## Subject: large arrays and transpose
Posted by Sean Raffuse on Mon, 01 Jul 2002 14:49:32 GMT
View Forum Message <> Reply to Message

Hello.

I have an array like so:

output = intarr(100, 3600,1800)

At the end of my program, I would like to do this:
output = TRANSPOSE(output, [1, 2, 0])

Problem is that the array is too large, and I am "unable to allocate memory
to make array."

I've tried using the TEMPORARY function with the same result.  Any idea on
how to achieve this operation without eating all of the rest of my memory?

thanks in advance,

Sean

## Subject: Re: large array
Posted by btt on Tue, 12 Oct 2004 17:35:39 GMT
View Forum Message <> Reply to Message

```
Wolf Schweitzer wrote:
>  Update.
>
>  I believe that IDL can STILL only work with arrays of 2 GB size due to
>  internal representation even on systems where more space is available.
>  Positioning along any huge file using point_lun and the Long64-function
>  and then reading a <2GB-part is no problem.
>
>  http://www.rsinc.com/services/techtip.asp?ttid=2597
>
>  But try:
>
>  a = intarr (2000,2000,400)
>  a [1999,1999,299] = 12
>  b = where (a eq 12)
>  help, b  ; it is a LONG ... and that is a problem in my eyes;
>           ; it shows they represent arrays with scalar counters and
>           ; as long as these are LONG there is no way an array can be
>           ; larger
>
```

```
>
> a = intarr (2000,2000,800)
> a [1999,1999,799] = 12
> b = where (a eq 12) ; (this really should crash your IDL)
>
> help, b    ; i would be shocked if you get that far, but, please
>            ; tell me what data type you obtain here
> print, b   ;  let me see this if you can, too
>
> Does anyone know when RSI will fix this serious limit? Until then, it'll
> probably be the chunk trick they wrote about.
>
>
>
>
> Wolf Schweitzer wrote:
>
>> I need to read a file that is ca. 7 GB large (the file size is
>> defineda s 2048 x 2048 x 900 are the dimensions, x 2 (integers) + 512
>> bytes header).
>>
>> The format is known (.ISQ) and I have a routine that deals with the
>> header information efficiently; I have some similarly formatted files
>> of the same scan process around 800-900 MB that I can read without
>> problems.
>>
>> Generally, I had no problems reading files up to 1.4 GB in size
>> directly into one variable under IDL.
>>
>> The data is read into an intarr (2048,2048,900) like this:
>>
>>     imagearray = assoc (.., intarr(...), headersize)
>>     imagearray = temporary (imagearray [0])
>>
>> Now, the reading takes a little while but it seems to work alright.
>> The machine - most of the time - does not crash (12 GB RAM, 64-bit AIX
>> version of IDL 6.1, IBM-Workstation).
>>
>> However, the visualisation of slice subscripts to this array later
>> does not display any interesting information; instead, the images of
>> these large data files look different each time and they do not
>> reflect the content of the data.
>>
>>
>> Question:
>>
>> What do I need to know in order to set up an array for very large
>> data? Is there a basic difference between arrays < 2 GB and arrays >
```

>>  GB of size? Are the subscript variables all multiplied before the
>>  array is really looked at, so do all of the subscript variables need
>>  to be Long64? Such as intarr (long64(x),long64(y),long64(z))?
>>

Hello,

I'm the last guy who will ever understand all of this - but, here's a couple of things I have picked-up on the way.

(1) The big-file thing is releated to the memory_bits and file_offset_bits IDL operates under.  It seems those are fixed by your platform and/or operating system.  You can determine what you have to play with for any given verion of IDL with ...

```
IDL> help, !Version,/str
** Structure !VERSION, 8 tags, length=76, data length=76:
   ARCH            STRING    'ppc'
   OS              STRING    'darwin'
   OS_FAMILY       STRING    'unix'
   OS_NAME         STRING    'Mac OS X'
   RELEASE         STRING    '6.1'
   BUILD_DATE      STRING    'Jul 14 2004'
   MEMORY_BITS     INT             32
   FILE_OFFSET_BITS
                   INT             32
```

(2) If you are fortunate enough to have 64-bit access then you can use the /L64 keyword to WHERE, HISTOGRAM, etc. so that 64-bit integers are returned.

(3) I don't follow what you mean here...

 >> However, the visualisation of slice subscripts to this array later
 >> does not display any interesting information; instead, the images of
 >> these large data files look different each time and they do not
 >> reflect the content of the data.
 >>

Could you explain that again?

Cheers,
Ben

---

Subject: Re: large array
Posted by Dick Jackson on Tue, 12 Oct 2004 17:43:48 GMT
View Forum Message <> Reply to Message

Hi Wolf,

IDL's WHERE function recently (6.0?) received a new keyword, /L64, with this
listing in the Online Help:

L64
By default, the result of WHERE is 32-bit integer when possible, and 64-bit
integer if the number of elements being processed requires it. Set L64 to
force 64-bit integers to be returned in all cases.

Note
Only 64-bit versions of IDL are capable of creating variables requiring a
64-bit result. Check the value of !VERSION.MEMORY_BITS to see if your IDL is
64-bit or not.

Perhaps something in this will help you out.

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

"Wolf Schweitzer" <shwi@irm.unizh.ch> wrote in message
news:416c0503$1@idnews.unizh.ch...
> Update.
>
> I believe that IDL can STILL only work with arrays of 2 GB size due to
> internal representation even on systems where more space is available.
> Positioning along any huge file using point_lun and the Long64-function
> and then reading a <2GB-part is no problem.
>
> http://www.rsinc.com/services/techtip.asp?ttid=2597
>
> But try:
>
```
> a = intarr (2000,2000,400)
> a [1999,1999,299] = 12
> b = where (a eq 12)
> help, b  ; it is a LONG ... and that is a problem in my eyes;
>          ; it shows they represent arrays with scalar counters and
>          ; as long as these are LONG there is no way an array can be
>          ; larger
>
>
```

> a = intarr (2000,2000,800)
> a [1999,1999,799] = 12
> b = where (a eq 12) ; (this really should crash your IDL)
>
> help, b ; i would be shocked if you get that far, but, please
> ; tell me what data type you obtain here
> print, b ; let me see this if you can, too
>
> Does anyone know when RSI will fix this serious limit? Until then, it'll
> probably be the chunk trick they wrote about.
>
>
>
>
> Wolf Schweitzer wrote:
>
>> I need to read a file that is ca. 7 GB large (the file size is defineda s
>> 2048 x 2048 x 900 are the dimensions, x 2 (integers) + 512 bytes header).
>>
>> The format is known (.ISQ) and I have a routine that deals with the
>> header information efficiently; I have some similarly formatted files of
>> the same scan process around 800-900 MB that I can read without problems.
>>
>> Generally, I had no problems reading files up to 1.4 GB in size directly
>> into one variable under IDL.
>>
>> The data is read into an intarr (2048,2048,900) like this:
>>
>> imagearray = assoc (.., intarr(...), headersize)
>> imagearray = temporary (imagearray [0])
>>
>> Now, the reading takes a little while but it seems to work alright. The
>> machine - most of the time - does not crash (12 GB RAM, 64-bit AIX
>> version of IDL 6.1, IBM-Workstation).
>>
>> However, the visualisation of slice subscripts to this array later does
>> not display any interesting information; instead, the images of these
>> large data files look different each time and they do not reflect the
>> content of the data.
>>
>>
>> Question:
>>
>> What do I need to know in order to set up an array for very large data?
>> Is there a basic difference between arrays < 2 GB and arrays > GB of
>> size? Are the subscript variables all multiplied before the array is
>> really looked at, so do all of the subscript variables need to be Long64?
>> Such as intarr (long64(x),long64(y),long64(z))?

>>
>> Thanks,
>> Wolf.

---

Subject: Re: large array
Posted by Wolf Schweitzer on Wed, 13 Oct 2004 16:19:33 GMT
View Forum Message <> Reply to Message

Dick Jackson wrote:

> L64
> By default, the result of WHERE is 32-bit integer when possible, and 64-bit
> integer if the number of elements being processed requires it. Set L64 to
> force 64-bit integers to be returned in all cases.

Thank you for this tip! Yes, I could get a sensible response from IDL
this time.

> Note
> Only 64-bit versions of IDL are capable of creating variables requiring a
> 64-bit result. Check the value of !VERSION.MEMORY_BITS to see if your IDL is
> 64-bit or not.

Yes, it's 64-bit.

--
Ben Tupper wrote:

```
 > IDL> help, !Version,/str
 > ** Structure !VERSION, 8 tags, length=76, data length=76:
 >    ARCH            STRING    'ppc'
 >    OS              STRING    'darwin'
 >    OS_FAMILY       STRING    'unix'
 >    OS_NAME         STRING    'Mac OS X'
 >    RELEASE         STRING    '6.1'
 >    BUILD_DATE      STRING    'Jul 14 2004'
 >    MEMORY_BITS     INT             32
 >    FILE_OFFSET_BITS
 >                    INT             32
```

OK, this reflected that I use an AIX 64-bit version of IDL 6.1.

> (2) If you are fortunate enough to have 64-bit access then you can use
> the /L64 keyword to WHERE, HISTOGRAM, etc. so that 64-bit integers are
> returned.

Thanks! I have found that the /L64-keyword is the key for these functions.

> >> However, the visualisation of slice subscripts to this array later
> >> does not display any interesting information; instead, the images of
> >> these large data files look different each time and they do not
> >> reflect the content of the data.
> >>
>
> Could you explain that again?

Here is what I meant:

Having read an integer image array of the dimensions ~[2048,2048,900] using the assoc function from a ~7 GB file, showing a slice such as TV (imagearray[*,*,12]), or [*,*,800] results in some type of noise, garbage, or however you want to put it. It looks off. Some of the slice images look like mixed byte order, some are simply noisy, and many are empty.

And it is not a windowing, byte order, variable type, tvscl, or file reading problem - I have smaller data sets of the same type that I can process perfectly to show that this is not the problem. It's the file size with certainty.

Here is what I did:

Encouraged and inspired that I am IN FACT using a system that DOES support large arrays "for the most part", I have established that my version of IDL 6.1 / 64-bit CAN handle integer arrays of, say, 2000 x 2000 x 900 items.

So, if it is not the variable itself - since that 'can be handled', it must be the variable set-up which happens when I read the volume data.

Now it seems that each time I try to read my 7.0 GB raw volume data file using READU or ASSOC, there must be problems transferring data into an array of >2 GB size, because any subscript later does not yield sensible results but garbage.

This is what I did:

```
imagearray = assoc (.., intarr[2000,2000,900], ..)
imagearray = temporary (imagearray [0])
...
```

Visualizing slices out of the imagearray variable does not yield sensible results. All of the slice image subscripts look simply off (as explained above).

I checked the integrity of the file ITSELF by reading just a couple of slices out of it into a small variable - such as just grabbing 400 MB -, and the file is a perfect scan. Just large.

Then I tried readu. This did not work at all but caused a crash each time:

```
...
readu, imagearray
...
```

So, I figured that large files can not be read in a single read process, so I tried this:

```
chunk=intarr (2000,2000,coupleofslices)

imagearray = intarr (2000,2000,900) ;--- A HUGE ARRAY!!!!!

...
point_lun, lun, ... long64(2000) * 2000 * slicecount
              ;-- position anywhere inside file using a L64 variable

readu, chunk   ;--- only read small portion into memory

imagearray [*,*,slicecount:slicecount+coupleofslices-1] = chunk

  or

imagearray [*,*, 0:coupleofslice-1] = chunk
...
```

This works very well. I can use imagearray as a huge array and still visualize whatever I want with it - filling this variable is the problem. All slice images subscripts of the type imagearray [*,*,slice#] work very well and without 'garbage' results as I experienced previously.

It appears to me, that IDL may not handle the READU or ASSOC function well with these large variables.

Is there a special way to handle these options using very large files? According to the handbook, there are no further options to be considered.

Do I need to run AIX off harddisk partitions that are formatted using the Large File Option - or is it enough to read the volume data from

Large File enabled partitions (which is what I am doing now)? Does IDL
try to create 'copies' of files in some temp-folder - then it would be
important for these temp-files to also be large file enabled.

Apparently, applications written for / compiled under AIX 4.2 and
earlier could only handle up to 2GB chunks - despite a filesystem being
able to store larger files. In IDL, !version.file_offset_bits returns '64'.

Since I can successfully read a smaller portion of the file, and then
store it into the large array that I have set up, I probably just have
to to rewrite this file-reading routine and then acquire the file-data
in chunks. I still wonder, what keeps 'assoc' and 'readu' to read these
large files.

Thanks for the valuable tips!

Wolf.

---

## Subject: Re: large array
## Posted by Dick Jackson on Wed, 13 Oct 2004 22:55:28 GMT

View Forum Message <> Reply to Message

---

"Wolf Schweitzer" <shwi@irm.unizh.ch> wrote in message
news:416d54e2$1@idnews.unizh.ch...

```
>
>  So, I figured that large files can not be read in a single read process,
>  so I tried this:
>
>  chunk=intarr (2000,2000,coupleofslices)
>
>  imagearray = intarr (2000,2000,900) ;--- A HUGE ARRAY!!!!!
>
>  ...
>  point_lun, lun, ... long64(2000) * 2000 * slicecount
>                ;-- position anywhere inside file using a L64 variable
>
>  readu, chunk   ;--- only read small portion into memory
>
>  imagearray [*,*,slicecount:slicecount+coupleofslices-1] = chunk
>
>   or
>
>  imagearray [*,*, 0:coupleofslice-1] = chunk
>  ...
```

It sounds like this approach is working for you, but I'll mention a tip
about this use of [*, *, ...]: I think you would be better off using [0, 0,

...] here, as discussed in these articles:

http://www.dfanning.com/misc_tips/submemory.html
http://www.dfanning.com/code_tips/asterisk.html

This could run you into trouble when working with big arrays!

Hope this helps.

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