
Subject: Re: strange behaviour of bytscl by large arrays
Posted by [lecacheux.alain](#) on Wed, 25 Apr 2012 19:24:15 GMT
[View Forum Message](#) <> [Reply to Message](#)

On 25 avr, 19:53, Karl <Karl.W.Schu...@gmail.com> wrote:
> On Tuesday, April 24, 2012 9:40:14 AM UTC-6, Chris Torrence wrote:
>> On Tuesday, April 24, 2012 8:50:46 AM UTC-6, alx wrote:
>>> On 23 avr, 22:22, Chris Torrence <gorth...@gmail.com> wrote:
>>>> On Monday, April 23, 2012 10:14:21 AM UTC-6, fawltyl...@gmail.com wrote:
>
>>>> > I think IDL's FINDGEN() implementation is wrong: it uses a float counter instead of an integer one. The following test shows the difference:
>
>>>> > pro test
>>>> > cpu, tpool_nthreads=1
>>>> > n=10l^8
>>>> > nn=n-1
>>>> > a1=findgen(n) ; real FINDGEN()
>>>> > a2=fltarr(n)
>>>> > count=0.0
>>>> > for j=0l, nn do a2[j]=count++ ; IDL's implementation
>>>> > a3=fltarr(n)
>>>> > count=0ll
>>>> > for j=0l, nn do a3[j]=count++ ; better implementation
>>>> > print, a1[nn], a2[nn], a3[nn], format='(3F15.3)'
>>>> > end
>
>>>> > (Multithreading must be disabled because the starting values for the threads are calculated as an integer. So the result of FINDGEN() depends on the number of your CPU cores, too :-)
>
>>>> > regards,
>>>> > Lajos
>
>>>> Well, wrong is perhaps too strong of a word. The real word is "fast". I just did a test where I changed the internal implementation of FINDGEN to use an integer counter. The "float" counter is 4 times faster than using an integer counter and converting it to floats.
>
>>>> However, perhaps we could look at the size of the input array, and switch to using the slower integer counter if it was absolutely necessary. I'll give it a thought.
>
>>>> Thanks for reporting this!
>
>>>> Cheers,
>>>> Chris
>>>> Exelis VIS
>
>>> It is risky to write a statement like "findgen(n)" while n is larger

>>> than the inverse of the floating point precision (given in IDL by
>>> long(1/machar().eps)). This is true in any programming language. It is
>>> mathematically incorrect to assume that such a "findgen" will behave
>>> as a "lindgen".
>>> IDL is not "wrong" here, but rather clever. Is'nt it ?
>>> alx.
>
>> Okay, alx has convinced me to not change anything. Try the following:
>
>> IDL> print, 16777216 + findgen(10), format='(f25.0)'
>> 16777216.
>> 16777216.
>> 16777218.
>> 16777220.
>> 16777220.
>> 16777220.
>> 16777222.
>> 16777224.
>> 16777224.
>> 16777224.
>
>> So even if you did the computation using long64's, as soon as you convert them back to
floats, you are going to get "jumps" in the findgen because of the loss of precision. I suppose you
could argue that this might be better than having the findgen get "stuck" on the number 16777216,
but I think the speed of findgen is more important.
>
>> Thanks.
>
>> -Chris
>> Exelis VIS
>
> Hi Chris,
>
> Interesting problem. FINDGEN is probably one of the oldest functions in IDL and it is hard to
imagine that it can still need some attention.
>
> I'd argue that skipping is better than getting stuck. Apps that use FINDGEN up in this range
are going to have to be aware of the precision issues. Those that do properly take this into
account would expect the skips and shouldn't be penalized by the "stuck" behavior.
>
> If you stay with the "stuck" implementation, then you'd have to document that the behavior is
undefined for $n > 1/\text{eps}$.
>
> Implementation-wise, couldn't you keep the performance by using the float for the first part of
the fill, and then switch to an integer for the rest? This would retain the performance for the
more common use cases.
>
> Karl

>
>

No, the performance penalty - as far as understand it - would not be due to some counting, in float, integer or whatever else, but to the needed integer to float conversions to get a floating vector. The solution is of the responsibility of the user which should know that "findgen" can be only used up to about $16 \cdot 10^6$, and "dindgen" must be used beyond. It would be certainly useful to have this reminder in the documentation.
alain.
