Subject: Re: compile a routine wich inlude a commun
Posted by JD Smith on Mon, 23 Jan 2006 18:19:01 GMT
View Forum Message <> Reply to Message

On Mon, 23 Jan 2006 08:03:48 -0700, David Fanning wrote:

> Maarten writes:

>> Non-resizable arrays
>> in objects (or structures for that matter). Aargh! And as a remedy:
>> let's introduce pointers. Why do they think I would use a
>> scripting/non-compiling language in the first place!
>
> To avoid pointers!? Are you a Luddite? Pointers are the coolest thing *in*
> IDL. Global, sticky, variables that act *exactly* like any other IDL
> variables. Fantastic! I think almost everyone would agree it is one thing
> RSI got *exactly* right.

I think he means pointers are a kludge for extensible arrays. In scripting languages like Python and Perl, arrays, structure members, object data, etc., are all extensible without any special tricks. You simply don't have to ask yourself "is this an array which will be fixed in size, or change later on, in which case I should use a pointer". *But*, and this is a big but, all that flexibility comes at some real cost in speed, which grows with the data size, perhaps non-linearly.

This is the real reason to use IDL, which certainly has many warts, and is surprisingly inelegant at some things compared to more modern object-oriented/scripting languages: for a non-statically-typed language, it is very fast at basic array operations.

Of course, Python has NumArray (and Numeric, and NumPy), and Perl and PDL, but these are much more IDL-like in the restrictions they place on you. It might be nice to have access to all the higher-order magic of push, pop, shift, and unshift on any array/structure in IDL, with the exception of those arrays you'd like to operate on quickly, but this setup begins to look more and more like IDL's "use a pointer if you want to extend it" design.

>> After half a year of using IDL I really wonder why anyone with half a
>> sane mind would use IDL for new projects. Let's enumerate the reasons to
>> use IDL:
>>
>> 1) Legacy code
>> 2) Can't afford Matlab
>> 3) Struggled to learn it, afraid to throw away that time to learn
>> something else

>> 4) Popular in the field of interest, so we struggle together, with a lot
>> of code available
>> 5) Haven't looked too well at other options 6) Masochism, believe others
>> when they say that IDL is really powerful

I'd add:

6) Want to share code which just runs with colleagues, avoiding the
package dependency and moving target problems of roll your own
solutions like Python + numarray, or numeric, or numpy, ...

Of course, this should include a footnote of {Rich colleagues who can
afford IDL licenses}.

By the way, for the interested, STSci has a nice IDL<->Python/numarray
mapping page:

http://www.stsci.edu/resources/software_hardware/numarray/id l2numarray

See also this extension of that page:

http://www.johnny-lin.com/cdat_tips/tips_array/idl2num.html

Another thing you'll notice with most of these packages (and, sadly,
even GDL) -- plotting is typically a compromise, borrowing a
pre-existing package like GnuPlot, or matplotlib, not very cleanly
integrated. It's a real pain to integrate decent graphics in a
compatible, cross-platform way. I think this problem will eventually
be solved, but for now, if I send you a Python+numpy+matplotlib
script, it probably wouldn't run out of the box.

> Yes, IDL is a messy language. But have you looked at programs you wrote 10
> years ago? 20? *28* years ago! Imagine keeping those programs you first
> punched on card decks backward compatible. Imagine trying to add new
> programming concepts to an old language. Yes, it is messy and compromised
> and .... well, you fill in the blank. I'm sure it is all that.
>
> Yet, there is no better alternative for a number of users. IDL objects are
> inelegant, agreed. They are far from a perfect implementation. But they
> bring additional power and capability to a language that can use them.
> I've certainly written programs with them that I didn't think were
> possible in IDL. So, I like them despite their obvious limitations.

IDL implements the 90% of object-orientation that is actually useful.
Encapsulation is probably the biggest missing thing. To Python users,
the fact that everything is not an object is grating, but most of the
benefits of OOP are there. Not pretty and seamlessly integrated, but
there. In fact, IDL's OOP is somewhat similar to SmallTalk, regarded

by many as the cleanest and simplest original implementation of OOP.

>> The code given there beyond the explicit loop I use here, is so hard to
>> read, (and therefore hard to maintain), that I simply put up with the
>> slow, but readable, explicit loop. I tried to rewrite one of the faster
>> algorithms shown there to a 2D version, but got nowhere. Any programming
>> language that forces you to write code that hard to read, has
>> fundamental problems, and IMHO should be avoided.
>
> Well, I guess this is my fault. I partly put that Drizzling page up there
> *because* it is so hard to understand. I certainly don't understand it.
> It's one of my little Coyote jokes, if you want to know the truth. But it
> would be hard to fault the elegance and simplicity of the small examples
> to illustrate the IDL Way page:
>
> http://www.dfanning.com/idl_way/smallexamples.html
>
> I don't know Python, but I would enter the examples found on that page in
> any Elegant Programming contest and expect to have a chance at winning.

As one of the perpetrators of that page, I have to agree, these examples (and many of the IDL Way) are not terribly obvious. Some have maintenance concerns, to be sure. But, they enable you, in a typeless language, to obtain the kind of speed of operation on large (many MB to many GB) piles of data that are simply otherwise unheard of.

Moreover, a elegant Python Drizzle would probably run 10x slower even than the straightforward loop-based IDL drizzle. At some point, you give up and write it quite simply in C, spending 95% of the time and C code figuring out how to communicate the results back with IDL. So, I agree with the original poster that the algorithms mentioned, among many others in IDL, are not at all transparent, while simple versions of the same are not at all fast. However, in my experience, this is the price you pay in the tradeoff of elegance and raw speed.

I think if RSI wants to do one thing to move the tradeoff forward, they should take MAKE_DLL and vastly expand its scope, allowing you to trivially stick *simple* bits of C-code callout which operate directly on IDL data in memory. Python has several projects aiming to do just this, and if any of them become standard, this may change the scientific coding landscape significantly.

JD