Subject: Pass by value and performance
Posted by Kenneth P. Bowman on Wed, 14 Dec 2005 05:00:03 GMT
View Forum Message <> Reply to Message

A cautionary note on argument passing ...

I have a large code that does a lot of interpolation in
multi-dimensional arrays. Being a clever IDL programmer (too clever by
half, as it turns out), I package these arrays into structures along
with various information about the arrays. When these arrays are then
passed, for example, into INTERPOLATE, as

result = INTERPOLATE(data.array, x, y, z)

the array is passed by value, which entails making a copy of the array.
When these arrays get large, this causes a big performance hit. So, I
am in the process of making my code less clever (and uglier) but much
faster.

Cheers, Ken Bowman

Subject: Re: Pass by value and performance
Posted by JD Smith on Thu, 15 Dec 2005 21:04:18 GMT
View Forum Message <> Reply to Message

On Thu, 15 Dec 2005 07:56:50 +0100, Antonio Santiago wrote:

> Kenneth P. Bowman wrote:
>> Perhaps someone can clarify this for me.
>>
>> I was doing this
>>
>> data = {values : FLTARR(...), $
>> other : other stuff ...}
>>
>> Then pass "data" to a procedure and do this
>>
>> result = INTERPOLATE(data.values, x, y, z)
>>

> I like to understand pointers in IDL in this way:
>
> 1.- 'a' is a conventional variable managed by IDL and its "garbage
> collector".

Sadly, IDL doesn't have garbage collection. It would be nice if it

did, but until then, it's up to you to free all of your heap variables
at the correct time (which is great when you know when that is).

> 2.- '*a' is a HEAP variable, where 'a' stores a reference to it. Also, the
> content of the variable 'a' is stored in the heap memory.
>
> Then 'a' is a reference for a "normal" variable that stores a reference,
> and '*a' is a reference to a HEAP variable that stores a 5.

I'd just say both a and *a are variables. One ordinary (local in
scope), the other heap (global in scope).

> junk, *a --> The content of the HEAP memory variable is pased by value.

This isn't correct. De-referenced pointer variables (aka "heap"
variables) are passed by reference, just like regular variables (which
they are, really). E.g. in Ken's original example:

```
 result = INTERPOLATE(*data.array, x, y, z) ; by reference
```

would indeed pass the pointer heap variable by reference and not by
value. As such it would be much faster (for large arrays) than
INTERPOLATE(data.array,x,y,z), which would require copying the full
array to a local variable, and would be equivalent to a simple
INTERPOLATE(array,x,y,z).

As pointed out in the pointer tutorial
(http://www.dfanning.com/misc_tips/pointers.html), there is no
difference between pointer heap variables and ordinary variables,
except in how you access them. Of course, that also means that a
structure member (or array element, etc.) of a dereferenced pointer
variable is (just like a member of an ordinary variable), still passed
by value:

```
 result = INTERPOLATE((*data).array, x, y, z) ; by value
```

Here `data' is a pointer to a structure with member "array", which is
passed here by value.

This equivalence also means that standard IDL variable tricks, like
re-assigning the memory contents of one variable to another without
copying, work just fine for pointer heap variables (and in between
plain old variables and pointer heap variables).

JD

Subject: Re: Pass by value and performance
Posted by Antonio Santiago on Fri, 16 Dec 2005 10:24:36 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
> On Thu, 15 Dec 2005 07:56:50 +0100, Antonio Santiago wrote:
>
>
>> Kenneth P. Bowman wrote:
>>
>>> Perhaps someone can clarify this for me.
>>>
>>> I was doing this
>>>
>>> data = {values : FLTARR(...), $
>>> other : other stuff ...}
>>>
>>> Then pass "data" to a procedure and do this
>>>
>>> result = INTERPOLATE(data.values, x, y, z)
>>>
>
>
>
>> I like to understand pointers in IDL in this way:
>>
>> 1.- 'a' is a conventional variable managed by IDL and its "garbage
>> collector".
>
>
> Sadly, IDL doesn't have garbage collection. It would be nice if it
> did, but until then, it's up to you to free all of your heap variables
> at the correct time (which is great when you know when that is).
>
>
>> 2.- '*a' is a HEAP variable, where 'a' stores a reference to it. Also, the
>> content of the variable 'a' is stored in the heap memory.
>>
>> Then 'a' is a reference for a "normal" variable that stores a reference,
>> and '*a' is a reference to a HEAP variable that stores a 5.
>
>
> I'd just say both a and *a are variables. One ordinary (local in
> scope), the other heap (global in scope).
>
>

Sorry, but unfortunately yesterday a bad boy was sitting in my chair and
writte the above misspelling words. Alse the bad boy is a bad englighs

witter :((like me ;)).

>> junk, *a --> The content of the HEAP memory variable is pased by value.
>
>
> This isn't correct. De-referenced pointer variables (aka "heap"
> variables) are passed by reference, just like regular variables (which
> they are, really). E.g. in Ken's original example:
>
>   result = INTERPOLATE(*data.array, x, y, z) ; by reference
>
> would indeed pass the pointer heap variable by reference and not by
> value. As such it would be much faster (for large arrays) than
> INTERPOLATE(data.array,x,y,z), which would require copying the full
> array to a local variable, and would be equivalent to a simple
> INTERPOLATE(array,x,y,z).
>
> As pointed out in the pointer tutorial
> (http://www.dfanning.com/misc_tips/pointers.html), there is no
> difference between pointer heap variables and ordinary variables,
> except in how you access them. Of course, that also means that a
> structure member (or array element, etc.) of a dereferenced pointer
> variable is (just like a member of an ordinary variable), still passed
> by value:
>
>   result = INTERPOLATE((*data).array, x, y, z) ; by value
>
> Here `data' is a pointer to a structure with member "array", which is
> passed here by value.
>
> This equivalence also means that standard IDL variable tricks, like
> re-assigning the memory contents of one variable to another without
> copying, work just fine for pointer heap variables (and in between
> plain old variables and pointer heap variables).
>
> JD
>

OK, I think I understand IDL pointer. Maybe my problem was to find the
similities between C pointers and IDL pointers. That is, when I saw '*a'
I read the C style: "the content where 'a' points to".

Following this I have:

a = 10
b = PTR_NEW(40)
c = PTR_NEW(BYTARR(100))

```
 Conventional Mem.                  |   HEAP memory
 (Managed by IDL but                |   (Jungle where you are
  without a Garbage Collector??? )  |    responsible to free)
-------------------------------------------------------------- ----
        a = 10                      |
        b --------------------------|--> 40
        c --------------------------|--> [0,1,2,.....,99]
```

Then if I call a function with:

call_to_procedure, *c

In bad C style I think I am passing the content of 'c', that is a BTYARR
of 100 (BAD ???)

In IDL is passed a reference to the content, that is like if I writte:

d = BYTARR(100)
call_to_procedure, d

Is this right??

Thanks a lot.

PD: I think I must talk seriously with the bad boy of my office :)

--

Antonio Santiago Pï¿½rez
(email: santiago<<at>>grahi.upc.edu   )
(www: http://www.grahi.upc.edu/santiago)
(www: http://asantiago.blogsite.org   )

GRAHI - Grup de Recerca Aplicada en Hidrometeorologia
Universitat Politï¿½cnica de Catalunya

Subject: Re: Pass by value and performance
Posted by Paolo Grigis on Fri, 16 Dec 2005 11:08:08 GMT
View Forum Message <> Reply to Message

JD Smith wrote:

> This isn't correct. De-referenced pointer variables (aka "heap"
> variables) are passed by reference, just like regular variables (which
> they are, really). E.g. in Ken's original example:
>
> result = INTERPOLATE(*data.array, x, y, z) ; by reference
>
> would indeed pass the pointer heap variable by reference and not by
> value. As such it would be much faster (for large arrays) than
> INTERPOLATE(data.array,x,y,z), which would require copying the full
> array to a local variable, and would be equivalent to a simple
> INTERPOLATE(array,x,y,z).

Since we are on the subject of performance, there's nothing like
a little benchmark to bring some light to shine upon the issue...

Let's try this (using rebin for simplicity):

Benchmark 1:

```
;initialize large arrays of data
N=2L^27

data={a:lindgen(N),b:ptr_new(lindgen(N))}

c=lonarr(N/2)

nrounds=10
nrebins=10

timevar=fltarr(nrounds)
timeptr=fltarr(nrounds)

;do benchmark
.run
FOR j=0,nrounds-1 DO BEGIN

    print,'Now doing round '+strtrim(string(j+1),2)

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(data.a,N/2)
    tend=systime(1)
    timevar[j]=tend-tstart

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(*data.b,N/2)
    tend=systime(1)
```

```
    timeptr[j]=tend-tstart
ENDFOR
end
```

This compares the data.array vs. *data.array performance.
As correctly claimed by JD, there is indeed a difference
between the two approaches:

```
;"data.array" case
IDL> print,timevar
      32.7094      34.0300      34.7631      33.0446      33.9109
      34.2302      34.2145      33.8960      34.2056      34.2010
;"*data.array" case
IDL> print,timeptr
      18.1812      18.4961      18.5838      17.8924      18.4376
      18.4548      18.0502      18.3959      18.7219      18.0366
```

However, if we don't have structures, is there a difference
between passing pointers and regular variables? How does
this compare with the structure case?

Benchmark 2:

```
;initialize large arrays of data
N=2L^27

a=lindgen(N)
b=ptr_new(lindgen(N))

c=lonarr(N/2)

nrounds=10
nrebins=10

timevar=fltarr(nrounds)
timeptr=fltarr(nrounds)

;do benchmark
.run
FOR j=0,nrounds-1 DO BEGIN

    print,'Now doing round '+strtrim(string(j+1),2)
```

```
    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(a,N/2)
    tend=systime(1)
    timevar[j]=tend-tstart

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(*b,N/2)
    tend=systime(1)
    timeptr[j]=tend-tstart
ENDFOR
end
------------------------------------------------
```

Here we get:

```
;"array" case
IDL> print,timevar
      17.6973     17.6340     17.6237     17.7584     17.6499
      17.7070     17.6797     17.6858     17.6515     17.6766
;"*array" case
IDL> print,timeptr
      17.6719     17.7895     17.6816     17.6413     17.7822
      17.6556     18.0883     17.6746     17.6907     18.1122
```

No difference (motto: "dereferenced pointer behave like normal variables" thus both passed by reference), and the performance is the same as the fastet of the previous case.

Summarizing: rebin(*data.array) is indeed faster than rebin(data.array), but rebin(*data.array), rebin(array) and rebin(*array) have all the same speed.

Again, JD was indeed absolutely right. I just thought it was nice to have an experimental confirmation... and it helped me to grasp the issue.

Cheers,
Paolo

>
> As pointed out in the pointer tutorial
> (http://www.dfanning.com/misc_tips/pointers.html), there is no
> difference between pointer heap variables and ordinary variables,
> except in how you access them. Of course, that also means that a
> structure member (or array element, etc.) of a dereferenced pointer

> variable is (just like a member of an ordinary variable), still passed
> by value:
>
> result = INTERPOLATE((*data).array, x, y, z) ; by value
>
> Here `data' is a pointer to a structure with member "array", which is
> passed here by value.
>
> This equivalence also means that standard IDL variable tricks, like
> re-assigning the memory contents of one variable to another without
> copying, work just fine for pointer heap variables (and in between
> plain old variables and pointer heap variables).
>
> JD
>

Subject: Re: Pass by value and performance
Posted by Rick Towler on Fri, 16 Dec 2005 17:29:18 GMT
View Forum Message <> Reply to Message

Kenneth P. Bowman wrote:
> A cautionary note on argument passing ...
>
> I have a large code that does a lot of interpolation in
> multi-dimensional arrays. Being a clever IDL programmer (too clever by
> half, as it turns out), I package these arrays into structures along
> with various information about the arrays. When these arrays are then
> passed, for example, into INTERPOLATE, as
>
> result = INTERPOLATE(data.array, x, y, z)
>
> the array is passed by value, which entails making a copy of the array.
> When these arrays get large, this causes a big performance hit. So, I
> am in the process of making my code less clever (and uglier) but much
> faster.

It's nice that we have to worry about these things in IDL.

I'm working in MATLAB right now helping a colleague run a large simulation and managing memory when you can only pass by value is a real pain/annoyance. Talk about a performance hit:

??? Error using ==> zeros
Out of memory. Type HELP MEMORY for your options.

The inability to pass by reference in MATLAB is *insane*. An annoyance when writing everyday code, a real hindrance when array sizes balloon.

-Rick

Subject: Re: Pass by value and performance
Posted by David Fanning on Fri, 16 Dec 2005 18:24:42 GMT
View Forum Message <> Reply to Message

Rick Towler writes:

> The inability to pass by reference in MATLAB is *insane*. An annoyance
> when writing everyday code, a real hindrance when array sizes balloon.

My goodness. Does the MatLab newsgroup know this? I would have
thought with the size of data ballooning daily that this alone
would have MatLab users clamoring to learn IDL.

Where's the number of the RSI marketing department...

Oh, wait. I guess I wouldn't want to explain "pass by
reference" in ten words or less to the unwashed masses either. :-(

Cheers,

David

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

Subject: Re: Pass by value and performance
Posted by JD Smith on Fri, 16 Dec 2005 18:40:18 GMT
View Forum Message <> Reply to Message

```
> a = 10
> b = PTR_NEW(40)
> c = PTR_NEW( BYTARR(100) )
>
>
>   Conventional Mem.                 |   HEAP memory
>   (Managed by IDL but               |   (Jungle where you are
>    without a Garbage Collector??? )    |    responsible to free)
>  ------------------------------------------------------------ ----
>          a = 10                    |
>          b --------------------------|--> 40
```

>   c ------------------------|--> [0,1,2,.....,99]

All memory is managed by IDL, without garbage collection. For normal variable memory, IDL takes care of allocating it when a variable goes into scope (e.g. you enter a procedure and a assign a value to a variable), and de-allocating when it goes out of scope (e.g. exiting a procedure). Heap memory is managed in the same way, except it only gets allocated when you act on a heap variable (via a pointer or object), and only gets de-allocated when you explicity free it (or use one of the heavy-handed clean-up routines like HEAP_FREE).

> Then if I call a function with:
>
> call_to_procedure, *c
>
> In bad C style I think I am passing the content of 'c', that is a BTYARR
> of 100 (BAD ???)

In C, everything is always passed by value. In IDL, everything is always passed by reference (more on this below). Here, you are passing by reference the heap variable which the pointer variable `c' points to. The fact that it is a pointer heap variable, and not a normal variable, is irrelevant.

In C, pointers are often used to avoid the pass-by-value overhead, so that the only thing passed by value is the lightweight pointer, and the full data it points to can be accessed efficiently and without copying. It's still passing by value, but it's such a small value, that you don't care, and, since the pointer gives you the address of the data you are really interested in, you can edit it at will. (As an aside, this form of lightweight pass-by-value is very likely what IDL uses at its C core to implement its default pass by reference behavior).

In IDL, pointers aren't normally used for this purpose, since everything is passed by reference by default. In IDL, pointers are used more for storing arbitrarily-sized data inside of structures, and objects, and keeping global persistent data around as you jump from procedure to procedure. IDL pointers shouldn't really even be called pointers; probably "references" is a better description of them. C pointers give you indirect hardware access to a block of memory. IDL pointers give you indirect access to a special pool of normal IDL variables called heap variables, special only in their lifetime and access semantics, but otherwise exactly the same as normal IDL variables.

> In IDL is passed a reference to the content, that is like if I writte:

>
> d = BYTARR(100)
> call_to_procedure, d
>
> Is this right??
>
> Thanks a lot.

Yep, again, IDL *always* passes by reference. True of pointer heap variables, and normal variables alike. As far as by-value vs. by-reference, normal vs. pointer heap variables makes no difference whatsoever. The way to think about IDL pass-by-value is as follows:

```
IDL> a=randomu(sd,100,100)
IDL> do_something, a[0:10,20:30]
```

When you pass a[0:10,20:30] as an argument, IDL creates a temporary array variable to hold the smaller subscripted array. It then passes this temporary array variable *by reference* into the procedure, just like normal. You can set this temporary array to another value inside the procedure, and it won't complain:

```
pro do_something, array
  array=12
end
```

However, as soon as your procedure completes, that temporary array variable is automatically destroyed, and you have not managed to set anything. So, it's not that IDL ever passes by value, just that it occasionally automatically creates and destroys temporary variables, which make it appear that arguments have been passed by value.

JD

Subject: Re: Pass by value and performance
Posted by Rick Towler on Fri, 16 Dec 2005 19:03:01 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Rick Towler writes:
>
>> The inability to pass by reference in MATLAB is *insane*. An annoyance
>> when writing everyday code, a real hindrance when array sizes balloon.
>
> My goodness. Does the MatLab newsgroup know this? I would have

> thought with the size of data ballooning daily that this alone
> would have MatLab users clamoring to learn IDL.

MATLAB newsgroup? Ha. There are a few very knowledgeable posters (and a pretty good showing from the mathworks.com domain) but the S/N is so poor it is very difficult to learn anything from it. But the IDL newsgroup... A valuable resource. Indispensable. And not a single reference to it on the RSI website. <sigh>

And I should clarify that MATLAB uses a "lazy" copy where it will not make a copy until you change the data. But this really doesn't help me right now. At least MATLAB has a 64bit linux version and a beta 64bit winXP version available. When in doubt, throw more memory at the problem.

As an aside, I think MATLAB definitely has some features over IDL (GUI builder and Java integration), and IDL over MATLAB (keywords and pointers and the VM). Ooooh the VM. You all should thank RSI for that. Mathworks charges a lot of money for the exact same thing.

-Rick

Subject: Re: Pass by value and performance
Posted by David Fanning on Fri, 16 Dec 2005 19:23:56 GMT
View Forum Message <> Reply to Message

Rick Towler writes:

> MATLAB newsgroup? Ha. There are a few very knowledgeable posters (and
> a pretty good showing from the mathworks.com domain) but the S/N is so
> poor it is very difficult to learn anything from it. But the IDL
> newsgroup... A valuable resource. Indispensable. And not a single
> reference to it on the RSI website. <sigh>

I think you forgot "fun". But RSI is supportive. I talked them out of an IDL T-shirt with less than an hour of shameless pandering. :-)

Cheers,

David

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

Subject: Re: Pass by value and performance
Posted by Dick Jackson on Mon, 19 Dec 2005 17:33:22 GMT
View Forum Message <> Reply to Message

Hi Rick,

"Rick Towler" <rick.towler@nomail.noaa.gov> wrote in message
news:dnv4mjhrc1@news.nems.noaa.gov...

> But the IDL newsgroup... A valuable resource. Indispensable. And not a
> single reference to it on the RSI website. <sigh>

Not meaning to be a stickler, but I remembered seeing something the other
day:

On this page:
http://www.rsinc.com/services/techres.asp

We find:
The comp.lang.idl-pvwave Newsgroup is an active, independent forum where
users exchange ideas and code. RSI occasionally provides input to this
forum, but does not regulate it.

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / +1-403-242-7398 / Fax: 241-7392

Subject: Re: Pass by value and performance
Posted by Rick Towler on Fri, 23 Dec 2005 18:04:12 GMT
View Forum Message <> Reply to Message

Dick Jackson wrote:

>> But the IDL newsgroup... A valuable resource. Indispensable. And not a
>> single reference to it on the RSI website. <sigh>
>
>
> Not meaning to be a stickler, but I remembered seeing something the other
> day:
>
> On this page:
> http://www.rsinc.com/services/techres.asp
>

> We find:
> The comp.lang.idl-pvwave Newsgroup is an active, independent forum where
> users exchange ideas and code. RSI occasionally provides input to this
> forum, but does not regulate it.

Dammit, Dick. O.K. Fine. Maybe I exaggerated :) ONE reference. My issue is that the newsgroup isn't front and center on the "community" page. When users, unaware of comp.lang.idl-pvwave, stumble to the RSI website looking for help they will most likely find the IDL user forum.

How many of you are posting there?

-Rick
