Subject: Solve memory problems
Posted by corinnefrey on Tue, 13 Jan 2009 13:28:12 GMT
View Forum Message <> Reply to Message

Hi everybody,

Again I am confrontend with memory problems in IDL. I am working with satellite images and their derivatives and therefore I am often confronted with huge datasets.

Well, running a routine once, some common tricks may help to enhance the available memory:

- Use the temporary-function,
- Put the variables, I don't need anymore, to zero,
- Working as much as possible with interger or byte arrays,
- etc.

However, specially when running an already optimized routine several times (e.g. for several scenes), available memory will disappear over short or long. I guess, this has to do with the growing fragmentation of the memory.

A solution could be to use the same variables for different grids in a subroutine. However, I prefer to name the variables according to their physical (or whatever) meaning. Otherwise, after some time, I don't have any chance to understand my programms anymore.

In my dreams, delvar is available also in routines. However, it's not, is it?

So, after this long introduction: What strategies are you using, to have more memory available, or to prevent the fragmentation of the memory?

Thanks for any comments.

Regards,
Corinne

Subject: Re: Solve memory problems
Posted by Carsten Lechte on Tue, 13 Jan 2009 13:57:44 GMT
View Forum Message <> Reply to Message

Corinne wrote:
> However, specially when running an already optimized routine several
> times (e.g. for several scenes), available memory will disappear over

> short or long.

All the memory you use in a function or procedure is given back when that
function returns, except for any variables that are handed back via
arguments or function return values.

This is pure conjecture, but maybe you use some memory in your function, then
allocate more memory for the return value, then return, freeing the earlier
memory, but leaving you with the return value allocated somewhere "in the
middle" of the heap memory, thereby fragmenting it more and more with each
call of the function. Then, it should make a difference if you allocate
your return array at the beginning of your function.

Maybe you could use memtest.pro to investigate the fragmentation problem?

chl

Subject: Re: Solve memory problems
Posted by corinnefrey on Tue, 13 Jan 2009 14:34:30 GMT
View Forum Message <> Reply to Message

hi carsten,

indeed, this is the case in my actual problem. i am using a programme,
which uses several subprogrammes and subsub-programmes. between each
level there is an exchange of variables. if i defined all variables
and return-variables at the beginning, i would run out of memory just
by defining the variables. but probably i could make a compilation of
the "most wanted" variables to have them at a constant place from the
beginning. could be worth checking out!

thanks,
corinne

> All the memory you use in a function or procedure is given back when that
> function returns, except for any variables that are handed back via
> arguments or function return values.
>
> This is pure conjecture, but maybe you use some memory in your function, then
> allocate more memory for the return value, then return, freeing the earlier
> memory, but leaving you with the return value allocated somewhere "in the
> middle" of the heap memory, thereby fragmenting it more and more with each
> call of the function. Then, it should make a difference if you allocate
> your return array at the beginning of your function.
>

> Maybe you could use memtest.pro to investigate the fragmentation problem?
>
> chl

Subject: Re: Solve memory problems
Posted by David Fanning on Tue, 13 Jan 2009 14:44:51 GMT
View Forum Message <> Reply to Message

Corinne writes:

> i'm sure, that my data sets will increase immediately after i would
> get a new OS :)

Yes, satellite data will become too low-level for you,
and you will need lidar data to know for sure. That
will increase your footprint another 30-40X. It is
a vicious path. Don't get started down it!! :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by Jean H. on Tue, 13 Jan 2009 15:03:47 GMT
View Forum Message <> Reply to Message

> Well, running a routine once, some common tricks may help to enhance
> the available memory:
>
> - Use the temporary-function,
> - Put the variables, I don't need anymore, to zero,
> - Working as much as possible with interger or byte arrays,
> - etc.

if not already in use, using pointers can be of great help. You can save
each band in a different pointer, or even each line etc. Therefore, the
size of the required contiguous memory space decreases.

> However, specially when running an already optimized routine several
> times (e.g. for several scenes), available memory will disappear over

> short or long. I guess, this has to do with the growing fragmentation
> of the memory.

As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
out what is happening. It could as well be a memory leak (you create a
pointer but don't destroy it). In this case, make a call to "heap_gc"
after your function.

> A solution could be to use the same variables for different grids in a
> subroutine. However, I prefer to name the variables according to their
> physical (or whatever) meaning. Otherwise, after some time, I don't
> have any chance to understand my programms anymore.
>
> In my dreams, delvar is available also in routines. However, it's not,
> is it?
>
> So, after this long introduction: What strategies are you using, to
> have more memory available, or to prevent the fragmentation of the
> memory?

Under windows, try to assign an array as big as possible, 1st thing in
your program. At least you are "reserving" the contiguous memory for IDL.

Switching to Linux made my life soooooo much easier, for any
memory-related problem!

Jean

>
> Thanks for any comments.
>
> Regards,
> Corinne
>

Subject: Re: Solve memory problems
Posted by David Fanning on Tue, 13 Jan 2009 15:33:40 GMT
View Forum Message <> Reply to Message

Jean H. writes:

> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
> out what is happening. It could as well be a memory leak (you create a
> pointer but don't destroy it). In this case, make a call to "heap_gc"
> after your function.

What!? What kind of advice is this!

Uh, do NOT be making a call to HEAP_GC unless your program
has completely and utterly failed and it is late Friday
afternoon and you are at wit's end. Believe me when I tell
you there are MUCH better ways to handle this!

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by Carsten Lechte on Tue, 13 Jan 2009 15:54:38 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Uh, do NOT be making a call to HEAP_GC unless your program
> has completely and utterly failed and it is late Friday
> afternoon and you are at wit's end.

It CAN be used for good! Trust me!

I find that the sequence

```
   HELP, /MEMORY
   HEAP_GC
   HELP, /MEMORY
```

is useful for DIAGNOSING forgotten pointers.

chl

Subject: Re: Solve memory problems
Posted by Allan Whiteford on Tue, 13 Jan 2009 15:56:35 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Jean H. writes:
>
>

>> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
>> out what is happening. It could as well be a memory leak (you create a
>> pointer but don't destroy it). In this case, make a call to "heap_gc"
>> after your function.
>
>
> What!? What kind of advice is this!
>
> Uh, do NOT be making a call to HEAP_GC unless your program
> has completely and utterly failed and it is late Friday
> afternoon and you are at wit's end. Believe me when I tell
> you there are MUCH better ways to handle this!
>
> Cheers,
>
> David

Perhaps a compromise:

Do a "help,/heap" and see how many pointers you have sitting, then do a
heap_gc followed immediately by a "help,/heap" again. If you're leaking
memory by not freeing pointers or destroying objects then chances are
the two results of help,/heap will be different. If they are the same
then the heap_gc didn't do anything and the problem is elsewhere.

help,/heap will even give you an idea of what heap variable is causing
the problem.

Thanks,

Allan

Subject: Re: Solve memory problems
Posted by Jean H. on Tue, 13 Jan 2009 16:28:30 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Jean H. writes:
>
>> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
>> out what is happening. It could as well be a memory leak (you create a
>> pointer but don't destroy it). In this case, make a call to "heap_gc"
>> after your function.
>
> What!? What kind of advice is this!
>
> Uh, do NOT be making a call to HEAP_GC unless your program

> has completely and utterly failed and it is late Friday
> afternoon and you are at wit's end. Believe me when I tell
> you there are MUCH better ways to handle this!
>
> Cheers,
>
> David

what is wrong with Heap_Gc? I agree it's better to destroy each pointer
the proper way, but I have no clue of what is wrong with it! I do use is
from time to time and it has never failed on me!

Jean

Subject: Re: Solve memory problems
Posted by David Fanning on Tue, 13 Jan 2009 17:14:57 GMT

View Forum Message <> Reply to Message

Jean H. writes:

> what is wrong with Heap_Gc? I agree it's better to destroy each pointer
> the proper way, but I have no clue of what is wrong with it! I do use is
> from time to time and it has never failed on me!

It is just such a...I don't know...admission of
failure. It's letting IDL do something the programmer
should have done. I guess the basic thing wrong with
it is it is just so damn inelegant. It's kind of like
using your crescent wrench as a hammer. :-)

Cheers,

David
--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by David Fanning on Tue, 13 Jan 2009 21:01:04 GMT

View Forum Message <> Reply to Message

Jean H. writes:

> if not already in use, using pointers can be of great help. You can save

> each band in a different pointer, or even each line etc. Therefore, the
> size of the required contiguous memory space decreases.

I was so upset with the HEAP_GC suggestion this morning
that I forgot to respond to this. But I don't immediately
see how this will help, unless, of course you have some
ability to read all three bands from the file as bands
and not as an image.

Otherwise, I think you would have to allocate twice
the image size to do the transfer to pointers.

Plus, you have to put the bands together again to display
the image, so I don't see where I am ahead of anything
here. What am I missing?

Cheers,

David

--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by Craig Markwardt on Wed, 14 Jan 2009 06:26:23 GMT
View Forum Message <> Reply to Message

On Jan 13, 10:33 am, David Fanning <n...@dfanning.com> wrote:
> Jean H. writes:
>> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
>> out what is happening. It could as well be a memory leak (you create a
>> pointer but don't destroy it). In this case, make a call to "heap_gc"
>> after your function.
>
> What!? What kind of advice is this!
>
> Uh, do NOT be making a call to HEAP_GC unless your program
> has completely and utterly failed and it is late Friday
> afternoon and you are at wit's end. Believe me when I tell
> you there are MUCH better ways to handle this!

Uh, like using any other high-level language that doesn't force you to
free your own variables?

Subject: Re: Solve memory problems
Posted by corinnefrey on Wed, 14 Jan 2009 11:56:06 GMT
View Forum Message <> Reply to Message

hi guys,

now i got lots of inputs! thanks so much! will check memtest.pro, as
well as this pointer stuff and heap_gc. if it helps, why not?

@david: my 'images' contain normally up to 9 bands. normally i handle
them separate (read in only exactly the band i need), as the file
would be too big otherwise.

best regards,
corinne

On Jan 14, 7:26 am, Craig Markwardt <cbmarkwa...@gmail.com> wrote:
> On Jan 13, 10:33 am, David Fanning <n...@dfanning.com> wrote:
>
>> Jean H. writes:
>>> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
>>> out what is happening. It could as well be a memory leak (you create a
>>> pointer but don't destroy it). In this case, make a call to "heap_gc"
>>> after your function.
>
>> What!? What kind of advice is this!
>
>> Uh, do NOT be making a call to HEAP_GC unless your program
>> has completely and utterly failed and it is late Friday
>> afternoon and you are at wit's end. Believe me when I tell
>> you there are MUCH better ways to handle this!
>
> Uh, like using any other high-level language that doesn't force you to
> free your own variables?

Subject: Re: Solve memory problems
Posted by Jean H. on Wed, 14 Jan 2009 12:55:35 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Jean H. writes:
>
>> if not already in use, using pointers can be of great help. You can save
>> each band in a different pointer, or even each line etc. Therefore, the
>> size of the required contiguous memory space decreases.
>

> I was so upset with the HEAP_GC suggestion this morning
> that I forgot to respond to this. But I don't immediately
> see how this will help, unless, of course you have some
> ability to read all three bands from the file as bands
> and not as an image.
>
> Otherwise, I think you would have to allocate twice
> the image size to do the transfer to pointers.
>
> Plus, you have to put the bands together again to display
> the image, so I don't see where I am ahead of anything
> here. What am I missing?
>
> Cheers,
>
> David

Hi,

Agreed for the memory cost associated with loading the data... but down the road (or down the program), it might still be useful to save memory.. for example, if the routine that is being called needs 10 times the memory of the input.

In my own program, I do all the analysis on a modified version of my original image (a classified land-use map, with the background values removed so the data is a 1D array), then, at the very end, I re-transform it to save and display it. I save a lot of memory! Moreover, all bands are saved in pointers, allowing the program to run on almost any computer, while the original version, which did not use much pointer, was making my work-beast run out of memory fairly quickly!

Jean

Subject: Re: Solve memory problems
Posted by corinnefrey on Wed, 14 Jan 2009 13:37:36 GMT

View Forum Message <> Reply to Message

hi jean,

i have never used pointers, so my question is: do you use separate pointers or whole pointer-arrays for your bands? what do you do, if your bands have different sizes due to different spatial resolutions (which you might interpolate later in the programme)?

i'm still trying to figure out, what the advantage of pointers is.

example: i have created an float array a with 4000x5000 elements and
don't need it anymore. so i want to get rid of it. does it make a
difference, if i put a=0 or if i set the value of the pointer of the
array to zero?

regards,
corinne

> In my own program, I do all the analysis on a modified version of my
> original image (a classified land-use map, with the background values
> removed so the data is a 1D array), then, at the very end, I
> re-transform it to save and display it. I save a lot of memory!
> Moreover, all bands are saved in pointers, allowing the program to run
> on almost any computer, while the original version, which did not use
> much pointer, was making my work-beast run out of memory fairly quickly!
>
> Jean

Subject: Re: Solve memory problems
Posted by David Fanning on Wed, 14 Jan 2009 14:03:18 GMT
View Forum Message <> Reply to Message

Craig Markwardt writes:

> Uh, like using any other high-level language that doesn't force you to
> free your own variables?

My goodness, people, whatever happened to craftsmanship?
I feel like I'm working with a bunch of Wal-Mart furniture
builders here. Quick, easy, cheap. :-(

Cheers,

David

P.S. I guess you probably know it is time to give it up when
you get to the point where you believe the world will go
straight to hell in a handbasket if you *do* give it up.
Just don't be sending me any code with HEAP_GC in it. Even
Coyote wouldn't lower himself *that* much. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by Jean H. on Wed, 14 Jan 2009 14:53:54 GMT
View Forum Message <> Reply to Message

Corinne,

setting var=0 or *ptr_var = 0 will have the same effect on memory.
Now, let's say you have 10 bands, 4000*5000. If you try to create an
array like this data = bytarr(4000,5000,10), you might run out of
memory, or, for the same reason, can not create any other variable (not
enough contiguous space in memory). With this example, you would need
about 8 bits * 4000 * 5000 * 10 = 1 600 000 000 bits of contiguous
memory. Now, if you use points, you can create an array of 10 pointers,
each holding a band.
ptr_data = ptrarr(10)
ptr_data[0] = ptr_new(bytarr(4000,5000)
ptr_data[1] = ptr_new(bytarr(4000,5000)
....

so now, the contiguous memory you need is only 8 bits * 4000 * 5000 =
160 000 000 bits.

If band 2 has a different size, no problem:
ptr_data[2] = ptr_new(bytarr(12,25)

Jean

Corinne wrote:
> hi jean,
>
> i have never used pointers, so my question is: do you use separate
> pointers or whole pointer-arrays for your bands? what do you do, if
> your bands have different sizes due to different spatial resolutions
> (which you might interpolate later in the programme)?
>
> i'm still trying to figure out, what the advantage of pointers is.
>
> example: i have created an float array a with 4000x5000 elements and
> don't need it anymore. so i want to get rid of it. does it make a
> difference, if i put a=0 or if i set the value of the pointer of the
> array to zero?
>
> regards,
> corinne
>
>
>> In my own program, I do all the analysis on a modified version of my
>> original image (a classified land-use map, with the background values
>> removed so the data is a 1D array), then, at the very end, I

>> re-transform it to save and display it. I save a lot of memory!
>> Moreover, all bands are saved in pointers, allowing the program to run
>> on almost any computer, while the original version, which did not use
>> much pointer, was making my work-beast run out of memory fairly quickly!
>>
>> Jean
>

Subject: Re: Solve memory problems
Posted by Paul Van Delst[1] on Wed, 14 Jan 2009 14:59:40 GMT
View Forum Message <> Reply to Message

Corinne wrote:
> hi guys,
>
> now i got lots of inputs! thanks so much! will check memtest.pro, as
> well as this pointer stuff and heap_gc. if it helps, why not?

Your unit test suite should include a test case that checks for valid pointers (and
objects if appropriate) after your cleanup routines. If there are still non-null pointers
floating around, your cleanup is incomplete. It's very simple to test for this stuff --
even in what I would refer to as "research grade" code.

I second David's incredulity at suggestions that generic garbage collection is an
acceptable solution when you have memory issues. When the problems stem from
structures/variables that the programmer created, it behooves said programmer to fix the
problems. Otherwise it's just sloppy programming -- and you're learning bad habits that,
when transferred to other languages, may not be so easily corrected.

It reminds me of the old chestnut about three properties of software: fast, good, cheap.
Pick any two. :o)

cheers,

paulv

>
> @david: my 'images' contain normally up to 9 bands. normally i handle
> them separate (read in only exactly the band i need), as the file
> would be too big otherwise.
>
> best regards,
> corinne
>
>
> On Jan 14, 7:26 am, Craig Markwardt <cbmarkwa...@gmail.com> wrote:

>> On Jan 13, 10:33 am, David Fanning <n...@dfanning.com> wrote:
>>
>>> Jean H. writes:
>>>> As Carsten has mentioned, play with memtest.pro (from ITTVIS) to find
>>>> out what is happening. It could as well be a memory leak (you create a
>>>> pointer but don't destroy it). In this case, make a call to "heap_gc"
>>>> after your function.
>>> What!? What kind of advice is this!
>>> Uh, do NOT be making a call to HEAP_GC unless your program
>>> has completely and utterly failed and it is late Friday
>>> afternoon and you are at wit's end. Believe me when I tell
>>> you there are MUCH better ways to handle this!
>> Uh, like using any other high-level language that doesn't force you to
>> free your own variables?
>

Subject: Re: Solve memory problems
Posted by Paul Van Delst[1] on Wed, 14 Jan 2009 15:08:23 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Craig Markwardt writes:
>
>> Uh, like using any other high-level language that doesn't force you to
>> free your own variables?
>
> My goodness, people, whatever happened to craftsmanship?
> I feel like I'm working with a bunch of Wal-Mart furniture
> builders here. Quick, easy, cheap. :-(

Hey! I often use woodworking craftmenship analogies when trying to get people to take some
pride (for pete's sake!) in the code they write. A lot of code I see is the equivalent of
a Homer Simpson spice rack (or barbeque) - it might qualify as abstract art, but it ain't
that functional.

> P.S. I guess you probably know it is time to give it up when
> you get to the point where you believe the world will go
> straight to hell in a handbasket if you *do* give it up.
> Just don't be sending me any code with HEAP_GC in it. Even
> Coyote wouldn't lower himself *that* much. :-)

Keep fighting the good fight, David. :o)

cheers,

paulv

>

Subject: Re: Solve memory problems
Posted by pgrigis on Wed, 14 Jan 2009 15:16:25 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Craig Markwardt writes:
>
>> Uh, like using any other high-level language that doesn't force you to
>> free your own variables?
>
> My goodness, people, whatever happened to craftsmanship?
> I feel like I'm working with a bunch of Wal-Mart furniture
> builders here. Quick, easy, cheap. :-(
>
> Cheers,
>
> David
>
> P.S. I guess you probably know it is time to give it up when
> you get to the point where you believe the world will go
> straight to hell in a handbasket if you *do* give it up.
> Just don't be sending me any code with HEAP_GC in it. Even
> Coyote wouldn't lower himself *that* much. :-)

Well, I must say that garbage collection is not such a bad idea,
or else I would have an unpleasant and smelly pile of stuff in
front of my house... ;-)

Ciao,
Paolo

>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by David Fanning on Wed, 14 Jan 2009 15:23:11 GMT
View Forum Message <> Reply to Message

Paolo writes:

> Well, I must say that garbage collection is not such a bad idea,
> or else I would have an unpleasant and smelly pile of stuff in
> front of my house... ;-)

What!? You put your compost pile in the *front* yard? :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by pgrigis on Wed, 14 Jan 2009 15:38:50 GMT
View Forum Message <> Reply to Message

But probably this is important only on windows 32-bit systems?

Ciao,
Paolo

Jean H. wrote:
> Corinne,
>
> setting var=0 or *ptr_var = 0 will have the same effect on memory.
> Now, let's say you have 10 bands, 4000*5000. If you try to create an
> array like this data = bytarr(4000,5000,10), you might run out of
> memory, or, for the same reason, can not create any other variable (not
> enough contiguous space in memory). With this example, you would need
> about 8 bits * 4000 * 5000 * 10 = 1 600 000 000 bits of contiguous
> memory. Now, if you use points, you can create an array of 10 pointers,
> each holding a band.
> ptr_data = ptrarr(10)
> ptr_data[0] = ptr_new(bytarr(4000,5000)
> ptr_data[1] = ptr_new(bytarr(4000,5000)
> ....
>
> so now, the contiguous memory you need is only 8 bits * 4000 * 5000 =
> 160 000 000 bits.
>
> If band 2 has a different size, no problem:
> ptr_data[2] = ptr_new(bytarr(12,25)

>
> Jean
>
> Corinne wrote:
>> hi jean,
>>
>> i have never used pointers, so my question is: do you use separate
>> pointers or whole pointer-arrays for your bands? what do you do, if
>> your bands have different sizes due to different spatial resolutions
>> (which you might interpolate later in the programme)?
>>
>> i'm still trying to figure out, what the advantage of pointers is.
>>
>> example: i have created an float array a with 4000x5000 elements and
>> don't need it anymore. so i want to get rid of it. does it make a
>> difference, if i put a=0 or if i set the value of the pointer of the
>> array to zero?
>>
>> regards,
>> corinne
>>
>>
>>> In my own program, I do all the analysis on a modified version of my
>>> original image (a classified land-use map, with the background values
>>> removed so the data is a 1D array), then, at the very end, I
>>> re-transform it to save and display it. I save a lot of memory!
>>> Moreover, all bands are saved in pointers, allowing the program to run
>>> on almost any computer, while the original version, which did not use
>>> much pointer, was making my work-beast run out of memory fairly quickly!
>>>
>>> Jean
>>

Subject: Re: Solve memory problems
Posted by Craig Markwardt on Wed, 14 Jan 2009 16:23:05 GMT

View Forum Message <> Reply to Message

On Jan 14, 9:03 am, David Fanning <n...@dfanning.com> wrote:
> Craig Markwardt writes:
>> Uh, like using any other high-level language that doesn't force you to
>> free your own variables?
>
> My goodness, people, whatever happened to craftsmanship?
> I feel like I'm working with a bunch of Wal-Mart furniture
> builders here. Quick, easy, cheap. :-(

Uh, I wasn't defending the use of HEAP_GC, but it's monumentally silly

that the IDL language designers had the choice to implement automatic
freeing of dangling pointers ("garbage collection") and did not [*].
I mean, would you really enjoy the "privilege" of freeing every
regular variable before returning from each IDL procedure? Of
course not. The IDL runtime has enough information to know *exactly*
when a pointer becomes dangling, so why not use that information?

I think I understand craftmanship -- I hope my public code speaks for
itself. But I don't think that has anything to do with masochistic
worship at the alter of POINTER_FREE.

Snark-Craig

[*] - I understand that for debugging purposes, automatic garbage
collection may be a nuisance. It would be straightforward to disable
it with a system variable.

>
> Cheers,
>
> David
>
> P.S. I guess you probably know it is time to give it up when
> you get to the point where you believe the world will go
> straight to hell in a handbasket if you *do* give it up.
> Just don't be sending me any code with HEAP_GC in it. Even
> Coyote wouldn't lower himself *that* much. :-)
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by David Fanning on Wed, 14 Jan 2009 16:48:29 GMT
View Forum Message <> Reply to Message

Craig Markwardt writes:

> I think I understand craftmanship -- I hope my public code speaks for
> itself. But I don't think that has anything to do with masochistic
> worship at the alter of POINTER_FREE.
>
> Snark-Craig

Yeah, I feel guilty snarking at you, of all people.

Best curve fitting code on the planet, etc., etc.
Please don't take it personally.

I've scheduled an appointment with the therapist to
see if I can get to the bottom of why this HEAP_GC
thing bothers me so much. :-)

> [*] - I understand that for debugging purposes, automatic garbage
> collection may be a nuisance. It would be straightforward to disable
> it with a system variable.

I guess I would be more concerned with how it would impact
the speed of FOR loops. ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Solve memory problems
Posted by Michael Galloy on Wed, 14 Jan 2009 17:55:36 GMT

View Forum Message <> Reply to Message

On Jan 14, 9:23 am, Craig Markwardt <cbmarkwa...@gmail.com> wrote:
> On Jan 14, 9:03 am, David Fanning <n...@dfanning.com> wrote:
>
>> Craig Markwardt writes:
>>> Uh, like using any other high-level language that doesn't force you to
>>> free your own variables?
>
>> My goodness, people, whatever happened to craftsmanship?
>> I feel like I'm working with a bunch of Wal-Mart furniture
>> builders here. Quick, easy, cheap. :-(
>
> Uh, I wasn't defending the use of HEAP_GC, but it's monumentally silly
> that the IDL language designers had the choice to implement automatic
> freeing of dangling pointers ("garbage collection") and did not [*].
> I mean, would you really enjoy the "privilege" of freeing every
> *regular* variable before returning from each IDL procedure? Of
> course not. The IDL runtime has enough information to know *exactly*
> when a pointer becomes dangling, so why not use that information?
>

> I think I understand craftmanship -- I hope my public code speaks for
> itself. But I don't think that has anything to do with masochistic
> worship at the alter of POINTER_FREE.
>
> Snark-Craig
>
> [*] - I understand that for debugging purposes, automatic garbage
> collection may be a nuisance. It would be straightforward to disable
> it with a system variable.

I must say that garbage collection is one of the features I really
appreciate when I use Python. Garbage collection is now even in OS X's
objective C.

That said, IDL does not have the same garbage collection algorithms.
HEAP_GC is *slow* when you have a lot of heap variables. I would
support IDL adding real garbage collection, but what is currently in
IDL is a debugging tool only.

Mike
--
www.michaelgalloy.com
Tech-X Corporation
Associate Research Scientist

Subject: Re: Solve memory problems
Posted by Craig Markwardt on Thu, 15 Jan 2009 07:41:14 GMT
View Forum Message <> Reply to Message

On Jan 14, 12:55 pm, "mgal...@gmail.com" <mgal...@gmail.com> wrote:
>
> That said, IDL does not have the same garbage collection algorithms.
> HEAP_GC is *slow* when you have a lot of heap variables. I would
> support IDL adding real garbage collection, but what is currently in
> IDL is a debugging tool only.

It doesn't really have to be garbage collection. Python uses
reference-counting, and IDL could easily do the same kind of thing,
with extremely low overhead. Internal to IDL, what is needed is one
additional integer for each heap variable and three extra instructions
per pointer assignment.

Garbage collection might still be useful for when cycles occur.

Craig

Subject: Re: Solve memory problems
Posted by corinnefrey on Thu, 15 Jan 2009 11:45:01 GMT
View Forum Message <> Reply to Message

Hi Jean,

Jap, got it! This is indeed an advantage. Will try to implement that
in my next programme.

Have a nice day,
Corinne

On Jan 14, 3:53 pm, "Jean H." <jghas...@DELTHIS.ucalgary.ANDTHIS.ca>
wrote:
> Corinne,
>
> setting var=0 or *ptr_var = 0 will have the same effect on memory.
> Now, let's say you have 10 bands, 4000*5000. If you try to create an
> array like this data = bytarr(4000,5000,10), you might run out of
> memory, or, for the same reason, can not create any other variable (not
> enough contiguous space in memory). With this example, you would need
> about 8 bits * 4000 * 5000 * 10 = 1 600 000 000 bits of contiguous
> memory. Now, if you use points, you can create an array of 10 pointers,
> each holding a band.
> ptr_data = ptrarr(10)
> ptr_data[0] = ptr_new(bytarr(4000,5000)
> ptr_data[1] = ptr_new(bytarr(4000,5000)
> ....
>
> so now, the contiguous memory you need is only 8 bits * 4000 * 5000 =
> 160 000 000 bits.
>
> If band 2 has a different size, no problem:
> ptr_data[2] = ptr_new(bytarr(12,25)
>
> Jean
>
> Corinne wrote:
>> hi jean,
>
>> i have never used pointers, so my question is: do you use separate
>> pointers or whole pointer-arrays for your bands? what do you do, if
>> your bands have different sizes due to different spatial resolutions
>> (which you might interpolate later in the programme)?
>
>> i'm still trying to figure out, what the advantage of pointers is.
>
>> example: i have created an float array a with 4000x5000 elements and

>> don't need it anymore. so i want to get rid of it. does it make a
>> difference, if i put a=0 or if i set the value of the pointer of the
>> array to zero?

>
>> regards,
>> corinne
>
>>> In my own program, I do all the analysis on a modified version of my
>>> original image (a classified land-use map, with the background values
>>> removed so the data is a 1D array), then, at the very end, I
>>> re-transform it to save and display it. I save a lot of memory!
>>> Moreover, all bands are saved in pointers, allowing the program to run
>>> on almost any computer, while the original version, which did not use
>>> much pointer, was making my work-beast run out of memory fairly quickly!
>
>>> Jean
>
>

Subject: Re: Solve memory problems
Posted by Guillaume Potdevin on Thu, 15 Jan 2009 12:10:29 GMT
View Forum Message <> Reply to Message

Hi!

I followed with great interest this discussion, as I sometimes run into
the same sort of problems, though for other sort of applications.

For reason of commodity, I often use structures. Do structures require a
contiguous space in memory? For example: does
struct = {image1 : BYTARR(4000,5000), image2 : BYTARR(4000,5000) }
require 8 bits * 4000 * 5000 * 2 of continuous space?

And, also the same question in the case of structure arrays: if we have
struct_array = REPLICATE(struct , many_times)
Does struct_array require many_times the space for struct as contiguous
space (in case struct needs a contiguous memory slot)?

But I see the immediate solution here to use arrays of pointers...

Guillaume.

On 14.01.2009 15:53, Jean H. wrote:
> Corinne,
>

> setting var=0 or *ptr_var = 0 will have the same effect on memory.
> Now, let's say you have 10 bands, 4000*5000. If you try to create an
> array like this data = bytarr(4000,5000,10), you might run out of
> memory, or, for the same reason, can not create any other variable (not
> enough contiguous space in memory). With this example, you would need
> about 8 bits * 4000 * 5000 * 10 = 1 600 000 000 bits of contiguous
> memory. Now, if you use points, you can create an array of 10 pointers,
> each holding a band.
> ptr_data = ptrarr(10)
> ptr_data[0] = ptr_new(bytarr(4000,5000)
> ptr_data[1] = ptr_new(bytarr(4000,5000)
> ....
>
> so now, the contiguous memory you need is only 8 bits * 4000 * 5000 =
> 160 000 000 bits.
>
> If band 2 has a different size, no problem:
> ptr_data[2] = ptr_new(bytarr(12,25)
>
> Jean
>