# Migrating Applications to the CRAY T90 Series IEEE Floating Point

*Doug Petesch*, Cray Research, Inc, Eagan, Minnesota

**ABSTRACT:** *As of the fall '95 CUG meeting, applications have been running in IEEE mode on the CRAY T90 for only a week. By the time the proceedings are published, more information will be available in the "Migrating to the CRAY T90 Series IEEE Floating Point" manual (SN2194 2.0). This paper documents the efforts to date of the Third Party Applications Group to help vendors of major Cray applications make their codes available in IEEE-mode. It will form the basis of a section in the SN2194 manual.*

## Introduction

This paper deals with migrating applications that already run on Cray PVP (Parallel Vector Processing) systems to the CRAY T90 series with IEEE floating point. Such applications probably don't need the added accuracy, multiple rounding modes or other special features of IEEE floating point arithmetic. However, over the years they may have developed dependencies on compiler, library or non-floating point hardware features that also have changed.

Most applications running on Cray PVP systems today are compiled with cf77 6.0 or scc 4.0 and loaded with UNICOS 7 or 8 libraries. None of these will be available on CRAY T90 systems with IEEE floating point. The first, and probably biggest step in migrating an application is to get it running on a CRAY C90 with the 2.0 Programming Environment (f90 2.0 and scc 5.0 compilers and associated libraries). This will isolate numeric differences due to the new compilers and libraries from those due to the new floating point hardware. This paper will concentrate on FORTRAN applications.

The next step is to adapt to the larger address space and other new features of all CRAY T90 systems, regardless of floating point format. The final step is to take into account the differences between CRAY and IEEE floating point. This may be the easiest step and only involve recompiling.

## CF77 to F90 Migration

F90 is a full Fortran 90 compiler. When the 2.0 version is released, it is intended to give performance equal to or better than cf77 on Fortran 77 code. Most existing applications do not need the new language features of Fortran 90. However, they may use cf77 Fortran 77 extensions that behave differently in f90. There are also some differences in Namelist and formatted I/O between cf77 and f90. For more information, see the "Fortran 90 Features and Differences" publication (TR-CF90 A).

CF77 treated variables of type POINTER almost the same as variables of type INTEGER. F90 enforces the FORTRAN 90 restrictions on what operations can be performed on pointers. Specifically, POINTER type variables cannot be multiplied or divided or be arguments to most functions.

The LOC intrinsic function returns a result of type POINTER. Applications that got away with improper usage of the LOC function with cf77 will have to be changed to run with f90. Figure 1 shows an example of code from a major application that had to be changed when compiled with f90.

```
C Illegal, but worked with cf77.
      INTEGER FUNCTION CORWDS(II,JJ)
      CORWDS = 1 + ABS( LOC(II)-LOC(JJ) )

C Correct usage in both cf77 and f90.
      INTEGER FUNCTION CORWDS(II,JJ)
C Convert result from pointer to integer.
      INTLOC = LOC(II) - LOC(JJ)
      CORWDS = 1 + ABS(INTLOC)
```

**Figure 1: LOC function usage.**

The f90 compiler optimizes code differently than cf77 did. For example, f90 enforces the rule that dummy argument arrays may not be indexed past their declared dimension, unless it is 1, 2 or '*'. If an array is dimensioned less than the vector length of the machine, f90 may treat loops referencing the array as if they were preceded by a CDIR$ SHORTLOOP directive. That can increase performance, but if the actual usage of the array extends past the declared dimension, the results are undefined.

In the example shown in figure 2, the actual number of iterations of the loop would be the remainder of 200 divided by the

machine vector length (ie. 72 on a CRAY C90 or CRAY T90 and 8 on a CRAY Y-MP or CRAY J90). Any application that uses array elements past the declared dimension of the dummy argument should change that dimension to '*', or be compiled with the f90 -Ooverindex. This tells the compiler not to perform this particular shortloop optimization.

```
dimension junk(200)
call fill(junk,200)
print*,junk
end

 subroutine fill(j,k)
 integer j(3) !should be j(1) or j(*)
    do i=1,k   !f90 treats as shortloop
    j(i)=i     ! unless -O overindex
 end do
 end
```

**Figure 2: Overindexing of a dummy array.**

Most of the changes required to get applications running with f90 involve fixing code that was illegal to begin with, but happened to work with cf77. Once those changes are made, it is possible to take advantage of the new optimization features. For example, f90 generates more efficient vector code for many types of nested loops.

## General CRAY T90 Series Features

There are other features of the CRAY T90 series, independent of floating point format, that need to be considered when migrating applications. Like the CRAY C90, it has 128 element vector registers compared to 64 on CRAY Y-MP systems. Assembly code that uses vector mask instructions on a CRAY Y-MP may have to be modified. Also like the CRAY C90, it has vector shift instructions that along with the longer vector registers make the order of additions different in reduction loops. This may cause numeric differences compared to a CRAY Y-MP even on a CRAY T90 system with CRAY floating point.

All CRAY T90 systems have a scalar cache. Most third party applications available on Cray T90 systems with Cray floating point run in CRAY C90 compatibility mode. They still get the advantage of the cache when run on a single processor. However the operating system must turn off the scalar cache if such an executable is multitasked and run with NCPUS set to a value larger than 1. Executables built in native T90-mode, with either Cray or IEEE floating point, will get the advantage of the cache even when run in parallel. The multitasking libraries make sure that each processor's cache is synchronized with main memory at appropriate points.

CRAY T90 systems have a larger address space than previous CRAY PVP systems. Address registers are 64-bits compared to 32-bits for the CRAY C90 or CRAY Y-MP. Certain scalar memory instructions require 4 parcels in T90-mode instead of 3. This can cause a slight loss of performance in scalar sections of code.

A major consequence of the larger address space is that a Fortran character descriptor (FCD) must be 2 words instead of 1 as it has been since the Cray-1. FCDs are used to pass Fortran character variables from one routine to another. Although it has always been illegal, there are times when it works to pass character actual arguments to integer or real dummy arguments or vice versa on previous CRAY PVP system. Figure 3 shows an example of such a mismatch of argument types that will cause errors in T90-mode.

```
C Illegal, but works on C90, Y-MP, J90.
C Fails on T90 because of 2 word FCD.
    CHARACTER*8 concat
    I=4Htest
    J=4Hcase
    CALL cat(I,J,concat)

    SUBROUTINE cat(I,J,concat)
    CHARACTER I*4, J*4, concat*(*)
    concat = I // J
    end
```

**Figure 3: Mismatch of dummy and actual argument types.**

The cft77 -Ra run-time argument checking option can be used to locate any such occurrences of mismatched types between caller and callee within Fortran. A similar error can occur when calling a C routine with a Fortran character variable as an actual argument. The _fcd macros in <fortran.h> must be used in the C routine to properly declare the type of the argument and to convert Fortran character variables to C strings and back. These procedures were formally required on previous CRAY PVP systems too, but some applications did get by without using them.

A number of Fortran-callable library routines that exist on CRAY C90 and CRAY Y-MP systems allow some arguments to be either character variables or Hollerith argument (characters packed in an integer or real word). Some of these routines (ex. FOPEN, FREAD, FWRITE, etc.) only allow character type actual arguments on CRAY T90 systems. The rest are not available at all (ex. GETOPT, GETENV, UNAME, STAT, etc.). Instead, new entry points with similar functionality have been created. When possible, the new interface routine follows the interface description provided in IEEE Std 1003.9-1992. The new entry points are available on all CRAY PVP systems.

## IEEE Floating Point Specifics

Once an application runs correctly using the new compilers on a CRAY T90 system with Cray floating point, the additional changes required to make it work on IEEE CPUs should be relatively minor. Any bit manipulations, including data initialization, of floating point values must adapt to the different number of mantissa and exponent bits. Figure 4 shows the IEEE 64-bit

format compared to the CRAY floating point format. Note that the mantissa in both formats is normalized. In IEEE the leading mantissa bit is implied, but not stored, which effectively gives it a 53-bit mantissa.

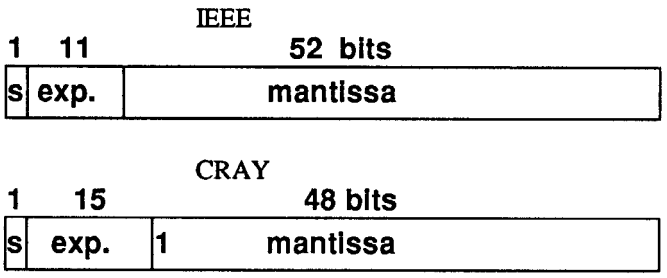


**Figure 4: 64-bit floating point formats.**

Generally, the extra mantissa bits make the IEEE mode calculations more accurate. Therefore, some tolerances used for integration or iterative algorithms may have to be changed. However, the smaller exponent field means 64-bit IEEE numbers are limited to a magnitude of about 10**308 compared to 10**2465 for CRAY floating point numbers.

The IEEE mode CRAY T90 CPUs support an instruction for SQRT. They also have a divide instruction instead of the CRAY floating point reciprocal approximation. These instructions could increase the performance of codes that perform many square roots or divides.

The IEEE mode compilers and libraries for the CRAY T90 series support the additional numeric concepts defined in the IEEE standard such as rounding modes, NaNs (Not a Number), signed infinity and signed zero for programmers who wish to use them. Comparison operations have to deal with these end cases so they may take slightly longer than the equivalent operation on a CRAY floating point CPU.

The IEEE standard does not define a 128-bit format. The CRAY T90 system uses the same 128-bit format as on SPARC systems. This format has more exponent bits than the 64-bit IEEE format. That means the first word of a double precision term will not be a valid single precision number like on previous Cray PVP machines. That means double precision arrays can't be passed as stride 2 single precision input arrays. Some shifting of bits is required to convert between single and double precision. Figure 5 shows the 128-bit double precision format used on IEEE systems compared to the CRAY floating point format.

The internal representation of Fortran LOGICAL variables is also different on CRAY T90 systems with IEEE floating point. It matches the representation used on Cray MPP and SPARC machines and in the C language. The logical true/false test is whether the word is non-zero or zero instead of negative or positive. Any inter-procedural aliasing of integer and logical variables that relies on the sign bit for the true/false test will have to be changed.

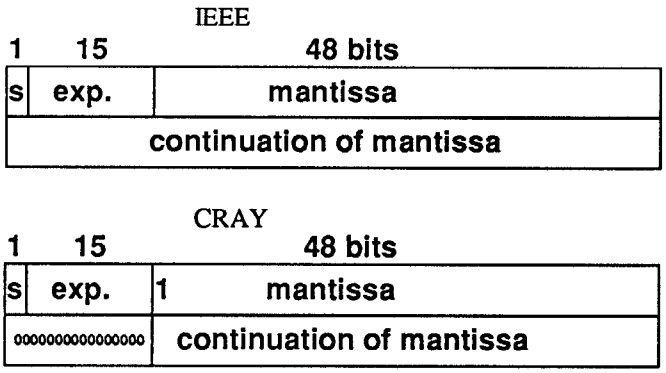


**Figure 5: 128-bit Floating point formats.**

## Data Conversion

CRAY T90 systems with IEEE floating point can potentially share unformatted data files more easily with other IEEE machines than previous Cray PVP machines could. However, the size of all integer and logical values is 64-bits on Cray systems and usually 32-bits on non-Cray systems. The UNICOS assign command and explicit data conversion library routines can often be used to share unformatted files with 32-bit IEEE as well as Cray floating point machines.

## Applilcation Case Study

ADAMS is a multibody dynamics code for modeling mechanical systems. The small sparse systems of equations can be quite stiff so it uses an implicit predictor-corrector time integration scheme. It has run on Cray systems for 15 years, but has always been numerically sensitive. Mechanical Dynamics, Inc. (MDI) has tried various error tolerances, but the Cray version often takes smaller time steps than when using 64-bit IEEE floating point.

ADAMS v7.0 has about 2500 routines and 500,000 lines of code. The existing Cray version called 3 of the Fortran interface routines not supported on CRAY T90 systems. After making changes to call alternate routines and working around 1 f90 bug that has since been fixed, ADAMS compiled and built correctly in IEEE mode the first try. Two tolerance values that had been changed by several orders of magnitude on previous Cray systems were restored to what MDI uses on other IEEE machines. ADAMS was also compiled in 128-bit double precision on the CRAY C90 with f90. In that case, the tolerances were changed by several orders of magnitude the other way.

The results of running a small test case in IEEE mode on a CRAY T90 system matched the 128-bit CRAY C90 results almost exactly. Both required fewer time steps than the 64-bit CRAY C90 run whose results differed by a few percent from the first 2 runs.