Subject: Re: compile a routine wich inlude a commun
Posted by David Fanning on Sun, 22 Jan 2006 14:21:12 GMT
View Forum Message <> Reply to Message

L. Testut writes:

> I am a beginner with IDL and I really enjoy this programmation
> langage, even if it is not easy at the beginning. I'm building at the
> moment an application to work with satellite altimetric data of four
> satellites.
> My question is "it is possible to compile a procedure or function
> including a COMMON which is not already defined" in others words : "is
> it possible to force compilation of procedure or function including a
> COMMON which is not already defined" ?
>
> Your answer will probably be : don't use common at all ! (but I don't
> know how to do without common)

Well, I admit I don't know much about common blocks,
having only used them a handful of times in
20 years of IDL programming, but I am perplexed
by your question. I keep asking myself, "How could
you NOT compile a procedure or function including
a COMMON which is not already defined?" Where else
would it be defined?

So, there must be some kind of problem you are
running into that I can't envision. What are you
trying to do, and what is preventing you from
doing that? Are you certain it is a COMMON block
that is preventing compilation? Why?

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: compile a routine wich inlude a commun
Posted by L. Testut on Sun, 22 Jan 2006 20:13:13 GMT
View Forum Message <> Reply to Message

Dear David,
I must explain you a bit more what I want to do.
I want to build an application which is able to treat the data of 4

satellites (I want my routine to be satellite independant). For the moment I'm just concerned with the reading of the data (the first step of my application). Data of each satellite is stored in a different directory that contain a certain number of binary files + an ASCII file (info_file.txt) containing information on how to read the binary files.

Then I've wrote a program (create_common.pro) that decode the info_file.txt and create an ASCII file : "common_base.pro" (a kind of batch file) with all the information I need to read the binary files + the coordinates of the point, + the names of all the parameters, and so on... (I define 3 common on my batch file common_base.pro)

```
*****************************
common_base.pro
  common info_base, para1, para2, ect....
  common info_head, head1, head2, ...
  common info_point, point1, point2,...
*****************************
```

I have written differents routine to decode the data, defined a sructures, plot the data... But many of them need one of the common to work.
for example:

```
PRO convert_binary_files, my_file, ...
COMMON info_head
....
END
```

So when I work with the IDL command line there no problem, because I first compile and execute the create_commo.pro, this operation create the file common_base.pro (that defined the different common I need for the other program). Then I load this common_base.pro (@common_base) and then from this point I can compile and execute all my other programs.

BUT..the problem arise when I try to put all this routines in a project because I can't compile all the files because in these case the common_base.pro has been created yet !

I think the best solution is probably to avoid the use of common, but I don't see exacly how to do that as simply as with the use of common ?

Cheers,
L. Testut
PS: I've bought your book this month, and I really enjoy to read it, but my wife thinks it's a bit strange to read a programming book in my bed before to sleep !

Subject: Re: compile a routine wich inlude a commun
Posted by Robert Barnett on Sun, 22 Jan 2006 21:37:42 GMT
View Forum Message <> Reply to Message

I don't think that many of us use @ to have inline IDL code. @ has a
few limitations as you have shown. Why don't you just have a procedure
called create_common? The procedure should be compiled on the fly when
it is called at runtime. If it is not compiled on the fly, then you
should check that you have set up your IDL paths correctly. You can
also force recompilation using RESOLVE_ROUTINE.

```
****************************
common_base.pro
pro common_base
  common info_base, para1, para2, ect....
  common info_head, head1, head2, ...
  common info_point, point1, point2,...
end
****************************
```

If you want to be really sneaky, you can use the structure autoload
feature to set up your common block. Admittedly, you might need a bit
more experience with IDL before trying this one at home.

```
****************************
common_base__define.pro
pro common_base__define
common, common_base
common_base = {common_base, head1: 23, head2: 34}
....
end
****************************
```

Robbie

Subject: Re: compile a routine wich inlude a commun
Posted by David Fanning on Sun, 22 Jan 2006 23:57:05 GMT
View Forum Message <> Reply to Message

L. Testut writes:

> PS: I've bought your book this month, and I really enjoy to read it,
> but my wife thinks it's a bit strange to read a programming book in my
> bed before to sleep !

My wife thought it strange that I got up in the middle
of the night to write it. :-)

The kind of application you are talking about would
be perfect for objects. You can easily abstract a
common object for your four satellites (each is associated
with a filename, has a defined header size, returns
data, etc., etc.). But then each satellite data object
is subclassed from this common object, and implements
the particulars for each individual satellite. With
objects, the interface to the objects is exactly
the same, but the details are different internally.
This means you could add a fifth satellite without,
for example, having to change any of your underlying
application code. This is MUCH smarter than common
blocks, and it allows you to extend your application
easily in ways you do not yet anticipate.

I'm available for consultation, if you need help with this. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: compile a routine wich inlude a commun
Posted by Maarten[1] on Mon, 23 Jan 2006 09:51:14 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> L. Testut writes:
>
>> PS: I've bought your book this month, and I really enjoy to read it,
>> but my wife thinks it's a bit strange to read a programming book in my
>> bed before to sleep !
>
> My wife thought it strange that I got up in the middle
> of the night to write it. :-)

<Grin>

> The kind of application you are talking about would
> be perfect for objects. You can easily abstract a
> common object for your four satellites (each is associated
> with a filename, has a defined header size, returns

> data, etc., etc.).

I take your word for it, but each time I picked up the RSI introduction to object programming in IDL, I threw it away in utter disgust after a few pages. With a object programming background in Objective-C (That is when I really "got" objects) and later in Python, the objects in IDL feel like hacks, and ugly ones at that. I do have your book, and it must be better than RSI's, but that doesn't really change the underlying ugliness.

Especially python puts the IDL objects to shame. 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!

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

When I got my current job, I was asked if I knew IDL. "No". "OK, fine", was the reply, "what tool have you used until now?" I said: "Igor Pro", to which I received: "Ah, the toy. IDL is much more powerful, but you'll learn it easily enough." After half a year I can honestly say that it is IDL that is the joke, that Igor is way ahead in interactive use, exploratory abilities, graphical abilities, and ease of use. OK, the latter may come from several years of use, but teaching first year students how to use Igor has shown me that its metaphors are easy to grasp. (And the final swich to Origin that I witnessed taught me that Origin is a joke unfit for any use, way worse than IDL).

I fall into category 4, but would switch to a python based solution as soon as someone puts together a working package, with a user interface that is close to that of Igor Pro.

Let me give one clear example, a simple regridding algorithm:

```
idx0 = long(lat/5.)
idx1 = long(lon/5.)
result = fltarr(long(180./5.), long(360./5.))
```

```
for ii=0,n_elements(data)-1 do $
   result[idx0[ii],idx1[ii]] += data[ii]
```

This is the 2D version of
   http://www.dfanning.com/code_tips/drizzling.html
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.

Does this make me popular in this newsgroup? Does it give me a chance of getting answers here? *ploink*, I guess.

Does it make me feel better? Yes, certainly.

Maarten

Subject: Re: compile a routine wich inlude a commun
Posted by Paolo Grigis on Mon, 23 Jan 2006 13:25:47 GMT
View Forum Message <> Reply to Message

Maarten wrote:
>
> [...]
>
> 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

Or, maybe, 7) one has learned to "think" the IDL Way, and now it is everything else that feels strange... ;-)

Ciao,
Paolo

Subject: Re: compile a routine wich inlude a commun
Posted by L. Testut on Mon, 23 Jan 2006 14:45:41 GMT
View Forum Message <> Reply to Message

Thanks Robbie and David,
I will consider the use of the structure autoload feature when I'll
have more skill on IDL ... or a Fanning consultation :)
I return to the book to understand more about your answers
Cheers,
Laurent

Subject: Re: compile a routine wich inlude a commun
Posted by David Fanning on Mon, 23 Jan 2006 15:03:48 GMT
View Forum Message <> Reply to Message

Maarten writes:

> I take your word for it, but each time I picked up the RSI introduction
> to object programming in IDL, I threw it away in utter disgust after a
> few pages. With a object programming background in Objective-C (That is
> when I really "got" objects) and later in Python, the objects in IDL
> feel like hacks, and ugly ones at that.

I know how you feel, Maarten. The Denver Broncos got
crushed yesterday in the AFC Football Championship and
most of the folks in my neighborhood think the world is over.
It will get better with time. :-)

And look at the bright side, we don't have to listen to the
Super Bowl hype for the next two weeks!

> Especially python puts the IDL objects to shame.

I'm sure of it.

> 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.

> 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

How much is item 3 figuring in your own evaluation of IDL?
In IDL programming courses I teach, I figure as many as a third
of the people in the course won't ever successfully use IDL,
simply because they can't bring themselves to give it a chance.
It's not Fortran, it's not C, it's not Python. The list goes
on and on.

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.

> 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.

> Does this make me popular in this newsgroup? Does it give me a chance
> of getting answers here? *ploink*, I guess.

Oh, I wouldn't worry about it. We are fools enough
to answer *anyone's* questions. :-)

> Does it make me feel better? Yes, certainly.

I hope so. And I hope the rest of the week goes
better than today! :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: compile a routine wich inlude a commun
Posted by Maarten[1] on Mon, 23 Jan 2006 16:59:46 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> It will get better with time. :-)
>
> And look at the bright side, we don't have to listen to the
> Super Bowl hype for the next two weeks!

I hope that the programming bit will get better over time, whatever the
language. As for "missing" any sports hype: I know how you feel, the
olympics are coming up, can't wait till they're over ;-)

>> 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.

And I maintain that requiring fixed sized arrays in structures is impractical. Having pointers is no substitute, as far as I'm concerned: way too easy to loose track of, and an array inside a structure needs a different syntax than a pointer to an array. References to data objects are needed as well, but not as the sole means of access to an object.

>> 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:

(note that I say *new* projects).

[snip]

> How much is item 3 [struggle to learn, afraid to throw out invested time]
> figuring in your own evaluation of IDL?

Not at all. Like I said: I'd jump ship the very moment I see the opportunity - it is just that I don't have the time to gather all the items I need on a python install. And the struggle isn't that heavy: apart from objects, where it is plain disgust holding me back, I don't have too much trouble picking things up. I do wonder "what were they thinking" too often though. And yes, 28 years is a long time, but still, I feel that the direct graphics to object graphics transtition could have been handled more elegantly.

> In IDL programming courses I teach, I figure as many as a third
> of the people in the course won't ever successfully use IDL,
> simply because they can't bring themselves to give it a chance.
> It's not Fortran, it's not C, it's not Python. The list goes
> on and on.

I'm _very_ aware of what IDL is _not_. It is just that I have a hard time figuring out what it _is_ instead.

> Yes, IDL is a messy language. But have you looked at
> programs you wrote 10 years ago? 20? *28* years ago!

My code cannot be that old. Well, technically it could, but only just. But yes, coding style changes over time, and generally imrpoves from leasons learnt. I've seen other programs handle backward compatibility in other ways, that improved the whole, and left us with slightly less

of a mess.

> 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.

I'm a (La)TeX user, I've done some pretty wild things over there. I know a thing or two about old code, legacy systems, backward compatibility and the mess it can create, not to mention programming languages that think in reverse gear (if you really want: try the language of bibtex one time. I always get the feeling of needing a reverse gear on my mind when *reading* the code -- literaly: reading it from the end to the beginning, and the code actually makes some sense. And that is just reading.)

Thing is, when starting _now_ you care about this () much about that legacy. What you see is the mess, and what you miss is what you had before (and can no longer use because of the change in the job).

> Yet, there is no better alternative for a number of users.

Why? What is unique to IDL that makes it the only option for some users? Legacy code is an obvious one, but the less obvious ones?

[snip]

> 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.

And yet: what that code does is something that is fairly common, and it is rather silly that it takes code that is _that_ hard. Well, I opt for readability and ease of programming, speed be damned.

> 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.

I'll have a look later on, and see if I can come up with an elegant
Python version for some of the samples.

> Oh, I wouldn't worry about it. We are fools enough
> to answer *anyone's* questions. :-)

Ah, lucky me ;-) On the whole this usenet group is friendly and
helpful. I must say that is makes it easier on me to stick with it.

>> Does it make me feel better? Yes, certainly.
>
> I hope so. And I hope the rest of the week goes
> better than today! :-)

Oh, today went rather well, so far at least. Given the local time, that
is not too bad.

Maarten

Subject: Re: compile a routine wich inlude a commun
Posted by David Fanning on Mon, 23 Jan 2006 17:17:32 GMT
View Forum Message <> Reply to Message

Maarten writes:

> Well, I opt for readability and ease of programming,
> speed be damned.

I would have thought IDL would have been *perfect* for
you. Lord knows thousands of users are writing "speed
be damned" programs. ;-)

Cheers,

David

P.S. And it is not true that 90% of them are astronomers.
It just seems that way. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: compile a routine wich inlude a commun

Posted by Craig Markwardt on Mon, 23 Jan 2006 17:37:21 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> writes:
>
> 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.

Uhhh, David, "exactly???"

No automatic garbage collection.

Separate "yet equal" object reference type.

/ALLOCATE_HEAP.

Awkward syntax for dereferencing some pointers.

Pointers are okay, but they are not exactly right.

Craig

Subject: Re: compile a routine wich inlude a commun
Posted by David Fanning on Mon, 23 Jan 2006 17:54:48 GMT
View Forum Message <> Reply to Message

Craig Markwardt writes:

> Uhhh, David, "exactly???"
>
> No automatic garbage collection.
>
> Separate "yet equal" object reference type.
>
> /ALLOCATE_HEAP.
>
> Awkward syntax for dereferencing some pointers.
>
> Pointers are okay, but they are not exactly right.

Oh, well, I mean "exactly in the IDL sort of way".
You know what I mean. Kind of in the way NLEVELS
gives you exactly that many levels in a Contour plot. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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

Subject: Re: compile a routine wich inlude a commun
Posted by Paul Van Delst[1] on Mon, 23 Jan 2006 22:24:32 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> 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.

Well, they're called pointers but they're not, really. You can't actually "point" to
anything - just make copies. But, not being a pointer expert, let me ask the question:
How *do* you use a pointer in IDL to, uh, well, "point" to an already created variable? Or
just parts of an already created array?

E.g.

```
IDL> x=indgen(4,4)
IDL> print, x
       0       1       2       3
       4       5       6       7
       8       9      10      11
      12      13      14      15
IDL> p=ptr_new(x[1:2,1:2])
IDL> print, *p
       5       6
       9      10
IDL> *p=*p+100
IDL> print, *p
     105     106
     109     110
IDL> print, x
       0       1       2       3
       4       5       6       7
       8       9      10      11
      12      13      14      15
```

If pointers in IDL worked right (and by "right", I mean how *I* intuitively expect them to
work <take grain of salt here>) I would expect the "print, x" command to output the following:

```
IDL> print, x
       0       1       2       3
       4     105     106       7
       8     109     110      11
      12      13      14      15
```

In my pointer-naivete, it seems to me that p should point to the memory that x occupies.
But that's not what happens. I expect it to work like the following Fortran95 program:

```
program testptr
  integer,parameter::n=4
  integer,target::x(0:n-1,0:n-1)
  integer,pointer::p(:,:)
  x=reshape((/(i-1,i=1,n*n)/),(/n,n/))
  write(*,'("Print x:")')
  write(*,'(4i5)')x
  p=>x(1:2,1:2)
  write(*,'("Print p:")')
  write(*,'(2i5)')p
  p=p+100
  write(*,'("Print p added to:")')
  write(*,'(2i5)')p
  write(*,'("Print x:")')
  write(*,'(4i5)')x
end program testptr

lnx:scratch : lf95 testptr.f90
Encountered 0 errors, 0 warnings in file testptr.f90.
lnx:scratch :  a.out
Print x:
    0    1    2    3
    4    5    6    7
    8    9   10   11
   12   13   14   15
Print p:
    5    6
    9   10
Print p added to:
  105  106
  109  110
Print x:
    0    1    2    3
    4  105  106    7
    8  109  110   11
   12   13   14   15
```

Maybe PTR_NEW() should be renamed to something else? Like, um, HEAPVAR_NEW() ?

I'm sure all of this has something to do with the pass-by-reference/pass-by-value nature of certain things in IDL.

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC