Squeezing the Most out of an Algorithm in CRAY FORTRAN

JACK J. DONGARRA
Argonne National Laboratory
and
STANLEY C. EISENSTAT
Yale University

This paper describes a technique for achieving supervector performance on a CRAY-1 in a purely FORTRAN environment (i.e., without resorting to assembler language). The technique can be applied to a wide variety of algorithms in linear algebra, and is beneficial in other architectural settings.

Categories and Subject Descriptors: G.1.3 [**Mathematics of Computing**]: Numerical Analysis—*numerical linear algebra*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Performance

Additional Key Words and Phrases: Vector processing, linear algebra, efficiency, unrolling

INTRODUCTION

There are three basic performance levels on the CRAY-1—*scalar, vector, and supervector* [4]:

Performance level	Rate of execution, MFLOPS[1]
Scalar	0–4
Vector	4–50
Supervector	50–160

The difference between scalar and vector modes is the use of vector instructions to eliminate loop overhead and take full advantage of the pipelined functional units. The difference between vector and supervector modes is the use of vector

[1] MFLOPS is an acronym for million floating-point operations (additions or multiplications) per second.

The first author's work was supported in part by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U. S. Department of Energy under Contract W-31-109-Eng-38.
The second author's work was supported in part by the Office of Naval Research under contract N00014-82-K-0184 and by the National Science Foundation under grant MCS-81-04874.
Authors' addresses: J. J. Dongarra, Argonne National Laboratory. 9700 South Cass Avenue, Argonne, IL 60439, S. C Eisenstat, Department of Computer Science and Research Center for Scientific Computation, Yale University, P.O. Box 2158, New Haven, CT 06520

registers to reduce the number of memory references (and thus avoid letting the one path to/from memory become a bottleneck).

Typically, programs written in FORTRAN run at scalar or vector speeds, so that one must resort to assembler language (or assembler language kernels) to improve performance. In this paper, we describe a technique for attaining supervector speeds *from FORTRAN*.[2]

THE IDEAL SETTING[3]

Most algorithms in linear algebra are easily vectorized. For example, consider the following subroutine which adds the product of a matrix and a vector to another vector:

```
      SUBROUTINE SMXPY (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM, *)
      DO 20 J = 1, N2
        DO 10 I = 1, N1
          Y(I) = Y(I) + X(J) * M(I,J)
10      CONTINUE
20    CONTINUE
      RETURN
      END
```

The innermost loop is a SAXPY [5] (adding a multiple of one vector to another) and would be detected by a good vectorizing compiler. Thus, the CRAY CFT FORTRAN compiler generates vector code of the general form:

Load vector Y
Load scalar X(J)
Load vector M(•, J)
Multiply scalar X(J) *times vector* M(•, J)
Add result to vector Y
Store result in Y

Note that there are *three* vector memory references for each *two* vector floating-point operations. Since there is only one path to/from memory and the memory bandwidth is 80 million words per second, the rate of execution cannot exceed ~53⅓ MFLOPS (less than 50 MFLOPS when vector start-up time is taken into account)—vector performance.

Thus to attain supervector performance, it is necessary to expand the scope of the vectorizing process to more than just simple vector operations. In this case, a closer inspection reveals that the vector Y is stored and then reloaded in successive SAXPYs. If instead we accumulate Y in a vector register (up to 64 words at a time) until all of the columns of M have been processed, we can avoid two of the three vector memory references in the innermost loop. The maximum rate of execution is then 160 MFLOPS (~148 MFLOPS when vector start-up time is taken into account)—supervector performance.

[2] We recognize that assembler code may be needed to achieve the highest level of performance, and that its use in a small number of "kernels" is not a significant barrier to transportability. However, the approach presented does lead to high levels of performance, is portable, and can be used to derive *algorithmic* improvements in a much wider class of problems than discussed in this paper.
[3] See [4] for a more complete discussion.

REALITY

The CRAY CFT compiler does not detect the fact that the result can be accumulated in a register (and not stored between successive vector operations). Thus, the rate of execution is limited to vector speeds.

But if we unroll [1] the outer loop (in this case to a depth of four) and insert parentheses to force the arithmetic operations to be performed in the most efficient order, then the innermost loop becomes

```
      DO 10 I = 1, N1
        Y(I) = ((((Y(I) + X(J-3) * M(I,J-3)) + X(J-2) * M(I,J-2))
     $                 + X(J-1) * M(I,J-1)) + X(J) * M(I,J)
10    CONTINUE.
```

Now the code generated by CFT has *six* vector memory references for each *eight* vector floating-point operations. Thus the maximum rate of execution is ~106⅔ MFLOPS (~100 MFLOPS when vector start-up time is taken into account) and the actual rate is ~77 MFLOPS—supervector performance from FORTRAN. The complete subroutine SMXPY4 is given in Appendix A.

GENERALIZATIONS

With this approach we can develop quite a collection of procedures from linear algebra. The key idea is to use two kernels—SMXPY and SXMPY (add a vector times a matrix to another vector; see Appendix II)—to do the bulk of the work. Since both kernels can be unrolled[4] to give supervector performance, the procedures themselves are capable of supervector performance.

Many processes which involve elementary transformations can be described in these terms, e.g., matrix multiplication, Cholesky decomposition, and LU factorization (see Appendix III and [4, 6]). However, the formulation is often not the "natural" one, which may be based on outer products of vectors or accumulating variable-length vectors, neither of which can be supervectorized in FORTRAN.

Tables I–IV summarize the results obtained for these procedures on a CRAY 1-S (as well as on the new CRAY 1-M[5] and CRAY X-MP[6]) when the subroutines SMXPY and SXMPY were unrolled to the specified depth. All runs used the CFT 1.11 FORTRAN compiler. By contrast, 30 MFLOPS is often cited as a "good rate for FORTRAN" on the CRAY 1-S [3] and 100 MFLOPS as a "good rate for CAL (Cray Assembler Language)" [3] (e.g., Fong and Jordan [4] report 107 MFLOPS for an assembler language implementation of *LU* decomposition with pivoting).

---

[4] Although there are only eight vector registers, this is sufficient for any depth of unrolling.

[5] The CRAY 1-M is essentially a CRAY 1-S with "slow" memory. It is faster in these tests because of a chaining anomaly—a vector load issues earlier on the CRAY 1-S, causing a scalar-vector multiply to miss chain-slot time.

[6] The CRAY X-MP is a multiprocessor, each processor having a cycle time of 9.5 ns (versus 12.5 ns for the CRAY 1-S) and three paths to/from memory (two for vector loads, one for vector stores). These timings were obtained using only one processor. While, in principle, the extra paths should remove the memory bottleneck, in practice the unrolled code still runs faster because there are fewer vector startups and less memory traffic (and thus fewer bank conflicts).

Table I. 300 × 300 Matrix Multiplication

Unrolled depth	MFLOPS		
	CRAY 1-M	CRAY 1-S	CRAY X-MP
1	39	40	106
2	60	53	151
4	83	72	161
8	101	86	170
16	111	96	177

Table II. 300 × 300 Cholesky Decomposition

Unrolled depth	MFLOPS		
	CRAY 1-M	CRAY 1-S	CRAY X-MP
1	31	33	68
2	48	45	99
4	67	60	118
8	81	70	131
16	86	78	139

Table III. 300 × 300 *LU* Decomposition with Pivoting

Unrolled depth	MFLOPS		
	CRAY 1-M	CRAY 1-S	CRAY X-MP
1	28	29	56
2	42	39	78
4	56	52	93
8	66	60	103
16	69	66	108

Table IV. 300 × 300 *LU* Decomposition with Pivoting (Using an Assembler Language Implementation of ISAMAX*)

Unrolled depth	MFLOPS		
	CRAY 1-M	CRAY 1-S	CRAY X-MP
1	30	32	62
2	46	43	96
4	64	59	117
8	78	68	129
16	83	76	136

* The search for the maximum element in the pivot column (ISAMAX [5]) does not vectorize and thus limits performance. These times were obtained using an assembler language implementation of ISAMAX.

CONCLUSIONS

We have described a technique that can produce significant gains in execution speed on the CRAY-1.[7] Moreover, to the extent that this approach reduces loop

[7] See [2] for another approach.

overhead and takes advantage of segmented functional units, it will be effective on more conventional computers as well as on other "supercomputer" architectures. Since optimized assembler language implementations of the SMXPY and SXMPY kernels are easy to code (as much so as any kernel) and frequently available, one can get most of the advantages of assembler language while programming in FORTRAN.

APPENDIX A

```
      SUBROUTINE SMXPY4 (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM,*)
C
C   PURPOSE:
C     Multiply matrix M times vector X and add the result to vector Y.
C
C   PARAMETERS:
C
C     N1 INTEGER, number of elements in vector Y, and number of rows in
C         matrix M
C
C     Y REAL(N1), vector of length N1 to which is added the product M*X
C
C     N2 INTEGER, number of elements in vector X, and number of columns
C         in matrix M
C
C     LDM INTEGER, leading dimension of array M
C
C     X REAL(N2), vector of length N2
C
C     M REAL(LDM,N2), matrix of N1 rows and N2 columns
C
C ----------------------------------------------------------------------
C
C   Cleanup odd vector
C
      J = MOD(N2,2)
      IF (J .GE. 1) THEN
         DO 10 I = 1, N1
            Y(I) = (Y(I)) + X(J)*M(I,J)
   10    CONTINUE
      ENDIF
C
C   Cleanup odd group of two vectors
C
      J = MOD(N2,4)
      IF (J .GE. 2) THEN
         DO 20 I = 1, N1
            Y(I) = ( (Y(I))
     $               + X(J-1)*M(I,J-1)) + X(J)*M(I,J)
   20    CONTINUE
      ENDIF
C
C   Main loop - groups of four vectors
C
      JMIN = J+4
      DO 40 J = JMIN, N2, 4
```

```
            DO 30 I = 1, N1
               Y(I) = ((( (Y(I))
     $                 + X(J-3)*M(I,J-3)) + X(J-2)*M(I,J-2))
     $                 + X(J-1)*M(I,J-1)) + X(J)  *M(I,J)
   30       CONTINUE
   40    CONTINUE
C
         RETURN
         END
```

APPENDIX B

```
         SUBROUTINE SMXPY (N1, Y, N2, LDM, X, M)
         REAL Y(*), X(*), M(LDM,*)
C
C     PURPOSE:
C       Multiply matrix M times vector X and add the result to vector Y.
C
C     PARAMETERS:
C
C       N1 INTEGER, number of elements in vector Y, and number of rows in
C           matrix M
C
C       Y REAL(N1), vector of length N1 to which is added the product M*X
C
C       N2 INTEGER, number of elements in vector X, and number of columns
C           in matrix M
C
C       LDM INTEGER, leading dimension of array M
C
C       X REAL(N2), vector of length N2
C
C       M REAL(LDM,N2), matrix of N1 rows and N2 columns
C
C ----------------------------------------------------------------------
C
         DO 20 J = 1, N2
            DO 10 I = 1, N1
               Y(I) = (Y(I)) + X(J)*M(I,J)
   10       CONTINUE
   20    CONTINUE
C
         RETURN
         END

         SUBROUTINE SXMPY (N1, LDY, Y, N2, LDX, X, LDM, M)
         REAL Y(LDY,*), X(LDX,*), M(LDM,*)
C
C     PURPOSE:
C       Multiply row vector X times matrix M and add the result to row
C       vector Y.
C
C     PARAMETERS:
C
C       N1 INTEGER, number of columns in row vector Y, and number of
C           columns in matrix M
C
```

```
C     LDY INTEGER, leading dimension of array Y
C
C     Y REAL(LDY,N1), row vector of length N1 to which is added the
C       product X*M
C
C     N2 INTEGER, number of columns in row vector X, and number of
C       rows in matrix M
C
C     LDX INTEGER, leading dimension of array X
C
C     X REAL(LDX,N2), row vector of length N2
C
C     LDM INTEGER, leading dimension of array M
C
C     M REAL(LDM,N1), matrix of N2 rows and N1 columns
C
C ----------------------------------------------------------------------
C
      DO 20 J = 1, N2
         DO 10 I = 1, N1
            Y(1,I) = (Y(1,I)) + X(1,J)*M(J,I)
   10    CONTINUE
   20 CONTINUE
C
      RETURN
      END
```

APPENDIX C

```
      SUBROUTINE MM (A, LDA, N1, N3, B, LDB, N2, C, LDC)
      REAL A(LDA,*), B(LDB,*), C(LDC,*)
C
C   PURPOSE:
C     Multiply matrix B times matrix C and store the result in matrix A.
C
C   PARAMETERS:
C
C     A REAL(LDA,N3), matrix of N1 rows and N3 columns
C
C     LDA INTEGER, leading dimension of array A
C
C     N1 INTEGER, number of rows in matrices A and B
C
C     N3 INTEGER, number of columns in matrices A and C
C
C     B REAL(LDB,N2), matrix of N1 rows and N2 columns
C
C     LDB INTEGER, leading dimension of array B
C
C     N2 INTEGER, number of columns in matrix B, and number of rows in
C        matrix C
C
C     C REAL(LDC,N3), matrix of N2 rows and N3 columns
C
C     LDC INTEGER, leading dimension of array C
C
C ----------------------------------------------------------------------
```

```
C
      DO 20 J = 1, N3
         DO 10 I = 1, N1
            A(I,J) = 0.0
   10    CONTINUE
         CALL SMXPY (N2,A(1,J),N1,LDB,C(1,J),B)
   20 CONTINUE
C
      RETURN
      END

      SUBROUTINE LLT (A, LDA, N, ROWI, INFO)
      REAL A(LDA,*), ROWI(*), T
C
C  PURPOSE:                                                    t
C     Form the Cholesky factorization  A = L*L   of a symmetric positive
C     definite matrix A with factor L overwriting A.
C
C  PARAMETERS:
C
C     A REAL(LDA,N), matrix to be decomposed; only the lower triangle
C         need be supplied, the upper triangle is not referenced
C
C     LDA INTEGER, leading dimension of array A
C
C     N INTEGER, number of rows and columns in the matrix A
C
C     ROWI REAL(N), work array
C
C     INFO INTEGER, = 0 for normal return
C                   = I if I-th leading minor is not positive definite
C
C ---------------------------------------------------------------------
C
      INFO = 0
      DO 30 I = 1, N
C
C      Subtract multiples of preceding columns from I-th column of A
C
         DO 10 J = 1, I-1
            ROWI(J) = -A(I,J)
   10    CONTINUE
         CALL SMXPY (N-I+1,A(I,I),I-1,LDA,ROWI,A(I,1))
C
C      Test for non-positive definite leading minor
C
         IF (A(I,I) .LE. 0.0) THEN
            INFO = I
            GO TO 40
         ENDIF
C
C      Form I-th column of L
C
         T = 1.0/SQRT(A(I,I))
         A(I,I) = T
         DO 20 J = I+1, N
            A(J,I) = T*A(J,I)
   20    CONTINUE
   30 CONTINUE
```

```
      40 RETURN
         END
      SUBROUTINE LU (A, LDA, N, IPVT, INFO)
      INTEGER IPVT(*)
      REAL A(LDA,*), T
C
C   PURPOSE:
C     Form the LU factorization of A, where L is lower triangular and U
C     is unit upper triangular, with the factors L and U overwriting A.
C
C   PARAMETERS:
C
C     A REAL(LDA,N), matrix to be factored
C
C     LDA INTEGER, leading dimension of the array A
C
C     N INTEGER, number of rows and columns in the matrix A
C
C     IPVT INTEGER(N), sequence of pivot rows
C
C     INFO INTEGER, = 0 normal return.
C                   = J if L(J,J) is zero (whence A is singular)
C
C ------------------------------------------------------------------------
C
      INFO = 0
      DO 40 J = 1, N
C
C        Form J-th column of L
C
         CALL SMXPY (N-J+1,A(J,J),J-1,LDA,A(1,J),A(J,1))
C
C        Search for pivot
C
         T = ABS(A(J,J))
         K = J
         DO 10 I = J+1, N
            IF (ABS(A(I,J)) .GT. T) THEN
               T = ABS(A(I,J))
               K = I
            END IF
   10    CONTINUE
         IPVT(J) = K
C
C        Test for zero pivot
C
         IF (T .EQ. 0.0) THEN
            INFO = J
            GO TO 50
         ENDIF
C
C        Interchange rows
C
         DO 20 I = 1, N
            T = A(J,I)
            A(J,I) = A(K,I)
            A(K,I) = T
   20    CONTINUE
C
```

```
C      Form J-th row of U
C
         A(J,J) = 1.0/A(J,J)
         CALL SXMPY (N-J,LDA,A(J,J+1),J-1,LDA,A(J,1),LDA,A(1,J+1))
         T = -A(J,J)
         DO 30 I = J+1, N
            A(J,I) = T*A(J,I)
   30    CONTINUE
   40 CONTINUE
C
   50 RETURN
      END
```

APPENDIX D

```
      SUBROUTINE LLTS (A, LDA, N, X, B)
      REAL A(LDA,*), X(*), B(*), XK
C
C   PURPOSE:
C     Solve the symmetric positive definite system Ax = b given the
C     Cholesky factorization of A (as computed in LLT).
C
C   PARAMETERS:
C
C     A REAL(LDA,N), matrix which has been decomposed by routine LLT
C         in preparation for solving a system of equations
C
C     LDA INTEGER, leading dimension of array A
C
C     N INTEGER, number of rows and columns in the matrix A
C
C     X REAL(N), solution of linear system
C
C     B REAL(N), right-hand-side of linear system
C
C ---------------------------------------------------------------------
C
      DO 10 K = 1, N
         X(K) = B(K)
   10 CONTINUE
C
      DO 30 K = 1, N
         XK = X(K)*A(K,K)
         DO 20 I = K+1, N
            X(I) = X(I) - A(I,K)*XK
   20    CONTINUE
         X(K) = XK
   30 CONTINUE
C
      DO 50 K = N, 1, -1
         XK = X(K)*A(K,K)
         DO 40 I = 1, K-1
            X(I) = X(I) - A(K,I)*XK
   40    CONTINUE
         X(K) = XK
   50 CONTINUE
```

```
C
      RETURN
      END
      SUBROUTINE LUS (A, LDA, N, IPVT, X, B)
      INTEGER IPVT(*)
      REAL A(LDA,*), X(*), B(*), XK
C
C     PURPOSE:
C       Solve the linear system Ax = b given the LU factorization of A (as
C       computed in LU).
C
C     PARAMETERS:
C
C       A REAL(LDA,N), matrix which has been decomposed by routine LU
C           in preparation for solving a system of equations
C
C       LDA INTEGER, leading dimension of the array A
C
C       N INTEGER, number of rows and columns in the matrix A
C
C       IPVT INTEGER(N), sequence of pivot rows
C
C       X REAL(N), solution of linear system
C
C       B REAL(N), right-hand-side of linear system
C
C ---------------------------------------------------------------------
C
      DO 10 K = 1, N
         X(K) = B(K)
   10 CONTINUE
C
      DO 20 K = 1, N
         L = IPVT(K)
         XK = X(L)
         X(L) = X(K)
         X(K) = XK
   20 CONTINUE
C
      DO 40 K = 1, N
         XK = X(K)*A(K,K)
         DO 30 I = K+1, N
            X(I) = X(I) - A(I,K)*XK
   30    CONTINUE
         X(K) = XK
   40 CONTINUE
C
      DO 60 K = N, 1, -1
         XK = X(K)
         DO 50 I = 1, K-1
            X(I) = X(I) + A(I,K)*XK
   50    CONTINUE
   60 CONTINUE
C
      RETURN
      END
```

ACKNOWLEDGMENTS

We would like to thank the National Magnetic Fusion Energy Computer Center for providing computer time to carry out some of the experiments, Cray Research for their cooperation, and Alan Hinds for many stimulating discussions on code optimization.

REFERENCES

1. Dongarra, J. J., and Hinds, A. R. Unrolling loops in FORTRAN. *Software—Practice and Experience 9* (1979), 219–229.
2. Duff, I. S. The solution of sparse linear equations on the CRAY-1. *CRAY Channels 4*, 3 (1982), 4–9.
3. Duff, I. S., and Reid, J. K. Experience of sparse matrix codes on the CRAY-1. *Comput. Phys. Commun. 26* (1982), 293–302.
4. Fong, K., and Jordan, T. L. *Some linear algebra algorithms and their performance on the CRAY-1*. UC-32. Los Alamos Scientific Laboratory, June 1977.
5. Lawson, C., Hanson, R., Kincaid, D., and Krogh, F. Basic linear algebra subprograms for FORTRAN Usage. *ACM Trans. Math. Softw.* 5 (1979), 308–371.
6. Orbits, D. A., and Calahan, D. A. Data flow considerations in implementing a full matrix solver with backing store on the CRAY-1. Systems Engineering Laboratory *Rep.* 98 Univ. of Michigan, Sept. 1976.