Squeezing the Most out of an Algorithm in Cray Fortran

Jack J. Dongarra[†] *and Stanley C. Eisenstat*[‡]

Research Report YALEU/DCS/RR-269
May 1983

[†] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

[‡] Department of Computer Science and Research Center for Scientific Computation, Yale University, P. O. Box 2158, New Haven, CT 06520.

The work presented in this paper was supported in part by the Office of Naval Research under contract N00014-82-K-0184, by the National Science Foundation under grant MCS-81-04874 and grant MCS-81-06181, and by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the Department of Energy under contract W-31-109-Eng-38.

Squeezing the Most out of an Algorithm in Cray Fortran

Jack J. Dongarra[†]

Mathematics and Computer Science Division
Argonne National Laboratory

Stanley C. Eisenstat[‡]

Department of Computer Science and
Research Center for Scientific Computation
Yale University

Abstract — This paper discusses a technique for achieving super-vector performance on a Cray in a purely Fortran environment (i.e., without resorting to assembly language). The technique can be applied to a wide variety of algorithms in linear algebra, and is beneficial in other architectural settings.

Introduction

There are three basic performance levels on the Cray-1 — *scalar, vector, and super-vector* [3]:

Performance Level	Rate of Execution*
Scalar	0–10 MFLOPS
Vector	10–50 MFLOPS
Super-Vector	50–160 MFLOPS

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 super-vector modes is the use of vector 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 assembly language (or assembly language kernels) to improve

[†] Work 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.

[‡] Work 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.

*MFLOPS is an acronym for Million FLoating-point OPerations (additions or multiplications) per Second.

performance. In this paper, we discuss a technique for attaining super-vector speeds *from Fortran.*

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 [4] (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* memory references for each *two* floating-point operations. Since there is only one path to/from memory and the memory bandwidth is 80 million words per second, the maximum rate of execution is ~53 MFLOPS (less than 50 MFLOPS when vector startup time is taken into account) — *vector performance.*

Thus to attain super-vector 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 SAXPY's. 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 memory references in the innermost loop. The maximum rate of execution is then 160 MFLOPS (~147 MFLOPS when vector startup time is taken into account) — *super-vector performance.*

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*** memory references for each ***eight*** floating-point operations. Thus the maximum rate of execution is ~100 MFLOPS — ***super-vector performance from Fortran.*** The complete subroutine SMXPY4 is given in Appendix I.

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* to give super-vector performance, the procedures themselves are capable of super-vector 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 [3, 5]). 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 super-vectorized in Fortran.

Tables 1-3 below summarize the results obtained for these procedures on a Cray 1-S (as well as on the new Cray 1-M† and Cray X-MP‡) when the subroutines SMXPY and SXMPY were unrolled to the specified depth. All runs used the CFT 1.11 Fortran compiler.

---

*Although there are only eight vector registers, the number required is largely independent of the depth of unrolling.

† The Cray 1-M is essentially a Cray 1-S with "slow" memory. Its is faster in these tests because of a chaining anomaly.

‡ The Cray X-MP is a multiprocessor with a cycle time of 9.5 ns (vs. 12.5 ns for the Cray 1-S) and three paths to/from memory. The timings were obtained using only one processor.

Table 1: 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 2: 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 3a: 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 3b: 300 × 300 LU Decomposition with Pivoting
(Using an Assembly 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

By contrast, 30 MFLOPS is often cited as a "good rate for Fortran" [2] and 100 MFLOPS as a "good rate for CAL (Cray Assembly Language)" [2] (e.g., Fong and Jordan [3] report 107 MFLOPS for an assembly language implementation of LU decomposition with pivoting).

Conclusion

We have described a technique that can produce significant gains in execution speed on the Cray-1. Moreover, to the extent that this approach reduces loop overhead and takes advantage of segmented functional units, it will be effective on more conventional computers as well as on other "super-computer" architectures. And since optimized assembly 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 assembly language while programming in Fortran.

ACKNOWLEDGMENTS

We would like to thank the National Magnetic Fusion Energy Computer Center for providing computer time to carry out some of the experiments, and Cray Research for their cooperation.

* The search for the maximum element in the pivot column (ISAMAX [4]) does not vectorize and thus limits performance. These times were obtained using an assembly language implementation of ISAMAX.

REFERENCES

[1] J. J. Dongarra and A. R. Hinds, "Unrolling loops in Fortran," *Software—Practice and Experience* **9** (1979), 219-229.

[2] I. S. Duff and J. K. Reid, *Experience of Sparse Matrix Codes on the Cray-1,* Report CSS-116, Computer Science and Systems Division, AERE Harwell, October 1981.

[3] Kirby Fong and Thomas L. Jordan, *Some Linear Algebra Algorithms and Their Performance on the CRAY-1,* Los Alamos Scientific Laboratory, UC-32, June 1977.

[4] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Transactions on Mathematical Software* **5** (1979), 308-371.

[5] D. A. Orbits and D. A. Calahan, *Data Flow Considerations in Implementing a Full Matrix Solver with Backing Store on the Cray-1,* Report #98, Systems Engineering Laboratory, University of Michigan, September 1976.

APPENDIX I

```
      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
```

```
      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
```

```
      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
   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) THEN
            INFO = I
            GO TO 40
         ENDIF
C
C        Form I-th column of L
C
         T = 1/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
C
   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) 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/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
```

```
      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   ADDITIONAL PARAMETERS (NOT PARAMETERS OF LLT):
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   ADDITIONAL PARAMETERS (NOT PARAMETERS OF LU):
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
```