Automatic Translation of Fortran 77 to Fortran 90 Using VAST-90

John K. Prentice
Quetzal Computational Associates
quetzal@aip.org

To appear in the *Fortran Journal*, vol 5, no 4, July/August 1993

Introduction

While there appear to be no surveys to back it up, there is a consensus in the computer community that Fortran is one of the most widely used programming languages in the world[1]. Certainly it is the dominant scientific programming language, despite recent inroads by languages such as C and C++. Most of the Fortran code in existence today, however, is written in Fortran 77. While this language served its purpose well, it is badly out of step with modern computing. This is especially apparent when comparing Fortran 77 with modern languages such as C++. Fortunately, the deficiencies of Fortran 77 have been addressed with the introduction of Fortran 90.

[1]Ellis Horowitz. *Fundamentals of Programming Languages*, Computer Science Press, 1983

Fortran 90 contains all the features expected of a modern imperative programming language, as well as providing some of the features of a modern object oriented language[2]. With Fortran 90, codes can be written that are more concise, efficient, readable, less error prone, and better suited to modern computer architectures than is possible with Fortran 77. These reasons alone make it desirable to use Fortran 90, but there are advantages even more significant than the simple wish to write cleaner and more efficient codes.

With the adoption of Fortran 90 as an ANSI/ISO standard, support for Fortran 77 compilers will slowly cease, just as happened with Fortran 66 compilers during the 1980's. While Fortran 77 codes can be compiled with Fortran 90 compilers[3], it seems likely that the bulk of the effort in developing optimization techniques for Fortran in the future will be concentrated in optimizing of the newer features of Fortran 90 such as array syntax. For those using high performance computers, the situation is even more pressing. Fortran 77 codes are not easily adapted to modern high performance computers, particularly massively parallel systems. The reasons for this are varied, but one of the most significant is that neither of these earlier languages provided any straightforward way to express parallelism. Fortran 90 does have features which assist in the expression of data parallelism and these features form the kernel of both the CM-Fortran language for the Connection Machine and the more widely applicable High Performance Fortran language. In addition,

[2] Lawrie Schonfelder. "Fortran 90 is Already Object Oriented (Well Almost)". *Fortran Journal*, 5(2): 4-6, 1993.

[3] The Fortran 90 standard requires that Fortran 77 codes compile with a standard conforming Fortran 90 compiler.

the use of features such as Fortran 90 array syntax forces programmers to consider issues that are important to parallelism, such as data dependence in loops.

Fortran 90 compilers are now commercially available from the Numerical Algorithms Group (NAG), Pacific-Sierra Research, and ParaSoft. In addition, Edinburgh Portable Compilers, Lahey, SunPro, Cray, and many other companies have compilers under development which will become available over the next year or so. The Connection Machines, the CM-2 and CM-5, use a dialect of Fortran which incorporates some of features of Fortran 90, and there are several universities and companies actively working to produce compilers for High Performance Fortran that add explicit parallelism constructs to Fortran 90. With all this activity by compiler developers, it is clearly a good time to begin migrating old codes to Fortran 90 and to start writing any new codes in it. Certainly for anyone interested in high performance computing, it makes very little sense to be doing Fortran programming in anything but Fortran 90. The difficulty however is that conversion of older Fortran codes to Fortran 90, if done by hand, can be a very time intensive process and consequently an extremely expensive one.

What makes the migration of old codes complicated is that the syntax of good Fortran 90 programs is entirely different than earlier Fortran and many of the features of Fortran 90 that make it especially valuable for high performance computing are completely new and require at least some amount of code restructuring. Clearly, tools are needed to assist in this process and several have been written to at least transform the syntax of Fortran 77 codes to Fortran 90. Pacific-Sierra Research has produced a tool which goes much further however. Their VAST-90 code is an extremely powerful tool for migrating Fortran 77 to Fortran 90, going far beyond a simple translation of the syntax. Using VAST-90, I have been able to migrate literally tens of thousands of lines of Fortran 77 to Fortran 90 in an extremely short amount of time. This article discusses some features of VAST-90 and gives some examples of how it works in practice.

VAST-90

VAST-90 is really two separate codes wrapped into one, a tool for automatically translating Fortran 77 into Fortran 90, and a Fortran 90 compiler. I will discuss only the Fortran 77 to Fortran 90 translation capability in this article[4]. A complete discussion of the translation capabilities of VAST-90 would require more space than I have available, so I will merely summarize and then discuss some specific examples. In translating Fortran 77 to Fortran 90, VAST-90 converts to Fortran 90 free format source form and modifies all declarations to Fortran 90 form. Where possible, VAST-90 converts arithmetic IF's and GOTO's to block IF's, adds CYCLE and EXIT commands to loops, changes computed GOTO's and block IF's to CASE statements, changes COMMONs into MODULEs, generates and includes INTERFACE files, and collapses loops to employ Fortran 90 array syntax. VAST-90's capability for modifying Fortran 77 code to use array syntax is impressive. It analyzes IF and DO loops, performs an analysis of the data flow and data

[4]For a discussion of the Fortran 90 compiler side of VAST-90, see John K. Prentice, "Performance of Fortran 90 Compilers". *Fortran Journal*, 5(3), 2-7, 1993.

dependence of the loop, and then generates the array syntax. Two options are available for generating array syntax. The default level translates loops that will be fairly clean after the translation and skips loops whose translation would lead to code that is complicated and difficult to read code. The second option is to aggressively generate array syntax, translating loops whenever possible, regardless of how hard the translated loop is to read. The aggressive array syntax option is meant for use when array syntax has significant performance implications, for example on vector or parallel computers. Where possible, the aggressive syntax option performs scalar promotion (using allocated arrays to replace the promoted scalars) and scalar folding, carry-around scalars are collected into a scalar loop and split out of the calculation, reduction functions are translated to Fortran 90 intrinsic reduction routines (such as SUM, MAXVAL, etc...) and conditional expressions in loops are replaced by WHERE or ELSEWHERE. VAST-90 also attempts to collapse nested loops into equivalent expressions involving array syntax, re-roll unrolled loops so as to use array syntax, reorder references to avoid data dependencies (using allocated arrays for temporaries), and peel loops to remove data dependencies. Most of these translation functions are performed by default, though for more complicated loops it may be necessary to toggle the VAST-90 aggressive array syntax option to fully exploit these features. During the translation, VAST-90 will also check to make sure that the Fortran 77 is valid and terminate if it is not. In addition, an optional listing can be generated that has diagnostics about the translation process.

The process of translating Fortran 77 to Fortran 90 with VAST-90 is surprisingly fast. For example, I recently ran VAST-90 to translate a code containing 8001 lines of Fortran 77, involving 85 individual routines. Using the default translation options, VAST-90 performed the Fortran 77 to Fortran 90 conversion in 14.0 seconds on a Sparc 10, including generating interface files and modules. Using the option to aggressively generate array syntax, the translation required 20.7 seconds.

The translation technology in VAST-90 is an outgrowth of much of the work Pacific-Sierra Research has done over the years in building optimization tools for supercomputers. For example, many of the optimization and vectorization tools used by Cray Research as part of their CF77 Compiling System were developed by Pacific-Sierra Research. Their considerable experience with this technology may account for the maturity of the Fortran 77 to Fortran 90 translation capability of VAST-90 and the fact that every Fortran 77 to Fortran 90 translation I have performed with VAST-90 has been error free.

In the course of developing Fortran 90 codes for the purpose of benchmarking Fortran 90 compilers, I have used VAST-90 to convert literally tens of thousands of lines of Fortran 77 code. In general, I run VAST-90 to make an initial conversion to Fortran 90 and then go in and make additional changes to further optimize the code. While VAST-90 can be used to perform the complete conversion, my general feeling is that it is best regarded as a tool that does about 90% of the work in migrating a code from Fortran 77 to Fortran 90. The remaining 10% of the migration must often be done by the programmer and involves improving VAST-90's translation. The Quetzal

Fortran 90 Benchmark Suite[5] has several codes which were converted from Fortran 77 to Fortran 90 in this fashion, for example, the gas dynamics and overlap codes.

An Example of the Use of VAST-90

As an example of the use of VAST-90 to convert a real production scientific code from Fortran 77 to Fortran 90, I ran VAST-90 on two routines randomly selected from the widely used United States Geological Survey MODFLOW code[6]. The first routine I looked at was BCF2AD. The original Fortran 77 code is shown below. I have removed the comments from the original code in order to shorten it.

```
      SUBROUTINE BCF2AD(IBOUND,HOLD,BOT,WETDRY,IWDFLG,ISS,
     1                  NCOL,NROW,NLAY)
      DIMENSION IBOUND(NCOL,NROW,NLAY),HOLD(NCOL,NROW,NLAY),
     1          BOT(NCOL,NROW,NLAY),WETDRY(NCOL,NROW,NLAY)
      COMMON /FLWCOM/LAYCON(80)
      IF(IWDFLG.EQ.0 .OR. ISS.NE.0) RETURN
      KB=0
      DO 100 K=1,NLAY
      KB=KB+1
      DO 90 I=1,NROW
      DO 90 J=1,NCOL
      IF(IBOUND(J,I,K).NE.0) GO TO 90
      IF(WETDRY(J,I,KB).EQ.0.) GO TO 90
```

[5]This benchmark suite is available via anonymous ftp from ftp.cs.unm.edu. Log in as anonymous and give your e-mail address as a password. The benchmark suite is in the directory pub/smith-quetzal. There is a copy of the most recent benchmark report in that directory also. The report discusses each of the benchmarks and gives execution times for code generated with different Fortran 90 compilers.

[6]Michael G. McDonald and Arlen W. Harbaugh. *A Modular Three-Dimensional Finite-Difference Ground-Water Flow Model.* Scientific Software Group, Washington, DC.

```
      HOLD(J,I,K)=BOT(J,I,KB)
   90 CONTINUE
  100 CONTINUE
      RETURN
      END
```

This is a simple and short routine, but one that has some curious features. The use of the KB variable, for example, is superfluous. The IF checks in the inner loop could be condensed also. On my first try, I ran VAST-90 with the default options. The result was:

```
      SUBROUTINE BCF2AD(IBOUND, HOLD, BOT, WETDRY, IWDFLG, ISS, NCOL,   &
     &   NROW, NLAY)
!...Translated by Pacific-Sierra Research VAST-90 1.02M   17:22:14   7/29/93
!...Switches:
!-----------------------------------------------
!   M o d u l e s
!-----------------------------------------------
      USE VFLWCOM
      IMPLICIT NONE
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
      INTEGER IWDFLG, ISS, NCOL, NROW, NLAY
      INTEGER, DIMENSION(NCOL,NROW,NLAY) :: IBOUND
      REAL, DIMENSION(NCOL,NROW,NLAY) :: HOLD, BOT, WETDRY
!-----------------------------------------------
!   L o c a l   V a r i a b l e s
!-----------------------------------------------
      INTEGER :: KB, K, I, J
!-----------------------------------------------
      IF (IWDFLG==0 .OR. ISS/=0) RETURN
      KB = 0
      WHERE (IBOUND==0 .AND. WETDRY(:,:,KB+1:NLAY+KB)/=0.) HOLD = BOT(:,&
     &   :,KB+1:NLAY+KB)
      RETURN
      END SUBROUTINE BCF2AD
```

Note that VAST-90 has converted the source form to Fortran 90 free source form, including changing the comments from the Fortran 77 form to the Fortran 90 form. In addition, the FLWCOM labeled common block in the original Fortran 77 code has been automatically converted to a module, the declaration of variables has been changed to the Fortran 90 form, and VAST-90 has grouped the declarations by whether they are in modules, they are dummy arguments, or they are local variables. The superfluous KB variable is still present (you can't have everything!), but VAST-90 has correctly recognized that incrementing it in the outer loop is not necessary. In fact, there is no outer loop anymore, nor any inner loops. VAST-90 has collapsed the three nested loops in the original Fortran 90 into a single statement employing array syntax, using the WHERE construct to handle the cases where the assignment should be skipped.

The next routine I converted was SBCF1C:

```
      SUBROUTINE SBCF1C(CR,CC,TRPY,DELR,DELC,K,NCOL,NROW,NLAY)
      DIMENSION CR(NCOL,NROW,NLAY), CC(NCOL,NROW,NLAY)
     2   , TRPY(NLAY), DELR(NCOL), DELC(NROW)
      YX=TRPY(K)*2.
      DO 40 I=1,NROW
      DO 40 J=1,NCOL
      T1=CC(J,I,K)
      IF(T1.NE.0.) GO TO 10
      CR(J,I,K)=0.
      GO TO 40
   10 IF(J.EQ.NCOL) GO TO 30
      T2=CC(J+1,I,K)
      CR(J,I,K)=2.*T2*T1*DELC(I)/(T1*DELR(J+1)+T2*DELR(J))
   30 IF(I.EQ.NROW) GO TO 40
      T2=CC(J,I+1,K)
      CC(J,I,K)=YX*T2*T1*DELR(J)/(T1*DELC(I+1)+T2*DELC(I))
   40 CONTINUE
      RETURN
      END
```

This routine is interesting because the use of the GOTO statements has generated some classic Fortran spaghetti code. Running this routine through VAST-90 with the default options produced the following:

```
      SUBROUTINE SBCF1C(CR, CC, TRPY, DELR, DELC, K, NCOL, NROW, NLAY)
!...Translated by Pacific-Sierra Research VAST-90 1.02M   17:29:59   7/29/93
!...Switches:
```

```
      IMPLICIT NONE
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
      INTEGER K, NCOL, NROW, NLAY
      REAL, DIMENSION(NCOL,NROW,NLAY) :: CR, CC
      REAL, DIMENSION(NLAY) :: TRPY
      REAL, DIMENSION(NCOL) :: DELR
      REAL, DIMENSION(NROW) :: DELC
!-----------------------------------------------
!   L o c a l   V a r i a b l e s
!-----------------------------------------------
      INTEGER :: I, J
      REAL :: YX, T1, T2
!-----------------------------------------------
      YX = TRPY(K)*2.
      DO I = 1, NROW
         DO J = 1, NCOL
            T1 = CC(J,I,K)
            IF (T1 /= 0.) GO TO 10
            CR(J,I,K) = 0.
            CYCLE
   10       CONTINUE
            IF (J /= NCOL) THEN
               T2 = CC(J+1,I,K)
               CR(J,I,K) = 2.*T2*T1*DELC(I)/(T1*DELR(J+1)+T2*DELR(J))
            ENDIF
            IF (I == NROW) CYCLE
            T2 = CC(J,I+1,K)
            CC(J,I,K) = YX*T2*T1*DELR(J)/(T1*DELC(I+1)+T2*DELC(I))
         END DO
      END DO
      RETURN
      END SUBROUTINE SBCF1C
```

VAST-90 replaced most of the GOTO's with a IF-THEN-ELSE construct and indented the code so as to make it much more readable. VAST-90 has also added a CYCLE command just before the line with statement label 10. The purpose of the CYCLE command is to skip to the end of the inner loop instead of continuing on to the next executable statement. Note that the remaining GOTO could have been eliminated in this routine, but this version is still clearly an improvement over the original Fortran 77!

VAST-90 also automatically generated an interface file for this routine. Fortran 90 interface files allow the compiler to catch discrepancies in the number and type of dummy arguments, as well as in the dimension of arrays. An option exists in VAST-90 to automatically generate interface files for a subroutine, as well as to automatically include interface files for subroutines or functions called by a routine. The interface file VAST-90 generated for subroutine SBCF1C was:

```
      INTERFACE
      SUBROUTINE SBCF1C (CR, CC, TRPY, DELR, DELC, K, NCOL, NROW, NLAY)
!...Generated by Pacific-Sierra Research VAST-90 1.02M   17:28:40   7/29/93
      REAL, DIMENSION(NCOL,NROW,NLAY), INTENT(OUT) :: CR
      REAL, DIMENSION(NCOL,NROW,NLAY), INTENT(INOUT) :: CC
      REAL, DIMENSION(NLAY), INTENT(IN) :: TRPY
      REAL, DIMENSION(NCOL), INTENT(IN) :: DELR
      REAL, DIMENSION(NROW), INTENT(IN) :: DELC
      INTEGER, INTENT(IN) :: K
      INTEGER, INTENT(IN) :: NCOL
      INTEGER, INTENT(IN) :: NROW
      INTEGER, INTENT(IN) :: NLAY
      END SUBROUTINE
      END INTERFACE
```

By including this interface file in any routines that call SBCF1C, I can now be assured that the dimension of the arrays passed as dummy arguments is consistent throughout the code. Further, note that VAST-90 determined the INTENT of the various dummy arguments. In this case, the input values of the CR array are not used by this routine, but the array values are changed in it. Hence, the INTENT for this routine is OUT. By contrast, the values passed into the routine for CC are both used by the routine and then modified, making the INTENT of this array INOUT. The values passed to this routine for all of the other dummy arguments are used by the routine but not modified, thus their INTENT is IN. Declaring the INTENT of dummy arguments in this way allows for the compiler to optimize the code in ways that are impossible with earlier versions of Fortran.

Next I turned on the VAST-90 option to aggressively generate array syntax. No array syntax was generated during the earlier translation because doing so would lead to code which is difficult to read. By turning on the option to aggressively generate array syntax, I have instructed VAST-90 to generate array syntax whenever possible, irrespective of how complicated or difficult it is to read. The file it generated was:

```
      SUBROUTINE SBCF1C(CR, CC, TRPY, DELR, DELC, K, NCOL, NROW, NLAY)
!...Translated by Pacific-Sierra Research VAST-90 1.02P   12:47:33   8/05/93
!...Switches: -xr
      IMPLICIT NONE
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
      INTEGER K, NCOL, NROW, NLAY
      REAL, DIMENSION(NCOL,NROW,NLAY) :: CR, CC
      REAL, DIMENSION(NLAY) :: TRPY
      REAL, DIMENSION(NCOL) :: DELR
      REAL, DIMENSION(NROW) :: DELC
!-----------------------------------------------
!   L o c a l   V a r i a b l e s
!-----------------------------------------------
      INTEGER :: I, J, J1
      REAL :: YX, T1, T2
      REAL , ALLOCATABLE :: T11U(:)
      LOGICAL , ALLOCATABLE :: L1V(:)
!-----------------------------------------------
      YX = TRPY(K)*2.
      DO I = 1, NROW
         IF (I == NROW) THEN
            ALLOCATE (T11U(NCOL), L1V(NCOL))
            L1V = .FALSE.
            T11U = CC(:,I,K)
            WHERE (T11U == 0.)
               CR(:,I,K) = 0.
            ELSEWHERE
               L1V = (/(J1,J1=1,NCOL)/) /= NCOL
            END WHERE
            WHERE (L1V) CR(:,I,K) = 2.*CC(2:NCOL+1,I,K)*T11U*DELC(I)/(  &
     &         T11U*DELR(2:NCOL+1)+CC(2:NCOL+1,I,K)*DELR)
            DEALLOCATE (T11U, L1V)
         ELSE
```

```
            DO J = 1, NCOL
               T1 = CC(J,I,K)
               IF (T1 /= 0.) GO TO 1053
               CR(J,I,K) = 0.
               CYCLE
 1053          CONTINUE
               IF (J /= NCOL) THEN
                  T2 = CC(J+1,I,K)
                  CR(J,I,K) = 2.*T2*T1*DELC(I)/(T1*DELR(J+1)+T2*DELR(J))
               ENDIF
               T2 = CC(J,I+1,K)
               CC(J,I,K) = YX*T2*T1*DELR(J)/(T1*DELC(I+1)+T2*DELC(I))
            END DO
         ENDIF
      END DO
      RETURN
      END SUBROUTINE SBCF1C
```

As advertised, the aggressive array syntax option has produced a considerably more complicated Fortran 90 version than that produced with the default VAST-90 option. Out of curiosity, I added a driver to these two Fortran 90 versions of SBCF1C and ran them on a Sparc 10. Using both VAST-90 and the NAG Fortran 90 compilers, the execution time for both versions was nearly identical. So, on this scalar RISC computer, the aggressive array syntax option neither helped nor hurt the execution speed. On a vector or parallel computer it is possible that the version generated with the aggressive array syntax option would be faster. If the use of array syntax is not a significant issue on your computer, however, the default translation is obviously easier to read and would therefore be preferable. In this particular case, it is also possible to hand translate this routine into a form that uses array syntax, but is much cleaner than what VAST-90 has produced here. This underscores my earlier point that some amount of effort on the part of the programmer is still required to achieve the optimal translation of codes from Fortran 77 to Fortran 90.

Availability of VAST-90

VAST-90 is currently available for the IBM RS6000, SUN Sparc, HP9000, DEC 5000, Convex, Cray, and VAX VMS. Pacific-Sierra Research will port to other machines as necessary. The price for UNIX workstations is $1,250, effective until September 30, 1993. For more information, contact

Pacific-Sierra Research Corporation
Computer Products Group
2901 28th Street, Suite 300
Santa Monica, CA 90405
USA

phone: (310) 314-2300
FAX: (310) 314-2323
e-mail: info@psrv.com

Conclusions

VAST-90 makes the task of migrating Fortran 77 codes to Fortran 90 a fairly painless and efficient task. The translations are often surprising sophisticated and with a little hand tweaking, it is possible to generate extremely efficient and clean Fortran 90 from even fairly messy Fortran 77 codes. VAST-90 runs fast, it generates good code, and it is reasonably priced.