Caltech, January 1980.

2 Duff I S, A survey of sparse matrix research, *Proceedings of IEEE*, vol. 65, no. 4, April 1977.

3 Eisenstat S C, Schultz M H and Sherman A H, Applications of an element model for Gaussian elimination, in *Sparse Matrix Computations*, Ed. Bunch J R and Rose D J, Academic Press, 1976, pp. 85 - 96.

4 George A J. Solution of linear systemss of equations: direct methods for finite element problems. In *Sparse Matrix Techniques*, Springer Verlag, Lecture Notes in Mathematics 572, 1977.

5 Harary F. *Graph Theory*. Addison-Wesley, 1972.

6 Kung H T and Leiserson C E. Algorithms for VLSI processor arrays. In *Introduction to VLSI systems*. Addison-Wesley 1980.

7 Kung S Y. VLSI matrix computation array processors. *The MIT Conference on Advanced Research in Integrated Circuits*. 1980.

8 Leuker G S and Booth K S, Linear algorithms to recognize interval graphs and test for the consecutive ones property, *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, 1975, pp. 255 - 265.

9 Lipton R J, Rose D J and Tarjan R E, Generalized nested dissection, *SIAM J Numer. Anal.*, Vol. 16, No. 2, April 1979, pp. 346 - 358.

10 Parter S. The use of linear graphs in Gaussian elimination. *SIAM Review*, vol. 3, no 2, 1961, pp. 119-130.

11 Rose D J. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph theory and computing*, Academic Press, 1971, pp. 183 - 217.

12 Rose D J and Whitten G F, A recursive analysis of dissection strategies, in *Spase Matrix Computations*, Ed. Bunch J R and Rose D J, Academic Press, 1976, pp. 59 - 83.

13 Rudin L. A graph theoretic study of concurrent algorithms for the solution of symmetric linear systems of equations. Computer Science, Caltech, 1980.

14 Tarjan R E. Complexity of combinatorial algorithms. *Report STAN-CS-77-609*, April 1977.

15 Tarjan R E. Graph theory and Gaussian elimination, in *Sparse Matrix Computations*, Ed. Bunch J R and Rose D J, Academic Press 1976, pp. 3 - 22.

16 Tinney W F and Walker J W. Direct solution of sparse network equations by optimally ordered triangular factorization. *Proceedings of IEEE*, vol. 55, no. 11, 1967, pp. 1801 - 1809.