APL/FORTRAN Translations*

By

V. L. Moruzzi

December 1971

Technical Report #6

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ONR Contract No. N00014-70-C-0188
Task No. NR 389-155

* Work supported in part by the Geography Programs, Office of Naval Research.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

IBM Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

APL/FORTRAN Translations

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Victor L. Moruzzi

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1971	9	0

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-70-C-0188	Technical Report No. 6
b. PROJECT NO. Task 389-155	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* RC #3644
c.	
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Geography Programs Office of Naval Research

13. ABSTRACT

The advantages and limitations of APL\360 and OS/360 are outlined and a set of simple rules for translating APL to FORTRAN is given. A translation library containing FORTRAN programs that in a limited sense simulate the results of some of the more commonly used APL functions is described and listed for interested users.

DD FORM 1 NOV 65 1473

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming Translations						
Computer Simulation						

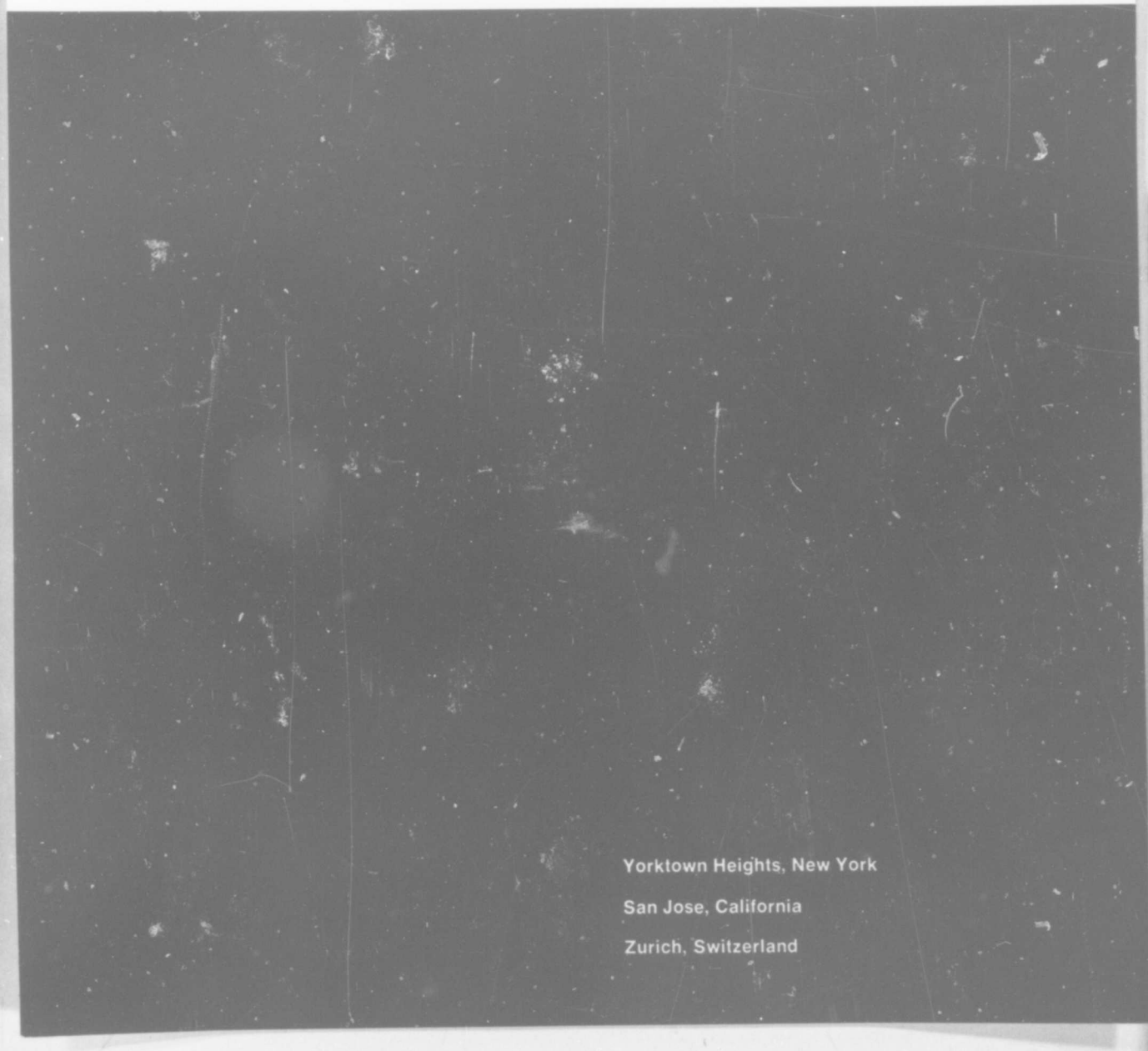
IBM Research

APL/FORTRAN TRANSLATIONS

V. L. Moruzzi

December 8, 1971

RC 3644



Yorktown Heights, New York

San Jose, California

Zurich, Switzerland

APL/FORTRAN Translations*

V. L. Moruzzi
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT: The advantages and limitations of APL\360 and OS/360 are outlined and a set of simple rules for translating APL to FORTRAN is given. A translation library containing FORTRAN programs that in a limited sense simulate the results of some of the more commonly used APL functions is described and listed for interested users.

RC 3644 (#16492)
December 8, 1971

*Supported in part by the Geography Programs, Office of Naval Research.
ONR Contract No. N00014-70-C-0188 Task No. NR 389-155

INTRODUCTION

From its inception, the APL language and the APL\360 time sharing system have attracted a host of dedicated followers. The mathematical compactness and explicitness of the language and the interactive features of the system appeal to technical users who do not require the memory, speed, and greater flexibility of the more cumbersome OS/360 compatible languages. Users can debug APL programs as they are written and can test complicated logical branching operations step by step. These features make APL\360 very attractive to a wide range of users.

However, APL\360 suffers from limitations which can be serious under certain circumstances. The relatively small workspaces normally available and the terminal related input/output restrictions limit the range of programming problems which can be undertaken. An even more serious, though less recognized, consideration is the waste in CPU time in executing APL programs whose FORTRAN counterparts involve significant use of the loop mode. Since APL recompiles each instruction at execution, such operations are costly and can result in severe drains on the system.

Although limited to somewhat more cumbersome languages, OS/360 avoids the above restrictions. Problems involving an order of magnitude more storage, speed, or input/output are very easily undertaken. An APL compiler, which would allow a user to run APL on OS, would therefore be very desirable, and would combine the best features of both. Until such a compiler is developed, an alternative less ambitious approach, is to translate APL programs directly to FORTRAN (or some other OS compatible language) by providing equivalent operations. This report gives simple prescriptions or rules for doing this in a limited sense and shows that APL/FORTRAN translations can be accomplished fairly easily.

TRANSLATION

Since APL is a function-based language, it is convenient to assemble a library of FORTRAN analogs of the APL functions. With a complete library one could, in principle, translate an APL program by coding an appropriate series of calls in FORTRAN. Because of the inherent differences in the two languages, however, serious problems arise in accumulating such a library. The APL functions are very general and can accommodate arguments of different types (logical, integer, or real) and different dimensions (scalar, vector, or matrix). Since there is nothing comparable to this in FORTRAN, we must specify the argument type and make the subroutines general enough to accommodate different dimensionality. This necessitates a different subroutine for each type of argument allowed.

A list of implemented programs is shown in Appendix 1. Although the list is far from complete, analogs of almost all APL functions can be written. In addition, the library contains some non-APL analogs like DELETI and RANDOM which have been found to be very useful. Subroutines, wherever possible, are given names that identify them with their APL analogs. When more than one type of argument is allowed, the last letter of the subroutine name identifies the argument type. Although the subroutines in the library were designed to accommodate arrays with a maximum dimensionality of two, they can be very easily extended if necessary. In any case, the maximum dimensionality of all arrays is clearly indicated in the argument lists.

One can immediately see from the Translation Table that there is no analog of the APL function RHO which gives the size of the array upon which it acts. Since a FORTRAN analog would be very difficult (if at all possible) to implement, the lengths of all arrays are carefuly traced throughout a translation. Whenever an array is introduced, its corresponding RHO is

defined. It is updated at every step in the program where it is subject to change. Multidimensional arrays must have the appropriate number of RHO's associated with them. (With some practice and discipline, this is not as difficult as it might seem.) To simplify matters all variables beginning with the letter R can be assumed to be integer by using the FORTRAN statement IMPLICIT INTEGER*4 (R). Thus, a vector A will have associated with it at all times, an integer RHOA which contains the instantaneous length of A. If A is an empty vector, RHOA=0.

Another difficulty arises from the array dimensioning requirement in FORTRAN. Since FORTRAN demands the reservation of a specific block of core storage for the exclusive use of each array used, we are forced to examine the arrays and to reserve enough storage to accommodate the maximum lengths involved. Since maximum array lengths can depend upon input data or upon random number generation (e.g., in problems involving stochastic processes), care must be taken to reserve adequate storage. In such cases it is necessary to design internal protection against exceeding array sizes. A simple test of the value of RHO of the array in question at the update points of the program, and the printing of a warning message when dimensions have been exceeded, can be very useful.

In an actual translation it is advisable to retain, whenever possible, the variable names used in the original and to reserve the statement numbers used in the APL program to locate the corresponding points in the FORTRAN program. The first considerations leads to the recommendation that all variables be defined in type statements so as to avoid difficulties due to the traditional FORTRAN implicit specification of variable type (according to the first character of the variable name). These two simple prescriptions have been found to be very useful in achieving trouble-free translations.

The APL distinction between local and global variables is very easily accommodated in FORTRAN by using COMMON. That is, global variables passed from program to program are simply put in a named COMMON. The usual FORTRAN rules for arrays in COMMON must, of course, be followed so that all COMMON areas have the same size in all pertinent programs.

A very simple example of an actual translation is shown in Appendix 2. Included is the listing of an APL program DISTR which is used to study the distribution of the elements in a vector L (and RD, a round-off program used in DISTR), and the listing of the FORTRAN translation. It is known that the elements of L are integers and that it is very unlikely that the maximum will ever exceed 100. The translation proceeds from right to left, top to bottom. Notice that the length of all arrays (including the temporary arrays TP and TMP) are traced throughout and are updated whenever they are used in calls to the translation library subroutines MAXI, IOTA, OUTEQI, and SUMI. In this case the APL original had no statement names and the statement numbering prescription was not used. The global variables VAR, XB, and N which are needed in other routines were put in the COMMON block CMDSTR.

RESULTS AND CONCLUSIONS

Experience has shown that with the translation library and the outlined prescriptions, fairly intricate APL programs can be translated with a minimum number of coding errors. Furthermore, this can be accomplished without ever having detailed knowledge of the actual function of the program. A copy of the APL original, and an indication of the input data type and range provide sufficient information for a translation.

Although our experience is limited, several sophisticated APL programs involving random-walk simulations of natural stream networks have been successfully translated. The APL originals involved approximately 80 statements while

the FORTRAN programs consist of approximately 650 statements. Thus each APL statement required approximately 8 FORTRAN statements. This count, of course, does not include the translation library.

Preliminary tests on translated programs indicate a gain of a factor of 30 in CPU time, i.e., translated programs (involving loop modes) run over 30 times faster than the original APL programs. Thus, for the sake of economy, and to ease the load on the system, routine APL programs which require large amounts of CPU time should be translated and run on OS.

We conclude by reiterating that APL/FORTRAN translations ccan be done mechanically and can achieve a 30-fold reduction in CPU time. Therefore APL users should seriously consider translations. In addition, FORTRAN users might consider writing and debugging in APL and translating to FORTRAN.

ACKNOWLEDGMENTS

We wish to thank J. S. Smart and P. M. Marcus for their encouragement, and the Geography Programs, Office of Naval Research, for their partial support of this work.

APPENDIX 1

TRANSLATION TABLE

ROUTINE	APL EQUIVALENT
CATI(C,A,RHOA,B,RHOB) CATR(C,A,RHOA,B,RHOB)	C←A,B
DATE(ID)	ID←⌶25 (SIMILAR)
DELETI(A,RHOA,B,C,RHOC) DELETR(A,RHOA,B,C,RHOC)	A←(~A∈B)/A
DROPI(B,N,A,RHOA) DROPR(B,N,A,RHOA)	B←N↓A
IOTA(A,RHOA)	A←⍳RHOA
KEEPI(B,N,A,RHOA) KEEPR(B,N,A,RHOA)	B←N↑A
MAXI(B,A,RHOAX,RHOAY) MAXR(B,A,RHOAX,RHOAY)	B←⌈/A
MEMBI(C,A,RHOA,B,RHOBX,RHOBY) MEMBL(C,A,RHOA,B,RHOBX,RHOBY) MEMBR(C,A,RHOA,B,RHOBX,RHOBY)	C←A∈B
MINI(B,A,RHOAX,RHOAY) MINR(B,A,RHOAX,RHOAY)	B←⌊/A
NOT(B,A,RHOAX,RHOAY)	B←~A
ORDUPI(B,A,RHOAX,RHOAY) ORDUPR(B,A,RHOAX,RHOAY)	B←⍋A
ORDWNI(B,A,RHOAX,RHOAY) ORDWNR(B,A,RHOAX,RHOAY)	B←⍒A
OUTEQI(C,A,RHOA,B,RHOB) OUTEQR(C,A,RHOA,B,RHOB)	C←A∘.=B

OUTPLI(C,A,RHOA,B,RHOB) OUTPLR(C,A,RHOA,B,RHOB)	C←A∘.+B
OUTPRI(C,A,RHOA,B,RHOB) OUTPRR(C,A,RHOA,B,RHOB)	C←A∘.×B
RAND(A,B)	B←?A
RANDOM(A,RHOA)	A←A[RHOA?RHOA]
RANKI(C,A,RHOA,B,RHOB) RANKR(C,A,RHOA,B,RHOB)	C←A⍳B
REVI(B,A,RHOAX,RHOAY) REVR(B,A,RHOAX,RHOAY)	B←⌽A
ROTI(B,N,A,RHOAX,RHOAY) ROTR(B,N,A,RHOAX,RHOAY)	B←N⌽A
SLINK	SETLINK (SIMILAR)
STRUCI(B,RHOBY,RHOBX,A,RHOA) STRUCR(B,RHOBY,RHOBX,A,RHOA)	B←(RHOBY,RHOBX)⍴A
SUMI(B,A,RHOAX,RHOAY) SUMR(B,A,RHOAX,RHOAY)	B←+/A
TIME(IT)	IT←I20 (SIMILAR)

APPENDIX 2

EXAMPLE OF APL/FORTRAN TRANSLATION

```
      ∇DISTR[⎕]∇
    ∇ DISTR L;∧;XM;XB2;P;R
[1]   →(0=N←⍴L)/0
[2]   X←+/(⍳XM←(⍳/L)∘.=L
[3]   XB←(+/X×⍳XM)÷N
[4]   XB2←(+/X×(⍳XM)*2)÷N
[5]   R←XB×P←XB÷XB+VAR←XB2-XB*2
[6]   'SAMPLE SIZE                    ';N
[7]   'MEAN                           ';3 RD XB
[8]   'STD DEV                        ';3 RD VAR*0.5
[9]   'DISTRIBUTION                   ';X
[10]  'PROB                           ';3 RD P
[11]  'CHECK                          ';3 RD R
    ∇

      ∇RD[⎕]∇
    ∇ R←D RD V
[1]   R←(10*-D)×⌊0.5+V×10*D
    ∇
```

```
      SUBROUTINE DISTRI(L,RHOL)
      IMPLICIT INTEGER*4 (R)
      INTEGER*4 XM,SUMTP,L(1),TP(100),TMP(100,100),X(100)
      REAL*4 R
      COMMON /CMDSTR/ VAR,XB,N
      N=RHOL
      IF(N.EQ.0) RETURN
      CALL MAXI(XM,L,RHOL,1)
      RHOTP=XM
      IF(XM.LE.100) GO TO 2
      WRITE(6,1)
    1 FORMAT('      ARRAY LENGTHS EXCEED STORAGE LIMIT IN DISTRI')
      RETURN
    2 CONTINUE
      CALL IOTA(TP,RHOTP)
      CALL OUTEQI(TMP,TP,RHOTP,L,RHOL)
      RTMPX=RHOL
      RTMPY=RHOTP
      CALL SUMI(X,TMP,RTMPX,RTMPY)
      RHOX=RTMPY
      RHOTP=XM
      CALL IOTA(TP,RHOTP)
      DO 3 I=1,RHOTP
    3 TP(I)=X(I)*TP(I)
      CALL SUMI(SUMTP,TP,RHOTP,1)
      XB =FLOAT(SUMTP)/FLOAT(N)
      RHOTP=XM
      CALL IOTA(TP,RHOTP)
      DO 4 I=1,RHOTP
    4 TP(I)=X(I)*TP(I)**2
      CALL SUMI(SUMTP,TP,RHOTP,1)
      XB2=FLOAT(SUMTP)/FLOAT(N)
      VAR=XB2-XB**2
      P=XB/(XB-VAR)
      SQVAR=SQRT(VAR)
      R=XB*P
      WRITE(6,5) N
      WRITE(6,6) XB
      WRITE(6,7) SQVAR
      WRITE(6,8) (X(I),I=1,RHOX)
      WRITE(6,9) P
      WRITE(6,10) R
    5 FORMAT('      SAMPLE SIZE = ',I10)
    6 FORMAT('      MEAN        = ',F10.5)
    7 FORMAT('      STD DEV     = ',F10.5)
    8 FORMAT('      DISTRIBUTION= ',16I6)
    9 FORMAT('      PROB        = ',F10.5)
   10 FORMAT('      CHECK       = ',F10.5)
      RETURN
      END
```