N89-16310

531-61
167055
13P.

WAS 528

Automated Fortran Conversion

Gregory Aharonian
Source Translation & Optimization
P.O. Box 404
Belmont, Ma 02178
617-489-3727

What to do with a million lines of Fortran code? Managers at every major Fortran installation are asking this question every day. Newer programming languages (C and ADA), and newer computer architectures (parallel, data flow) pose a serious dilemma. How will the algorithms and mathematical techniques in tens of thousands of Fortran programs be moved to these environments? Further, since no language will dominate the science and engineering arena, another question arises. With strained programming staffs and budgets, how will algorithms be maintained in multiple languages and architectures?

There are three solutions. The first is to hire additional staff to translate programs across languages, to coordinate and maintain large libraries of subroutines in the different languages using existing software tools. Most of the conversion will be from Fortran to C and ADA, a project with many unresolved issues (in particular array handling). The solution is unfeasible economically, when you consider the number of combinations of environments (a language out of Fortran,C,ADA,any other) with a new architecture (out of Cray, FPS, CSPI, Alliant, etc.). The staff requirements and overhead will be excessive, even if you could find enough people willing to do the very boring work of translating and maintaining software.

The second solution is to develop completely automatic language translation programs, using all of the breakthroughs in software engineering, language theory, and artificial intelligence. The problems here are many. First noone has developed an efficient automatic translation system. The few on the market either are not completely automatic, or produce very ugly and inefficient code. It is impossible for a computer (and even many humans) to translate a piece of Fortran code that operates on different dimensioned arrays passed to the same subroutine with some EQUIVALENCE and COMMON usage. Further you don't want exact translations. Fortran programs were written within the limitations of Fortran, when in the newer languages the algorithms can be expressed more clearly and efficiently.

The third, and most practical solution, which STO and a few others have adopted, uses an intermediate language that is easy to translate Fortran into, and allows for source code in others languages to be generated automatically. The intermediate language is the union of all other programming languages (and the trick is to create a useful union) with some extensions that reflect the nature of the algorithms. The benefits of this approach are many. First the original Fortran program has to be rewritten only once, and then only parts of the program; most Fortran code passes through without any change (i.e. assignment and simple IF statements). Software tools are provided to ease this initial translation. Once in the intermediate language, the algorithm can then be obtained in any other language automatically.

Some of the conversions (as options) include array indice reversal (where A(B(C,D),E(F,G)) in Fortran becomes in C A[E[G][F]][B[D][C]]), many precision support (constants appended with EO,DO etc., subroutine and function names are suffixed, ABSR, ABSD, ABSC), and insertion of timing/frequency analysis. Manual conversion introduces errors, hindering the testing of the translated programs.

Figure 1 shows an example of a subroutine from the Eispack library in ten different languages. First, the subroutine is rewritten in STO's intermediate language, and is shorter than most of the final programs. Then, the subroutine is automatically generated in the other languages (and back into Fortran). We have successfully converted Linpack (and its test drivers), and produced tested C, Pascal, Basic, and Fortran 77 versions (and if anyone has compilers for other languages, we will provide the code for verification).

What are the disadvantages of this approach? There are two main problems, which are present even if you adopt another solution to converting Fortran programs. The first problem is that many of the newer languages are incapable of supporting numerical algorithms as easily as Fortran does. Pascal does not allow subroutines to accept arrays of different sizes, making subroutine libraries all but impossible (actually some Pascal compilers do, but there are at least two incompatible implementations). Modula-2, a (weak) attempt to fix Pascal, also doesn't allow subroutines to handle different sized multiple dimensional arrays (only 1D). Neither Pascal nor Modula-2 allow complex numbers (the suggested solution of using records and turning arithmetic expressions into series of subroutine or function calls being pathetic). These languages also provide limited multiple precision support, and not the most useful looping control structures. Modula has no GOTO, and while most GOTOs can be removed from Fortran subroutines, some very important subroutines have GOTOs that are extremely difficult to remove. At least in C and ADA you can use GOTOs for these tricky subroutines (like the *INVIT algorithms in the Eispack library). C supports Fortran programs well; its only deficiency is the lack of COMPLEX numbers used with +-*/ (hint ANSI committee!!!).

The other main problem arises with ADA. ADA has many powerful capabilities that forces you to start from scratch to fully take advantage of ADA. Generics, exceptions, and other features can only be generated if the intermediate language is as expressive as ADA, in which case just use ADA/DIANA to begin with. Unfortunately there are many installations with millions of lines of Fortran code that probably don't need all of the power of ADA, in which case automated translation becomes reasonable. Then languages like Occam (for parallel processing) require additional design considerations (in this case to efficiently use the parallel architecture).

At STO, we are undertaking a project to convert SLATEC to multiple languages via the intermediate language; when successful, packages such as Spice, Nastran, and Gaussian 84 will be converted. These projects are quite important to the design of the intermediate langauge in the translation challenges provided. It is important to realize that the recoding is a small part of the translation process. Creating software environments for multi-language software maintenance is the more critical task. To do so will require flexible software generation programs, in particular, ...e based on the use of an intermediate language.

The approach taken by STO and others (Boyle at Argonne, Waters at MIT,de Maine at Auburn, Diana for ADA, Lexeme) of using an intermediate language and associated software tools will allow Fortran installations to move their Fortran programs into new environments with minimal problems. While not a perfect solution, it is less costly than having larger programming staffs, and more realistic than relying on completely automatic translators.

```
TYPE ARRAY1DR IS ARRAY (INTEGER RANGE <>) OF REAL;
TYPE ARRAY2DR IS ARRAY (INTEGER RANGE <>,
                        INTEGER RANGE <>) OF REAL;

PROCEDURE ORTRNR (N: IN INTEGER; LOW: IN INTEGER;
             HIGH: IN INTEGER; A: IN ARRAY2DR;
             ORT: IN OUT ARRAY1DR; Z: IN OUT ARRAY2DR) IS
    I, J, KL, MM, MP, MP1: INTEGER ;
    G: REAL  ;
BEGIN
--
--
--      EISPACK SUBROUTINE ORTRAN IN ADA
--
--
    FOR J IN 1..N LOOP
        FOR I IN 1..N LOOP
            Z(I,J) := 0.0E+0 ;
        END LOOP ;
        Z(J,J) := 1.0E+0 ;
    END LOOP ;
    KL := HIGH - LOW - 1 ;
    FOR MM IN 1..KL LOOP
        MP := HIGH - MM ;
        IF A(MP,MP - 1) \= 0.0E+0 THEN
            MP1 := MP + 1 ;
            FOR I IN MP1..HIGH LOOP
                ORT(I) := A(I,MP - 1) ;
            END LOOP ;
            FOR J IN MP..HIGH LOOP
                G := 0.0E+0 ;
                FOR I IN MP..HIGH LOOP
                    G := G + ORT(I) * Z(I,J) ;
                END LOOP ;
                G := (G / ORT(MP)) / A(MP,MP - 1) ;
                FOR I IN MP..HIGH LOOP
                    Z(I,J) := Z(I,J) + G * ORT(I) ;
                END LOOP ;
            END LOOP ;
        END IF ;
    END LOOP ;
END ;
```

```
ORTRND (N, LOW, HIGH, A, ORT, Z)
int N, LOW, HIGH ;
double **A ;
double **Z, *ORT ;
{
    int I, J, KL, MM, MP, MP1 ;
    double G ;
/**/
/*

            EISPACK SUBROUTINE ORTRAN IN C

*/
    for ( J = 1; J <= N; J += 1 ) {
        for ( I = 1; I <= N; I += 1 ) {
            Z[I][J] = 0.0E+0 ;
        }
        Z[J][J] = 1.0E+0 ;
    }
    KL = HIGH - LOW - 1 ;
    for ( MM = 1; MM <= KL; MM += 1 ) {
        MP = HIGH - MM ;
        if (A[MP][MP - 1] != 0.0E+0) {
            MP1 = MP + 1 ;
            for ( I = MP1; I <= HIGH; I += 1 ) {
                ORT[I] = A[I][MP - 1] ;
            }
            for ( J = MP; J <= HIGH; J += 1 ) {
                G = 0.0E+0 ;
                for ( I = MP; I <= HIGH; I += 1 ){
                   G = G + ORT[I] * Z[I][J] ;
                }
                G = (G / ORT[MP]) / A[MP][MP - 1];
                for ( I = MP; I <= HIGH; I += 1 ){
                   Z[I][J] = Z[I][J] + G * ORT[I];
                }
            }
        }
    }
}
```

```
      SUBROUTINE ORTRND (N,LOW,HIGH,A,LDA,ORT,Z,LDZ)
      INTEGER LDA, LDZ
      INTEGER N, LOW, HIGH
      DOUBLE PRECISION A(LDA,1)
      DOUBLE PRECISION Z(LDZ,1), ORT(1)
      INTEGER I, J, KL, MM, MP, MP1
      DOUBLE PRECISION G
C
C
C          EISPACK SUBROUTINE ORTRAN IN FORTRAN
C
C
      DO 210 J = 1 , N
          DO 190 I = 1 , N
              Z(J,I) = 0.0D+0
190       CONTINUE
          Z(J,J) = 1.0D+0
210   CONTINUE
      KL = HIGH - LOW - 1
      IF (KL .LT. 1)  GOTO 411
      DO 410 MM = 1 , KL
          MP = HIGH - MM
          IF (A(MP - 1,MP) .EQ. 0.0D+0)  GOTO 400
              MP1 = MP + 1
              DO 290 I = MP1 , HIGH
                  ORT(I) = A(MP - 1,I)
290           CONTINUE
              DO 390 J = MP , HIGH
                  G = 0.0D+0
                  DO 340 I = MP , HIGH
                      G = G + ORT(I) * Z(J,I)
340               CONTINUE
                  G = (G / ORT(MP)) / A(MP - 1,MP)
                  DO 380 I = MP , HIGH
                      Z(J,I) = Z(J,I) + G * ORT(I)
380               CONTINUE
390           CONTINUE
400       CONTINUE
410   CONTINUE
411   CONTINUE
      RETURN
      END
```

```
PROCEDURE: ORTRNR ()
        INTEGER ARG: N
        INTEGER ARG: LOW
        INTEGER ARG: HIGH
        ANY ARG: A
        ANY ARG: ORT/VAR
        ANY ARG: Z/VAR
END PROCEDURE
PUBLIC: ORTRNR

PROCEDURE: ORTRNR
        INTEGER : I, J, KL, MM, MF, MP1
        REAL : G
260 REM
262 REM
264 REM       EISPACK SUBROUTINE ORTRAN IN BASIC
266 REM
270 REM
320     FOR J = 1 TO N
340         FOR I = 1 TO N
360             Z(I,J) = 0.0E+0
380         NEXT
400         Z(J,J) = 1.0E+0
420     NEXT
440     KL = HIGH - LOW - 1
459     IF KL < 1 THEN  GOTO 821
460     FOR MM = 1 TO KL
480         MP = HIGH - MM
500         IF A(MP,MP - 1) = 0.0E+0 THEN 800
520             MP1 = MP + 1
540             FOR I = MP1 TO HIGH
560                 ORT(I) = A(I,MP - 1)
580             NEXT
600             FOR J = MP TO HIGH
620                 G = 0.0E+0
640                 FOR I = MP TO HIGH
660                   G = G + ORT(I) * Z(I,J)
680                 NEXT
700                 G = (G/ORT(MP)) / A(MP,MP - 1)
720                 FOR I = MP TO HIGH
740                   Z(I,J) = Z(I,J) + G * ORT(I)
760                 NEXT
780             NEXT
800         REM END OF IF BLOCK
820     NEXT
821     REM END OF IF BLOCK
840     REM RETURN
END PROCEDURE
```

```
ORTRNR:
        PROC  (N, LOW, HIGH, A, ORT, Z) ;
        DCL (N, LOW, HIGH) FIXED BIN (15) ;
        DCL A(*,*) FLOAT DEC (6) ;
        DCL (Z(*,*), ORT(*)) FLOAT DEC (6);
        DCL (I, J, KL, MM, MP, MP1) FIXED BIN (15);
        DCL G FLOAT DEC (6);
/*

        EISPACK SUBROUTINE ORTRAN IN PLI

*/
        DO J = 1 TO N ;
            DO I = 1 TO N ;
                Z(I,J) = 0.0E+0 ;
            END  ;
            Z(J,J) = 1.0E+0 ;
        END  ;
        KL = HIGH - LOW - 1 ;
        IF KL >= 1 THEN  DO;
        DO MM = 1 TO KL ;
            MP = HIGH - MM ;
            IF A(MP,MP - 1) != 0.0E+0 THEN  DO;
                MP1 = MP + 1 ;
                DO I = MP1 TO HIGH ;
                    ORT(I) = A(I,MP - 1) ;
                END  ;
                DO J = MP TO HIGH ;
                    G = 0.0E+0 ;
                    DO I = MP TO HIGH ;
                        G = G + ORT(I) * Z(I,J) ;
                    END  ;
                    G = (G / ORT(MP)) / A(MP,MP - 1);
                    DO I = MP TO HIGH ;
                        Z(I,J) = Z(I,J) + G * ORT(I);
                    END  ;
                END  ;
            END ;
        END  ;
        END ;
 END ORTRNR ;
```

```
PROC ORTRNR (N, LOW, HIGH, A: ORT, Z); BEGIN
    ITEM N S ;
    ITEM LOW S ;
    ITEM HIGH S ;
    TABLE A[*,*] F ;
    TABLE Z[*,*] F ;
    TABLE ORT[*] F ;
    ITEM I S ;
    ITEM J S ;
    ITEM KL S ;
    ITEM MM S ;
    ITEM MP S ;
    ITEM MP1 S ;
    ITEM G F ;
""
""
"         EISPACK SUBROUTINE ORTRAN IN JOVIAL"
""
""
    FOR J : 1 BY 1 WHILE J <= N ;BEGIN
        FOR I : 1 BY 1 WHILE I <= N ;BEGIN
            Z[I,J] = 0.0E+0;
        END;
        Z[J,J] = 1.0E+0;
    END;
    KL = HIGH - LOW - 1;
    IF KL >= 1;  BEGIN
    FOR MM : 1 BY 1 WHILE MM <= KL ;BEGIN
        MP = HIGH - MM;
        IF A[MP,MP - 1] <> 0.0E+0;  BEGIN
            MP1 = MP + 1;
            FOR I : MP1 BY 1 WHILE I <= HIGH ;BEGIN
               ORT[I] = A[I,MP - 1];
            END;
            FOR J : MP BY 1 WHILE J <= HIGH ;BEGIN
               G = 0.0E+0;
               FOR I : MP BY 1 WHILE I <= HIGH ;BEGIN
                   G = G + ORT[I] * Z[I,J];
               END;
               G = (G / ORT[MP]) / A[MP,MP - 1];
               FOR I : MP BY 1 WHILE I <= HIGH ;BEGIN
                   Z[I,J] = Z[I,J] + G * ORT[I];
               END;
            END;
        END
    END;

    RETURN;
    END
```

```
TYPE ARRAY1DR = SUPER ARRAY [1..*] OF REAL8;
TYPE ARRAY2DR = SUPER ARRAY [1..*,1..*] OF REAL8;

PROCEDURE ORTRNR (N:INTEGER; LOW:INTEGER;
                  HIGH:INTEGER; VAR A:ARRAY2DR;
                  VAR ORT:ARRAY1DR; VAR Z:ARRAY2DR);
VAR I, J, KL, MM, MP, MP1: INTEGER  ;
    G: REAL8  ;
BEGIN
(*

        EISPACK SUBROUTINE ORTRAN IN PASCAL

*)
    FOR J := 1 TO N DO BEGIN
        FOR I := 1 TO N DO BEGIN
            Z[I,J] := 0.0E+0 ;
        END ;
        Z[J,J] := 1.0E+0 ;
    END ;
    KL := HIGH - LOW - 1 ;
    IF (KL >= 1) THEN  BEGIN
    FOR MM := 1 TO KL DO BEGIN
        MP := HIGH - MM ;
        IF (A[MP,MP - 1] <> 0.0E+0) THEN  BEGIN
            MP1 := MP + 1 ;
            FOR I := MP1 TO HIGH DO BEGIN
                ORT[I] := A[I,MP - 1] ;
            END ;
            FOR J := MP TO HIGH DO BEGIN
                G := 0.0E+0 ;
                FOR I := MP TO HIGH DO BEGIN
                   G := G + ORT[I] * Z[I,J] ;
                END ;
                G := (G/ORT[MP]) / A[MP,MP - 1];
                FOR I := MP TO HIGH DO BEGIN
                   Z[I,J] := Z[I,J] + G * ORT[I];
                END ;
            END ;
        END ;
    END ;
    END ;
END; {ORTRNR}
```

```
CONST NEIG =
TYPE ARRAY1DR = ARRAY [1..NEIG] OF REAL;
TYPE ARRAY2DR = ARRAY [1..NEIG,1..NEIG] OF REAL;

PROCEDURE ORTRNR (N:INT ; LOW:INT ; HIGH:INT;
                  A:ARRAY2DR; VAR ORT:ARRAY1DR;
                  VAR Z:ARRAY2DR);
VAR I, J, KL, MM, MP, MP1: INT  ;
    G: REAL  ;
(*

        EISPACK SUBROUTINE ORTRAN IN MODULA-2

*)
BEGIN
    FOR J := 1 TO N DO
        FOR I := 1 TO N DO
            Z[I,J] := 0.0E+0 ;
        END ;
        Z[J,J] := 1.0E+0 ;
    END ;
    KL := HIGH - LOW - 1 ;
    IF (KL >= 1) THEN
    FOR MM := 1 TO KL DO
        MP := HIGH - MM ;
        IF (A[MP,MP - 1] <> 0.0E+0) THEN
            MP1 := MP + 1 ;
            FOR I := MP1 TO HIGH DO
                ORT[I] := A[I,MP - 1] ;
            END ;
            FOR J := MP TO HIGH DO
                G := 0.0E+0 ;
                FOR I := MP TO HIGH DO
                    G := G + ORT[I] * Z[I,J] ;
                END ;
                G := (G / ORT[MP]) / A[MP,MP - 1] ;
                FOR I := MP TO HIGH DO
                    Z[I,J] := Z[I,J] + G * ORT[I] ;
                END ;
            END ;
        END ;
    END ;
    END ;
END
```

```
~TYPE ARRAY1DR : ~ARRAY 1..* ~OF ~REAL
~TYPE ARRAY2DR : ~ARRAY 1..*,1..* ~OF ~REAL

~PROCEDURE ORTRNR (N: ~INT, LOW: ~INT, HIGH: ~INT,
                   A:ARRAY2DR, ~VAR ORT:ARRAY1DR,
                   ~VAR Z:ARRAY2DR)
~VAR I, J, KL, MM, MP, MP1: ~INT
     G: ~REAL
%
%
%       EISPACK SUBROUTINE ORTRAN IN TURING
%
%
     ~FOR J : 1..N
          ~FOR I : 1..N
              Z(I,J) := 0.0e+0
          ~END FOR
          Z(J,J) := 1.0e+0
     ~END FOR
     KL := HIGH - LOW - 1
     ~IF KL >= 1 ~THEN
     ~FOR MM : 1..KL
          MP := HIGH - MM
          ~IF A(MP,MP - 1) ~NOT = 0.0e+0 ~THEN
              MP1 := MP + 1
              ~FOR I : MP1..HIGH
                  ORT(I) := A(I,MP - 1)
              ~END FOR
              ~FOR J : MP..HIGH
                  G := 0.0e+0
                  ~FOR I : MP..HIGH
                    G := G + ORT(I) * Z(I,J)
                  ~END FOR
                  G := (G/ORT(MP)) / A(MP,MP - 1)
                  ~FOR I : MP..HIGH
                    Z(I,J) := Z(I,J) + G * ORT(I)
                  ~END FOR
              ~END FOR
          ~END IF
     ~END FOR
     ~END IF
~END ORTRNR
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
~PROC ORTRNR = (~INT N, ~INT LOW, ~INT HIGH,
               [,]~REAL A, ~REF []~REAL ORT,
                 ~REF [,]~REAL Z ) ~VOID:
~BEGIN
~CO

     EISPACK SUBROUTINE ORTRAN IN ALGOL-68

~CO
    ~INT I, J, KL, MM, MP, MP1 ;
    ~REAL G ;
    ~FOR J ~FROM 1 ~TO N ~DO
        ~FOR I ~FROM 1 ~TO N ~DO
            Z[I,J] := 0.0e+0 ;
        ~OD ;
        Z[J,J] := 1.0e+0 ;
    ~OD ;
    KL := HIGH - LOW - 1 ;
    ~IF KL ~GE 1 ~THEN
    ~FOR MM ~FROM 1 ~TO KL ~DO
        MP := HIGH - MM ;
        ~IF A[MP,MP - 1] ~NE 0.0e+0 ~THEN
            MP1 := MP + 1 ;
            ~FOR I ~FROM MP1 ~TO HIGH ~DO
               ORT[I] := A[I,MP - 1] ;
            ~OD ;
            ~FOR J ~FROM MP ~TO HIGH ~DO
               G := 0.0e+0 ;
               ~FOR I ~FROM MP ~TO HIGH ~DO
                  G := G + ORT[I] * Z[I,J] ;
               ~OD ;
               G := (G/ORT[MP]) / A[MP,MP - 1];
               ~FOR I ~FROM MP ~TO HIGH ~DO
                  Z[I,J] := Z[I,J] + G * ORT[I];
               ~OD ;
            ~OD ;
        ~FI ;
    ~OD ;
    ~FI ;
    RETURN: ;
~END
```