Subject: Re: Compiling IDL ... ever likey ?
Posted by steinhh on Fri, 26 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <4e8p4h$9oe@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) writes:
|> steinhh@amon.uio.no (Stein Vidar Hagfors Haugan) writes:
|> > ... 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.
|>
|> Yeah, but then it wouldn't be IDL. You might as well write it in FORTRAN at
|> that point, IMHO.
|>

Hi there Bill,

True, it wouldn't be IDL, and the language could just as well look (something) like FORTRAN, but it would be *inside* IDL, and you needn't tell anyone to compile the source code, place the shareable objects anywhere special, or...

|> Almost all the IDL code that I write expects to be able to ingest data in a
|> variety of data types and dimensionality. That's what I like about IDL, and
|> a good part of why I use it.
|>

I agree. But many (maybe most) people write IDL programs for their own use only, and have no need to supply programs that cope with "everything". For many computationally intensive applications that people write, it's only one kind of data that's going through the pipeline. The sizes of the arrays may change, but seldom the number of dimensions, the array types etc.

|> 1. To be able to distribute IDL code without having to require other people to
|>   buy IDL. It was that possibility I was considering in my previous post. I
|>   think that it is perfectly possible to do this, and still let IDL be IDL.

And I agree that from this perspective, no compilation is necessary (and it would probably be very bug-prone as well).

|>
|> 2. To speed up execution time on tasks that cannot easily be vectorized (or
|>   which are not efficiently written). I don't see anyway of doing this
|>   without making fundamental changes in the way IDL works.
|>

I don't want to change the existing functionality of IDL. I just want
to be able to write something like:

```
------------------------------
 ;; Normal IDL
 a = findgen(10)
 b = findgen(10)

 :
 ;; Manipulations of a and b (preserving the type)
 :

 tmp = 0.0;

 compileblock( C : FLTARR(N:INTEGER) = A, $  ; Declarations and "name assosiation"
               D : FLTARR(M:INTEGER) = B, $  ; If A, B or TMP don't fit the bill,
               T : FLOAT = TMP)              ; we want a run-time error.
    i : INTEGER
 begin
    ;; Here goes the compiling statements

    IF M NE N THEN ASSERTION_FAILED("C and D unequal size")
    FOR i = 0,N-1 DO T = T + C(i)*D(i)

 endb

 ;; Normal IDL again
 PRINT,TEMP  ;; Has the value of TOTAL(A*B)
             ;; But *without* calculating temp = A*B
             ;; and then taking TOTAL(temp)

------------------------------
```

The key here is that even for operations that are possible to
vectorize, IDL wastes a lot of time because it's an interpreting
language: Ok, multiply A and B. Let's see: A is a float array, 10
elements, and B is a float array, 10 elements, so the result will be
float, 10 elements: allocate space for that. Do the multiplication and
store the result element by element. Done. Now, take the total of the
temporary. Let's see, what was it again, oh yes, it's a 10 element float,
so i'll use the code for adding up contiguous floats, 10 pieces of it.

All this "figuring out", plus actually storing the temporary takes time
and space.

Doing an atomic array function in IDL is *extremely* optimized, though:
I tried to beat IDL's array operations once, when I was doing some
Fourier filtering of real (as opposed to complex) data. I used every

trick in the book (the book being Num. Recipes), but I didn't gain anything (it might have been a few per cent).

On the other hand, when wanted to sum up some square differences between two arrays with a windowing function, i.e., taking

chisq = sum-over-i [(A(i)-B(i))^2*1.0/(1.0+i^2/const)]

then the call_external code (in C) beat the hell out of IDL by a substantial factor.

If the "pseudocode" language is kept very simple, then it shouldn't be difficult to compile such operations into quite efficient code (although not as optimized as IDL's own).

In my opinion, the fact that call_external exists indicates that there is a need for *both* high-level IDL *and* in some cases a low-level compiling language, so why not lower the threshold a little?

Of course cost is a concern, but if PV-WAVE gets this pseudocode and IDL doesn't, I know which one I'd choose (if starting from scratch, at least) for serious work.

The other thing that could change my mind is supplying a proper handle/pointer syntax. It's such a waste of screen space writing

```
HANDLE_VALUE,ID,A,/NO_COPY
PRINT,A.MESSAGE(5)
HANDLE,ID,A,/SET,/NO_COPY
```

instead of simply

```
PRINT,A^.MESSAGE(5)
```

I don't need pointer arithmetic or anything, nor do I need to be able to point to elements inside an array, I just want a compression of statements like the two extra lines above down to *one* character. Please!

Stein Vidar