This paper was presented at the Symposium on Sparse Matrix and Their Applications, in Knoxville, Tennessee, November 2-3, 1978.

Software for Sparse Gaussian Elimination with Limited Core Storage

S. C. Eisenstat, M. H. Schultz, and A. H. Sherman

Research Report #152

This research was supported in part by ONR Grant N00014-76-C-0277 and AFOSR Grant F49620-77-C-0037.

ABSTRACT

A variant of Gaussian elimination is presented for solving sparse symmetric systems of linear equations on computers with limited core storage, without the use of auxiliary storage such as disk or tape. The method is based on the somewhat unusual idea of recomputing rather than saving most nonzero entries in the reduced triangular system, thus trading an increase in work for a decrease in storage. For a nine-point problem with the nested dissection ordering on an n x n grid, fewer than $\frac{7}{2}n^2$ nonzeroes must be saved versus $\sim\frac{93}{12}n^2\log_2 n$ for sparse elimination, while the work required at most doubles. The use of auxiliary storage in sparse elimination is also discussed.

1. Introduction

Consider the system of linear equations

$$(S) \qquad A x = b$$

where the coefficient matrix A is a sparse N x N symmetric positive definite matrix such as arise in finite-difference and finite-element approximations to elliptic boundary-value problems. Direct methods for solving (S) are generally variations of (symmetric) Gaussian elimination: We use the k^{th} equation to eliminate the k^{th} variable from the remaining N-k equations for k = 1,...,N, and then solve the resulting triangular system (the reduced system) for x. Unfortunately, as the elimination proceeds, some coefficients that were zero in the original system of equations become non-zero (fill-in), increasing the work and storage required.

As an example, consider the following model problem which arises from the familiar nine-point finite-difference discretization of the Poisson equation on the unit square with homogeneous Dirichlet boundary conditions. Given a uniform n x n grid in the plane (see Figure 1), we associate a variable u_{ij} with each mesh-point (i,j) and form the system of linear equations

$$\begin{aligned} &8u_{ij} - u_{i-1,j-1} - u_{i-1,j} - u_{i-1,j+1} - u_{i,j-1} \\ &\quad - u_{i+1,j+1} - u_{i+1,j} - u_{i+1,j-1} - u_{i,j+1} = f_{ij} \qquad 1 \le i,j \le n \end{aligned}$$

where

$$u_{ij} = 0 \qquad i = 0,\ n+1 \quad \text{or} \quad j = 0,\ n+1$$

There are $N = n^2$ unknowns u_{ij} and, taking account of symmetry, $\sim 6n^2$ nonzeroes in A and b. But with the nested dissection ordering of the variables [7] (which is optimal to within a constant factor [8]), the number of nonzeroes in the reduced system is $\sim \frac{93}{12} n^2 \log_2 n$.

Our model problem illustrates the behavior one often encounters in using Gaussian elimination to solve large sparse systems: The storage required can easily exceed the core storage available for even moderately large N, even though the problem and solution (i.e., A, b, and x) can be represented in core. Thus, although we could store the nonzero coefficients, right-hand side, and solution for our model problem in $\sim 7n^2$ locations, the reduced system would require an additional $\sim \frac{93}{12} n^2 \log_2 n$ locations.

In this paper we discuss variants of sparse elimination which can solve (S) with minimal core storage. The methods are based on the following assumptions:

(1) The nonzero entries of A and b are inexpensive to generate on demand (e.g., they can be stored in core or computed with little effort).

(2) There is enough core storage for the solution vector x, plus a small amount of working storage, but not enough to store the entire reduced system.

We count only the working storage required to solve (S) in stating the storage requirements of such methods. All other storage (i.e., storage for A, b, and x) is associated with the linear system rather than the method of solution and is ignored.

The standard solution is not to store the entire reduced system in memory, but to use auxiliary storage (e.g., disk or tape) as well.[1] An alternate approach is not to store the entire reduced system at all (cf. [4]). Instead, we throw away most nonzero entries and recompute them as necessary during back-solution. The result is an algorithm which can solve our model problem with approximately twice as much work as sparse elimination, but which requires that fewer than $\frac{7}{2}n^2$ nonzero entries be stored, versus $\sim\frac{93}{12}n^2\log_2 n$ for the reduced system.

In Section 2, we review the nested dissection ordering [7] and the "Disaster Strikes" algorithm for solving the model problem [3]. We introduce an element model of elimination in Section 3 in order to generalize the algorithm to non-model problems, and show how to implement such a scheme in Section 4. In Section 5, we present some experimental results, and in Section 6 the applications of these ideas to auxiliary storage sparse elimination.

---

[1] Virtual memory systems appear to have large amounts of memory but in reality constitute a hidden use of auxiliary storage.

2. The Nested Dissection Ordering

The work and storage required to solve a large sparse system of linear equations clearly depend upon the zero-nonzero structure of the coefficient matrix. But since this matrix is positive definite and symmetric, we could equally well solve the permuted system

$$PAP^t\, y = P\, b, \quad P\, x = y$$

given any permutation matrix P [11]. The permuted system corresponds to reordering the variables and equations of the original system, and the net result can often be a significant reduction in the work and storage required to form the reduced system. For the case of the nine-point operator, George [7] has discovered a nearly optimal ordering known as the nested dissection ordering.

The easiest way to describe the nested dissection ordering is in reverse order; that is, we shall describe the last group of variables to be eliminated, then the second last, and so forth. The exact order in which variables in each group are eliminated does make some difference to the total work and storage required, but we shall ignore this difference. Although George described his ordering in terms of independent sets, we shall use the recursive description given by Rose and Whitten [10].

The basic step consists of numbering the 2n+1 variables on a central dividing cross (see Figure 2). These unknowns are the last to be eliminated. When we delete them, the grid splits into four $\sim\frac{n}{2} \times \sim\frac{n}{2}$

subgrids. Each of these subgrids is structurally similar to the original grid so that we can number it in the same fashion.

Theorem: (George [7]) For the nine-point model problem with the nested dissection ordering on an n x n grid, sparse elimination requires

$$O_{ND}(n) = 10n^3 + O(n^2 \log_2 n)$$

multiply-adds, and the reduced system has

$$S_{ND}(n) = \frac{93}{12} n^2 \log_2 n + O(n^2)$$

nonzero entries.

When the last variable has been eliminated, we have generated all the nonzero entries in the reduced system of equations that remains, and have merely to solve this system for the vector of unknowns x. Yet suppose disaster strikes and only those entries in the last 2n+1 rows are saved. Then we could still solve for the last 2n+1 variables, i.e., the values of the unknowns on the central dividing cross. But, given the values of these variables, our n x n problem splits into four smaller $\sim\frac{n}{2}$ x $\sim\frac{n}{2}$ problems of the same form. These subproblems can now be solved in the same fashion.

"Disaster Strikes" Algorithm [3]:

(1) Solve for the unknowns on a central dividing cross.

(2) The problem splits; solve each of the subproblems in the same

fashion.

Of course, throwing away nonzero entries means that we will have to do some additional work to recompute them. But how much more? The cost of solving an n x n problem is just $O_{ND}(n)$ plus the cost of solving four $\sim\frac{n}{2}$ x $\sim\frac{n}{2}$ problems. Letting $O_{MS}(n)$ denote this cost, we have

$$O_{MS}(n) = O_{ND}(n) + 4\, O_{MS}(\sim\tfrac{n}{2})$$

$$= 10n^3 + 4\, O_{MS}(\sim\tfrac{n}{2}) + O(n^2 \log_2 n) \quad .$$

Thus,

$$O_{MS}(n) = 20n^3 + O(n^2 \log_2 n) \quad ,$$

and we are doing approximately twice as much work. But, as we shall see, the saving in storage is more significant.

3. An Element Model of Elimination

How much storage is required to perform sparse elimination on our model problem with the nested dissection ordering if we only need to save the nonzero entries in the last 2n+1 rows of the reduced system? How do we generalize the "Disaster Strikes" algorithm to non-model problems? How do we implement such an algorithm? In this section, we introduce an element model of elimination which will help to resolve these questions.

The element model emulates Gaussian elimination by a sequence of transformations on a collection E of sets of variables called elements. Initially,

$$E^{(0)} = \{\{x_i, x_j\} \mid a_{ij} \neq 0,\ i \leq j\}.$$

Corresponding to using the k^{th} equation to eliminate the k^{th} variable from the remaining N-k equations, we transform $E^{(k-1)}$ to $E^{(k)}$ by

(1) merging all elements in $E^{(k-1)}$ which contain x_k to form a new element E_k;

(2) deleting those elements in $E^{(k-1)}$ which contain x_k;

(3) adding E_k.

As an example, consider Gaussian elimination as applied to our nine-point model problem with the nested dissection ordering on a 3 x 3 grid (see Figure 3; the initial elements corresponding to the nonzero entries of A are not shown). As each variable is eliminated, one new element is created and the elements merged to form this element are deleted. Since A is irreducible, the final element contains all the variables in the grid. Note that some of the elements are equal.

The order in which the variables are eliminated determines the elements which are created during the elimination process. We can describe the element merges that take place by an element merge tree:[2]

(1) The nodes of the tree are the elements which were created during

the elimination process (however, if two elements are equal, then they are identified with the same node).

(2) A node (i.e., element) E_i is a son of another node E_j if and only if $E_i \neq E_j$ and E_i was merged to form E_j when variable x_j was eliminated.

The merge tree for the previous example appears in Figure 4.

Note the following properties of element merge trees:

(1) Every variable v_k corresponds to some node in the element merge tree, namely the node identified with E_k; however, several variables will correspond to the same node in the tree if the corresponding elements are equal.

(2) If values were known for all the variables x_i whose corresponding E_i is equal to the root element of the tree, then the problem would split into subproblem(s), which could be solved in the same fashion.

This allows us to generalize the "Disaster Strikes" Algorithm to non-model problems.

Minimal Storage Sparse Elimination (MSSE) Algorithm:

(1) Construct the element merge tree corresponding to the given order

---

[2] The element merge tree is actually a forest unless the matrix A is irreducible.

of elimination.

(2) Solve for the variables corresponding to the root element in the tree.

(3) The problem splits; solve each of the subproblems in the same fashion.

Moreover, there is no reason why we cannot solve for more than just the variables corresponding to the root node at each stage. Instead, we could solve for as many variables as we have storage for the necessary nonzero entries in the reduced system. The result will be less work, although, since the bulk of the work is in the first one or two stages, the savings will be primarily in bookkeeping operations rather than in actual arithmetic. But how do we implement such a scheme?

4. An Implementation of MSSE

In this section, we present an alternate formulation of Gaussian elimination [6], and show how it leads to an implementation of the MSSE algorithm similar to the assembly approach for solving finite-element equations [9].

Gaussian elimination can be expressed as a sequence of operations on the coefficients and right-hand sides of the original system of equations leading to the reduced system:

SET $a_{ij}^{(0)} = a_{ij}, \quad b_i^{(0)} = b_i \qquad 1 \le i \le j \le N$

FOR $k = 1, 2, \ldots, N-1$ DO

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} - \dfrac{a_{ki}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} & k < i \le j \le N \\ a_{ij}^{(k-1)} & \text{otherwise} \end{cases}$$

$$b_i^{(k)} = \begin{cases} b_i^{(k-1)} - \dfrac{a_{ki}^{(k-1)} b_k^{(k-1)}}{a_{kk}^{(k-1)}} & k < i \le N \\ b_i^{(k-1)} & \text{otherwise} \end{cases}$$

Here $a_{ij}^{(k)}$ and $b_i^{(k)}$ are the entries of the partially reduced system after $x_1,\ldots,x_k$ have been eliminated. The importance of this formulation lies in the following result:

Theorem: If

$$a_{ij}^{(k)} \neq 0, \quad k < i \le j \le N,$$

then there exists an element E_m in $E^{(k)}$ such that

$$x_i, x_j \in E_m, \quad k < i \le j \le N.$$

If we exclude the possibility of exact cancellation, then the converse is also true.

This result suggests how to store the nonzero entries of the partially reduced systems. Associate with each element $E_m \in E^{(k)}$ a matrix $C_m = [c^{(k)}_{x_i,x_j}]$ and a vector $d_m = [d^{(k)}_{x_i}]$ whose rows and columns correspond to those $x_i \in E_m$ with $i > m$; and whose values represent corrections to the entries of A and b resulting from the elimination of those variables $x_i \in E_m$ with $i \leq m$. For the nine-point problem on a 3 x 3 grid, the C_1, d_1 and C_5, d_5 corresponding to E_1 and E_5 are shown in Figure 5.

To compute the corrections associated with a new element, we perform an assembly and elimination process similar to that used in finite-element solutions (see Figure 6):

(1) Set up a workspace with the rows and columns corresponding to those variables $x_i \in E_k$ with $i \geq k$ and an additional column corresponding to the right-hand side; the workspace is initialized to zeroes.

(2) The initial elements merged to form E_k are of the form $\{x_k, x_i\}$ where $a_{ki} \neq 0$; replace the (x_k, x_i) entry in the workspace by a_{ki} and the (x_k, b) entry by b_k.

(3) For those elements E_m merged to form E_k, add the corrections associated with E_m to the appropriate entries of the workspace.

(4) Eliminate x_k; the first row of the workspace is the corresponding row of the reduced system; the remaining rows are the C_k and d_k associated with E_k.

If several variables all correspond to the same node in the tree, then we eliminate them simultaneously.

Note that as long as all variables corresponding to descendants of a node are eliminated before variables corresponding to that node, then the "same" operations are done during the elimination process. Thus, instead of eliminating variables in a breadth-first or bottom-up order, we could equally well eliminate them in a depth-first order (see Figure 7). This cuts the amount of storage required and greatly simplifies storage management [2]. It is straight-forward but tedious to show that

Corollary: For the nine-point model problem with the nested dissection ordering on an n x n grid, at most $\frac{7}{2}n2$ nonzero entries have to be stored at any stage of the elimination.

5. Experimental Results

In this section, we present the results of some experiments with a code [2] which implements the Minimal Storage Sparse Elimination algorithm, and discuss the advantages and disadvantages of such a code versus a good sparse elimination code.

Tests were run on the nine-point model problem with the nested dissection ordering on a 31 x 31 grid. As a base for comparison, the same problem was solved using the symmetric codes from the Yale Sparse Matrix Package (YSMP) [5]. The times[3] are presented in Table 1.

In the table, Full Recursion refers to solving for only the variables corresponding to the root element at each stage; No Recursion refers to saving the entire reduced system in core, and solving for all the variables in the first stage; and Partial Recursion refers to saving as much of the reduced system as possible at each stage, and solving for the corresponding variables.

As the times show, the MSSE implementation is nearly competitive with YSMP when the amount of storage is sufficiently large, but less so when the amount of storage is reduced. The differences arise both from the additional bookkeeping required to implement sparse elimination in this manner and the additional work required to recompute discarded values.

The principal advantages of the YSMP code are speed and the ability to solve efficiently additional systems with the same coefficient matrix but different right-hand sides (since the entire reduced system is retained). The principal advantages of MSSE are the ability to solve problems in significantly less core, and to trade off an increase in execution time for a decrease in core.

---

[3] All experiments were run on a DEC KL-20 computer with the FORTRAN-20 optimizing compiler.

6. Applications to Auxiliary Storage Sparse Elimination

In this section, we examine how the ideas underlying MSSE can be applied to auxiliary storage sparse elimination.

Auxiliary storage devices are characterized by long access times and high transfer rates. Therefore it is important to transfer as many words as possible as infrequently as possible, consistent with carrying out the elimination process. Unfortunately, neither of the standard approaches to sparse elimination -- the row-by-row approach [5] and the outer-product approach [1] -- allows auxiliary storage to be integrated in this manner. Both would be characterized by excessive input/output to random-access storage. Thus input/output could easily dominate computation in the total run-time. However, by using the fundamental ideas of MSSE, we can avoid this problem.

Assume that there is enough storage to perform the MSSE algorithm. Recall that MSSE must generate the entire reduced system while solving for the set of variables corresponding to the root element. Clearly it suffices to output these values to auxiliary storage instead of throwing them away, and then read them back in during the back-solution process. Since each value is read and written once, the minimal amount of input/output is performed and, by appropriate buffering, this can be done efficiently.

But what if the amount of storage is insufficient to carry out MSSE? It seems likely that interposing a paging scheme to simulate a larger amount of available core would be a promising approach.

REFERENCES

1. I. S. Duff, N. Munksgaard, H. B. Nielsen, and J. K. Reid. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. Harwell Report CSS-44, January 1977.

2. S. C. Eisenstat. A minimal storage sparse elimination code. To appear.

3. S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Applications of an element model for Gaussian elimination. In J. R. Bunch and D. J. Rose editors, Proceedings of the Symposium on Sparse Matrix Computations, Argonne National Laboratory, September 1975, pp 85-96.

4. S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Minimal storage band elimination. In A. H. Sameh and D. Kuck editors, Proceedings of the Symposium on High Speed Computer and Algorithm Organization, University of Illinois at Champagne-Urbana, April 1977, pp 273-286.

5. S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale Sparse Matrix Package: I. The Symmetric Codes. Yale Computer Science Research Report #112, July 1977.

6. G. E. Forsythe and C. B. Moler. Computer Solution of Linear Algebraic Systems. Prentice-Hall, 1967.

7. J. A. George. Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis 10:345-363, 1973.

8. A. J. Hoffman, M. S. Martin, and D. J. Rose. Complexity bounds for regular finite difference and finite element grids. SIAM Journal on Numerical Analysis 10:364-369, 1973.

9. B. M. Irons. A frontal solution program for finite elements. International Journal for Numerical Methods in Engineering 2:5-32, 1970.

10. D. J. Rose and G. F. Whitten. Automatic nested dissection. Proceedings of the 1974 ACM National Conference, pp 82-88.

11. J. H. Wilkinson. The Algebraic Eigenvalue Problem. Clarendon Press, 1965.

Figure 1:

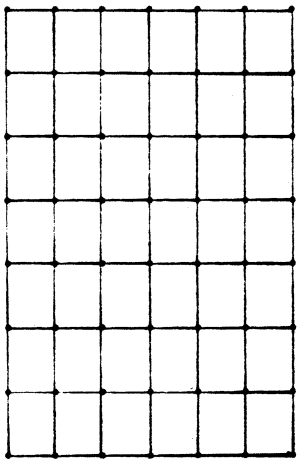
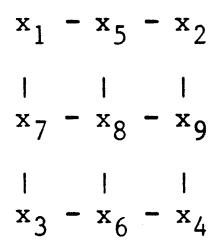


Figure 2:

```
o o o x o o o                          o x o     o x o
o o o x o o o                          x x x     x x x
o o o x o o o                          o x o     o x o
x x x x x x x          ----->
o o o x o o o                          o x o     o x o
o o o x o o o                          x x x     x x x
o o o x o o o                          o x o     o x o
```

Figure 3:



$E^{(0)} = \{ \ldots \}$

$E^{(1)} = \{ E_1=\{x_1,x_5,x_7,x_8\}, \ldots \}$

$E^{(2)} = \{ E_1, E_2=\{x_2,x_5,x_8,x_9\}, \ldots \}$

$E^{(3)} = \{ E_1, E_2, E_3=\{x_3,x_6,x_7,x_8\}, \ldots \}$

$E^{(4)} = \{ E_1, E_2, E_3, E_4=\{x_4,x_6,x_8,x_9\}, \ldots \}$

$E^{(5)} = \{ E_3, E_4, E_5=\{x_1,x_2,x_5,x_7,x_8,x_9\}, \ldots \}$

$E^{(6)} = \{ E_5, E_6=\{x_3,x_4,x_6,x_7,x_8,x_9\}, \ldots \}$

$E^{(7)} = \{ E_7=\{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9\}, \ldots \}$

$E^{(8)} = \{ E_8=E_7, \ldots \}$

$E^{(9)} = \{ E_9=E_7 \}$

Figure 4:

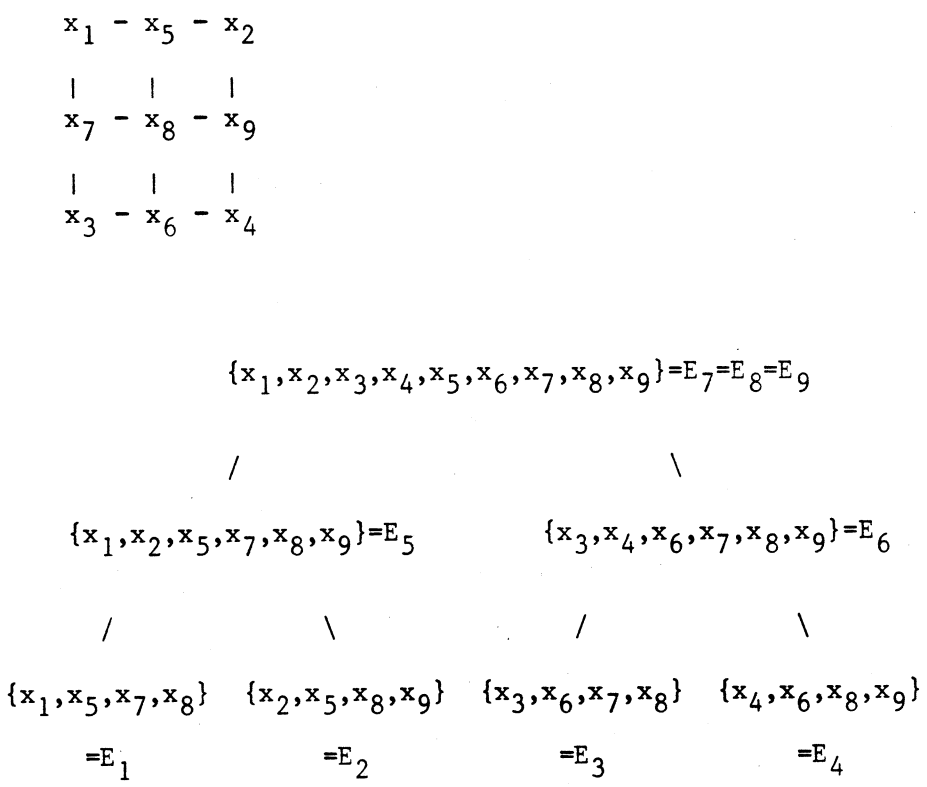

$x_1 - x_5 - x_2$
$x_7 - x_8 - x_9$
$x_3 - x_6 - x_4$
$\{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9\}=E_7=E_8=E_9$
$\{x_1,x_2,x_5,x_7,x_8,x_9\}=E_5$
$\{x_3,x_4,x_6,x_7,x_8,x_9\}=E_6$
$\{x_1,x_5,x_7,x_8\}$ $=E_1$
$\{x_2,x_5,x_8,x_9\}$ $=E_2$
$\{x_3,x_6,x_7,x_8\}$ $=E_3$
$\{x_4,x_6,x_8,x_9\}$ $=E_4$

Figure 5:

$$
\begin{array}{ccccc}
x_1 & - & x_5 & - & x_2 \\
| & & | & & | \\
x_7 & - & x_8 & - & x_9 \\
| & & | & & | \\
x_3 & - & x_6 & - & x_4
\end{array}
$$

$$E_1 = \{x_1, x_5, x_7, x_8\}$$

$$c^{(1)}_{x_i,x_j} = - \frac{a^{(0)}_{1i}\, a^{(0)}_{1j}}{a^{(0)}_{11}} \qquad i \leq j, \quad i,j \in \{5,7,8\}$$

$$d^{(1)}_{x_i} = - \frac{a^{(0)}_{1i}\, b^{(0)}_{1}}{a^{(0)}_{11}} \qquad i \in \{5,7,8\}$$

$$E_5 = \{x_1, x_2, x_5, x_7, x_8, x_9\}$$

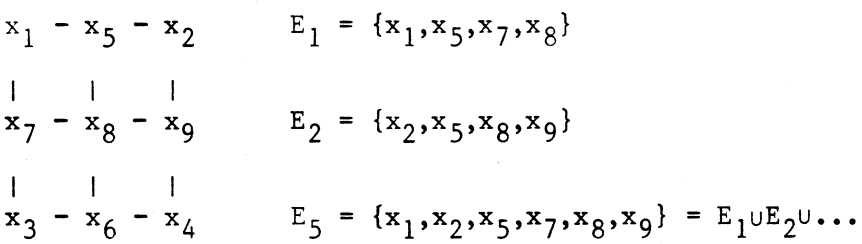
$$c^{(5)}_{x_i,x_j} = - \frac{a^{(0)}_{1i}\, a^{(0)}_{1j}}{a^{(0)}_{11}} - \frac{a^{(1)}_{2i}\, a^{(1)}_{2j}}{a^{(1)}_{22}} - \frac{a^{(4)}_{5i}\, a^{(4)}_{5j}}{a^{(4)}_{55}}$$

$$i \leq j, \quad i,j \in \{7,8,9\}$$

$$d^{(5)}_{x_i} = - \frac{a^{(0)}_{1i}\, b^{(0)}_{1}}{a^{(0)}_{11}} - \frac{a^{(1)}_{2i}\, b^{(1)}_{2}}{a^{(1)}_{22}} - \frac{a^{(4)}_{5i}\, b^{(4)}_{5}}{a^{(4)}_{55}}$$

$$i \in \{7,8,9\}$$

Figure 6:



WORKSPACE

	x_5	x_7	x_8	x_9	b
x_5	A+1+2	A+1	A+1+2	A+2	b+1+2
x_7		1	1	0	1
x_8			1+2	2	1+2
x_9				2	2
			C_k		d_k

A and b denote contributions from an initial element

1 denotes a contribution from E_1

2 denotes a contribution from E_2

Figure 7:

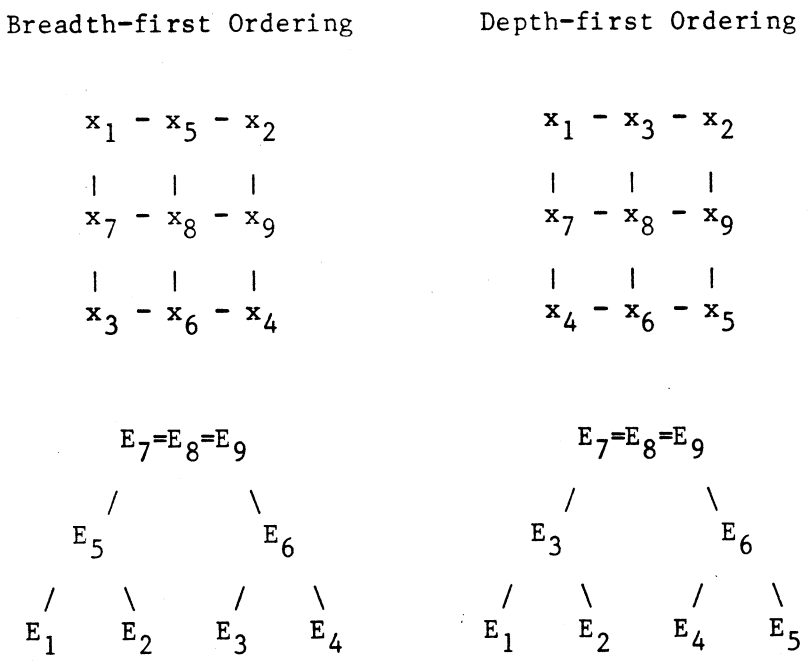

Breadth-first Ordering
Depth-first Ordering
x1 - x5 - x2
| | |
x7 - x8 - x9
| | |
x3 - x6 - x4
x1 - x3 - x2
| | |
x7 - x8 - x9
| | |
x4 - x6 - x5
E7=E8=E9
E5 E6
E1 E2 E3 E4
E7=E8=E9
E3 E6
E1 E2 E4 E5

Table 1: Nine-point operator on a 31 x 31 grid

	Time (seconds)		Additional Storage
YSMP			
Preprocessing (symbolic factorization)	0.309		
Numerical factorization and solution	3.291		
Total		3.600	29198
MSSE (Full Recursion)			
Preprocessing (element merge tree)	0.547		
Numerical factorization and solution	10.624		
Total		11.171	7426
MSSE (No Recursion)			
Numerical factorization and solution	4.178		
Total		4.725	32169
MSSE (Partial Recursion)			
Numerical factorization and solution	6.814		
Total		7.361	8124