

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3

ADA 125738

*Randomly Sparse Equation Solution
by Loopless Code Generation
on the CRAY-1*

D.A. CALAHAN

May 1, 1982

DTIC
ELECTE

Sponsored by Directorate of Mathematical and Information
Sciences, Air Force Office of Scientific Research,
under Grant No. ~~80-0158~~

AFOSR-80-0158



Systems Engineering Laboratory

Approved for public release;
distribution unlimited.

DTIC FILE COPY

83 03 14 012

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0077	2. GOVT ACCESSION NO. <i>AD-A125 738</i>	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) RANDOMLY SPARSE EQUATION SOLUTION BY LOOPLESS CODE GENERATION ON THE CRAY-1		5. TYPE OF REPORT & PERIOD COVERED Interim
7. AUTHOR(s) D. A. Calahan		6. PERFORMING ORG. REPORT NUMBER SEL #162
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Michigan Dept. of Elec. & Computer Engring. Ann Arbor, MI, 48109		8. CONTRACT OR GRANT NUMBER(s) AFOSR 80-0158
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research (NM) Bolling AFB, Washington DC 20332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F <i>2304/A3</i>
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE May 1, 1982
		13. NUMBER OF PAGES 27
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Sparse matrices Parallel processing Vector processing Linear algebra		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) To solve directly a sparse unsymmetrix equation $Ax = b$, an equation-ordering algorithm based on local equation decoupling is proposed to maintain a high flow rate of scalar computations within a floating point pipeline. Software is described to solve highly-sparse unpatterned systems efficiently via explicit code generation. Rates in the range of 15 MFLOPS on the CRAY-1 are achieved.		

**Randomly-Sparse Equation Solution
by Loopless Code Generation
on the CRAY-1**

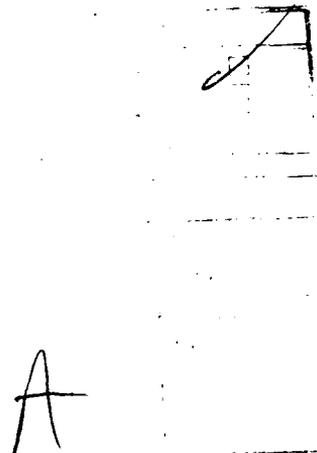
D. A. Calahan

**Systems Engineering Laboratory
University of Michigan
Ann Arbor, Michigan 48109
August 1, 1982**

SEL Report #162



**Sponsored by the Directorate of Mathematical
and Information Sciences, Air Force
Office of Scientific Research, under
Grant 80-0158**



AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFOSR)
NOTICE: This report is the property of the Air Force Office of Scientific Research and is loaned to your organization; it and its contents are not to be distributed outside your organization.
approv
Distrib
MATTHEW J. ...
Chief, Technical Information Division

Abstract

To solve directly a sparse, unsymmetric matrix equation $Ax = b$, an equation-ordering algorithm based on local equation decoupling is proposed to maintain a high flow rate of scalar computations within a floating point pipeline. Software is described to solve highly-sparse unpatterned systems efficiently via explicit code generation. Rates in the range of 15 MFLOPS on the CRAY-1 are achieved.

I. Introduction

Vector processors have forced a reconsideration of traditional computational algorithms. In the solution of sparse systems of equations, such study has resulted in a proliferation of methods to service a variety of sparsity characteristics.

Early work in general vectorized sparse methods [1], yielding codes that appeared to the user similar to "traditional" scalar counterparts [2], had limited intelligence to identify vector operations. For important classes of both highly-sparse and relatively dense systems, special codes were later found to achieve speedups of 3:1 to 20:1 over the general vector code. From an algorithmic viewpoint, it appears that the notion of a general sparsity code for vector architectures may be an anachronism (however, see [17]).

An exception occurs where such speedups can be tolerated in the interest of user convenience; for example, in a small highly-sparse system, the equation solution time may be a small fraction of the equation-formulation time and such inefficiencies may be acceptable.

In general, such speedups are achieved either by

- (a) locally decoupling of equations so that the pipelines can be "crammed" with independent computations associated with uncoupled equations; such methods are useful in highly-sparse systems;
- (b) local coupling of equations so that vectors can be defined within dense banded or locally blocked sparse systems.

Figure 1 illustrates that each of these approaches can be further classified. In the case of dense systems, usually associated with elliptic finite element and finite difference problems, coupling is exploited either (a) within a grid point (node) - with many unknowns/node [3] - or within a finite element [4] or (b) across grid points, yielding banded and profile systems

[5]. These are termed **intranodal** (intra-element), and **internodal** (inter-element) coupling, respectively. In general, internodal coupling yields denser submatrices, longer vectors, and so higher execution rates.

When highly-sparse equations are decoupled, a distinction must be made between **patterned** and **unpatterned** systems.

In the former, it is presumed that (a) submatrices with identical sparsity patterns can be identified, and (b) these submatrices are stored with similarly-positioned elements a constant stride apart. (This latter restriction is more important for highly sparse systems, where one cannot afford to remap the matrix by gather/scatter operations; however, the existence of patterns usually implies that subsystem matrices can be simultaneously formulated in vector mode so that (b) is often satisfied.) These conditions apply, for example, to large electronic circuit matrices [7][8] and assure vectorizability.

The above vectorizable dense and patterned sparse matrix cases account for the majority of sparse problems. Indeed, sparse matrices become large usually by means of a formulation algorithm that guarantees vectorizability.

However, there do exist relatively small (≤ 5000 equations) highly-sparse problems with undiscernable patterns: some electronic circuits, electrical power systems, small dissected 2-D finite element grids [6], occurring perhaps as a part of a 3D iterative solution. In more exotic formulations, an unpatterned matrix may represent only part of a large sparse system [7]. In any case, such structures pose a most difficult algorithmic challenge, apart from their arguable utility. It is to these problems that the report is addressed. The results of the report were first given in [14] and [15].

One additional caveat is offered before proceeding. It will be necessary to preprocess the sparsity structure before the numerical solution is carried out. On the CRAY-1, this preprocessing time is several hundred times the matrix solution time. Therefore, this procedure is appropriate only when multiple numerical solutions are required with the same sparsity structure.

II. Algorithms

A. Parallel Solution

Consider an unsymmetric matrix equation of the form

$$Ax = b$$

where A is an $n \times n$ matrix and x and b are $n \times 1$ vectors. This equation is to be solved by LU factorization, viz,

1. Factor $A = LU$, where L and U are lower and upper triangular matrices, respectively.
2. Solve $Ly = b$ for y (forward substitution).
3. Solve $Ux = y$ for x (backward substitution).

The matrix A is considered locally decoupled if the combined structure of its LU factors has the form [11].

$$\begin{array}{cccc}
 D_{11} & & & U_{12} \\
 & D_{22} & & U_{23} \dots \\
 L_{21} & & D_{33} \dots & \\
 & L_{32} \dots & &
 \end{array}$$

where D_{rr} is a diagonal matrix and $L_{r+1,r}$ and $U_{r,r+1}$ extend from D_{rr} to the matrix boundary. The ordered steps to reduce the r th pivot block and the associated right hand side components are

$$D_{rr} \leftarrow D_{rr}^{-1} \quad (\text{reciprocation}) \quad (1)$$

$$L_{r+1,r} \leftarrow L_{r+1,r} D_{rr}^{-1} \quad (\text{multiplication}) \quad (2)$$

$$A_{r+1,r+1} \leftarrow A_{r+1,r+1} - L_{r+1,r} U_{r,r+1} \quad (\text{mult./subtraction}) \quad (3)$$

$$Y_{r+1} \leftarrow Y_{r+1} - L_{r+1,r} Y_r \quad (\text{mult./subtraction}) \quad (4)$$

where $A_{r+1,r+1}$ represents the unreduced southeast corner of the matrix and Y_{r+1} is the associated right hand side at the r th reduction step. The block back substitution has the form

$$Y_r \leftarrow Y_r - U_{r,r+1} Y_{r+1} \quad (5)$$

$$X_r \leftarrow Y_r D_{rr} \quad (6)$$

where x_r is the r th block component of the solution vector. Equations (1)-(4) can be performed in three parallel steps. That is, except for the subtraction, all right hand side matrix elements - operands of unary and binary floating point computations - are known on entry to the step. (The subtractions can be processed efficiently at the coding level but can not always be performed in parallel.) Indeed, the sparser the equations, the greater the decoupling (dimension of D_{rr}) and the more parallel the solution.

B. Pipelined Solution

Calculation of vector or scalar results in a pipeline requires that operations in the pipeline at any time be independent. Without this independence, results must be secured in registers or, worse, main memory, before they can be operands for a succeeding computation. Independence ideally permits pipelines to be crammed with vector or scalar operands. Thus, parallel and pipelined architectures make a similar demand on the organization of an efficient solution algorithm; equations (1)-(6) are, therefore, also the basis of the proposed pipelined solution.

C. Code generation

If the elements of the D_{rr} are stored a fixed address increment apart, then conceivably floating point operations could be performed in vector mode. However, assuming column-ordered matrix storage for compatibility with existing programs, the cost of gathering the diagonals into this vector storage format will likely not be worth the advantage of vectorization. Similar arguments apply to the other two highly sparse parallel steps. For the CRAY-1, with slow gather/scatter operations, it is assumed that floating point operations should instead be performed in scalar mode.

To achieve the highest speed scalar operation, it was decided to generate explicit loopless scalar code in the manner of Gustavson [9]. This avoids the issuing of address operations - costly on the CRAY-1 - since the addresses are imbedded in the scalar code. Thus, when a series of consecutive reciprocations, (or multiplications, or subtractions) is to be performed, the code generator produces, in a preprocessing step, a sequence of similar scalar operations with different addresses. Because the instructions are identical except for addresses, the associated scalar

fetches, floating point operations, and stores can be overlapped in a predictable manner.

Table 1 gives a summary of the asymptotic rates of each of these generated code sequences, with and without overlapping. Since the multiplies/subtracts dominate the other computation in any but the sparsest matrix, the execution rate for a large matrix should approach that of the multiply/subtract kernel (14.8 MFLOPS). The detailed kernel overlapped timings from a CRAY-1 simulator [10] are given in Table 2.

D. Ordering

The restricted utility of this class of sparse matrix algorithms to small highly sparse systems suggests that available ordering techniques and software be modified, rather than new software be developed. For this reason, the following procedure represents a variation on the so-called minimum-degree algorithm, but applied to unsymmetric matrices. It is accepted a priori that specialized ordering software may execute more efficiently.

First, the conventional MD minimizing algorithm is reviewed. At the k th step in the ordering, let $pr(m)$ and $pc(m)$, $m = 1, \dots, n$, represent the row- and column-ordering permutation vectors, with $pr(m) = pc(m) = m$ for $k = 1$. Also, let $n_{pr(i)}$ and $n_{pc(j)}$ be the number of $k+1, \dots, n$ and $j = k, k+1, \dots, n$, respectively. There, among the non-zero elements $e_{pr(i), pc(j)}$, the pivot positions i_k and j_k are chosen such that

$$\{i_k, j_k\} = \{i, j: \min(n_{pr(i)} - 1, n_{pc(j)} - 1); e_{pr(i), pc(j)} \neq 0; k \leq i \leq n, k \leq j \leq n\}$$

The modified algorithm insures the local decoupling of equations and variables, as follows. When pivot positions i_k and j_k are selected within

block r , the rows and columns coupled to the block's pivot positions are marked; pivoting is then prohibited on non-zeros in these rows and columns. When no non-zero, unmarked pivots exist, a new block is initiated by incrementing r and clearing markers.

E. Limited Decoupling

The memory hierarchy of the CRAY-1 suggests that the full decoupling allowed by this ordering should not be exploited. It is preferred that the results from the first two steps of (2) and (3) be maintained in 64 scalar (T) registers. This necessitates that the total number of elements in D_{rr} and $L_{r+1,r}$ be no greater than 64, since all elements of these two matrices are required in (3) and (4). By correspondingly limiting the dimension of D_{rr} in the ordering algorithm, the minimal degree criterion is, on the average, less constrained during a pivot selection than if maximum decoupling were demanded within each block. In the limit, if the dimension of D_{rr} is constrained to be unity, a true MD criterion results and the MD operation count should be achieved.

Viewed another way, since the scalar register file size is limited at 64, a family of matrices increasing in size should be less impacted by the limited-decoupling strategy as the size increases. Thus a matrix of large dimension should achieve a nearly minimal (MD) operation count.

A flow chart of the limited decoupling algorithm is shown in Figure 3.

III. Software

A. Introduction

The program has two parts:

- (1) A Fortran symbolic preprocessor that (a) orders the equations according to the limited decoupling algorithms, (b) generates CRAY-1 machine code in a buffer array, and (c) writes this code into a file in unformatted form.
- (2) A program that (a) reads the code into main memory, (b) formulates a set of equations of the prescribed sparsity from random-valued numerical data, and (c) calls (from Fortran) a short interface program that jumps to the code.

The flow chart is given in Figure 2. Note that the same code suffices to solve multiple numeric solutions.

B. Inputs To Symbolic Phase

The symbolic phase reads the following data.

1. N - the number of equations.
2. NRMAP = 0 if numeric values stored in column order; NRMAP = 1 if order of numeric values is given in NUMN
3. JA - an array of dimension N+1; JA(J) points to the first element of the Jth column in array IA; JA(N+1) points to one beyond the last element of IA.
4. IA - an array of dimension NA = JA(N+1)-1, containing the column-ordered row indices.
5. NUMN - an array, usually of dimension NA; NUMN(J) gives the location in data array A of the element corresponding to IA(J).

Although written as a self-contained research program (e.g., facilities are provided to generate randomly-positioned matrices, to count operations, and produce a printer map), it is relatively clear which facilities should be deleted for production use. Also, the above data could be transferred in an argument list.

If NRMAP = 0, it is assumed that NUMN(J) = J, and NUM(J) is not referenced further; the dimension of NUM need then be only unity.

C. Numeric Solution Phase

A Fortran test driver (Table 3) was developed to formulate a randomly-valued matrix of the sparsity prescribed by JA and IA. Moreover, the values are mapped according to NUMN and, to insure numerical dominance of the pivot positions, pivots are located from an array passed from the symbolic program. The right hand side is formulated so that the solution vector X has the value $X(J) = J + 1$.

The linkage in the code that performs the LU factorization, the forward substitution, and the back substitution is made by the subroutine invocation

```
CALL EXEC(INST, A, B, X, N)
```

where

INST is an array containing the machine code

A is the matrix numeric values, packed according to NUMN and IA

B is the right hand side

X is the solution

N is the number of equations.

All reciprocations are half precision.

D. Ordering Subroutine

The equation-ordering program, being a critical element of this software package, deserves separate documentation. A list of its calling arguments follows.

```
CALL SPCPIV(N, NA, IA, JA, IROW, JCOL,
           JROW, ICOL, INUM, JNUM, IMIN,
           JMIN, IPR, IPC, ISIZE, IBLC,
           IBLR, IBLOCK, IMINT, JMINT,
           IMAP, ICALC, NMAP, NBL, IPIV,
           ICMAX, IDP).
```

where

N*	is the number of equations
NA*	is the number of non-zero elements
NMAP	is the number of elements of MAP
NBL	is the expected number of diagonal blocks; .LE.N
ICMAX*	must be set to 64 by user
ISIZE*	is the maximum expected number of non-zeros of L and U (combined)
IPIV*	=0 if limited decoupling is desired =1 if no decoupling is desired; MD ordering criterion is then used
IDP*	=0 for unspecified tie-breaking in MD ordering =1 for diagonal preference in tie-breaking
IA(J)*	contains column-ordered row number of non-zero positions of matrix; dimension is NA
IBLOCK(J)	is the row number (=column number) of first element of Jth diagonal block; dimension at least NBL+1; IBLOCK(N+1) = NBL+1

(The following arrays must have a dimension of at least N or N+1)

JA(J)* is the location in JA of the first non-zero element
in the Jth column; JA(N+1) = NA+1

IPR(J) is the Jth pivot row number

IPC(J) is the Jth pivot column number

ICALC(J) is the number of floating point operations to factor
the matrix through the Jth column

JNUM, INUM, IMINT, JMINT, IBLR, IBLC, IMIN, JMIN, JROW, ICOL

are working arrays

The following arrays must have a dimension equal to the number of expected
non-zeros of L and U combined, plus N, the number of equations.

IROW and JCOL are working arrays

IMAP contains information related to the map of L and U in
alternating row- and column-order; diagonal elements are
represented twice, requiring N additional locations.

IV. Performance

A. Choice of examples

Because the dimension of the matrix is limited by the size of code
stored in main memory, the number of applications of this procedure is lim-
ited. On the other hand, within this class of highly-sparse systems, the code
length and other performance aspects appear to be relatively sensitive to
sparsity features from different applications. Therefore, illustrative
problems have been chosen from a number of applications, namely,

1. Electronic circuit analysis,

*Input data to subroutine.

2. General elliptic PDE solution (by nested dissection),
3. Oil reservoir analysis,
4. Electrical power systems analysis.

The first class of problems is unsymmetrical in structure; the latter three classes are symmetrical in structure, but are assumed, for purposes of this study, unsymmetrical in value. In all cases, off-diagonal pivoting is allowed.

B. Effect of ordering

It is well-known that the operation counts associated with the orderings of highly-sparse matrices are sensitive to the tie-breaking procedure. The current ordering algorithm is not necessarily optimized in this respect (see [12] and [13]). However, two options have been incorporated in the program:

- (a) choosing the "first-found" tied pivot, and
- (b) preference for diagonal pivots.

In general, it has been found desirable for symmetrically-structured matrices to favor diagonal pivots. Unsymmetrically-structured matrices yield mixed results.

Floating-point operation counts for a number of problems are given in Table 3, with MD ordering and with the limited decoupling algorithm. The penalty incurred for decoupling is moderate and decreases on a fractional basis as the matrix size increases (as previously predicted). These results are not surprising, since in many model finite difference problems [6], minimal operation counts are associated with the decoupling proposed here.

C. Code length and performance

Table 4 gives the timing and storage results of a number of problems. The "effective" MFLOPS are the actual MFLOPS multiplied by the degradation due to the extra floating point computations necessitated by forced decoupling, *visa vis* MD ordering (see Table 3).

The following should be noted.

(1) The code length is approximately equal to the number of floating point operations, in 64-bit words. This allows one to estimate the feasibility of code generation for a problem with a known complexity. For example, from Table 4, a million-word memory would seem to be adequate to store code for the largest real-valued electrical power system problem and, perhaps, a 1000-equation complex-valued system. Electronic circuits in the range of 5000 equations should be readily handled. Five-point 2-D square finite difference grids solved by nested dissection [6] have by a known solution complexity of $\approx 20n^3$; these can be solved for $n \leq 36$.

(2) The code generation time, exclusive of writing the code to a file, is approximately 18 μ sec per floating-point operation, or 200-400 times the equation solution time. Together with the storage results above, approximately 18 seconds suffice to generate a million words of code. In general, the code generation time is less than the equation-ordering time for highly-sparse problems; denser matrices, such as those associated with D-4 ordered reservoir grids, have the opposite relation.

(3) The execution rates (MFLOPS) is relatively insensitive to variations in the matrix size and density. For example, the highly-sparse power system and electronic circuit matrices yield rates in

the range of 11.6-15 MFLOPS, whereas the denser grid-related matrices can be solved at 16-18 MFLOPS. (Of course, model grid problems can be solved at for higher rates by band-related methods [16]). This insensitivity is due to the independent (parallel) element-level operations that are associated both with dense matrices and with decoupled sparse matrices.

It is reasonable to conclude that 11 MFLOPS represents a lower bound of the solution rate of any sparse matrix requiring fewer than one million floating point operations.

References

- [1] Calahan, D. A., P. G. Buning, and W. N. Joy, "Vectorized General Sparsity Algorithms with Backing Store," Report #96, Systems Engineering Laboratory, University of Michigan, January, 1977.
- [2] Gustavson, F. G., "Some Basic Techniques for Solving Sparse Systems of Linear Equations," in *Sparse Matrices and Their Applications*, Ed. by D. J. Rose, and R. A. Willoughby, Plenum Press, 1972.
- [3] Calahan, D. A., "A Block-Oriented Sparse Equation Solver for the CRAY-1," Report #136, Systems Engineering Laboratory, University of Michigan, December, 1980.
- [4] Duff, I. S., and J. K. Reid, "Experience of Sparse Matrix Codes on the CRAY-1," Report CSS 116, AERE Harwell, October, 1981.
- [5] Calahan, D. A., "High Performance Banded and Profile Equation-Solvers for the CRAY-1: The Unsymmetric Case," Report #160, Systems Engineering Laboratory, University of Michigan, February, 1982.
- [6] George, J. A., "Nested Dissection of a Regular Finite Element Mesh," *SIAM J. Num. Anal.*, vol. 10, 1973, pp. 345-363.
- [7] Calahan, D. A., "Multi-level Vectorized Sparse Solution of LSI Circuits," *Proc. IEEE Conf. on Circuits and Computers*, Rye, N.Y., October, 1980, pp. 976-979.
- [8] Vladimirescu, A. and Pederson, D. O., "A Computer Program for the Simulation of Large-Scale-Integrated Circuits," *Proc. International Conf. on Circuits and Systems*, Chicago, April, 1981.
- [9] Gustavson, F. G., W. M. Liniger, and R. A. Willoughby, "Symbolic Generation of an Optimal Crout Algorithm in Sparse Systems of Linear Equations," *J. ACM*, vol. 17, pp. 87-109.

- [10] Orbits, D. A., "A CRAY-1 Simulator," report #118, Systems Engineering Laboratory, University of Michigan, Ann Arbor, 671 Engineering Laboratory, University of Michigan, Ann Arbor, September, 1978.
- [11] Calahan, D. A., "Parallel Solution of Sparse Simultaneous Equations," Proc. 11th Allerton Conf. on Circuits and Systems Theory, University of Illinois., 1973, pp. 729-738.
- [12] Wing, O., and J. W. Huang, "A Computational Model of Parallel Solution of Linear Equations," IEEE Trans., vol. C29, no. 12, pp. 632-638.
- [13] Lipton, R. J., D. J. Rose, and R. E. Tarjan, "Generalized Nested Dissection," SIAM J. Numer. Anal., vol. 16, 1979, pp. 346-358.
- [14] Calahan, D. A., "Direct Solution of Linear Equations on the CRAY-1," Cray Channels, pub. by Cray Research, Inc., vol. 3, No. 1, pp. 2-5.
- [15] Calahan, D. A., "Decoupled Solution of Circuit Matrices in Pipelined Processors," ICC 82 Conf., N. Y., October, 1982.
- [16] Calahan, D. A., "High Performance Banded and Profile Equation Solvers for the CRAY-1: The Unsymmetric Case," Report #160, Systems Engineering Laboratory, University of Michigan, February, 1982.
- [17] Calahan, D. A., "A Vectorized General Sparsity Solver," Report #168, Systems Engineering Laboratory, University of Michigan, October, 1982.

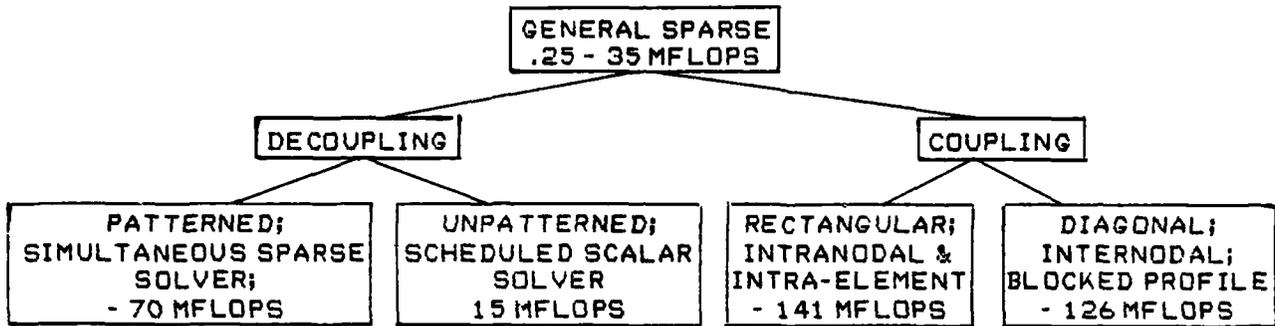
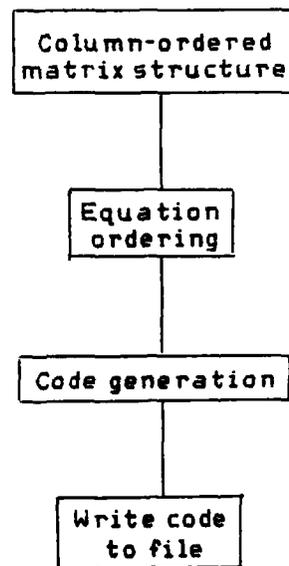
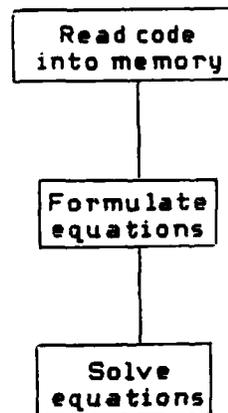


Figure 1. Classification of sparse matrix vectorized algorithms and CRAY-1 available software

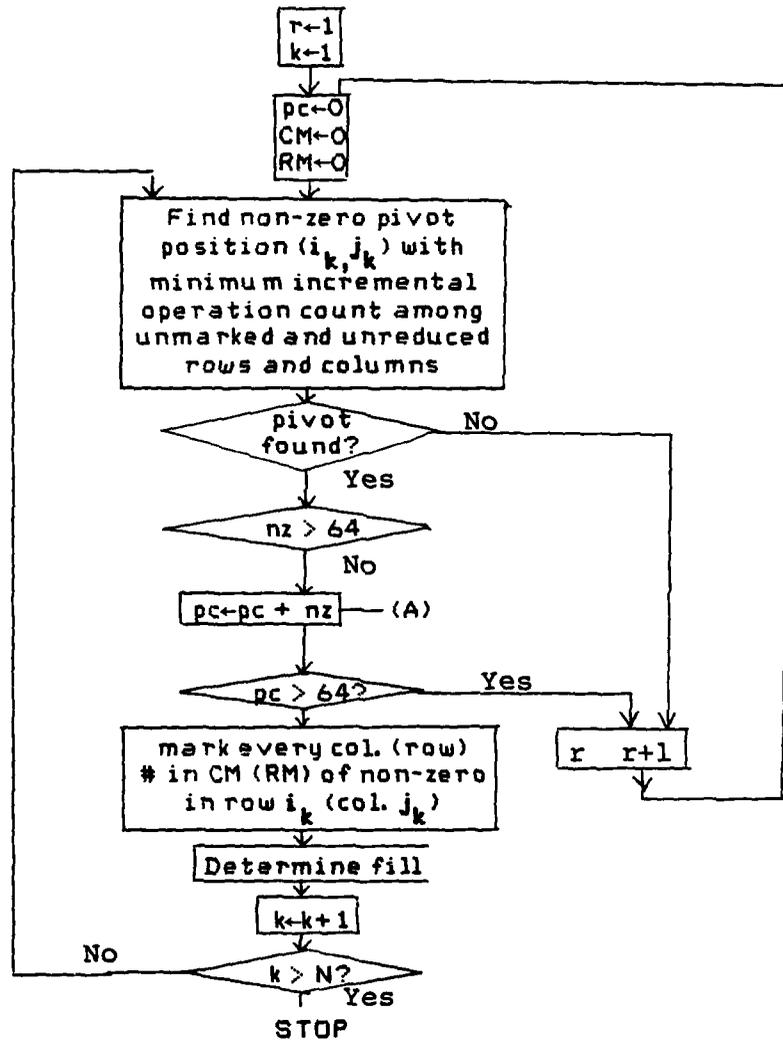


(a) Symbolic preprocessor phase



(b) Numeric solution phase

Figure 2. General flow chart of equation solver



- pc - counter of total non-zero positions in D_{rr} and $L_{r+1,r}$
 RM(CM) - row (column) marker array of dimension N ; only $N-k+1$ representing unreduced rows (columns) are zeroed and tested.
 r - block counter
 k - pivot counter
 nz - number of non-zero positions in $L_{r+1,r}$ and D_{rr} ($= 1$) added during k th pivot step.

Figure 3. Flow chart of limited-decoupling ordering algorithm;
 (A) represents coding to process case when more than 64
 non-zeros are in a column

Kernel	Inner loop timingx (Clocks)		Execution rate (MFLOPS)	
	Non-overlapped+	Overlapped	Non-overlapped+	Overlapped
Reciprocation*	27.2	7.50	2.94	10.7
Multiplication	23.8	7.25	3.36	11.0
Multi./Subt.	29.5	10.8	5.42	14.8

xTiming includes instruction fetching.

*Half-precision.

+Result store overlapped with next operand fetch.

Table 1. CRAY-1 kernel performance

Table 2(a). Reciprocation kernel activity report

ST. A	T	INSTRUCTION	P-ADDR	CP	S R R R	MEMORY BANKS	A	A. REG	S	S. REG
Y	G				K K K K		R		R	
IS	0				1 A B C	0123456789ABCDEF	A	01234567 A	01234567	
IS Y	S1	141,A1	60A	90 Y						Y
IS	<BLANK>			91	Y					Y
IS Z	S2	1,A1	60C	92 Z	Y					YZ
IS	<BLANK>			93						YZ
				94						YZ
				95	Z	Y				YZ
IS 0	S3	J,A1	61A	96 0	Z	Y				YZ0
IS	<BLANK>			97	0	Z				YZ0
				98	0	Z				YZ0
				99	0	Z				YZ0
				100		Z	0			YZ0
IS 1	S1 /HS1		61C	101		Z	0			1Z0
IS 2	S4	5,A1	61D	102 2			0			1Z02
IS	<BLANK>			103	2		0			1Z02
				104		2				1Z02
IS 3	S2 /HS2		62B	105		2				1302
IS 4	S5	7,A1	62C	106 4			0			13024
IS	<BLANK>			107	4					13 24
IS 5	S3 /HS3		63A	108		4				13524
IS 6	S6	11,A1	63B	109 6		4				135246
IS	<BLANK>			110	6		4			135246
				111		6				135246
				112		6				135246
				113		6				135746
IS 7	S4 /HS4		63D	114 8		6				1357468
IS 8	S7	13,A1	64A	115	8		6			357468
IS	<BLANK>			116		8	6			357468
IS 9	T40 S1		64C	117 A		8	6			357 68
IS A	1,A7 S1		64D	118	A		8			357 68
IS	<BLANK>			119		A	8			57B68
IS B	S5 /HS5		65B	120 C		A	8			C 57B 8
IS C	S1	15,A1	65C	121	C		8			C 57B 8
IS	<BLANK>			122		C	8			C 7B 8
IS D	T41 S2		66A			C				C 7B 8

Table 2(b). Multiply kernel activity report

ST. A G	T	INSTRUCTION	P-ADDR	CP	S R R R	MEMORY BANKS	A R	A. REG	S R	S. REG
IS L	S1	142,A1	140A	335	L			01234567 A	01234567	L
IS	<BLANK>			336	L					L
IS M	S2 T40		140C	337	L	A			M	L
IS N	S3	2,A1	140D	338	N	K			L	N
IS	<DLANK>			339		K			L	N
IS O	S4 T41		141B	340	N	K			L	N
				341	N				L	N
				342	N	L			L	N
				343	N	L			L	N
				344	N	N			L	N
				345	N	N			L	N
				346	N	N			L	N
IS P	S6 S1*FS2		141C	347	Q				L	N
IS Q	S1	4,A1	141D	348	Q				L	N
IS	<BLANK>			349	Q				L	N
IS R	S2 T42		142B	350	Q				L	N
				351	Q				L	N
IS S	S7 S3*FS4		142C	352	Q				L	N
IS T	S3	6,A1	142D	353	T				L	N
IS	<BLANK>			354	T				L	N
IS U	S4 T43		143B	355	T				L	N
IS V	T00 SC		143C	356	T				L	N
				357	T				L	N
IS W	S6 S1*FS2		143D	358	T				L	N
IS X	S1	10,A1	144A	359	X				L	N
IS	<BLANK>			360	X				L	N
IS Y	S2 T44		144C	361	X				L	N
IS Z	T01 S7		144D	362	X				L	N
				363	X				L	N
IS 0	S7 S3*FS4		145A	364	X				L	N
IS 1	S3	12,A1	145D	365	X				L	N
IS	<BLANK>			366	X				L	N
IS 2	S4 T45		145D	367	X				L	N
IS 3	T02 S6		146A	368	X				L	N
				369	X				L	N


```

C**** GENERAL TEST DRIVER FOR NUMERIC SOLVER
C*** THE DIMENSION OF THE FOLLOWING SHOULD BE .GE. # OF EQUATIONS + 1
      DIMENSION SUMR(801), SUMC(801), IPIV(801), B(801), X(801), JA(801)
C *** THE DIMENSION OF THE FOLLOWING SHOULD BE .GE. # OF NONZEROS OF
C MATRIX
      DIMENSION IA(9000)
C*** THE DIMENSION OF THE FOLLOWING SHOULD BE .GE. # OF NONZEROS OF LU
      DIMENSION NUMN(10000), A(10000)
C*** DIMENSION THE FOLLOWING ARRAY TO HOLD THE CODE
      DIMENSION INST(100000)
C*** SOME EQUIVALENCES COULD BE MADE BETWEEN ABOVE ARRAYS
      READ (5,10) N, NRMAP
10  FORMAT (16I5)
      NP1 = N + 1
      READ (5,10) (JA(J),J=1,NP1)
      NA = JA(NP1) - 1
      READ (5,10) (IA(J),J=1,NA)
      IF (NRMAP .NE. 0) READ (5,10) (NUMN(J),J=1,NA)
C*** ROW INDEX OF PIVOT POSITIONS
      READ (3,10) (IPIV(I),I=1,N)
C*** ZERO LU STORAGE
      DO 20 I = 1, 10000
20  A(I) = 0.
C*** UNIFORMLY-DISTRIBUTED NEGATIVE OFF-DIAGONAL VALUES
      NNN = 999
      DO 30 J = 1, NA
          JJ = J
          IF (NRMAP .NE. 0) JJ = NUMN(J)
30  A(JJ) = -URAND(NNN)
      DO 40 J = 1, N
          SUMR(J) = 0.
          SUMC(J) = 0.
40  B(J) = 0
C*** FORMULATE EQUATIONS SO SOLUTION IS X(I) = I + 1
      DO 60 I = 1, N
          I1 = JA(I)
          I2 = JA(I + 1) - 1
          DO 50 J = I1, I2
              ICOL = IA(J)
              JJ = J
              IF (NRMAP .NE. 0) JJ = NUMN(J)
              SUMC(I) = SUMC(I) - A(JJ)
              SUMR(ICOL) = SUMR(ICOL) - A(JJ)
50  B(ICOL) = B(ICOL) + A(JJ) * (I + 1)
60  CONTINUE

```

Table 3. Example driver for numeric phase

```

C***  FIND PIVOTS AND FORCE DOMINANCE
DO 80 I = 1, N
  I1 = JA(I)
  I2 = JA(I + 1) - 1
  DO 70 J = I1, I2
    ICOL = IA(J)
    JJ = J
    IF (NRMAP .NE. 0) JJ = NUMN(J)
    IF (ICOL .NE. IPIV(I)) GO TO 70
    B(ICOL) = B(ICOL) - A(JJ) * (I + 1)
    A(JJ) = .01 + 1.1E0 * AMAX1(SUMC(I) + A(JJ), SUMR(ICOL) + A(JJ)
1    )
    B(ICOL) = B(ICOL) + A(JJ) * (I + 1)
    GO TO 80
70  CONTINUE
80  CONTINUE
  READ (8,90) NINSTW
90  FORMAT (I6)
  READ (8) (INST(I), I=1, NINSTW)
C***  A IS COLUMN-ORDERED PACKED MATRIX
C      B IS RIGHT HAND SIDE
C      X IS SOLUTION
C      N IS # OF EQUATIONS
  CALL SOLVE(INST, A, B, X, N)
  WRITE (6,100) (X(I), I=1, N)
100  FORMAT (5E12.4)
  STOP
  END

```

Table 3. Continued

# of equations	Description	No Decoupling	Decoupling
289	17x17 5 pt. 2-D grid	52060	59626
443	Elec. Power System	7528	9394
450	Electronic circuit 4-bit adder	6931	7122
507	Oil reservoir	96479	108478
2323	Oil reservoir	1360000	1407069
5300	Elec. power system	465000	534077

Table 4. Floating point operation counts to factorize matrices

# of equat.	Description	Code stor. (64-bit wds)	FP oper.	MFLOPS	Eff. MFLOPS	Ordering	Time (msec) Code Gen.	Solu.
160	Elec. Power Sys.	7691	7945	15.3	--	90.5	145	.520
289	17x17 5-pt grid nested dissect.	63375	59102	14.5	12.6	758	1250	4.06
391	Oil reserv. D-4 ordered	43096	46296	16.5	--	583	796	2.79
443	Elec. Power Sys.	14157	14001	14.7	11.7	311	250	.948
450	Elec. cir. 4-bit adder	12791	12370	14.3	13.9	314	213	.864
507	Oil reserv. D-4 ordered	113566	125117	17.3	15.4	1823	2260	7.24
1746	Elec. cir. 16-bit adder	45758	43779	14.3	14.2	3520	695	3.06
5300	Elec. Power Sys.	585253	634837	11.6	10.1	54313	25700	58.37

Table 5. Result summary. Different operating systems (CCOS and CTSS) were used to solve different problems; unexplained variability was noted in CTSS timings.

ATE
LME
8