High-Performance Banded Equation Solver for the CRAY-1

II. The Symmetric Case

D.A. CALAHAN

October 1, 1982

Sponsored jointly by
Air Force Flight Dynamics Laboratory
Wright-Patterson Air Force Base
and Directorate of Mathematical and Information Sciences,
Air Force Office of Scientific Research
under Grant No. 86-0178

Systems Engineering Laboratory
**Title:** High-performance banded equation solver for the CRAY-1: II. The symmetric case

**Authors:** D. A. Calahan

**Performing Organization:** University of Michigan

**Contract/Offer Grant Number:** AFOSR 80-0158

**DOE Office:** Air Force Office of Scientific Research

**Number of Pages:** 18

**Abstract:**

This report describes algorithms, performance, applications, and user information associated with a code which solves a memory-resident single banded matrix equation on the CRAY-1. The code is available as part of a library of CAL-coded equation-solvers for the CRAY-1.
High Performance Banded Equation Solver for the CRAY-I

II. The Symmetric Case

D. A. Calahan

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

SEL Report #166
Sponsored jointly by
Air Force Flight Dynamics Laboratory
Wright-Patterson Air Force Base, and
Directorate of Mathematical and Information Sciences
Air Force Office of Scientific Research,
under Grant 80-0158
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>2</td>
</tr>
<tr>
<td>II. Symmetric Banded Solution</td>
<td>3</td>
</tr>
<tr>
<td>A. Basic Algorithm</td>
<td>3</td>
</tr>
<tr>
<td>B. Partitioned Solution</td>
<td>5</td>
</tr>
<tr>
<td>Figure 1</td>
<td>6</td>
</tr>
<tr>
<td>Table 1</td>
<td>7</td>
</tr>
<tr>
<td>III. Software Description</td>
<td>9</td>
</tr>
<tr>
<td>A. Storage Options</td>
<td>9</td>
</tr>
<tr>
<td>B. Calling Sequences</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2</td>
<td>10</td>
</tr>
<tr>
<td>C. Comments</td>
<td>12</td>
</tr>
<tr>
<td>D. Driver Program</td>
<td>12</td>
</tr>
<tr>
<td>IV. Performance</td>
<td>13</td>
</tr>
<tr>
<td>Table 2</td>
<td>14</td>
</tr>
<tr>
<td>References</td>
<td>15</td>
</tr>
<tr>
<td>Appendix A</td>
<td>16</td>
</tr>
</tbody>
</table>
ABSTRACT

This report describes algorithms, performance, applications, and user information associated with a code which solves a memory-resident single banded symmetric matrix equation on the CRAY-I.

The code is available as part of a library of CAL-coded equation-solvers, [13].

PREFACE

The mathematical software described herein is the result of experimental research on vector algorithms for the direct solution of finite element grids arising in structural analysis. It represents what is thought to be the best compromise between vectorizability, sparsity exploitation, and user convenience for such problems for the CRAY-I.
I. Introduction

When direct methods are used in the solution of equations associated with 2-D finite element systems, the majority of production codes require a "frontal" approach [1], i.e., the finite elements are assembled and reduced in batches along a front that moves across the grid. This procedure saves storage of the entire profile matrix and so conserves memory, a major issue for the scalar scientific processors of the 1970's with fast storage often less than 100,000 words. Only relatively small research problems can be completely assembled and then completely solved in main memory. The principal difficulty with a frontal solution is programming complexity due to solution partitioning and to I/O management.

In contrast, vector processors with one- and two-megaword storage permit the memory-resident solution of the larger problems commensurate with their speed. Roughly, matrices associated with square grids three times larger on a side can now be solved by a vector processor, in the same computation time and at the storage limit of main memory.

A previous study [2] indicated that, for unsymmetric matrices, profile solution was marginally faster than banded solution on the CRAY-1. For this small speedup, significant preprocessing is required to block the profile structure. It was not considered worthwhile to produce a symmetric version of the block profile solution.
II. Symmetric Banded Solution

A. Basic algorithm

Consider an nxn symmetric banded matrix A, with half-bandwidth m. The solution of

\[ AX = B \]

is performed in two steps, viz, (1) triangular factorization

\[ A = U^T D U \]

where D is a diagonal matrix, U is an upper triangular matrix, and \( U^T \) is the transpose of U, and (2) forward and backward substitution

\[ Y_1 = (U^T)^{-1} B \]
\[ Y_2 = D^{-1} Y_1 \]
\[ X = U^{-1} Y_2 \]

Asymptotically in n, factorization of a symmetric matrix requires 1/2 the computation of an unsymmetric matrix; the substitution steps require the same computation.

The performance of the algorithm depends on the vector length for small bandwidths and on the data flow between the vector registers and main memory for all bandwidths. Mathematically, the average vector length is restricted to 1/2 that of the unsymmetric case. The data flow is minimized – and performance optimized – when accumulation is made into a single row or column from previously-factored rows and columns; a poor algorithm creating excessive data flow would be the common one based on an outer product of a row or column.

A column-oriented accumulation suggested by Jordan [3] and modified in [2] could be used with reduced success in the symmetric case, due to the halved vector length. However, the accumulation
kernel discussed in [2] suffers, for small bandwidths, from being instruction-bound and, for large bandwidths, from a shift operation in the chained sequence of vector innerloop instructions. In contrast, the following algorithm kernel consists of simply a vector-matrix multiply, which can be made quite efficient.

The accumulation is represented as being made into a row rather than a column (Figure 1). The product of the row vector to the left of the main diagonal and the triangular matrix above the accumulant to the right of the main diagonal is then added to the accumulant row. However, this simple kernel is complicated by the need to maintain components of both U and DU or UD.

The organization of the accumulation kernel has the following form (see Figure 1a). Let

\[
\begin{align*}
Y & \quad \text{be the accumulant row} \\
R & \quad \text{be the row vector, initially stored as a column of DU above the pivot} \\
C & \quad (\text{current column}) \text{ be a column of } U \text{ to be computed from and stored over } R \\
D & \quad \text{be an appropriate segment of the diagonal, before the current pivot position} \\
M & \quad \text{be the triangular matrix } DU \text{ above } Y \text{ and including } R
\end{align*}
\]

then the reduction kernel has the form

\[
\begin{align*}
T & \leftarrow D^{-1}R \\
Y & \leftarrow Y + TM \\
C & \leftarrow T
\end{align*}
\]

The current column C is a column of U; the accumulant Y is a row of DU. T is a temporary vector and resides in a vector register.

An illustrative Fortran program incorporating this algorithm
is given in Table 1. This will be exercised later to obtain performance comparisons with the more efficient assembly code to follow.

B. Partitioned Solution

When the half-bandwidth is greater than 64, so that vector lengths must be truncated, it remains efficient to preserve the concept of a vector-matrix multiply. The oversized matrix is then partitioned into bandedge and interior matrices, all of maximum dimension 64. Figure 1b illustrates this partitioning process in both the row and column directions. The circled numbers 1..4 represent the nesting level of the computation, with 1 being the innermost loop. Loops 3 and 4 relate the order of the block processing; this ordering is also represented by the circled letters a...f.

Two vector-matrix multiply kernels are now required: one for triangular bandedge matrices and one for rectangular (interior) matrices. The later kernels can achieve very high performance with short vector lengths. For vector lengths (VL) greater than 8, the execution rates are given by

$$\text{MFLOPS} = 160 \left( \frac{VL}{VL+8} \right)$$

i.e., a rate of 80 MFLOPS for $VL = 8$ and 142 MFLOPS for $VL = 64$. 
Figure 1. Organization and terminology of a reduction step
**ILLUSTRATIVE SYMMETRIC BANDED EQUATION SOLVER**

**UNPARTITIONED; FCW STORAGE ONLY**

DIMENSION A(1000), B(100), TEMP(100)

**M IS HALF-BANDWIDTH; N IS NUMBER OF EQUATIONS**

10 READ (5,20,END=90) M, N

20 FORMAT (2I10)

MP1 = M + 1
M2 = 2 * M + 1
DO 30 I = 1, N

30 B(I) = 0.

**FORMULATE EQUATIONS AND RHS TO HAVE SOLUTION B(J)=J**

DO 50 I = 1, MP1

DO 50 J = 1, N

40 IX = I + (J - 1) * MP1
A(IX) = 0.
M1 = MAX(1,M - I + 2)
IF (J .GE. M1) A(IX) = -1.
IF (I .EQ. MP1) A(IX) = M2
IY = I + J - M - 1
IF (IY .GT. 0 .AND. I .NE. MP1) B(J) = B .+. (IY) * A(IX)
IF (IY .GT. 0) B(IY) = B(IY) + (J) * A(IX)

50 CONTINUE

**TRIANGULARLY FACTOR MATRIX**

CALL FACTOR(A, TEMP, M, N)

**FORWARD AND BACK SUBSTITUTE**

CALL SOLVE(A, B, M, N)

DO 60 J = 1, N

AJ = J
IF (ABS(B(J) - AJ) .GT. 1.E-6) GO TO 70

60 CONTINUE

GO TO 10

70 WRITE (6,80) J, (B(I),I=I,N)

80 FORMAT (' FIRST WRONG SOLUTION VARIABLE IS', I5/(5E12.4))

90 STOP

**END**

SUBROUTINE FACTOR(A, TEMP, M, N)

DIMENSION TEMP(1), A(1)

MP1 = M + 1
A(MP1) = 1.E0 / A(MP1)

DO 50 J = 2, N

JM1 = J - 1
M1 = MAX(1,J - M)
ID = M1 * MP1 - MP1
IX = M1 + J * M - 1

CDIR$ IVDEP

DO 10 I = M1, JM1

IX = IX + 1
ID = ID + MP1

10 TEMP(I) = A(IX) * A(ID)

---

*Table 1. Simplified Fortran version of code*
DO 30 I = M1, JM1
   IX = I - J + J * MP1
   (S = TEMP(I)
   IZ = MINO(N,M + I)
   IL = I + J * M - M
   LJ = J * MP1 - M
CDIR$ IVDEP
   DO 20 L = J, IZ
      LJ = LJ + M
      IL = IL + M
20   A(LJ) = A(LJ) - A(IL) * US
30 CONTINUE
   IX = J * M + M1 - 1
CDIR$ IVDEP
   DO 40 I = M1, JM1
      IX = IX + 1
40   A(IX) = TEMP(I)
      JJ = J * MP1
      A(JJ) = 1.E0 / A(JJ)
RETURN
END
SUBROUTINE SOLVE(A, B, M, N)
DIMENSION A(1), B(1)
MP1 = M + 1
NM1 = N - 1
DO 10 I = 1, NM1
   IP1 = I + 1
   IL = I * MP1
   M1 = MINO(N,I + M)
CDIR$ IVDEP
   DO 10 L = IP1, M1
      IL = IL + M
10   B(L) = B(L) - A(IL) * B(I)
CDIR$ IVDEP
   II = 0
   DO 20 I = 1, N
      II = II + MP1
20   B(I) = B(I) * A(II)
   DO 30 L = 1, NM1
      LL = N - L + 1
      LLM1 = LL - 1
      M1 = MAXO(1,LL - M)
      ILL = M1 + LL * M - 1
CDIR$ IVDEP
   DO 30 I = M1, LLM1
      ILL = ILL + 1
30   B(I) = B(I) - A(ILL) * B(LL)
RETURN
END

Table 1. Continued
III. Software Description

A. Storage Options

It is common to store the diagonal and the U matrix in compressed form in an array of dimension \( N \times (M+1) \). Figure 2 illustrates eight possible regularly-addressed storage patterns; in each case, \( u_{ij} \) may be replaced by \( l_{ji} \) to represent the storage of \( L(=U^T) \) rather than \( U \). Fortunately, all of these cases may be accommodated by defining suitable parameters of the argument lists of the following routines. The key to the generality is the passing of the \((1,1)\) position of the matrix rather than the first element of the matrix storage array; all indexing is then performed off of this base.

B. Calling Sequences

Factorization

\[ \text{CALL SDAN7}(N,M,A(N1L),NDIAG,NDROW) \]

where

- \( N \) is the number of equations
- \( M \) is the half-bandwidth (not including the diagonal)
- \( A(N1L) \) is the \((1,1)\) element of the matrix
- \( NDIAG \) is the storage increment between successive diagonal elements
- \( NDROW \) is the storage increment between successive column elements

Substitution

\[ \text{CALL SPANS}(N,M,A(N1L),NDIAG,NDROW,Y) \]

*See following discussion for symbol definitions*
Figure 2. Permitted compressed storage; $d_i$ is $i$th diagonal element; replace $u_{ij}$ by $j_i$ when $L$ is stored; $NDCOL = NDIAG - NDROW$. 
Figure 2. (Continued)
where

\[ N \ldots NDROW \text{ are defined above, except that } A(N11) \text{ is the (1,1) element of the factorized matrix} \]

\[ Y \text{ is the right hand side on entry and the solution on exit.} \]

C. Comments

1. Let \( NDCOL = NDIAG - NDROW \) be the distance between successive elements in a row. When \( |NDCOL| \) is a multiple of eight, performance of both the factorization and forward substitution step is severely degraded by a factor approaching four. When \( |NDROW| \) or \( |NDIAG| \) are multiples of eight, some degradation will also be noted for small bandwidths.

2. The dimension of \( Y \) must be at least \( N + M + 1 \).

3. The storage of the matrix may have to be increased to assure that certain data outside the normal matrix storage can be operated upon as floating point numbers. These positions are indicated by asterisks in Figure 1. For example, in Figure 1(a), the solver will access the data "above" the normal matrix storage, use it as operands for floating point add and multiply, but will not store the results. In this case additional storage need not be allocated, since these operands will simply be fetched from the preceding column. Only when this fetched data represents a fixed point or instruction format can floating point exceptions be expected.

D. Driver Program

Appendix A contains a listing of a Fortran driver program that formulates equations that are diagonally dominant and that are stored
so that neither |NROW|, |NDCOL|, or |NDIAG| are multiples of eight, thus avoiding memory bank conflicts.

IV. Performance

Table 2 gives the measured solution times and execution rates associated with solving 1024 equations on the CRAY-1. Among the more interesting results are the rates for solving small-bandwidth cases, in comparison with the unsymmetric solver of [2] that has twice the average vector length. For half-bandwidths of 8, 16, and 32, the unsymmetric factorization executes at 18, 44, and 88 MFLOPS, respectively. From Table 2 the corresponding rates are 13.4, 34.0, and 68.9 MFLOPS. The asymptotic rates are similar for both solvers.

It should be pointed out that the timings in [2] were obtained on the COS operating system; CTSS was used to produce the results of Table 2.
<table>
<thead>
<tr>
<th>Half-Bandwidth</th>
<th>Factorization</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>.00508/5.03</td>
<td>.00122/14.2</td>
</tr>
<tr>
<td>8</td>
<td>.00616/13.4</td>
<td>.00122/27.6</td>
</tr>
<tr>
<td>16</td>
<td>.00865/34.0</td>
<td>.00122/54.1</td>
</tr>
<tr>
<td>32</td>
<td>.0160/68.9</td>
<td>.00201/64.7</td>
</tr>
<tr>
<td>64</td>
<td>.0401/105.</td>
<td>.00323/78.9</td>
</tr>
<tr>
<td>65</td>
<td>.0554/78.7</td>
<td>.00406/63.7</td>
</tr>
<tr>
<td>68</td>
<td>.0559/85.1</td>
<td>.00449/60.2</td>
</tr>
<tr>
<td>80</td>
<td>.0687/95.0</td>
<td>.00450/70.1</td>
</tr>
<tr>
<td>96</td>
<td>.0894/104.</td>
<td>.00468/80.2</td>
</tr>
<tr>
<td>128</td>
<td>.139/117.</td>
<td>.00552/89.2</td>
</tr>
<tr>
<td>129</td>
<td>.164/101.</td>
<td>.00624/79.4</td>
</tr>
<tr>
<td>132</td>
<td>.165/105.</td>
<td>.00667/75.9</td>
</tr>
<tr>
<td>144</td>
<td>.189/108.</td>
<td>.00674/81.4</td>
</tr>
<tr>
<td>160</td>
<td>.218/115.</td>
<td>.00686/88.1</td>
</tr>
<tr>
<td>196</td>
<td>.328/113.</td>
<td>.00886/82.0</td>
</tr>
<tr>
<td>197</td>
<td>.328/114.</td>
<td>.00886/82.3</td>
</tr>
<tr>
<td>200</td>
<td>.335/115.</td>
<td>.00886/83.5</td>
</tr>
</tbody>
</table>

Table 2. Execution time (sec) and rate (MFLOPS) to solve 1024 equations
References


Appendix A

Listing of Driver Program
C*** DRIVER FOR SYMMETRIC BAND SOLVER
1
2 DIMENSION A(500000), B(3000)
3 DO 96 J=1,21
4 10 READ (5,20,END=100) M, N, NT
5 20 FORMAT (31I10)
6 MP1 = M + 1
7 M2 = 2 * M + 1
8 DO 30 I = 1, N
9 30 B(I) = 0.
10 C ROW STORAGE
11 NDIAG = MP1
12 NROW = 1
13 N11 = MP1
14 IF (NT .EQ. 0) GO TO 40
15 C COLUMN STORAGE
16 NDIAG = 1
17 NROW = -(N - 1)
18 N11 = 1
19 NDCOL = NDIAG * N
20 IF((NDCOL/8) * 8 - NDCOL .EQ. 0) GO TO 114
21 WRITE(6,77)
22 77 FORMAT(' EXECUTION WILL BE SLOWED BY BANK CONFLICTS:')
23 1' NDCOL OR NDIAG WILL BE CHANGED')
24 IF (NDCOL .LE. 1) NDCOL = NDCOL - N
25 IF (NDIAG .LE. 1) NDIAG = MP1 + 1
26 NDCOL = NDIAG - NDCOL
27 114 DO 50 I=1,N
28 50 A(I) = 0.
29 DO 60 J = 1, N
30 60 DO 61 I = 1, MP1
31 NCOL = J + I - 1
32 IF (NCOL .GT. N) GO TO 60
33 NA = N11 + NDIAG * (J - 1) + NDCOL * (I - 1)
34 A(NA) = -1.
35 IF (I .EQ. 1) A(NA) = M2
36 B(J) = B(J) + NCOL * A(NA)
37 IF (I .NE. 1) B(NDCOL) = B(NDCOL) + J * A(NA)
38 61 CONTINUE
39 T1=SECOND() CALL SBANF(N, M, A(N11), NDIAG, NROW)
40 T1=SECOND() - T1
41 T2=SECOND() C N2=N11+20
42 C WRITE(6,91) (A(J),J=1,N2)
43 91 FORMAT(5E12.4)
44 CALL SBANSN(M, A(N11), NDIAG, NROW, B)
45 T2=SECOND() - T2
46 DO 70 J = 1, N
47 AJ = J
48 IF (ABS(A(J)) .GT. 1.E-6) GO TO 80
49 70 CONTINUE
50 WRITE(6,92) M, N, NT
51 92 FORMAT(' M=',M,' N=',N,' NT=',NT)
52 WRITE(6,93) T1, T2
53 93 FORMAT(' TIMINGS: ',2F16.6)
54 96 CONTINUE
55 96 CONTINUE
56 GO TO 100