Appendix B

Idioms and Transformations

This appendix shows the CM Fortran output code that CMAX generates from particular Fortran 77 loop idioms and other constructions.

B.1 Fortran 90 and CM Fortran Language Constructs

For descriptions of these constructs, see the *CM Fortran Reference Manual*. These examples use one- and two-dimensional arrays; in general, multidimensional forms are recognized.

B.1.1 Array Syntax

This subroutine performs an elementwise addition of two arrays:

```
SUBROUTINE EXAMPLE_ARRAY_ADD(A, B, C, N)
REAL A(N,N), B(N,N), C(N,N)
DO J = 1,N
   DO I = 1,N
      C(I,J) = A(I,J) + B(I,J)
   END DO
END DO
RETURN
END
```

It translates to code which uses array syntax:

Copyright © 1994 Thinking Machines Corporation

```
      SUBROUTINE EXAMPLE_ARRAY_ADD(A,B,C,N)
      REAL A(N,N), B(N,N), C(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
CMF$  LAYOUT C(:NEWS,:NEWS)
      C = A + B
      RETURN
      END
```

This subroutine computes a more complicated function of several arrays, using scalars which need to be promoted:

```
      SUBROUTINE EXAMPLE_ARRAY_CRUNCH(X1, X2, Y1, Y2, D, N)
      REAL X1(N), X2(N), Y1(N), Y2(N), D(N)
      DO I = 1,N
         D1 = X1(I)-X2(I)
         D2 = Y1(I)-Y2(I)
         D(I) = SQRT(D1*D1 + D2*D2)
      END DO
      RETURN
      END
```

It translates to code which uses array syntax and promoted scalar arrays:

```
      SUBROUTINE EXAMPLE_ARRAY_CRUNCH(X1,X2,Y1,Y2,D,N)
      REAL X1(N), X2(N), Y1(N), Y2(N), D(N)
CMF$  LAYOUT D(:NEWS)
CMF$  LAYOUT X1(:NEWS)
CMF$  LAYOUT X2(:NEWS)
CMF$  LAYOUT Y1(:NEWS)
CMF$  LAYOUT Y2(:NEWS)
      REAL  D2000(N)
CMF$  LAYOUT D2000(:NEWS)
      REAL  D1000(N)
CMF$  LAYOUT D1000(:NEWS)
      D1000 = X1 - X2
      D2000 = Y1 - Y2
      D = SQRT(D1000 * D1000 + D2000 * D2000)
      RETURN
      END
```

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

This subroutine performs a smoothing operation using nearest−neighbor values:

```
      SUBROUTINE EXAMPLE_ARRAY_SMOOTH(IN, OUT, N)
      REAL IN(N,N), OUT(N,N)
      DO J = 2,N-1
         DO I = 2,N-1
            OUT(I,J) = 0.25*(IN(I-1,J-1) + IN(I-1,J+1) +
     .          IN(I+1,J-1) + IN(I+1,J+1))
         END DO
      END DO
      RETURN
      END
```

It translates to code which uses array sections:

```
      SUBROUTINE EXAMPLE_ARRAY_SMOOTH(IN,OUT,N)
      REAL IN(N,N), OUT(N,N)
CMF$  LAYOUT IN(:NEWS,:NEWS)
CMF$  LAYOUT OUT(:NEWS,:NEWS)
      OUT(2:N - 1,2:N - 1) =
     &    0.25 * (IN(:N - 2,:N - 2) + IN(:N - 2,3:)
     &    + IN(3:,:N - 2) + IN(3:,3:))
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.1.2 WHERE

This subroutine performs a simple conditional elementwise operation:

```
      SUBROUTINE EXAMPLE_WHERE_1(TEMP, N, DTEMP)
      REAL TEMP(N)
      DO I=1,N
         IF (TEMP(I) .GT. 98.6) TEMP(I) = TEMP(I) - DTEMP
      END DO
      RETURN
      END
```

It translates to code which uses a **WHERE** statement:

```
      SUBROUTINE EXAMPLE_WHERE_1(TEMP,N,DTEMP)
      REAL TEMP(N)
CMF$  LAYOUT TEMP(:NEWS)
      WHERE (TEMP .GT. 98.6) TEMP = TEMP - DTEMP
      RETURN
      END
```

This subroutine performs a slightly more complex conditional elementwise operation:

```
      SUBROUTINE EXAMPLE_WHERE_2
     &       (PRESSURE,TEMP,RAINING,N,DPRESSURE)
      REAL PRESSURE(N), TEMP(N)
      LOGICAL RAINING(N)
      DO I=1,N
         IF (PRESSURE(I) .LT. 1.0) THEN
            TEMP(I) = TEMP(I) - 5.0
         ELSE
            RAINING(I) = .TRUE.
         END IF
      END DO
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

It translates to code which uses a **WHERE** construct:

```
      SUBROUTINE EXAMPLE_WHERE_2
     &       (PRESSURE,TEMP,RAINING,N,DPRESSURE)
      REAL PRESSURE(N), TEMP(N)
      LOGICAL RAINING(N)
CMF$  LAYOUT PRESSURE(:NEWS)
CMF$  LAYOUT RAINING(:NEWS)
CMF$  LAYOUT TEMP(:NEWS)
      WHERE (PRESSURE .LT. 1.0)
      TEMP = TEMP - 5.0
      ELSEWHERE
      RAINING = .TRUE.
      ENDWHERE
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.1.3 FORALL

This subroutine computes APL's **IOTA** function:

```
      SUBROUTINE EXAMPLE_FORALL_1(A, N)
      INTEGER A(N)
      DO I = 1,N
         A(I) = I
      END DO
      RETURN
      END
```

It translates to code which uses a **FORALL**:

```
      SUBROUTINE EXAMPLE_FORALL_1(A,N)
      INTEGER A(N)
CMF$  LAYOUT A(:NEWS)
      FORALL (I = 1:N) A(I) = I
      RETURN
      END
```

This subroutine computes a two-dimensional mask which is true for the "red" squares of a checkerboard:

```
      SUBROUTINE EXAMPLE_FORALL_2(RED, N)
      LOGICAL RED(N,N)
      DO J = 1,N
         DO I = 1,N
            RED(I,J) = (MOD(I+J,2) .EQ. 0)
         END DO
      END DO
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_FORALL_2(RED,N)
      LOGICAL RED(N,N)
CMF$  LAYOUT RED(:NEWS,:NEWS)
      FORALL (J = 1:N, I = 1:N)
     &       RED(I,J) = (MOD(I + J,2) .EQ. 0)
      RETURN
      END
```

This subroutine multiplies a two-dimensional array with a one-dimensional array spread across its columns:

```
      SUBROUTINE EXAMPLE_FORALL_3(V, A, N)
      REAL V(N), A(N,N)
      DO J = 1,N
         DO I = 1,N
            A(I,J) = A(I,J) * V(I)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_FORALL_3(V,A,N)
      REAL V(N), A(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT V(:NEWS)
      FORALL (J = 1:N, I = 1:N) A(I,J) = A(I,J) * V(I)
      RETURN
      END
```

Copyright © 1994 Thinking Machines Corporation

This subroutine adds an array with the transpose of another array:

```
      SUBROUTINE EXAMPLE_FORALL_4(A, B, N)
      REAL A(N,N), B(N,N)
      DO J = 1,N
         DO I = 1,N
            A(I,J) = A(I,J) + B(J,I)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_FORALL_4(A,B,N)
      REAL A(N,N), B(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
      FORALL (J = 1:N, I = 1:N) A(I,J) = A(I,J) + B(J,I)
      RETURN
      END
```

This subroutine zeros the upper triangle of an array:

```
      SUBROUTINE EXAMPLE_FORALL_5(A, N)
      REAL A(N,N)
      DO J = 1,N
         DO I = 1,J-1
            A(I,J) = 0.0
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_FORALL_5(A,N)
      REAL A(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
      FORALL (J = 1:N, I = 1:N, I .LT. J) A(I,J) = 0.0
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

This subroutine does complicated indirect addressing:

```
      SUBROUTINE EXAMPLE_FORALL_6(A, B, I1, I2, N)
      REAL A(N,N), B(N,N)
      INTEGER I1(N,N), I2(N)
      DO J = 1,N
         DO I = 1,N
            A(I,J) = A(I,J) + B(I1(I,J),I2(J))
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_FORALL_6(A,B,I1,I2,N)
      REAL A(N,N), B(N,N)
      INTEGER I1(N,N), I2(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
CMF$  LAYOUT I1(:NEWS,:NEWS)
CMF$  LAYOUT I2(:NEWS)
      FORALL (J = 1:N, I = 1:N)
     &   A(I,J) = A(I,J) + B(I1(I,J),I2(J))
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.2 Fortran 90 and CM Fortran Intrinsic Funtions

For descriptions of these intrinsics, see the *CM Fortran Reference Manual*. These examples use one- and two-dimensional arrays; in general, multidimensional forms are recognized.

B.2.1 ALL

This subroutine computes the logical AND of all the elements in an array:

```
      SUBROUTINE EXAMPLE_ALL_1(L, FLAG, N)
      LOGICAL L(N,N), FLAG
      DO J=1,N
         DO I=1,N
            FLAG = FLAG .AND. L(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_ALL_2(L,FLAG,N)
      LOGICAL L(N,N), FLAG
CMF$  LAYOUT L(:NEWS,:NEWS)
      FLAG = FLAG .AND. ALL(L)
      RETURN
      END
```

This subroutine computes **ALL** over the second dimension of an array:

```
      SUBROUTINE EXAMPLE_ALL_2(L, FLAGS, N)
      LOGICAL L(N,N), FLAGS(N)
      DO J=1,N
         DO I = 1,N
            FLAGS(I) = FLAGS(I) .AND. L(I,J)
         END DO
      END DO
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

```
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_ALL_2(L,FLAGS,N)
      LOGICAL L(N,N), FLAGS(N)
CMF$  LAYOUT FLAGS(:NEWS)
CMF$  LAYOUT L(:NEWS,:NEWS)
      FLAGS = FLAGS .AND. ALL(L,DIM=2)
      RETURN
      END
```

Copyright © 1994 Thinking Machines Corporation

B.2.2 ANY

This subroutine computes the logical OR of all the elements in an array:

```
      SUBROUTINE EXAMPLE_ANY_1(L, FLAG, N)
      LOGICAL L(N,N), FLAG
      DO J=1,N
         DO I=1,N
            FLAG = FLAG .OR. L(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_ANY_1(L,FLAG,N)
      LOGICAL L(N,N), FLAG
CMF$  LAYOUT L(:NEWS,:NEWS)
      FLAG = FLAG .OR. ANY(L)
      RETURN
      END
```

This subroutine computes **ANY** over the first dimension of an array:

```
      SUBROUTINE EXAMPLE_ANY_2(L, FLAGS, N)
      LOGICAL L(N,N), FLAGS(N)
      DO J=1,N
         DO I = 1,N
            FLAGS(J) = FLAGS(J) .OR. L(I,J)
         END DO
      END DO
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_ANY_2(L,FLAGS,N)
      LOGICAL L(N,N), FLAGS(N)
CMF$  LAYOUT FLAGS(:NEWS)
CMF$  LAYOUT L(:NEWS,:NEWS)
      FLAGS = FLAGS .OR. ANY(L,DIM=1)
      RETURN
      END
```

Copyright © 1994 Thinking Machines Corporation

B.2.3 COUNT

This subroutine counts the number of zero elements in an array:

```
      SUBROUTINE EXAMPLE_COUNT_1(A, K, N)
      REAL A(N,N)
      DO J = 1,N
         DO I = 1,N
            IF (A(I,J) .EQ. 0.0) K = K + 1
         END DO
      END DO
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_COUNT_1(A,K,N)
      REAL A(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
      K = K + COUNT(A .EQ. 0.0)
      END
```

This subroutine computes **COUNT** over the second dimension of an array:

```
      SUBROUTINE EXAMPLE_COUNT_2(A, K, N)
      REAL A(N,N)
      INTEGER K(N)
      DO J=1,N
         DO I = 1,N
            IF (A(I,J) .EQ. 0.0) K(I) = K(I) + 1
         END DO
      END DO
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_COUNT_2(A,K,N)
      REAL A(N,N)
      INTEGER K(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT K(:NEWS)
      K = K + COUNT(A .EQ. 0.0,DIM=2)
      RETURN
      END
```

Copyright © 1994 Thinking Machines Corporation

B.2.4 DOTPRODUCT

This subroutine computes **DOTPRODUCT**:

```
      SUBROUTINE EXAMPLE_DOTPRODUCT(X, Y, N, A)
      REAL X(N), Y(N), A
      DO I = 1,N
         A = A + X(I) * Y(I)
      END DO
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_DOTPRODUCT(X,Y,N,A)
      REAL X(N), Y(N), A
CMF$  LAYOUT X(:NEWS)
CMF$  LAYOUT Y(:NEWS)
      A = A + DOTPRODUCT(X,Y)
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.2.5 MATMUL

These subroutines perform matrix multiplies:

```
SUBROUTINE EXAMPLE_MATMUL_1(A, B, C, N1, N2, N3)
REAL A(N1, N2), B(N2, N3), C(N1, N3)
DO J=1,N3
   DO I=1,N1
      C(I,J)=0.0
   END DO
END DO
DO K=1,N2
   DO J=1,N3
      DO I=1,N1
         C(I,J)=C(I,J)+A(I,K)*B(K,J)
      END DO
   END DO
END DO
RETURN
END


SUBROUTINE EXAMPLE_MATMUL_2(A, B, C, N1, N2, N3)
REAL A(N1, N2), B(N2, N3), C(N1, N3)
DO I=1,N1
   DO J=1,N3
      C(I,J)=0.0
      DO K=1,N2
         C(I,J)=C(I,J)+A(I,K)*B(K,J)
      END DO
   END DO
END DO
RETURN
END
```


Copyright © 1994 Thinking Machines Corporation

They translate to:

```
      SUBROUTINE EXAMPLE_MATMUL_1(A,B,C,N1,N2,N3)
      REAL A(N1, N2), B(N2, N3), C(N1, N3)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
CMF$  LAYOUT C(:NEWS,:NEWS)
      C = 0.0
      C = C + MATMUL(A,B)
      RETURN
      END


      SUBROUTINE EXAMPLE_MATMUL_2(A,B,C,N1,N2,N3)
      REAL A(N1, N2), B(N2, N3), C(N1, N3)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
CMF$  LAYOUT C(:NEWS,:NEWS)
      C = MATMUL(A,B)
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.2.6 MAXLOC

This subroutine finds the maximum element of an array and its location:

```
      SUBROUTINE MAXLOC_EXAMPLE_1(A, AMI, AMJ, AMAX, N1, N2)
      REAL A(N1,N2), AMAX
      INTEGER AMI, AMJ
      DO J=1,N2
         DO I=1,N1
            IF (A(I,J) .GT. AMAX) THEN
               AMAX = A(I,J)
               AMI = I
               AMJ = J
            END IF
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE MAXLOC_EXAMPLE_1(A,AMI,AMJ,AMAX,N1,N2)
      REAL A(N1,N2), AMAX
      INTEGER AMI, AMJ
CMF$  LAYOUT A(:NEWS,:NEWS)
      INTEGER IJ(2)
CMF$  LAYOUT IJ(:NEWS)
      IJ = MAXLOC(A)
      IF (IJ(1) .GT. 0 .AND. IJ(2) .GT. 0
     &    .AND. A(IJ(1),IJ(2)) .GT. AMAX) THEN
         AMAX = A(IJ(1),IJ(2))
         AMI = IJ(1)
         AMJ = IJ(2)
      ENDIF
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

This subroutine finds the maximum element of an array and its location subject to a mask:

```
      SUBROUTINE MAXLOC_EXAMPLE_2(A,MASK,AMI,AMJ,AMAX,N1,N2)
      REAL A(N1,N2), AMAX
      LOGICAL MASK(N1,N2)
      INTEGER AMI, AMJ
      DO J=1,N2
         DO I=1,N1
            IF (MASK(I,J).AND.(A(I,J).GT.AMAX)) THEN
               AMAX = A(I,J)
               AMI = I
               AMJ = J
            END IF
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE MAXLOC_EXAMPLE_2(A,MASK,AMI,AMJ,AMAX,N1,N2)
      REAL A(N1,N2), AMAX
      LOGICAL MASK(N1,N2)
      INTEGER AMI, AMJ
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT MASK(:NEWS,:NEWS)
      INTEGER IJ(2)
CMF$  LAYOUT IJ(:NEWS)
      IJ = MAXLOC(A,MASK=MASK)
      IF (IJ(1) .GT. 0 .AND. IJ(2) .GT. 0
     &    .AND. A(IJ(1),IJ(2)) .GT. AMAX) THEN
         AMAX = A(IJ(1),IJ(2))
         AMI = IJ(1)
         AMJ = IJ(2)
      ENDIF
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.2.7 MAXVAL

This subroutine finds the maximum value of an array:

```
      SUBROUTINE EXAMPLE_MAXVAL_1(A, X, N)
      REAL A(N,N), X
      DO J=1,N
         DO I=1,N
            X = MAX(X,A(I,J))
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_MAXVAL_1(A,X,N)
      REAL A(N,N), X
CMF$  LAYOUT A(:NEWS,:NEWS)
      X = MAX(X,MAXVAL(A))
      RETURN
      END
```

This subroutine finds the maximum values along the first dimension of an array:

```
      SUBROUTINE EXAMPLE_MAXVAL_2(A, X, N)
      REAL A(N,N), X(N)
      DO J=1,N
         DO I = 1,N
            X(J) = MAX(X(J),A(I,J))
         END DO
      END DO
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_MAXVAL_2(A,X,N)
      REAL A(N,N), X(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT X(:NEWS)
      X = MAX(X,MAXVAL(A,DIM=1))
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.2.8 MINLOC

This subroutine finds the minimum element of an array and its location:

```
      SUBROUTINE MINLOC_EXAMPLE_1(A, AMI, AMJ, AMIN, N1, N2)
      REAL A(N1,N2), AMIN
      INTEGER AMI, AMJ
      DO J=1,N2
         DO I=1,N1
            IF (A(I,J) .LT. AMIN) THEN
               AMIN = A(I,J)
               AMI = I
               AMJ = J
            END IF
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE MINLOC_EXAMPLE_1(A,AMI,AMJ,AMIN,N1,N2)
      REAL A(N1,N2), AMIN
      INTEGER AMI, AMJ
CMF$  LAYOUT A(:NEWS,:NEWS)
      INTEGER IJ(2)
CMF$  LAYOUT IJ(:NEWS)
      IJ = MINLOC(A)
      IF (IJ(1) .GT. 0 .AND. IJ(2) .GT. 0
     &    .AND. A(IJ(1),IJ(2)) .LT. AMIN) THEN
         AMIN = A(IJ(1),IJ(2))
         AMI = IJ(1)
         AMJ = IJ(2)
      ENDIF
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

This subroutine finds the minimum element of an array and its location subject to a mask:

```
      SUBROUTINE MINLOC_EXAMPLE_2(A,MASK,AMI,AMJ,AMIN,N1,N2)
      REAL A(N1,N2), AMIN
      LOGICAL MASK(N1,N2)
      INTEGER AMI, AMJ
      DO J=1,N2
         DO I=1,N1
            IF (MASK(I,J).AND.(A(I,J).LT.AMIN)) THEN
               AMIN = A(I,J)
               AMI = I
               AMJ = J
            END IF
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE MINLOC_EXAMPLE_2(A,MASK,AMI,AMJ,AMIN,N1,N2)
      REAL A(N1,N2), AMIN
      LOGICAL MASK(N1,N2)
      INTEGER AMI, AMJ
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT MASK(:NEWS,:NEWS)
      INTEGER IJ(2)
CMF$  LAYOUT IJ(:NEWS)
      IJ = MINLOC(A,MASK=MASK)
      IF (IJ(1) .GT. 0 .AND. IJ(2) .GT. 0
     &   .AND. A(IJ(1),IJ(2)) .LT. AMIN) THEN
         AMIN = A(IJ(1),IJ(2))
         AMI = IJ(1)
         AMJ = IJ(2)
      ENDIF
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.2.9 MINVAL

This subroutine finds the minimum value of an array:

```
      SUBROUTINE EXAMPLE_MINVAL_1(A, X, N)
      REAL A(N,N), X
      DO J=1,N
         DO I=1,N
            X = MIN(X,A(I,J))
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_MINVAL_1(A,X,N)
      REAL A(N,N), X
CMF$  LAYOUT A(:NEWS,:NEWS)
      X = MIN(X,MINVAL(A))
      RETURN
      END
```

This subroutine finds the minimum values along the second dimension of an array:

```
      SUBROUTINE EXAMPLE_MINVAL_2(A, X, N)
      REAL A(N,N), X(N)
      DO J=1,N
         DO I = 1,N
            X(I) = MIN(X(I),A(I,J))
         END DO
      END DO
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_MINVAL_2(A,X,N)
      REAL A(N,N), X(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT X(:NEWS)
      X = MIN(X,MINVAL(A,DIM=2))
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.2.10 PRODUCT

This subroutine computes the product of all the elements in an array:

```
      SUBROUTINE EXAMPLE_PRODUCT_1(A, X, N)
      REAL A(N,N), X
      DO J=1,N
         DO I=1,N
            X = X * A(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_PRODUCT_1(A,X,N)
      REAL A(N,N), X
CMF$  LAYOUT A(:NEWS,:NEWS)
      X = X * PRODUCT(A)
      RETURN
      END
```

This subroutine computes the product of elements along the first dimension of an array:

```
      SUBROUTINE EXAMPLE_PRODUCT_2(A, X, N)
      REAL A(N,N), X(N)
      DO J=1,N
         DO I = 1,N
            X(J) = X(J) * A(I,J)
         END DO
      END DO
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_PRODUCT_2(A,X,N)
      REAL A(N,N), X(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT X(:NEWS)
      X = X * PRODUCT(A,DIM=1)
      RETURN
      END
```

This subroutine computes the product of all the positive elements of an array:

```
      SUBROUTINE EXAMPLE_PRODUCT_3(A, X, N)
      REAL A(N,N), X
      DO J=1,N
         DO I=1,N
            IF (A(I,J) .GT. 0.0) X = X * A(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_PRODUCT_3(A,X,N)
      REAL A(N,N), X
CMF$  LAYOUT A(:NEWS,:NEWS)
      X = X * PRODUCT(A,MASK=A .GT. 0.0)
      RETURN
      END
```

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.2.11 SUM

This subroutine computes the sum of all the elements in an array:

```
      SUBROUTINE EXAMPLE_SUM_1(A, X, N)
      REAL A(N,N), X
      DO J=1,N
         DO I=1,N
            X = X + A(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_SUM_1(A,X,N)
      REAL A(N,N), X
CMF$  LAYOUT A(:NEWS,:NEWS)
      X = X + SUM(A)
      RETURN
      END
```

This subroutine computes the sum of elements along the second dimension of an array:

```
      SUBROUTINE EXAMPLE_SUM_2(A, X, N)
      REAL A(N,N), X(N)
      DO J=1,N
         DO I = 1,N
            X(I) = X(I) + A(I,J)
         END DO
      END DO
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

It translates to:

```
      SUBROUTINE EXAMPLE_SUM_2(A,X,N)
      REAL A(N,N), X(N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT X(:NEWS)
      X = X + SUM(A,DIM=2)
      RETURN
      END
```

This subroutine computes the sum of all the elements of an array under a mask:

```
      SUBROUTINE EXAMPLE_SUM_3(A, MASK, X, N)
      REAL A(N,N), X
      LOGICAL MASK(N,N)
      DO J=1,N
         DO I=1,N
            IF (MASK(I,J)) X = X + A(I,J)
         END DO
      END DO
      RETURN
      END
```

It translates to:

```
      SUBROUTINE EXAMPLE_SUM_3(A,MASK,X,N)
      REAL A(N,N), X
      LOGICAL MASK(N,N)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT MASK(:NEWS,:NEWS)
      X = X + SUM(A,MASK=MASK)
      RETURN
      END
```

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.3 CM Fortran Utility Library Subroutines

For descriptions of these subroutines, see the *CM Fortran Utility Library Reference Manual*. These examples use one- and two-dimensional arrays; in general, multidimensional forms are recognized.

B.3.1 CMF_SCAN_

This upward, inclusive add scan:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_1(A,B,N)
      REAL A(N), B(N)
      LAST = 0
      DO I=1,N
         LAST = LAST + A(I)
         B(I) = LAST
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_1(A,B,N)
      REAL A(N), B(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT B(:NEWS)
      INCLUDE '/usr/include/cm/CMF_defs.h'
      LAST = 0
      CALL CMF_SCAN_ADD(B,A,CMF_NULL,1,CMF_UPWARD,
     &       CMF_INCLUSIVE,CMF_NONE,.TRUE.)
      RETURN
      END
```

Copyright © 1994 Thinking Machines Corporation

This downward, inclusive add scan:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_2(A,B,N)
      REAL A(N), B(N)
      LAST = 0
      DO I=N,1,-1
         LAST = LAST + A(I)
         B(I) = LAST
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_2(A,B,N)
      REAL A(N), B(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT B(:NEWS)
      INCLUDE '/usr/include/cm/CMF_defs.h'
      LAST = 0
      CALL CMF_SCAN_ADD(B,A,CMF_NULL,1,CMF_DOWNWARD,
     &       CMF_INCLUSIVE,CMF_NONE,.TRUE.)
      RETURN
      END
```

This upward, exclusive add scan:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_3(A,B,N)
      REAL A(N), B(N)
      LAST = 0
      DO I=1,N
         B(I) = LAST
         LAST = LAST + A(I)
      END DO
      RETURN
      END
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

translates to:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_3(A,B,N)
      REAL A(N), B(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT B(:NEWS)
      INCLUDE '/usr/include/cm/CMF_defs.h'
      LAST = 0
      CALL CMF_SCAN_ADD(B,A,CMF_NULL,1,CMF_UPWARD,
     &       CMF_EXCLUSIVE,CMF_NONE,.TRUE.)
      RETURN
      END
```

This upward, inclusive, segmented add scan:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_4(A,B,SEGMENT,N)
      REAL A(N), B(N)
      LOGICAL SEGMENT(N)
      LAST = 0
      DO I=1,N
         IF (SEGMENT(I)) LAST = 0
         LAST = LAST + A(I)
         B(I) = LAST
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_4(A,B,SEGMENT,N)
      REAL A(N), B(N)
      LOGICAL SEGMENT(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT B(:NEWS)
CMF$  LAYOUT SEGMENT(:NEWS)
      INCLUDE '/usr/include/cm/CMF_defs.h'
      LAST = 0
      CALL CMF_SCAN_ADD(B,A,SEGMENT,1,CMF_UPWARD,
     &       CMF_INCLUSIVE,CMF_SEGMENT_BIT,.TRUE.)
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

This masked, upward, inclusive, segmented add scan:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_5(A,B,SEGMENT,MASK,N)
      REAL A(N), B(N)
      LOGICAL SEGMENT(N), MASK(N)
      LAST = 0
      DO I=1,N
         IF (SEGMENT(I)) LAST = 0
         IF (MASK(I)) THEN
            LAST = LAST + A(I)
            B(I) = LAST
         END IF
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_SCAN_ADD_5(A,B,SEGMENT,MASK,N)
      REAL A(N), B(N)
      LOGICAL SEGMENT(N), MASK(N)
CMF$  LAYOUT A(:NEWS)
CMF$  LAYOUT B(:NEWS)
CMF$  LAYOUT MASK(:NEWS)
CMF$  LAYOUT SEGMENT(:NEWS)
      INCLUDE '/usr/include/cm/CMF_defs.h'
      LAST = 0
      CALL CMF_SCAN_ADD(B,A,SEGMENT,1,CMF_UPWARD,
     &       CMF_INCLUSIVE,CMF_SEGMENT_BIT,MASK)
      RETURN
      END
```

This upward, inclusive, segmented copy scan:

```
      SUBROUTINE EXAMPLE_SCAN_COPY_1(A,B,SEGMENT,N)
      REAL A(N), B(N)
      LOGICAL SEGMENT(N)
      LAST = A(1)
      DO I=1,N
         IF (SEGMENT(I)) LAST = A(I)
         B(I) = LAST
      END DO
```


Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

```
      RETURN
      END
```

translates to:

```
      SUBROUTINE  EXAMPLE_SCAN_COPY_1(A,B,SEGMENT,N)
      REAL  A(N),  B(N)
      LOGICAL  SEGMENT(N)
CMF$  LAYOUT  A(:NEWS)
CMF$  LAYOUT  B(:NEWS)
CMF$  LAYOUT  SEGMENT(:NEWS)
      INCLUDE  '/usr/include/cm/CMF_defs.h'
      LAST  =  A(1)
      CALL  CMF_SCAN_COPY(B,A,SEGMENT,1,CMF_UPWARD,
     &       CMF_INCLUSIVE,CMF_SEGMENT_BIT,.TRUE.)
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

B.3.2 CMF_SEND_

This subroutine, which is in essence a combining send:

```
      SUBROUTINE EXAMPLE_SEND_ADD_1(A, B, V1, V2, N1, N2)
      REAL A(N1,N2), B(N1,N2)
      INTEGER V1(N1,N2), V2(N1,N2)
      DO I=1,N1
         DO J=1,N2
            IF (B(I,J).GT.0.0) THEN
               A(V1(I,J),V2(I,J)) = A(V1(I,J),V2(I,J)) + B(I,J)
            END IF
         END DO
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_SEND_ADD_1(A,B,V1,V2,N1,N2)
      REAL A(N1,N2), B(N1,N2)
      INTEGER V1(N1,N2), V2(N1,N2)
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B(:NEWS,:NEWS)
CMF$  LAYOUT V1(:NEWS,:NEWS)
CMF$  LAYOUT V2(:NEWS,:NEWS)
      INTEGER GRIDTEMP(N1,N2)
CMF$  LAYOUT GRIDTEMP(:NEWS,:NEWS)
      CALL CMAX_GRID_TO_SEND_ADDR(GRIDTEMP,A,V1,V2)
      CALL CMF_SEND_ADD(A,GRIDTEMP,B,B .GT. 0.0)
      RETURN
      END
```

NOTE: This output must be linked with the CMAX library, since it constructs send addresses using **CMAX_GRID_TO_SEND_ADDR** (a CMAX library routine) rather than the CM Fortran Utility Library routines.

Version 2.0, May 1994
Copyright © 1994 Thinking Machines Corporation

B.3.3 CMF_FE_ARRAY_TO_CM and CMF_FE_ARRAY_FROM_CM

This subroutine, which transfers one array from the serial control processor to the parallel processors and one array in the other direction:

```
      SUBROUTINE EXAMPLE_TRANSFERS(CM_A, FE_A, CM_B, FE_B, N)
      REAL CM_A(N,N,N), FE_A(N,N,N)
      REAL CM_B(N,N,N), FE_B(N,N,N)
CMF$  LAYOUT CM_A(:NEWS,:NEWS,:NEWS)
CMF$  LAYOUT FE_A(:SERIAL,:SERIAL,:SERIAL)
CMF$  LAYOUT CM_B(:NEWS,:NEWS,:NEWS)
CMF$  LAYOUT FE_B(:SERIAL,:SERIAL,:SERIAL)
      DO K = 1,N
         DO J = 1,N
            DO I = 1,N
               CM_A(I,J,K) = FE_A(I,J,K)
               FE_B(I,J,K) = CM_B(I,J,K)
            END DO
         END DO
      END DO
      RETURN
      END
```

translates to:

```
      SUBROUTINE EXAMPLE_TRANSFERS(CM_A,FE_A,CM_B,FE_B,N)
      REAL CM_A(N,N,N), FE_A(N,N,N)
      REAL CM_B(N,N,N), FE_B(N,N,N)
CMF$  LAYOUT CM_A(:NEWS,:NEWS,:NEWS)
CMF$  LAYOUT FE_A(:SERIAL,:SERIAL,:SERIAL)
CMF$  LAYOUT CM_B(:NEWS,:NEWS,:NEWS)
CMF$  LAYOUT FE_B(:SERIAL,:SERIAL,:SERIAL)
      CALL CMF_FE_ARRAY_TO_CM(CM_A,FE_A)
      CALL CMF_FE_ARRAY_FROM_CM(FE_B,CM_B)
      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation