NOV 17 1967

OAK RIDGE NATIONAL LABORATORY
operated by
UNION CARBIDE CORPORATION
NUCLEAR DIVISION
for the
U.S. ATOMIC ENERGY COMMISSION

MASTER

ORNL - TM - 2007

COPY NO. - 10

DATE - October 25, 1967

Removal of Constant Terms from DO Loops

J. G. Sullivan

Abstract

Techniques are described which can cause significant decreases in the execution time of Fortran programs. Examples are given which illustrate the techniques.

NOTICE This document contains information of a preliminary nature and was prepared primarily for internal use at the Oak Ridge National Laboratory. It is subject to revision or correction and therefore does not represent a final report.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

- A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or
- B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Removal of Constant Terms from DO Loops

The Fortran DO-loop is a convenient facility made available to the programmer in order to simplify repetitive processes.

Consider the following coding:

Example 1

```
DO 1 I=1,1000
  1 Y(I)=X(I)*SQRT(3.14159)/2.
```

This is perfectly legal and proper coding but what is being performed on the computer is a matter strictly between the Fortran compiler and statement 1. Some compilers (very few in number) would discover that there are two constants involved, the $\sqrt{\pi}$ and 2., and remove them to the outside of the DO loop. Other compilers might change the division by 2. to a multiplication by .5 (division takes more time on most computers than multiplication†). The majority of compilers would perform the statement as it exists so that 90% (at a guess) of the time used in performing the DO loop would be spent on the calculation of the $\sqrt{\pi}/2$.

Rewrite the code as follows:

Example 1a

```
RTPIO2=SQRT(3.14159)*.5
DO 1 I=1,1000
  1 Y(I)=X(I)*RTPIO2
```

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

The only loss is that a certain visual reference to what is being calculated has disappeared or, at least, been hidden to some extent. So, if this is a great loss, do the following:

†On the IBM 360/75, division averages 3.9 μ sec and multiplication averages 2.1 μ sec.

Example 1b

```

      RTPIO2=SQRT(3.14159)*.5
      DO 1 I=1,1000
C      *** Y(I) IS X(I) * SQRT(PI)/2.
      1 Y(I)=X(I)*RTPIO2

```

If this really sizable gain in time is not reason enough for using "good" coding, add a few more similar statements and then you are wrong not to worry about the machine time involved:

Example 2

```

      DO 1 I=1,1000
      DO 1 J=1,1000
      Y(I,J)=X(I,J)*SQRT(3.14159)/2.
      1 X(I,J)=X(I,J)*SQRT(2.*3.14159)/2.

```

Consider this:

Example 3

```

      DO 1 I=1,1000
      DO 1 J=1,1000
      1 X(I,J)=X(I,J)*W(I)/2.+X(I,J)

```

Example 3a is a better version of Example 3:

Example 3a

```

      DO 1 I=1,1000
      WIOV2=W(I)*.5
      DO 1 J=1,1000
      1 X(I,J)=X(I,J)*WIOV2+X(I,J)

```

Example 3a could also be better written as:

Example 3b

```
DO 1 I=1,1000
WIOV2=W(I)*.5+1.
DO 1 J=1,1000
1 X(I,J)=X(I,J)*WIOV2
```

Some coding practices recur quite often. One point is that a DO-loop such as:

Example 4

```
DO 1 I=1,10
X=(I-1)*DX
etc.
```

requires a conversion from the integer form of I to the floating point form before the multiplication by DX can be performed. Conversion requires tests for the sign of I, the storing of I into a special sort of machine number (a floating point number having a fractional part of zero and a special exponent) and the normalization of this number. Nowadays we can write "mixed" expressions but the computation which occurs is no different than:

```
X=FLOAT(I-1)*DX
```

The simple way to avoid the time spent for the conversion is

Example 4a

```

      EXTRA=0.

      DO 1 I=1,10

      X=EXTRA*DX

      etc.

1  EXTRA=EXTRA+1.

```

EXTRA now runs from 0. to 9. as I runs from 1 to 10. But note that even here the multiplication by DX could be completely avoided:

Example 4b

```

      X=0.

      DO 1 I=1,10

      etc.

1  X=X+DX

```

Another practice frequently seen is:

Example 5

```

      DO 1 I=1,4

      Y = X**I

      etc.

1  CONTINUE

```

First of all an appearance of an expression of the form:

```

      X**1

```


causes a library function to be executed which has to check on the sign and value of both X and I and then performs as few multiplications as possible to calculate X^I . In the loop above the function would be invoked four times for X^1, X^2, X^3, X^4 , or 0, 1, 2, 2, multiplications respectively (5 in all for the loop) plus all the checking which must take place each time. The time needed for the checking is probably equivalent to the time required for three multiplications so that we have forced about $5 + 4 \times 3 = 17$ multiplication operations by writing the loop this way. Another way is available:

Example 5a

```

      EXTRA=X
      DO 1 I=1,4
      Y=EXTRA
      etc.
1  EXTRA=EXTRA*X

```

which would cause only four multiplications rather than 17. Multiplication time is relatively slow compared to addition time so we might even gain by the following, which saves one multiplication:

Example 5b

```

      EXTRA(1)=X
      EXTRA(2)=X*X
      EXTRA(3)=EXTRA(2)*X
      EXTRA(4)=EXTRA(2)*EXTRA(2)
      DO 1 I=1,4
      Y=EXTRA(I)
      etc.
1  CONTINUE

```

An expression such as $X^{**}\underline{2}$ is quite different from $X^{**}\underline{2.}$ because the exponents are of differing types. In the first case many compilers check the exponents and if it is a small integer (less than 6, or less than 4 for example) may generate "in-line" coding to perform the multiplication. In the second case the number, 2., is not checked for having an exact integer value and the expression is treated just as $X^{**}A$ would be. That is, $X^{**}A = e^{A \ln X}$ and every expression which causes exponentiation to a floating point (real) power involves the employment of two library functions and instead of $X^{**}A$ one might just as well have written in Fortran: $\text{EXP}(A * \text{ALOG}(X))$. Both EXP and ALOG are usually computed by evaluation of an approximating polynomial or continued fraction and each would perform several multiplications and divisions. One often sees:

$$Y = X^{**.5}$$

or

$$Y = X^{**1.5}$$

which is approximately the same as:

$$Y = \text{SQRT}(X)$$

or

$$Y = X * \text{SQRT}(X)$$

The SQRT routine employs three or four Newton iterations and involves fewer operations than either the EXP or ALOG routine. It might also be noted that SQRT is usually coded for accuracy to the last one or two bits. EXP and ALOG are rarely coded to this accuracy over their whole range.

It should go without saying that:

Example 6

```
DO 1 I=1,10
  X(I)=Y(I)*2.**I
  etc.
1 CONTINUE
```

should be written as:

Example 6a

```
EXTRA=2.
DO 1 I=1,10
  X(I)=Y(I)*EXTRA
  etc.
1 EXTRA=EXTRA+EXTRA
```

Another common practice is to write equations inside DO loops with constant values (so far as the DO loop is concerned) which, depending on the complexity of the DO loop and on the compiler, may be reevaluated each time through the loop. Several examples of this appear in the Appendix. One should remember that programs distributed for general use are not always run under the "best" of compilers and the extra trouble taken can protect the naive user of the program.

In calculations involving sines and cosines one might see:

Example 7

```
A=COS(X)*SIN(Y)+SIN(X)*SIN(Z)
B=COS(X)*COS(Y)+SIN(X)*COS(Z)
```

The $\text{COS}(X)$ and $\text{SIN}(X)$ are computed in both equations when one could have written:

Example 7a

```
XCOS=COS(X)
XSIN=SIN(X)
A=XCOS*SIN(Y)+XSIN*SIN(Z)
B=XCOS*COS(Y)+XSIN*COS(Z)
```

Let us consider the problem of calculating the first nine Legendre coefficients using: $x = \cos \theta$, $P_0(x) = 1$, $P_1(x) = x$, and $(n+1) P_{n+1}(x) = (2n+1)x P_n(x) - n P_{n-1}(x)$.

Fortran subscripts must run from 1 up to 10 rather than from 0 to 9.

So, "straightforwardly":

Example 8

```
P(1)=1.
P(2)=X
DO 1 L=3,10
1 P(L)=((2*L-3)*X*P(L-1)-(L-2)*P(L-2))/(L-1)
```

There are at least two things "wrong" about this loop. The first and (in this case) least important is the expression $2*L-3$. This could, and should, be written as $L+L-3$. The more important point is that the use of the integer L mixed in the expression with variables of type real implies at least one extra set of coding to convert L to a real (floating point) value before it can be used in the calculation. This, depending on the compiler (and the computer), can cause anywhere from five instructions to a library function call to perform the conversion. The following change removes both of these pitfalls:

Example 8a

```

      P(1)=1.
      P(2)=X
      FL=3.
      DO 1 L=3,10
      P(L)=((FL+FL-3.)*X*P(L-1)- (FL-2.) *P(L-2))/(FL-1.)
1  FL=FL+1.

```

If this section of coding is used very often then the coefficients should be precomputed and the equation reduced to

$P(L)=A(L)*X*P(L-1)-B(L)*P(L-2):$

Example 8b

```

      SUBROUTINE LEGEND(X)
      COMMON P(10)
      DIMENSION A(10),B(10)
      DATA (INIT=0)
      IF(INIT)1,2,1
2  INIT=1
      FLML=2.
      DO 3 L=3,10
      B(L)=1.-1./FLML
      A(L)=1.+B(L)
3  FLML=FLML+1.
1  P(1)=1.
      P(2)=X
      DO 4 L=3,10
4  P(L)=A(L)*X*P(L-1)-B(L)*P(L-2)
      RETURN
      END

```

Notice particularly the use of the DATA statement. At the time LEGEND is first called, INIT will have the value 0 by virtue of the DATA statement. This will cause a transfer to statement 2 where the computation of the coefficients A and B can be performed. The value of INIT is changed from 0 so that on all subsequent calls the test of INIT will transfer around the section of coding beginning at statement 2 and begin computing at statement 1.

This "trick" of using a DATA statement to preset a value which can be tested on entry to a subroutine or function is an extremely useful tool at the programmer's disposal. One more point to note is that the running time could even be further improved by removing the DO loop on statement 4 and replacing it by:

$$P(3) = A(3) * X * P(2) - B(3) * P(1)$$

$$P(4) = A(4) * X * P(3) - B(4) * P(2)$$

$$\vdots$$

$$P(10) = A(10) * X * P(9) - B(10) * P(8)$$

From this point only marginal gains can be accomplished; e.g., using the DATA statement to set the values of A and B:

$$\text{DATA}(A(3)=1.5), (A(4)=1.6666667), \text{ etc.}$$

$$\text{DATA}(B(3)=.5), (B(4)=.6666667), \text{ etc.}$$

This would allow the removal of the test on INIT.

If one were to suggest that all of this is very fine but the number of coefficients (10 in the example) is not fixed and may vary from CALL to CALL, say, CALL LEGEND(X, LMAX) and, therefore, the above methods would not be applicable, one would be wrong.

If LMAX were, generally, 4 or less one should not be calling a subroutine as the values of the P's could be written out without recourse to a subroutine at all. If LMAX ranged randomly from 4 to 10 one could use Example 8b with DO 4 L=3,10 replaced by DO 4 L=3,LMAX. If LMAX ranged from 7 to 10, one could write out explicitly the first 7 P's and then test LMAX and branch to a RETURN for LMAX=7, branch to a section calculating P(8) and RETURN for LMAX=8, etc.

Summary

Generally, equations are written in Fortran in their most easily recognized forms. These forms, unfortunately, involve lengthy recomputations (within loops particularly) of constant factors and even of functions in some instances. Another source of lost time is the requirement for the use of a variable as both an index (integer form) and a value (floating point form). A bit of effort to "simplify" DO-loops and to avoid the use of library functions to convert numbers from one form to another can yield a significant decrease in execution time.

Examples

Three monitor input tapes, consisting of approximately 30 jobs, were listed and examined. All of them consisted of short jobs for debugging. The point of the examination was to study the way programmers were writing their Fortran programs.

It must be emphasized that all of these programs were either in the debugging phase or were short, one-time programs to compute a few results. Excerpts from these programs (statements taken out of context!) appear in the appendix together with comments as to how they would be improved if they were debugged and used in production very often. Some of them were of the one-shot type (which are usually sloppily written) but these are also treated as if they were massive users of computer time. Once the Fortran programs have been debugged presumably a treatment as suggested in the comments would be in order. No attempt is made to change the subscripting since this has been described previously in TM-1969.

$$22 \quad MO = 2 * (MO/4) + 1 \quad (A)$$

GO TO 24

$$23 \quad MO = 2 * (MO/8) + 1 \quad (B)$$

A. Faster would be:

$$22 \quad MOX = MO/4$$

$$MO = MOX + MOX + 1$$

B. Faster would be:

$$23 \quad MOX = MO/8$$

$$MO = MOX + MOX + 1$$

$$KP = K/2 * 2$$

Faster would be:

$$KP = K/2$$

$$KP = KP + KP$$

$$T = 5. * \text{LOGF}(10.0)$$

(A)

$$C = 4.0 * T$$

$$\text{PI15} = (1.0/3.14159265)**1.5$$

(A)

(B)

A. $5. * \text{LOGF}(10.0)$ is a constant.

$(\frac{1}{\pi})^{3/2}$ is a constant.

B. Faster execution would occur with:

$$\text{PI15} = 1. / \text{SQRT}(3.14159265**3)$$

DO 20 K = 1, TX

IF(ALF(K) .LT. 0.) 19, 20

19 ALF(K) = 0.5 * ALFX(K)

XX(K,1) = -1.0 * ALF(K)

(A)

20 CONTINUE

A. This is better written as:

$$\text{XX}(K,1) = -\text{ALF}(K)$$

```

      U(1) = COSF(THETR)
      U(2) = -SINF(THETR)
      U(3) = 0.
      V(1) = SINF(THETR)
      V(2) = COSF(THETR)

```

Faster is:

```

      U(1) = COSF(THETR)
      U(2) = -SINF(THETR)
      U(3) = 0.
      V(1) = -U(2)
      V(2) = U(1)

```

```

      DO 5 I = 1,N1
      DPMS(I) = CASM(I) * RMF
      PIC(I) = DPMS(I)/2.22
5     AMC(I) = PIC(I)/ 1000000.0

```

The loop would be faster as:

```

      CON = 1. / 2.22
      DO 5 I = 1, N1
      DPMS(I) = CASM(I) * RMF
      PIC(I) = DPMS(I) * CON
5     AMC(I) = PIC(I) * 1.0E-6

```

SELTH = SQRT(SELTH / (K - 3.))

SELSG = SQRT(SELSG / (K - 3.))

RESIG = SQRT(RESIG / (K - 3.))

$\frac{1}{K-3.0}$ is a constant for these equations, so:

XK = 1. / (K - 3.)

SELTH = SQRT(SELTH * XK)

SELSG = SQRT(SELSG * XK)

RESIG = SQRT(RESIG * XK)

Q1 = .010320 * EXPF(4. * 1.17202 * SQRT(S1) / (1. + .4 * SQRT(S1)))

Notice that SQRT(S1) appears twice.

```
DO 10 I = 1,12
10 BB(I) = B(100) * BB(I) / 48.542
```

$B(100) / 48.542$ is a constant inside this loop.

Rewriting as:

```
EX = B(100) / 48.542
DO 10 I = 1,12
10 BB(I) = BB(I) * EX
```

would save eleven divisions.

```
T2P = 1./EXPF(XR)
EREPA = 6. * B * EXPF(-XR)
```

$EREPA6 = 6. * B * T2P$ is faster.

```

YC = 0.5 * A/GP1(1)
FI = Y(J) - YC
DERIV(1) = -0.5 * A/(GP1(1))**2
DERIV(2) = 0.5/GP1(1) * X(J,1)
DERIV(3) = 0.5/GP1(1) * X(J,2)

```

Rather:

```

YCC = .5/GP1(1)
YC = A * YCC
FI = Y(J) - YC
DERIV(1) = -YC/GP1(1)
DERIV(2) = X(J, 1) * YCC
DERIV(3) = X(J,2) * YCC

```

The original has five multiply and four divide operations. The revised version has three multiply and two divide operations. One can also write:

```

XGP = 1./GP1(1)
YCC = .5 * XGP
etc.
DERIV(1) = -YC * XGP
etc.

```

and have five multiply and one divide operations.

```

1  DENOM = 2 * IMAX - 1      (A)
   IMAX = 2 * IMAX
   DELTA = (TWMAX - TWONE)/DENOM
   GO TO 3
2  DENOM = IMAX - 1
   DELTA = (TWMAX - TWONE)/DENOM
3  CONTINUE

```

} (B)

A. These two statements could well be reversed:

```

1 IMAX = IMAX + IMAX
   DENOM = IMAX - 1

```

B. Note that these five statements could also be written:

```

GO TO 4
2 DENOM = IMAX - 1
4 DELTA = (TWMAX - TWONE)/DENOM
3 CONTINUE

```

No machine time is saved or lost but the amount of memory used would be reduced.

UDZD = UDZDO * SQRT(E)

UL = ULO * SQRT(E)

Faster is:

SQRTE = SQRT(E)

UDZD = UDZDO * SQRTE

UL = ULO * SQRTE

H11 = H11 - W4(A,10.,15.) * COS(4. * D)

H12 = H12 - W3(A,10.,24.) * COS(4. * D)

H13 = H13 + W4(A,8.,12.) * COS(2. * D)

H14 = H14 + W3(A,8.,16.) * COS(4. * D)

Note that COS(4. * D) is computed three times.

```

      DO 6 I = 1, N1
      DPMS(I) = DPMS(I)/WG(I)
      PIC(I) = PIC(I)/WG(I)
6     AMC(I) = AMC(I)/WG(I)

```

The loop would run slightly faster on most computers if written as:

```

      DO 6 I = 1, N1
      EX = 1. / WG(I)
      DPMS(I) = DPMS(I) * EX
      PIC(I) = PIC(I) * EX
6     AMC(I) = AMC(I) * EX
360/75  Multiply = 2.1 μsec.
      Divide = 3.9 μsec.

```

so that 1 divide + 3 multiplies = 10.2 μsec.

and 3 divides = 11.7 μsec.

```

E1 = EXP(-UF * ZZ - UDIF * V * Z/CO)
E2 = EXP(-UF * ZZ - UDIF * V * Z/CO - UDZD)
TK = (E1 - E2)/(1.0 - EXP(-UDZD))

```

Faster and yielding the same results would be:

```

E1 = EXP(-UF * ZZ - UDIF * V * Z/CO)
E2 = E1 * EXP(-UDZD)
TK = E1

```

```

YDIREA = A(1) + A(2) * D + A(3) * D**2 + A(4) * D**3 + A(5) * D**4 + A(6) * D**5
YINREA = A(1) * D + (A(2)/2.) * D**2 + (A(3)/3.) * D**3 + (A(4)/4.) * D**4 +
1  (A(5)/5.) * D**5 + (A(6)/6.) * D**6

```

This would be considerably faster if written as:

```

YDIREA = (((((A(6) * D + A(5)) * D + A(4)) * D + A(3)) * D + A(2)) * D + A(1)
YINREA = (((((A(6)/6. * D + A(5) * .2) * D + A(4) * .25) * D +
1  A(3)/3.) * D + A(2) * .5) * D + A(1)) * D

```

$$V(5) = 3.1416 * X * (R(5)**2 - R(4)**2) + 2.4 * (R(5)**3 - R(4)**3)$$

$$V(6) = 3.1416 * X * (R(6)**2 - R(5)**2) + 2.4 * (R(6)**3 - R(5)**3)$$

Notice the use of $R(5)**2$ and $R(5)**3$. This sort of operation is faster if done as follows:

$$CONA = 3.1416 * X$$

$$R5SQ = R(5) * R(5)$$

$$R4SQ = R(4) * R(4)$$

$$R5CU = R5SQ * R(5)$$

$$V(5) = CONA * (R5SQ - R4SQ) + 2.4 * (R5CU - R4SQ * R(4))$$

$$R6SQ = R(6) * R(6)$$

$$V(6) = CONA * (R6SQ - R5SQ) + 2.4 * (R6SQ * R(6) - R5CU)$$

$$DC(1) = F * R - 4. * PI * D * (X * C(2) * C(1) + Y * C(2) * C(1))$$

$$DC(4) = 4. * PI * D * RII * C(2) * C(2)$$

$$DC(5) = 4. * PI * D * (Y * C(2) * C(3) + RII * C(2) * C(3))$$

$$DC(6) = 4. * PI * D * (X * C(2) * C(1) + X * C(2) * C(3))$$

The calculation of $4. * PI * D$ appears in each statement.

$Y * C(2)$ appears two times.

$X * C(2)$ appears three times.

$C(2) * C(3)$ appears three times.

$C(1) * C(2)$ appears three times.

This would be faster if written as:

$$DPI4 = 4. * PI * D$$

$$C2C1 = C(2) * C(1)$$

$$C2C3 = C(2) * C(3)$$

$$DC(1) = F * R - DPI4 * C2C1 * (X + Y)$$

$$DC(4) = DPI4 * RII * C(2) * C(2)$$

$$DC(5) = DPI4 * C2C3 * (Y + RII)$$

$$DC(6) = DPI4 * X * (C2C1 + C2C3)$$

The original contained 27 multiplications. The revised version contains 14 multiplications.

```

      DO 52 I = 1,K
      SCALEX = MAX1F(SCALEX,ABSF(X(I)))
      SCALEY = MAX1F(SCALEY,ABSF(Y(I)))
      IF(W(I)) 51, 51, 52

```

The use of the library function MAX1F could be avoided by:

```

      DO 52 I = 1,K
      EX = ABSF(X(I))
      IF(SCALEX-EX) 520, 521, 521
520  SCALEX = EX
521  EY = ABSF(Y(I))
      IF(SCALEY-EY) 522, 523, 523
522  SCALEY = EY
523  IF(W(I)) 51, 51, 52

```

Z = COSF(A) * COSF(C) + SINF(A) * SINF(C) * COSF(B)

(A)

AZ = SQRTF(1. - Z * Z)

Z = (COSF(A) - COSF(C) * Z) / (SINF(C) * AZ)

(A)

Z = AMIN1(1., Z)

(B)

Z = AMAX1(-1., Z)

D = ATANF(SQRTF(1. - Z * Z) / Z)

RETURN

END

A. The cosine of A is computed twice.

The cosine of C is computed twice.

The sine of C is computed twice.

Rewrite as:

COSA = COSF(A)

COSC = COSF(C)

SINC = SINF(C)

Z = COSA * COSC + SINF(A) * SINC * COSF(B)

AZ = etc.

Z = (COSA - COSC * Z) / (SINC * AZ)

B. IF(Z-1.) 1, 3, 2

2 Z = 1.

GO TO 3

1 IF(Z+1.) 4, 3, 3

4 Z = -1.

3 D = ATANF(SQRTF(1. - Z * Z)/Z)

Note also that when Z is set to either $\pm 1.$, $D = 0.$, so the first three statements in B. could be rewritten as:

IF(Z-1.) 10,20,20

20 D = 0.

RETURN

10 IF(Z + 1.) 20, 20, 3

and similarly for $Z \leq -1.$

```

*****
      DO 26 L = 1, IMAX
26    F(L,I) = F(L,I) + T(LQ,I) * U(LLQ,1) * (2 * LI + 1) * A(L) * (2 * LP + 1)*.0796
*****

```

Note that only values which are functions of L need be recomputed in the loop.

```

      EXTRA = T(LQ,I) * U(LLQ,1) * (2 * LI + 1) * (2 * LP + 1) * .0796
      DO 26 L = 1, IMAX
26    F(L,I) = F(L,I) + EXTRA * A(L)

```

Also: $2 * LI + 1 \rightarrow LI + LI + 1$

and $2 * LP + 1 \rightarrow LP + LP + 1$

```

      DO 21 L = 1,30
      XA = XA + FP(L, N1) * P(2 * L - 1, J1)
21    X = X + TOTAL(L,N1) * P(2 * L - 1, J1)

```

For $L = 1$, $2 * L - 1 = 1$

$L = 2$, $2 * L - 1 = 3$

$L = 3$, $2 * L - 1 = 5$

Faster is:

```

      LXM1 = 1
      DO 21 L = 1,30
      XA = XA + FP(L, N1) * P(LXM1, J1)
      X = X + TOTAL(L, N1) * P(LXM1, J1)
21    LXM1 = LXM1 + 2

```

```

      DO 2 I = 1,7
      Z = (I-.175)*.7854/6.825
      C(I+1) = COSF(Z)
      D(I+1) = SINF(Z)
2     E(I+1) = 0.

```

Rewriting as follows would save considerable time:

```

      CON = .7854/6.825
      Z = .825 * CON
      DO 2 I = 1,7
      C(I+1) = COSF(Z)
      D(I+1) = SINF(Z)
      E(I+1) = 0.
2     Z = Z + CON

```

```

DO 1 I = 1,19                                (A)
X = (I - 1.) * 5.
B = (I - 1.) * .087267
DO 6 J = 1,50                                (B)
C = (J - .5) * .030416

```

A. Note that:

$I = 1 \rightarrow X = 0.$ and $B = 0.$

$I = 2 \rightarrow X = 5.$ and $B = 1. * .087267$

$I = 3 \rightarrow X = 10.$ and $B = 2. * .087267$

Rewrite as:

$B = 0.$

$X = 0.$

DO 1 I = 1,19

·

etc.

·

$B = B + .087267$

1 $X = X + 5.$

B. Note that:

$J = 1 \rightarrow C = .5 * .030416$

$J = 2 \rightarrow C = 1.5 * .030416$

Rewrite as:

$C = .5 * .030416$ or $C = .015208$

DO 6 J = 1,50

etc.

6 $C = C + .030416$

```

      INTVL = 100
      H = 1./FLOAT(INTVL)      (A)
      DO 23 I = 1, INTVL
      S = 1. - (FLOAT(I-1) + .4226497) * H      (B)
C    FORTRAN STATEMENTS
23  CONTINUE

```

A. $H = .01$

B. Note that $\text{FLOAT}(I-1)$ can be removed:

```

      S = 1. - .4226497 * H
      DO 23 I = 1, INTVL
C    FORTRAN STATEMENTS
23  S = S - H

```

```

      DO 31 I = 1,N
      Z = COSF(0.01745329 * THETA(I))
      P(1,I) = 1.
      P(2,I) = Z
      P(3,I) = 1.5 * Z * Z - 0.5
      DO 32 K = 2,57
      (A) 32 P(K+2,I) = ((2. * K + 1.) * Z * P(K+1,I) - K * P(K,I))/(K+1)
      31 CONTINUE

```

A. Both K and K + 1 will be converted from integer to floating point each time through the loop on statement 32.

Rewrite as:

```

      XK = 2.
      XKA = 5. * Z
      XKAINC = Z + Z
      DO 32 K = 2,57
      XKP = XK + 1.
      P(K+2, I) = (XKA * P(K+1, I) - XK * P(K, I)) / XKP
      XKA = XKA + XKAINC
      32  XK = XKP

```

```

      DO 71 NK = 1,K
      YCALC = A(1)
      DO 61 I = 2,LIM
61    YCALC = YCALC + A(I) * (X(NK) / SCALEX)**(I-1)
      71  SIGR2 = SIGR2 + (Y(NK) / SCALEY - YCALC)**2

```

Statement 61 is:

$$YCALC = YCALC + A(I) * \left\{ \frac{X(NK)}{SCALEX} \right\}^{I-1}$$

$\frac{X(NK)}{SCALEX}$ is a constant for the loop on 61 so:

```

      DO 71 NK = 1, K
      YCALC = A(1)
      CON = X(NK) / SCALEX
      EXTRA = CON
      DO 61 I = 2, LIM
      YCALC = YCALC + A(I) * EXTRA
61    EXTRA = EXTRA * CON
      71  SIGR2 = SIGR2 + (Y(NK) / SCALEY - YCALC)**2

```

```

*****
      DO 60 K = 1, N
      DO 60 L = 1, M
      SQI = SQRT(Y(L))
      G(K,L) = -A1 * DELZ(K) * SQI / (1. + B1 * SQI) - C1 * Y(L) + D1 * Y(L)
60    X(K,L) = G(K,L) * Q(L)
*****

```

These DO loops would be much faster if reversed and written as:

```

      D1MC1 = D1 - C1
      DO 60 L = 1, M
      SQI = SQRT(Y(L))
      AX = -A1 * SQI / (1. + B1 * SQI)
      BX = Y(L) * D1MC1
      DO 60 K = 1, N
      G(K, L) = AX * DELZ(K) + BX
60    X(K, L) = G(K,L) * Q(L)

```

```

      DO 2200 J = 1, NOBS
      NZ = NOZ(J)
      X(J) = XI(J) / WTDOWN(NZ)
      DO 2200 I = 1, 11
      FRAC(I) = FLOAT(I - 1) / 10.
      IF(XI(J) - FRAC(I)) 2100, 2100, 2200
2100   NFRAC(I) = NFRAC(I) + XINTF(WT(J))
      XFRAC(I) = FLOATF(NFRAC(I)) / EXPSUM
2200   CONTINUE

```

(A)

A. XI = 0.

```

      DO 2201 I = 1, 11
      FRAC(I) = XI
      IF(etc.) 2100, 2100, 2201
2201 XI = XI + .1
2200 CONTINUE

```

would be faster.

Since the computation of FRAC(1) to FRAC(11) is performed identically for each J, the outer loop, speed would increase by:

```

      FRAC(1) = 0.
      DO 3000 IJ = 2, 11
3000 FRAC(IJ) = FRAC(IJ) + .1
      DO 2200 J = 1, NOBS
      etc.
      DO 2200 I = 1, 11
      etc.
2200 CONTINUE

```

Notice that the FRAC's are now computed once rather than NOBS times.

```

      DO 1 I = 1, N
      C(I) = 0.0
      DO 1 J = 1, N
1      B(I,J) = 0.
      DO 2 I = 1, N
      DO 2 J = 1, M
2      C(I) = C(I) + Y(J) * X(I,J)
      DO 5 I = 1, N
      DO 5 J = 1, N
      DO 5 K = 1, M
5      B(I,J) = B(I,J) + X(J,K) * X(I,K)

```

This can be "collapsed" as follows:

```

      DO 1 I = 1, N
      C(I) = 0.0
      DO 2 J = 1, N
      B(I,J) = 0.
      DO 2 K = 1, M
2      B(I,J) = B(I,J) + X(J,K) * X(I,K)
      DO 3 J = 1, M
3      C(I) = C(I) + Y(J) * X(I,J)
1      CONTINUE

```

There are now four DO loops instead of seven and the code will run faster.

```

      COEF = 2. * PI / (PI * BOLZ * TEMPK)**1.5      (A)
      DO 40 I = 1, K
      YI = I - 1
      X(I) = X1 + YI * XINC
      Y(I) = COEF * SQRTF(X(I)) / E**(X(I) / (BOLZ * TEMPK))
      XEV(I) = X(I) / ERG      (B)
      YEV(I) = Y(I) * ERG
      40 CONTINUE

```

} (C)

A. Faster would be:

```

      EX = 1. / SQRTF(PI * (BOLZ * TEMPK)**3)
      COEF = EX + EX

```

B. Note that $E^{**}(-BOLZ * TEMPK)$ and $1./ERG$ are constant for the loop.

C.	<u>I</u>	<u>YI</u>	<u>X(I)</u>	$\frac{E^{X(I)}}{E^{X1}}$
	1	0.	X1	E^{X1}
	2	1.	X1 + XINC	$E^{X1} * E^{XINC}$
	3	2.	X1 + 2. * XINC	$E^{X1} * E^{XINC} * E^{XINC}$

BTK = BOLZ * TEMPK

EX = COEF / E**(X1/BTK)

EXINC = 1. / E**(XINC/BTK)

CONA = 1. / ERG

XA = X1

DO 40 I = 1, K

X(I) = XA

Y(I) = EX * SQRTF(XA)

XEV(I) = XA * CONA

YEV(I) = Y(I) * ERG

EX = EX * EXINC

40 XA = XA + XINC

```

      DO 12 J = K, NR
      IPRJ = (J - 1) * NA + IPR      (A)
      Z = A(IPRJ)
      KJ = (J - 1) * NA + K          (A)
      A(IPRJ) = A(KJ)
12    A(KJ) = Z
      DO 13 J = 1, NV
      IPRJ = (J - 1) * NX + IPR      (A)
      Z = X(IPRJ)
      KJ = (J - 1) * NX + K          (A)
      X(IPRJ) = X(KJ)
13    X(KJ) = Z

```

A. Notice that $(J-1) * NA$ and $(J-1) * NX$ are each computed twice.

This would be much faster if done as follows:

```

      JNA = (K-1) * NA
      DO 12 J = K, NR
      IPRJ = JNA + IPR
      Z = A(IPRJ)
      KJ = JNA + K
      A(IPRJ) = A(KJ)
      A(KJ) = Z
12    JNA = JNA + NA
      JNX = 0
      DO 13 J = 1, NV
      IPRJ = JNX + IPR
      Z = X(IPRJ)
      KJ = JNX + K
      X(IPRJ) = X(KJ)
      X(KJ) = Z
13    JNX = JNX + NX

```

Now, instead of $2 * (NV + NR - K)$ multiplications there is only one multiplication and all the other multiplications have been turned into additions.

```

DO 54 I = 1,K
YC = Y(I)/SCALEY
XC = X(I)/SCALEX
DO 54 J = 1,LIM
JA = J + J - 1
52 SUM(JA) = SUM(JA) + XC**(JA-1) * W(I)
53 SUM(JA+1) = SUM(JA+1) + XC**(JA) * W(I)
54 SUM(J+15) = SUM(J+15) + XC**(J-1) * W(I) * YC

```

The loops would be faster as:

```

DO 54 I = 1,K
YC = Y(I)/SCALEY
XC = X(I)/SCALEX
XCJML = YC * W(I)     $\left[ XC^{J-1} * YC * W(I) \right]$ , J-1 = 0    for statement 54

XCJAML = W(I)          $\left[ XC^{JA-1} * W(I) \right]$ , JA-1 = 0    for statement 52
XCSQ = XC * XC
JA = 1
DO 54 J = 1,LIM
52 SUM(JA) = SUM(JA) + XCJAML
53 SUM(JA+1) = SUM(JA+1) + XCJAML * XC
54 SUM(J+15) = SUM(J+15) + XCJML

XCJML = XCJML * XC
XCJAML = XCJAML * XCSQ
54 JA = JA + 2

```

```

DO 1 I = 2, NI
SUM = 0.
M = 2**(I - 2)
S = 2**(I - 1)
DO 2 J = 1, M
2 SUM = SUM + ((S - 2. * J + 1.) * A + (2. * J - 1.) * B)/S
C(2,1) = C(1,1)/2. + (B - A) * SUM/S
DO 3 J = 2, I
S = 4.*(J - 1)
3 C(2,J) = (S * C(2,J-1) - C(1,J-1))/(S-1.)
1 C(1,J) = C(2,J)

```

Set up the problem as follows:

```

M = 1
S = 2.
DO 1 I = 2, NI
SUM = 0.
FJ = 1. (FJ will replace 2. * J - 1.)
DO 2 J = 1, M
SUM = SUM + ((S - FJ) * A + FJ * B)/S
2 FJ = FJ + 2.
C(2, 1) = etc. (Use SS instead of S so that S will not have
SS = 4. to be recomputed)
DO 3 J = 2, I
C(2,J) = (SS * C(2, J-1) - C(1, J-1)) / (SS-1.)
3 SS = SS * 4. or SS = SS + SS (two adds and a store
C(1, J) = C(2, J) 3 SS = SS + SS are slightly faster than
1 S = S + S a multiply on the IBM
360/75)

```

DE = 4.

EI = INTF(VC)

DO 20 I = 1, 40

E = EI + DE * (I - 1)

(A)

RM = ZZ/E

S12 = 0.

DO 10 J = 1, 5

S12 = S12 + R2 * 2.** (2 * J) * (2.** (2 * J) - 1.) * B(2 * J)

(B) 1 * (((RM - R1)/R2)**(2 * J + 1) - (-R1/R2)**(2 * J + 1)) / (2. * J + 1)

10 CONTINUE

20 CONTINUE

A. I-1 runs from 0. to 39. for the loop so:

XI = 0.

or, if DE is not changed inside the loop:

DO 20 I = 1, 40

XI = 0.

E = EI + DE * XI

DO 20 I=1, 40

etc.

E = EI + XI

20 XI = XI + 1.

etc.

20 XI = XI + DE

and, finally, if E, EI, and DE are not changed inside the loop:

E = EI

DO 20 I = 1, 40

RM = ZZ/E

etc.

20 E = E + DE

B. Notice that $2.** (2*J)$ could run with the loop:

```

      J = 1   →   2.2J = 4.
      J = 2   →   2.2J = 16.

      TWOJ = 4.

      DO 10  J = 1, 5

      etc.

10  TWOJ = TWOJ * 4.

```

As only $2*J$ is used and J itself is never directly used in the loop:

```

      DO 10  J2 = 2, 10, 2

      and use J2 wherever  $2*J$  appears.

```

The inner loop would then be:

```

      RX = (RM - R1)/R2
      RY = -R1 / R2
      RXSQ = RX * RX
      RYSQ = RY * RY
      RX = RX * RXSQ
      RY = RY * RYSQ
      TWOJ = 4.

      DENOM = 3.

      DO 10  J2 = 2, 10, 2
      S12 = S12 + R2 * TWOJ * (TWOJ - 1.) * B(J2) * (RX - RY) / DENOM
      TWOJ = TWOJ * 4.
      RX = RX * RXSQ
      RY = RY * RYSQ

10  DENOM = DENOM + 2.

```

One other "trick" could be used here since $R2$ multiplies each element of the sum, $S12$. Remove the multiplication by $R2$ which is inside the loop on statement 10 and after statement 10, put $S12 = S12 * R2$.

```

35  R(8) = T
    DO 10 I = 1,8           (A)
10  R(8) = R(8) + X(I)
    Y = 2. * (Z - .575) * R(8) - .5 * (1. + T + X(1) + X(2) + X(3) +
1  X(4)) - 2. * (X(1) + X(9))
    S = Y * (2. * X(2) * (T + X(1)) + X(2)**2)           (C)
    IF(ABS((B * R(8) - S)/B/R(8)) .GT. 1.E-6) 30, 31 (D)
30  T = T - (B * R(8) - S)/(B - 2. * (Z - .825) * (2. * X(2) * (T + X(1))
1  + X(2)**2) - 2. * X(2) * Y)
    M = M + 1
    IF(M.GT.10) GO TO 31
    GO TO 35

```

The code is an iterative loop.

A. Notice that X(1) thru X(8) are not changed during the loop although R(8) is. Therefore:

```
35  XSUM = 0.
```

```
DO 10 I = 1, 8
```

```
10  XSUM = X(I) + XSUM
```

```
36  R(8) = T + SUM
```

etc., and GO TO 35 becomes GO TO 36

B. $2.0 * (Z - .575)$ is a constant during the loop.

$1.0 + X(1) + X(2) + X(3) + X(4)$ is a constant during the loop.

$2.0 * (X(1) + X(9))$ is a constant during the loop.

C. $2.0 * X(2)$ is a constant and also appears twice in statement 30.

$X(2) ** 2$ is a constant and also appears in statement 30.

D. $B * R(8) - S$ is used both in the IF test and in statement 30.

If all the "constant" terms were removed:

36 $R(8) = T + XSUM$

$Y = C1 * R(8) - C2 - .5 * T$

$ACON = C3 * T + C4$

$S = Y * ACON$

$BCON = B * R(8) - S$

IF(ABS(BCON * C5/R(8)) .GT. 1.E-6) 30, 31

30 $T = T - BCON / (B - C6 * ACON - C7 * Y)$

```

      DO 2 I = 1, NSHIFT
      DO 1 J = 1, NUM
1     FENERGY(2 * N - 1, J) = FENERGY (N, J)      (A)
2     N = N - 1
      DO 3 I = 2, NUM, 2
      DO 4 J = 1, NAWEXP, 2
      CY((J + 1)/2) = FENERGY(J, I)              (B)
4     CX((J + 1)/2) = XNEP(J)
      DO 5 K = 1, NSHIFT
      XP = XNEP(2 * K)                            (C)
      FENERGY(2 * K, I - 1) = FENERGY(1, I - 1)  (C)
      FEP = FENERGY(2 * K - 1, I)                (C)
5     FENERGY (2 * K, I) = FEP                    (C)
3     CONTINUE

```

A. $2 * N - 1$ is constant within the DO loop on statement 1.

```

      DO 2 I = 1, NSHIFT
      NX = N + N - 1
      DO 1 J = 1, NUM
1     FENERGY(NX, J) = FENERGY(N, J)
2     N = N - 1

```

B. J runs from 1 to NAWEXP in steps of 2.

$\frac{J+1}{2}$ runs from 1 to $\frac{NAWEXP+1}{2}$ in steps of 1.

Replace $\frac{J+1}{2}$ by JX and:

JX = 1

DO 4 J = 1, NAWEXP, 2

CY(JX) = FENERGY(J, I)

CX(JX) = XNEP(J)

4 JX = JX + 1

C. Let the DO loop on statement 5 run from 2 to NSHIFT * 2 in steps of 2 since K by itself is not used in the loop. If K were used in the loop then use K2 = K + K instead of 2 * K.

NS2 = NSHIFT + NSHIFT

DO 5 K = 1, NSHIFT

DO 5 K2 = 2, NS2 or

K2 = K + K

XP = XNEP(K2)

XP = XNEP(K2)

etc.

FENERGY(K2, I - 1) = FENERGY(1, I - 1)

FEP = FENERGY(K2 - 1, I)

5 FENERGY(K2, I) = FEP

```

DO 3 I = 1,12
  IX = IT
  DO 4 J = IX,100
    IT = J - 1
    IF(B(J) - BB(1)) 4, 4, 5
4  CONTINUE
5  XJ = (BB(I) - B(J-1)) / (B(J) - B(J-1))
  IZ = II(I)
  DO 3 K = 1, IZ
    IJ = I/2
    AJ = I/2. - IJ
    AI = A(IT) * (1. - XJ) + A(IT+1) * XJ
    PA = AB(IT) * (1. - XJ) + AB(IT+1) * XJ
    AK = (.785398 - PA) * (K - 1)/(IZ - 1. + AJ)
    C(IK) = SIN(AI) * COS(AK)
    E(IK) = COS(AI)
    D(IK) = SIN(AI) * SIN(AK)
3  IK = IK + 1

```

A. Note that IJ depends on I and is a constant for the loop on K
 AJ depends on IJ. Both could be moved into the I loop.

```

IJ = I/2
AJ = .5 * I - IJ
DO 3 K = 1, IZ

```

Note also that:

I	IJ	I/2.	AJ
1	0	.5	.5
2	1	1.	0.
3	1	1.5	.5
4	2	2.	0.

and AJ alternates between .5 and 0., so:

```

      AJ = .5
      DO 30 I = 1, 12
        etc.
      DO 3 K = 1, IZ
        etc.
3    IK = IK + 1
      IF(AJ) 31, 31, 32
31   AJ = .5
      GO TO 30
32   AJ = 0.
30   CONTINUE

```

B. The loop on K contains AI and PA, neither of which depend on K and, consequently, the SIN_F(AI) and COS_F(AI) can be computed before the loop:

```

      XK = 0.
      DO 3 K = 1, IZ
        AK = CON * XK
        C(IK) = CONA * COSF(AK)
        E(IK) = CONB
        D(IK) = CONA * SINF(AK)
        IK = IK + 1
3    XK = XK + 1.

```

INTERNAL DISTRIBUTION

- 1-2. Central Research Library
- 3. Document Reference Section
- 4-8. Laboratory Records
- 9. Laboratory Records - Record Copy
- 10-24. Division of Technical Information Extension
- 25. Laboratory and University Div., ORO
- 26. ORNL Patent Office
- 27-76. J. G. Sullivan
- 77. V. R. Cain
- 78. W. B. Gardner
- 79. A. A. Brooks
- 80-766. ORCID List

EXTERNAL DISTRIBUTION

- 767. J. W. Givens, Jr. - Applied Mathematics Division
Argonne National Laboratory, 9700 South Cass Avenue
Argonne, Illinois
- 768. R. P. Leinius - University of Wisconsin Computing Center
1210 W. Dayton Street, Madison Wisconsin
- 769. J. A. Thompson - Control Data Corporation
4201 N. Lexington Avenue, St. Paul, Minnesota
- 770. W. F. Miller - Stanford Linear Accelerator Center
P. O. Box 4349, Stanford, California