U.S. DEPARTMENT OF COMMERCE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
NATIONAL WEATHER SERVICE
NATIONAL METEOROLOGICAL CENTER

OFFICE NOTE 347

SUGGESTIONS FOR WRITING ANSI-STANDARD FORTRAN ON THE CYBER 205

James J. Tuccillo
Automation Division

January 9, 1989

SUGGESTIONS FOR WRITING ANSI-STANDARD FORTRAN ON THE CYBER 205

J. Tuccillo
Dec 1988

I. INTRODUCTION

With the maturation of the CDC FORTRAN compiler and the availability of the VAST-2 FORTRAN preprocessor, it is now possible to write ANSI-standard FORTRAN for the CYBER 205 that is as efficient as hand vectorized code. The compiler will automatically vectorize many constructs which we currently write in semicolon or vector function form. Certain constructs which are not vectorized by the CDC compiler will be converted to vectorized code by the VAST-2 preprocessor. VAST-2 is a FORTRAN-in/FORTRAN-out code preprocessor used prior to compilation. For the advantages of readability, portability, and maintainability, it behooves us to write new code for the CYBER in the ANSI standard and use the automatic vectorizer and VAST-2 preprocessor to achieve vectorization. This assumes, of course, that the data structures and algorithms of the program have been designed for vectorization. This document will address the topic of writing ANSI-standard code as opposed to using CDC non-ANSI extensions.

There may always be situations where hand vectorization is necessary. These situations will, in general, represent a small percentage of the code. The non-ANSI standard code can usually be confined to small, easily identified subroutines and functions so that the vast majority of the code is in the ANSI standard. This is a widely accepted practice in software engineering.

The intended audience for this paper is CYBER 205 users, therefore, a general familiarity with the 205 is assumed.

II. EXAMPLES

In this section several code segments which are common to many NMC codes will be presented. The CDC compiler transparently vectorizes many of these constructs. The VAST-2 preprocessor is needed on others. VAST-2 generates FORTRAN with CDC extensions from FORTRAN. The processed FORTRAN will have semicolon notation and CDC FORTRAN vector functions and will look similar to the hand vectorization present in most NMC codes. Depending on your code, VAST-2 may or may not be needed. If it is used prior to compilation it performs the vectorization analysis which the CDC compiler would normally perform.

II.1 SIMPLE DO LOOPS

The most common construct in our CYBER codes, which we have traditionally hand vectorized, is the simple DO LOOP shown below. In this case simple means a loop with no conditional testing.

```
      DO 1 I = 1, LEN
         A(I) = A(I) + B(I)
    1 CONTINUE
```

This is usually hand vectorized as follows:

```
      A ( 1; LEN ) = A ( 1; LEN ) + B ( 1; LEN )
```

The hand vectorization is not needed. The DO LOOP will be automatically vectorized by the compiler if the automatic vectorization compiler option is used. If the value of LEN is not known at compile time then the UNSAFE vectorization option must be included. Automatic strip-mining of the loop will be performed if the vector length is greater than 65535. Strip-mining is the process by which a loop is processed, transparently to the programmer, in several steps so as not to exceed the loop iterate limit of 65535 on the CYBER.

VAST-2 will convert the DO LOOP to semicolon form with code to perform strip-mining if LEN is greater than 65535 or if LEN is not known at compile time.

In the hand vectorized code, the value of LEN must be less than 65536. If the compiler does not know the value of LEN at compile time it assumes you have taken the proper steps to insure that it is less than 65536. If it is not then the results may be incorrect because the actual vector length will be modulo 65535. If the compiler knows the value of LEN and it is greater than 65535 it will issue a compiler error. It will not stripmine code written in semicolon form.

Clearly it is safer and more general to let either the automatic vectorizer or the VAST-2 preprocessor perform vectorization of simple loops. The loops can, of course, be more complex as long as the basic requirements of vectorization are met. (see the CDC FORTRAN manual for precise requirements for vectorization).

II.2 INDIRECT ADDRESSING

Indirect addressing is also automatically vectorized by the CDC compiler. An example is shown below.

```
      DO 1 I = 1, LEN
         A ( I ) = B ( INDEX ( I ) )
    1 CONTINUE
```

where INDEX is an integer array which has been previously assigned. We normally use the GATHER instruction on the CYBER as follows:

```
A ( 1; LEN ) = Q8VGATHR ( B ( 1; LEN ),
                          INDEX ( 1; LEN );
                          A ( 1; LEN ) )
```

Indirect addressing on the left hand side, shown below, is also automatically vectorized.

```
      DO 1 I = 1, LEN
         A ( INDEX ( I ) ) = B ( I )
    1 CONTINUE
```

The use of the SCATTER instruction is the normal procedure for hand vectorization as shown below.

```
A ( 1; LEN ) = Q8VSCATR ( B ( 1; LEN ),
                          INDEX ( 1; LEN );
                          A ( 1; LEN ) )
```

VAST-2 will convert the DO LOOPs to GATHERs and SCATTERs with strip-mining, if necessary.

II.3 VECTORIZED IF-THEN-ELSE

The CDC FORTRAN compiler will not, at this time, vectorize IF-THEN-ELSE constructs, however, the VAST-2 preprocessor will. An example is given below.

```
      DO 1 I = 1, LEN
         IF ( A ( I ) .LT. 10.0 ) THEN
            A ( I ) = 10.0
         ELSE
            A ( I ) = A ( I ) * A ( I )
         END IF
    1 CONTINUE
```

The normal hand-vectorized code would be as follows:

```
      WHERE ( A ( 1; LEN ) .LT. 10.0 )
            A ( 1; LEN ) = 10.0
      OTHERWISE
            A ( 1; LEN ) = A ( 1; LEN ) * A ( 1; LEN )
      END WHERE
```

VAST-2 will convert the IF-THEN-ELSE structure directly to the WHERE-OTHERWISE structure.

II.4 COMPRESSION AND DECOMPRESSION

Often a series of calculations need to be done on a subset of an array. There are two procedures for handling this. The first procedure, represented by II.3, is to perform the calculations on the entire array and store the results according to a conditional test. This is the so-called WHERE block or bit-vector controlled store. The second procedure is to compress out the subset of points into a smaller contiguous array, perform the needed calculations, and then decompress the data back into the desired array. The first procedure is generally preferred when most of the array requires the calculations or the number of calculations required on the subset of the array is small and the overhead of compression/decompression is greater than the computational work on the subset. The second procedure is preferred when many calculations are to be performed on a small percentage of the array. Compression/decompression will be automatically vectorized by VAST-2. The intermediate arrays will be allocated from dynamic space and will be transparent to the programmer. The only requirements are the inclusion of a VAST-2 directive (they begin with 'C#') before and after the loop and use of the SC compiler option. An example follows.

```
C
C#ASSERT USE(CMPRS XPND)
C
      DO 1 I = 1, LEN
         IF ( A ( I ) .GT. 0.0 ) THEN
C
C           computational work
C
         END IF
    1 CONTINUE
C#ASSERT USE(CONTROL BITS)
```

II.5 32 - BIT CONSTANTS

The CDC compiler requires that 32-bit constants be identified as such using the following form:

nSx

where n is a string of digits
and x is a signed integer.

This is analogous to the 'E' notation normally used for scientific notation. An example is as follows:

R = 2.8704 S + 2

If this form is not followed then the arithmetic may be 64-bit which would defeat one of the two reasons for using 32-bit; an increase in execution speed. The above form is not ANSI standard. The declaration of the variable R would be as follows:

HALF PRECISION R

or the more general form for an entire program section

IMPLICIT HALF PRECISION (A-H, O-Z).

Neither of these statements is ANSI-STANDARD and there is no mechanism for having a 32-bit code without including them. The 'S' notation for specifying the value of a constant can be avoided, however, by setting the value with a PARAMETER statement. The PARAMETER statement is ANSI-standard. Any value in a PARAMETER statement will be typed by its declaration and all occurrences of the non-ANSI standard 'S' notation can be avoided. An example follows:

```
IMPLICIT HALF PRECISION ( A-H, O-Z )

PARAMETER ( R = 2.8704 E + 2 )
```

R will be typed as 32-bit according to the IMPLICIT statement and will represent a 32-bit constant. If all numeric values are handled in this manner then a code can be converted to 32-bit on the CYBER simply by including the IMPLICIT statement using an 'include' facility. This will result in a completely ANSI-standard code.

II.6 CONTROLLED STORE WITH PRESET BIT VECTORS

One common operation in the NMC codes is a controlled store with a bit vector that does not changes. The bit vector is often setup at the beginning of the execution and is not reset. Since a bit vector is not ANSI standard we will need to create the mask with a real or integer array and allow the compiler to generate the controlled store. The overhead will be the repeated creation of a bit vector (transparently to the programmer). This overhead should be small, however, since many vector operations are usually done prior to the controlled store. For example, if 50 arithmetic operations are done prior to the controlled store the overhead would be 2%. An integer or real array for the mask will require more storage than a bit vector, however, the total increase for the program will be generally small. VAST-2 will be required to vectorize the following code sample.

```
C
C     CREATE INTEGER ARRAY FOR MASK ( CONTROLLED STORE )
C
      DIMENSION IMASK ( LEN )
C
C     CREATE MASK PATTERN
C     MASK = 1 FOR STORE, = 0 FOR NO STORE
C
C     CALCULATION OF B
C
      DO 1 I = 1, LEN
C
C       code to compute B
C
    1 CONTINUE
C
C     STORE B IN A UNDER CONTROL OF MASK
C
      DO 1 I = 1, LEN
         IF ( MASK(I) .EQ. 1 ) THEN
            A(I) = B(I)
         END IF
    1 CONTINUE
```

II.7 FORTRAN INTRINSIC FUNCTIONS

When using FORTRAN supplied intrinsic functions and 32-bit arithmetic you should code using the generic names as opposed to the specific 32-bit names. The FORTRAN compiler will choose the correct function based on the typing of the arguments. Recall that the 32-bit specific intrinsic functions are prefaced with an 'H' and therefore are non-ANSI standard while the generic names are ANSI-standard.

II.8 LINKED TRIADS

Linked triads are a combination of two vectors and a scalar or two scalars and a vector joined by multiplication, addition, or subtraction. This instruction executes at the same rate as a vector multiply (after a slightly longer startup) but produces twice the work. Needless to say you should strive for linked triads in your code. You can help the compiler generate linked triads by factoring your equations into combinations of vectors and scalars as indicated below.

```
      DO 1 I = 1, LEN
         A ( I ) = R * ( B ( I ) - B ( I + NX ) ) +
                   ( CP + R * C ( I ) )
    1 CONTINUE
```

The hand vectorized equivalent in triadic and diadic form follows.

```
      TEMP1( 1; LEN ) = R * (B ( 1; LEN ) - B ( NX+1; LEN ))
      A ( 1; LEN ) = CP + R * C ( 1; LEN )
      A ( 1; LEN ) = A ( 1; LEN ) + TEMP1 ( 1; LEN )
```

Both code segements will results in two linked triads and one vector add. The intermediate results from the DO LOOP segment will be stored in dynamic space. There is no need for hand vectorized triadic and diadic code. The compiler is very good at identifying linked triads and will generally minimize the amount of temporary storage in dynamic space.

On the ETA10 architecture there may be a disadvantage to coding in diads and triads. The architecture will shortstop intermediate vector calculations and produce faster execution. It may not identify the opportunities for short stopping if the code is in triadic and diadic form.

II.9 REDUCTION FUNCTIONS (from ETA VAST-2 Manual)

A reduction function is an operation that condenses array operands into one scalar value. VAST-2 will vectorize the following operations (presented with the Q8 equivalent):

OPERATION	Q8 EQUIVALENT
S = S + A (I)	Q8SSUM
S = S * A (I)	Q8SPROD
S = AMAX1(S,A(I))	Q8SMAX
S = AMIN1(S,A(I))	Q8SMIN
S = S + A(I) * B(I)	Q8SDOT
index of maximum element	Q8SMAXI
index of minimum element	Q8SMINI
IF (L(I)) N = N + 1	Q8SCNT

III. SUMMARY

Techniques for writing vectorizable ANSI FORTRAN versions of the most common hand vectorized code segments in NMC programs have been presented. Once a code has been properly designed for vectorization (appropriate data structures and algorithms) it can be implemented primarily in ANSI FORTRAN if the VAST-2 preprocessor is applied and the appropriate CDC FORTRAN compiler options are used. ANSI FORTRAN is more readable, portable and maintainable than hand vectorized code. Some operations may need hand vectorization. These code segments will generally be small and should be confined to easily identified routines.

The reader is encouraged to read the ETA VAST-2 manual prior to using it. There are many more examples of code segments than presented here as well as detailed instructions for using the various options of VAST-2.