Subject: IDL & F90
Posted by carohde on Tue, 01 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Has anybody used IDL in conjunction with Fortran 90? I can get them to pass values, but not arrays. I'm using a C style passing convention (compiler flag) but I still can't seem to resolve the arrays correctly. I've dumped the pointer locations and the mem. address of the arrays seems out of place (differing by several orders of magnitude from the pointers to the simple integers). This is either extremely lousy memory management on NT's part ( something I wouldn't put by it) or something is wrong... I suspect the latter because when I try to resolve the array pointer, it  biffs and seg faults on me. Any ideas would be helpful.

Thanks

chuck

---------- ----------

Charles A. Rohde  E-mail:
623 Main St. Apt 1  carohde@mtu.edu
Dodgeville, MI 49921
Phone: (906)487-5401  Finger: carohde@pacemaker.cts.mtu.edu

,,,,
"Imagination is more important than knowledge" - A Einstein  (O O)
---------- --o0o(_)o0o---

Subject: Re: IDL & F90
Posted by William Clodius on Tue, 01 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Chuck Rohde wrote:
>
> Has anybody used IDL in conjunction with Fortran 90? I can get them to pass
> values, but not arrays. I'm using a C style passing convention (compiler
> flag) but I still can't seem to resolve the arrays correctly.
> <snip>

I have not yet tried interfacing F90 code with IDL, but have considered doing so and have looked at some of the issues. Some things to consider

0. Always specify the compiler you are using. I think there are four or five vendors with F90 compilers for Windows NT.

1. Name mangling by the Fortran compiler, F77 or F90. While name

mangling for Fortran is not as bad as for C++, some name mangling is done to avoid accidental name conflicts with system routines and to consistently handle upper and lower case names. Things to look for include leading and trailing special characters (usually, but not always, underscores) and case conversions.

2. Name mangling and F90 modules. Some F90 compilers handle scoping rules for entities in F90 modules by prepending or appending the module's name to that of public procedures, derived types, or entities.

3. No name mangling in C calling conventions. It is possible that in specifying C calling conventions with their more limited name mangling, a conflict is generated with a system procedure.

4. Long variable names.  Both IDL 4.+ and F90 nominally accept names of about 31 characters, but the name mangling issues above and the potential use of hashing to resolve name conflicts might cause additional problems for long names.

5. Watch how you declare array arguments in F90. Some of the newer syntaxes require that the compiler pass more information to the procedure than just the address of the array. For example, specifying

```
SUBROUTINE EXAMPLE(X)
REAL :: X(:)
...
```

the colon in X(:) indicates to the compiler that X is a one dimensional array and that it will be using its conventions to pass the array limits. Those conventions are typically to create a small structure containing the array dimensions and the starting address of the array and pass as the argument a pointer to the structure. Such structures are commonly termed array descriptors, and are probably similar to the structures IDL uses to pass array and type information. Note, however, that F90 only says what information needs to be passed, and does not state the details of the implementation. Some Fortran compilers, i.e., those which use message passing over networks of machines, can and will use different methods. You need to consult your manual to find the details of the implementation. If it is an array descriptor, you probably need to use the argument pasing mechanism for the equivalent C struct. This sounds like the most likely problem, but I can't be certain.

6. Be careful with derived types. Unless the derived type has the sequence attribute the compiler has great liberties in its implementation. For example an array of a derived type can be, and sometime is, implemented as separate arrays of each component. Read the manual, but do not be surprised if you need to specify the sequence

attribute, and have to lay things out carefully to avoid padding.

7. Be careful to use explicit interfaces in F90 code. Unless it has access to such interfaces the F90 compiler is allowed to assume F77 calling conventions. This increases the portability of old Fortran code, but can cause problems with new code. I suspect this causes more problems interfacing to other F90 code than to IDL code, but you can never be certain. Explicit interfaces can be provided by INTERFACE constructs, including the procedure in the same module, or including the procedure in a module and using the module.

8. If you can at all avoid it don't pass Fortran character strings to and from other languages. The compiler typically needs to pass some very simple additional information for these arrays, i.e., the length of the string. The information is so simple, that there seems to be a different method on each compiler. Depending on the compiler it might create a simplified version of an array descriptor and pass a pointer to the descriptor, or pass the descriptor by value, or pass the string length as a hidden argument, prepended to the argument list, appended to the argument list, preceding the pointer to the string, following the pointer to the string, etc. The hidden argument implementation, in particular, can cause maintenance nightmares if you modify the argument list.

9. Examine closely the meaning of "C style calling conventions". Does that mean that the F90 procedure can be called easilly in C, or does that mean that in certain situation some of the procedures it calls are assumed to be generated by a C compiler? Are the C conventions those of the system or those of the vendor? Note also that the C style calling convention typically is to pass scalars by value, but you seem to be implying that your scalars are passed by pointer.

--

William B. Clodius  Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2  FAX: (505)-667-3815
PO Box 1663, MS-C323  Group office: (505)-667-5776
Los Alamos, NM 87545  Email: wclodius@lanl.gov

---

Subject: Re: IDL & F90
Posted by nhbkmich on Wed, 02 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Chuck Rohde (carohde@mtu.edu) wrote:

: Has anybody used IDL in conjunction with Fortran 90? I can get them to pass
: values, but not arrays. I'm using a C style passing convention (compiler

: flag) but I still can't seem to resolve the arrays correctly. I've dumped
: the
: pointer locations and the mem. address of the arrays seems out of place
: (differing by several orders of magnitude from the pointers to the simple
: integers). This is either extremely lousy memory management on NT's part
: ( something I wouldn't put by it) or something is wrong... I suspect the
: latter
: because when I try to resolve the array pointer, it  biffs and seg faults
: on me. Any ideas would be helpful.

: Thanks

: chuck

Hi Chuck!

Yes, I've used IDL's call_external to Fortran 90 several times successfully. This was under Unix, so I don't really know whether my experiences are of any help for you. But maybe there are some similarities.

First, passing arrays with assumed-shape arguments on the f90 side will be very difficult, due to the dope vector passing mechanism, f90 usually uses with this kind of argument. Your weird pointer values are possibly caused by the f90 routine interpreting the pointer to the array as a pointer to a dope vector. Use fixed-size or assumed-size instead.

Since no f90 compiler, I have access to, can handle the C calling convention, I always put a C wrapping function in between and compile the f90 part as usual. Example:

A Fortran 90 function, returning the product of all elements of an array:

```
  FUNCTION prod(n, array)
    IMPLICIT NONE
    INTEGER             :: prod
    INTEGER             :: n
    INTEGER, DIMENSION(n):: array

    prod = PRODUCT(array)

  END FUNCTION prod
```

The corresponding C wrapper. You will probably have to use a different name than "prod_" for the f90 function, according to the naming conventions of your f90 compiler:

```
  int prod_(int *n, int *array);  /* f90 prototype */
```

```
int prod_argv(int iargc, void *argv[])
{
  return prod_(argv[0], argv[1]);
}
```

These functions are compiled and put into a shared object (called DLL under Windows?). I chose the name "prod.so" here. The call from IDL then looks like:

```
FUNCTION prod, array
  RETURN, CALL_EXTERNAL('prod.so', 'prod_argv', $
                N_ELEMENTS(array), $
                LONG(array), $
                VALUE = [0b, 0b])
END
```

This example works with IDL 4.0.1 under HP-UX 10.20 together with NAG f90 (and a libf90.sl shared library, kindly supplied by NAG on request. For some reasons, most vendors for HP compilers apparently think that dynamic linkage is not of interest with f90...)

Hope this helps and I'm curious about your experiences!
Michael
--
********** **********
Michael Steffens Institut f. Meteorologie u. Klimatologie
Tel.: +49-511-7624413 Universitaet Hannover
email: Michael.Steffens@mbox.muk.uni-hannover.de Herrenhaeuser Str. 2
steffens@muk.uni-hannover.de D-30419 Hannover

PGP fingerprint = FA BE 6C 1C F6 C3 EC 33 DD 42 6B 7F DE CF 84 B8