Subject: Re: IDL as programming language?
Posted by steinhh on Sun, 28 Aug 1994 11:14:05 GMT
View Forum Message <> Reply to Message

These kinds of questions seem to recur quite often, so I'm posting instead
of just replying by E-mail.

In article <1994Aug28.122441.32497@waikato.ac.nz>, abz@waikato.ac.nz writes:
|>[...]
|>   We have a number of questions, concerning
|> comparisons between IDL and other programming languages (particularly FORTRAN).
|> We are currently running an older version of IDL (2.2.2) on a Sun SPARC
|> station.
|>
|> (i) Accuracy. Our current version of IDL seems to prefer doing calculations
|> in single precision, while we prefer double. Has this been improved in the
|> latest version? (e.g. in our current version, routines like LUDCMP work in
|> s.p., despite being passed d.p. arguments.)
|>

I have little experience with this particular problem, but if you already
have written routines in FORTRAN that does specific jobs in double precision
(I guess you have),the use of the CALL_EXTERNAL routine makes it quite easy
to use those routines whenever the built-ins are not sufficient.

A note of advice on CALL_EXTERNAL: Don't let the hopeless bureaucratic style
of the supplementary information in "call_external.doc" baffle you. Simply
use your favourite editor to delete everything that has nothing to do with
your machine, operating system, programming language etc, and then do the
same with the examples. It's amazing how easy it is.

|> (ii) Speed. Some of us Grads are running some really time consuming programs
|> (large arrays, large loops). How does IDL compare with (say) FORTRAN in
|> general, speedwise? (my impression is that it's pretty slow, but I could be
|> wrong...)

For exactly those scenarios (FORTRAN loops processing large arrays), IDL
should be at worst a very few percent slower. If you're doing FFT's, the
built-in routine in IDL outperforms the standard Coley-Tukey version
in Numerical Recipes. I had a problem using FFT's for calculation of
autocorrelation functions of real-valued data, so I could use the
shortcuts in Num. Recipes twice (starting with real-valued data, and
the inverse transformation is also with real-valued data)
when implementing it in C, but I was only able to improve speed by
about 5-10% after squeezing it all out. I had thought that just writing
a straightforward implementation (without shortcuts) would save time, but
I ended up using *more* time! And the FFT's in IDL are written for
arbitrary array lengths, not just 2^N elements.

|>
|> (iii) Memory. How does IDL's memory management compare? Again, some of our
|> programs (FORTRAN) have a tendency to gobblelarge chunks of memory (probably
|> bad programming, but still...)
|>

This depends somewhat on programming style. For calculation-intensive
applications, it shouldn't be much worse than FORTRAN, although if you
are *really* "good" at it, you can make IDL spend a *lot* of memory.
(But this would happen in most languages, anyway).

|> (iv) What is a large IDL code like to debug?
|>

Generally, I find it very easy to debug. It's much easier to get
an overview of the programs compared to FORTRAN or C versions due to
the fact that array processing usually takes just one line instead
of being an explicit loop. Also, the ability to stop at any point
in the code, plotting data whenever you like etc. is very nice.

I have also found that it's very comfortable to use IDL as a development
tool, testing the algorithms in a very interactive way using IDL, and then,
if speed is essential, it's always easy to implement the hard work from
the IDL code into FORTRAN or C subroutines via CALL_EXTERNAL, or, in
extreme cases, to write a stand-alone program based on the IDL code.
The amount of time saved in testing and developing the algorithms is
tremendous!

|> (v) How 'robust' is IDL as a programming language? We have a variety of
|> different programming styles here -- some prefer 'quick and dirty' programming,
|> others a more structured approach. Forgive my possible ignorance, but I have
|> the impression that IDL as a language is more suited to the 'quick and dirty'
|> approach. Is this true? Does IDL as a programming language have many
|> glitches or inconveniences from a mathematical programmers point of view?
|>

I'm not sure I understand what you mean by 'robust', but here goes: IDL
doesn't have explicit declarations of variables, so you could say that
it's somewhat "quick and dirty" in that respect. Also, calling a routine
with too many parameters are only detected at runtime, and if you're
using too few parameters, it's up to the routine itself to check if
everything's OK (parameter typing etc as well) -- otherwise you could
get "Undefined" messages from the subroutine without really being aware
that you just didn't give it that particular piece of information. This
isn't a serious problem, though. Compared to, say C, IDL is *very* robust,
without any type checking (it doesn't core dump, and on PC's, you don't
risk deleting your harddisk if something goes haywire!). Implementation of

parameter #/type checks etc is, however, just as easy(-ier)
as any explicit declaration thereof would be in other languages. Also, an
advantage is that you can write a single version of most functions, and
they work just fine whether you'd like to pass it a scalar integer or a
double precision array, returning scalars or arrays respectively.

|> Any info/advice would be much appreciated. The types of stuff we do
|> here are generally large numerical (finite difference) codes on 2D and 3D
|> grids.

That's what IDL's for :-)

Stein Vidar
(Yes, I *do* like IDL)

Subject: Re: IDL as programming language?
Posted by thompson on Mon, 29 Aug 1994 14:19:07 GMT

View Forum Message <> Reply to Message

abz@waikato.ac.nz writes:

> (ii) Speed. Some of us Grads are running some really time consuming programs
> (large arrays, large loops). How does IDL compare with (say) FORTRAN in
> general, speedwise? (my impression is that it's pretty slow, but I could be
> wrong...)

Loops are very slow. On the other hand, you usually can do it without loops in
IDL. If you can't, there's always CALL_EXTERNAL or LINKIMAGE.

> (iii) Memory. How does IDL's memory management compare? Again, some of our
> programs (FORTRAN) have a tendency to gobblelarge chunks of memory (probably
> bad programming, but still...)

IDL does the same thing. I doubt if it would be any better than Fortran, but
don't know if it's any worse.

> (iv) What is a large IDL code like to debug?

Certainly no worse than Fortran, and one can fiddle with things to see how they
affect behavior much, much faster. According to the release notes for version
3.6, there is now a Motif-based interactive debugging tool for Unix platforms.
I haven't used it, but it should help things out quite a bit.

> (v) How 'robust' is IDL as a programming language? We have a variety of
> different programming styles here -- some prefer 'quick and dirty' programming,
> others a more structured approach. Forgive my possible ignorance, but I have
> the impression that IDL as a language is more suited to the 'quick and dirty'

> approach. Is this true? Does IDL as a programming language have many
> glitches or inconveniences from a mathematical programmers point of view?

I would agree with IDL as being oriented towards "quick and dirty" programming. That doesn't mean that one can't put together large software systems--we do. On the other hand, it doesn't enforce things like datatypes that other languages might. This can either be a blessing or a curse, depending on your point of view.

As far as inconveniences go, from a mathematical point of view, one can point out the fact that IDL doesn't support a double-precision complex data type.

> Any info/advice would be much appreciated. The types of stuff we do
> here are generally large numerical (finite difference) codes on 2D and 3D grids.
> I'll wait a while before sending this to see if there is/has been any
> discussion on this topic in the newsgroup.

My impression has always been that IDL is best suited towards people (like myself) who are working with actual data, with all the grungy details that entails. People who use only "ideal" theoretical data tend to be more pleased with packages like Mathematica or Mathcad.

Subject: Re: IDL as programming language?
Posted by Geoff.Sobering on Mon, 29 Aug 1994 18:08:10 GMT
View Forum Message <> Reply to Message

In article <1994Aug28.122441.32497@waikato.ac.nz>, abz@waikato.ac.nz wrote:

> OK. One the lecturers in our department recently went to
> Hawaii & America for a few weeks (lucky him) and came back
> with the idea of using IDL as a complete programming
> language rather than as just a plotter for FORTRAN generated data.
> ...

I don't want to use bandwidth, time, etc. repeating the comments previous posters have made to this thread (which I mostly agree with).
I just wanted note that I use IDL for almost all of my programing, both scientific and otherwise.
For example, I have a network monitoring program that polls our network-hub (via 'spawn'ed telnet) and keep track of network traffic. It may also be interesting to note that RSI wrote the GUI-builder that is part of 3.6.12 *in* IDL.

--
Geoff Sobering (Geoff.Sobering@nih.gov)
In Vivo NMR Research Center
National Institutes of Health

Subject: Re: IDL as programming language?
Posted by shapiro on Mon, 29 Aug 1994 20:50:01 GMT
View Forum Message <> Reply to Message

Geoff.Sobering@nih.gov (Geoff Sobering) writes:

> In article <1994Aug28.122441.32497@waikato.ac.nz>, abz@waikato.ac.nz wrote:

>> OK. One the lecturers in our department recently went to
>> Hawaii & America for a few weeks (lucky him) and came back
>> with the idea of using IDL as a complete programming
>> language rather than as just a plotter for FORTRAN generated data.
>> ...

> I don't want to use bandwidth, time, etc. repeating the comments previous
> posters have made to this thread (which I mostly agree with).
> I just wanted note that I use IDL for almost all of my programing, both
> scientific and otherwise.
> For example, I have a network monitoring program that polls our network-hub
> (via 'spawn'ed telnet) and keep track of network traffic. It may also be
> interesting to note that RSI wrote the GUI-builder that is part of 3.6.12
> *in* IDL.

RSI has a problem with quality control. I have been developing software
in IDL for the past three years. IDL has been no better or worse than
any other language. Initially IDL seems like a wonderfull language, it
has all sorts of canned routines and a fairly easy GUI builder. However
every time a new version of the language is released I spend at least a
week patching up old software that the new version breaks. If you are
only writing little programs with a limited lifespan then IDL is
perfect. If you are working in a larger project (ours is 6000+ lines)
the instability in the language can be a real pain.

RSI seems to have little vision of where there product is headed. So
features are added haphazardly and with little forthought. As a result
IDL is ripe with kludge and nuance. They also have a problem with
documenting changes when they happen. Once they included a whole
paragraph in the release notes arguing that some bug was a feature due
to backward compatability (to version 1.1) while the same release notes
failed to mention that widget_info had changed from a procedure to a
function. More recently RSI has added pointers to IDL. They never
mentioned this in the release notes, only that something about pointers
was broken and has now been fixed.

I would never recomend IDL 'as just a plotter for FORTRAN generated
data.' There are far to many cheep or free ploting packages already
available. As a language it seems better suited to little programs and
'one off' projects. Large scale and long term projects are best
attempted in a more stable language (like C).

-Andrew T. Shapiro
CSES/CIRES University of Colorado
shapiro@cses.colorado.edu Campus Box 216
(303) 492-5539 Boulder, CO 80309-0216, USA

Subject: Re: IDL as programming language?
Posted by n9140397 on Tue, 30 Aug 1994 23:51:45 GMT
View Forum Message <> Reply to Message

In article <1994Aug28.122441.32497@waikato.ac.nz> abz@waikato.ac.nz writes:
>
>
> OK. One the lecturers in our department recently went to Hawaii & America for a
> few weeks (lucky him) and came back with the idea of using IDL as a complete
> programming language rather than as just a plotter for FORTRAN generated data.
IMHO, this is a very bad idea... Unfortunately, I have only a few things to
back this up. Mostly it's a feeling, which doesn't hold much water, but,
see below...

> (i) Accuracy. Our current version of IDL seems to prefer doing calculations
> in single precision, while we prefer double. Has this been improved in the
> latest version? (e.g. in our current version, routines like LUDCMP work in
> s.p., despite being passed d.p. arguments.)

My first reason for not liking IDL is this. You have to very explicit
about double precision variables. This has not been improved to my
knowledge, but, I don't have the latest version, either.

> (ii) Speed. Some of us Grads are running some really time consuming programs
> (large arrays, large loops). How does IDL compare with (say) FORTRAN in
> general, speedwise? (my impression is that it's pretty slow, but I could be
> wrong...)

This is my impression as well... it's slow, even though code-wise, it's really
nice to do large array computations because you can often skip explicitly
writing loops.

> (iii) Memory. How does IDL's memory management compare? Again, some of our
> programs (FORTRAN) have a tendency to gobblelarge chunks of memory (probably
> bad programming, but still...)

Really bad. IDL is always sucking up lots of memory, in my experience.

> (iv) What is a large IDL code like to debug?

UGH! Again, my opinion. Some people will probably like debugging idl code

over FORTRAN code (or some other legitamate language). But, I would venture
that they do not know how to use a real debugger like dbx...

> (v) How 'robust' is IDL as a programming language? We have a variety of
> different programming styles here -- some prefer 'quick and dirty' programmin
> others a more structured approach. Forgive my possible ignorance, but I have
> the impression that IDL as a language is more suited to the 'quick and dirty'
> approach. Is this true? Does IDL as a programming language have many
> glitches or inconveniences from a mathematical programmers point of view?

IDL is the epitome of quick and dirty. Yet another reason I don't like it.
It's almost impossible to make idl code organized, it seems.

Other things you should consider: IDL does not, to my knowlege, create
executable code which you store on disk or which runs independently.
If you want to share something you wrote with a friend, they have to have
idl also, and maybe even the same version. Actualy languages like
C and FORTRAN are much more "portable".

Also, a good optomizing FORTRAN compiler is probably still the fastest
thing around as far as an executable goes. This from a C advocate...

IDL is nice for making plots, and you can write quick and dirty, but,
I sure wouldn't want to write a model in it...

My 2 cents....

Mike

******Opinions are my own and not necessarily those of my employer*********

Subject: Re: IDL as programming language?
Posted by olsen on Wed, 31 Aug 1994 15:23:59 GMT

View Forum Message <> Reply to Message

In article 15667@henson.cc.wwu.edu, n9140397@gonzo.cc.wwu.edu (Michael Hamilton) writes:
> In article <1994Aug28.122441.32497@waikato.ac.nz> abz@waikato.ac.nz writes:
>
>> (v) How 'robust' is IDL as a programming language?
> reply by Henson>
 >Other things you should consider: IDL does not, to my knowlege, create
> executable code which you store on disk or which runs independently.
> If you want to share something you wrote with a friend, they have to have
> idl also, and maybe even the same version. Actualy languages like
> C and FORTRAN are much more "portable".
>>
>

portability is relative.......
I can use IDL on all 4 of my main platforms (SUN,SGI, PC(portable), and MAC),
with virtually no changes (set_plot, ...).   fortran compilers and c compilers
are not all that standard from platform to platform....
 just try and compile the new ImageMagick..........

rc olsen

Subject: Re: IDL as programming language?
Posted by stl on Thu, 01 Sep 1994 07:03:58 GMT

View Forum Message <> Reply to Message

I just have to give a little plug for IDL...

In article <1994Aug30.235145.15667@henson.cc.wwu.edu> n9140397@gonzo.cc.wwu.edu
(Michael Hamilton) writes:
> In article <1994Aug28.122441.32497@waikato.ac.nz> abz@waikato.ac.nz writes:
>
>> (i) Accuracy. Our current version of IDL seems to prefer doing calculations
>> in single precision, while we prefer double. Has this been improved in the
>> latest version? (e.g. in our current version, routines like LUDCMP work in
>> s.p., despite being passed d.p. arguments.)
>
IDL allows types to be Dynamic. Double precision is not a problem, you
just must make sure that the variables you are dealing with at the time
are all DOUBLES. One of the nicest features with IDL is that you can
interact with variables of different types, and generally not worry
about them, almost no other language allows you to do this (low level
language at least) Therefore, if you initially define all your
variables as DOUBLES, you should have NO problem.
>
>> (ii) Speed. Some of us Grads are running some really time consuming programs
>> (large arrays, large loops). How does IDL compare with (say) FORTRAN in
>> general, speedwise? (my impression is that it's pretty slow, but I could be
>> wrong...)
>
The responses to this are extremely interesting. When programmed
correctly (ie: a few loops as possible, accessing memory in the correct
order, etc) IDL SCREAMS. ANd the speed is obtained with very little
code. When dealing with matrix calculations, I have hardly heard anyone
complain about its speed. GRANTED, if you do not use the built in
operators, use lots of loops, and program as if you have to handle all
the indexing,.. then its pretty slow..
>
>> (iii) Memory. How does IDL's memory management compare? Again, some of our
>> programs (FORTRAN) have a tendency to gobblelarge chunks of memory (probably
>> bad programming, but still...)

>
No idea really. (probably not even close to C, maybe to Fortran but I
really have no idea) But, hey, memory is cheap... (usually)
>
>> (iv) What is a large IDL code like to debug?
>
> UGH! Again, my opinion. Some people will probably like debugging idl code
> over FORTRAN code (or some other legitamate language). But, I would venture
> that they do not know how to use a real debugger like dbx...
HUM, hate to respond against someone elses answer, but I disagree here.
As for DBX, well.. no comment. If your code is programmed Modularly,
ie: one routine per file, I find it very nice to debug because of the
following:
 -IDL tells you the exact line the error occured (I hear that the PC
 and MAc version even thoughs you into the file to debug it.. NICE)
 -The error messages usually are pretty meaningfull
 -When an error does occure, you are left in that environment, and
 can test any variable you want (no need to start a debugger, and set
 all variables you want to look at, etc)
 -You can even find the problem, fix it, recompile the code and
 continue where you die before. Or you can try out new ways to do
 what your code does in an interactive mode.

Granted, it has no GUI oriented interface for a debugger, but I think
its easier then debugging compiler/linker/memory and other problems that
you might encounter in C. Its A a higher level language then C and
fortran, so its pretty tough to compare them.

>
>> (v) How 'robust' is IDL as a programming language? We have a variety of
>> different programming styles here -- some prefer 'quick and dirty' programmin
>> others a more structured approach. Forgive my possible ignorance, but I have
>> the impression that IDL as a language is more suited to the 'quick and dirty'
>> approach. Is this true? Does IDL as a programming language have many
>> glitches or inconveniences from a mathematical programmers point of view?
>
Okay, this is a critical question. Normally I find it VERY stable. But
I am learning that when working with IDL on a network with different
versions of UNIX (Solaris and SUNOS) and apparently therefore different
X servers, it gets somewhat flaky.

Another thing worth thinking about is building huge applications with
IDL. It CAN be done. And more and more easily with every release, but
you need to plan, and define standards just like any other language. It
does tend to be a quick and dirty langauge, but MODULAR code is
possible. Just be neat, define a regular tree structure to code, put
each routine in a seperate file, define the paths in the startup file,
etc..   I have been doing it for numerous big projects. Also with the

introduction of WIdgets, interfaces are possible, and are standard
across platforms (somewhat). But don't expect control over them as if
you were prograsmming from Motif, but they are really not bad.

Finally, yes, there is no possible way to build a runtime of code
without an IDL license to run it. Kind of a bummer, but unterstandable
when you are marketing a product, and most other higher level langauges
are like this.. But you can compile the code, and save it for execution
later, this works fine.

Also, now you can call IDL functions via RPC from someother langauge.
ANd the conectablitlity to C and Fortran LIbraries is pretty good. Its
still growing, and really is an amazing analysis tool first and formost!
But don't expect that just because its easy to use that an
nonexperienced programmer can build a huge application in it just
because its a relatively easy langauge..

Well thats
> My 2 cents....

-stephen
--
Stephen C Strebel / SKI & TELE TO DIE
strebel@sma.ch / and
Swiss Meteorological Institute, Zuerich / LIVE TO TELL ABOUT IT
01 256 93 85 / (and pray for snow)

Subject: Re: IDL as programming language?
Posted by scisoft on Mon, 05 Sep 1994 16:43:51 GMT
View Forum Message <> Reply to Message

William Thompson (thompson@orpheus.gsfc.nasa.gov) wrote:
: abz@waikato.ac.nz writes:
: >Speed. Some of us Grads are running some really time consuming programs
: >(large arrays, large loops). How does IDL compare with (say) FORTRAN in
: >general, speedwise? (my impression is that it's pretty slow, but I could
: >be wrong...)
: Loops are very slow. On the other hand, you usually can do it without
: loops in IDL. If you can't, there's always CALL_EXTERNAL or LINKIMAGE.

Another alternative is the IDL and PV-Wave interface to the optimized and
parallelized math library that our company sells. Dr. Thompson is
certainly correct that A=B*C is a faster matrix multiple than the
equivalent nested loops, but CALL GEMM(A,B,C) is about three times
faster than that even on a single-CPU box. On the multi-processors,
we're unbeatable.

scisoft@well.sf.ca.us for more information, end of plug.