Subject: Re: Compiling IDL ... ever likey ?
Posted by steinhh on Mon, 22 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <4dttc7$cmb@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) writes:
|>
|> James Tappin <sjt> writes:
|>
|> >It stikes me that the biggest technical problem would be handling the way IDL
|> >is able to change the type of variables without the user knowing explicitly
|> >that it is doing so. ...
|>
|> Actually, there's nothing to keep a "compiled" version of an IDL program from
|> acting the same way as in the normal version of IDL. Such a compiled
|> executable would consist of two parts:
|>
|> 1. A program section, which contains a restricted version of the basic IDL
|>   executable. It would be able to do everything IDL can do, except compile
|>   procedures, or accept commands from a command line.
|>
|> 2. A data section, which contains the IDL procedures to be executed, in a
|>   binary interpreted format--presumably the same format that IDL stores the
|>   procedure in a SAVE file.
|>
|> Such an object would act just like an executable--i.e. it would be a single
|> file that one would simply run--but it would preserve all the
|> interpretive-language advantages that IDL currently has. It would still be
|> IDL.
|>

In fact, every interpreted --- uh, most! (self-modifying languages such
as lisp are a bit awkward) -- programming languages can be compiled.
I guess the IDL interpreter goes in a loop that looks something
like

```
  WHILE <program-not-ended> DO BEGIN
     TOKEN = PROGRAM_STORE(PROGRAM_COUNTER)
     PROGRAM_COUNTER = PROGRAM_COUNTER + 1
     CASE TOKEN OF
       <ASSIGNMENT_OPERATION> : DO_ASSIGNMENT(PROGRAM_COUNTER)
       <PRINT_STATEMENT>: DO_PRINT_STATEMENT(PROGRAM_COUNTER)
       :
       :
      END

  END
```

The various functions like DO_ASSIGNMENT or e.g., EVALUATE_EXPRESSION

have branching code to deal with different data types/dimensions etc.

In the same way that this program is possible to compile into somehing
machine executable, the *tokens* themselves could have been compiled
into executable machine code (e.g., mainly just subroutine calls).
This is not very difficult, but there's not much time saved!

|> Some people might be disappointed that the performance would be the same.
|> There would not be the performance increase that compiled binaries typically
|> enjoy.

The key to improving performance is declaring the type and
dimensionality of the data that are to be manipulated. Very often,
IDL subroutines are made to deal with very specific data,
but there's no way to tell IDL about this -- it has to do all
the checks all the time. In the survey about the future of IDL
I suggested the possibility of having "pseudocode blocks", where
all the data to be manipulated are declared in the beginning.
If some of the input data do not match the declaration, a
runtime error occurs.
The statement part of the code would look like e.g., F90, and
it would be quite easy to compile into native machine language.
This is especially suitable for array operations that are not
possible to do without FOR loops -- there are some, consider:

```
B(0) = A(0)
for i=1,N-1 do B(i) = B(i-1) + A(i)
```

or:

```
tmp = 0.0
for i=0,N-1 do tmp = tmp + A(i) * B(i)
```

(In order to get array performance on the last one, one would
have written tmp = total(A*B), but IDL translates this to
temp = A * B
tmp = total(temp) ; -- which is a waste of time and space!)

Stein Vidar
(In the hope that this will some day come true)
