Subject: strange behaviour of bytscl by large arrays
Posted by Klemen on Mon, 23 Apr 2012 14:01:51 GMT
View Forum Message <> Reply to Message

Hi folks,

is there any explanation of why I don't get the same or at least similar results using the code below by:
a) using DINDGEN in line 3
b) using FINDGEN in line 3

```
pro test
  s = 10000
  a = sin(findgen(s, s)/100000.)
  b = bytscl(a)
  write_tiff, 'b.tif', b
end
```

The tif file I get using the DINDGEN function has waves all over the image. The option using FINDGEN produces strange results (a couple of waves and then wide bands of constant values). See the following link for the (resized) results.
https://picasaweb.google.com/112572300011512591455/Eumetsat# 5734593216558178098

I came across this problem as I tried to scale (using HIST_EQUAL and BYTSCL functions) 16-bit 5-band RapidEye data to 24-bit RGB image. Scaling the whole image produced results that were all black, smaller subsets seemed ok.

Does anybody have a suggestion how to handle this issue?

Cheers, Klemen

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Klemen on Tue, 24 Apr 2012 11:14:26 GMT
View Forum Message <> Reply to Message

Sorry, the performance of BYTSCL seems to be somehow connected to the OS/hardware... Anyway, it works fine on my office PC (WIN7 enterprise, 64-bit, 8 GB RAM).

Klemen

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Craig Markwardt on Tue, 24 Apr 2012 12:32:03 GMT
View Forum Message <> Reply to Message

On Monday, April 23, 2012 4:22:08 PM UTC-4, Chris Torrence 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.

It's pretty awesome when the product vendor tweaks the product for you just to test a hunch. :-)

Craig

Subject: Re: strange behaviour of bytscl by large arrays
Posted by lecacheux.alain on Tue, 24 Apr 2012 14:50:46 GMT
View Forum Message <> Reply to Message

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.

Subject: Re: strange behaviour of bytscl by large arrays
Posted by chris_torrence@NOSPAM on Tue, 24 Apr 2012 15:40:14 GMT
View Forum Message <> Reply to Message

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

Subject: Re: strange behaviour of bytscl by large arrays
Posted by manodeep@gmail.com on Tue, 24 Apr 2012 17:10:30 GMT
View Forum Message <> Reply to Message

On Apr 24, 10:40 am, Chris Torrence <gorth...@gmail.com> 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

It looks like IDL is actually behaving "correctly" - consider the
following C code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{
  float  fx = powf(2.0,24);
  double dx = pow(2.0,24);
  const double d_one = 1.0;
  const float f_one = 1.0;
  const int N=10;
  for(int i=0;i<N;i++){
    fprintf(stderr,"i=%5d (float) x = %25.10f  (double) x = %25.10lf
\n",i,fx+(i*f_one),dx+(i*d_one));
  }

  return EXIT_SUCCESS;
}
```

The output from the code is:

```
i=    0 (float) x =       16777216.0000000000  (double) x =
16777216.0000000000
i=    1 (float) x =       16777216.0000000000  (double) x =
16777217.0000000000
i=    2 (float) x =       16777218.0000000000  (double) x =
16777218.0000000000
i=    3 (float) x =       16777220.0000000000  (double) x =
16777219.0000000000
i=    4 (float) x =       16777220.0000000000  (double) x =
16777220.0000000000
i=    5 (float) x =       16777220.0000000000  (double) x =
16777221.0000000000
i=    6 (float) x =       16777222.0000000000  (double) x =
16777222.0000000000
```

i= 7 (float) x = 16777224.0000000000 (double) x =
16777223.0000000000
i= 8 (float) x = 16777224.0000000000 (double) x =
16777224.0000000000
i= 9 (float) x = 16777224.0000000000 (double) x =
16777225.0000000000

which is exactly what IDL prints out for findgen.

Cheers,
Manodeep

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Lajos Foldy on Tue, 24 Apr 2012 17:30:10 GMT
View Forum Message <> Reply to Message

On Monday, April 23, 2012 10:22:08 PM UTC+2, Chris Torrence wrote:

> 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

I could not reproduce this 4x slowdown. The integer counter + conversion method is only 30% slower in the following C test program (Intel Core i5-2500, 64 bit Linux):

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

double timediff(struct timeval* tv1, struct timeval* tv2)
{
  return tv2->tv_sec-tv1->tv_sec+(tv2->tv_usec-tv1->tv_usec)*1e-6;
}

int main()
{
  int n=1000000000, j;
```

```
  float* x=malloc(n*sizeof(float));
  float f;
  struct timeval tv1, tv2;

  gettimeofday(&tv1, NULL);
  for (j=0; j<n; j++) x[j]=j;
  gettimeofday(&tv2, NULL);
  printf("integer counter: %lf %f\n", timediff(&tv1, &tv2), x[n-1]);

  gettimeofday(&tv1, NULL);
  f=0.0;
  for (j=0; j<n; j++) x[j]=f++;
  gettimeofday(&tv2, NULL);
  printf("float counter: %lf %f\n", timediff(&tv1, &tv2), x[n-1]);
}
```

Also, IDL help says:

The FINDGEN function creates a floating-point array of the specified dimensions. Each element of the array is set to the value of its one-dimensional subscript.

So it should be equivalent to float(lindgen()), as one-dimensional subscript is an integer.

But I don't want to convince you, I can accept that it is a feature :-)

regards,
Lajos

Subject: Re: strange behaviour of bytscl by large arrays
Posted by David Fanning on Tue, 24 Apr 2012 17:54:55 GMT
View Forum Message <> Reply to Message

fawltylanguage@gmail.com writes:

> I could not reproduce this 4x slowdown. The integer counter +
> conversion method is only 30% slower in the following C test
> program (Intel Core i5-2500, 64 bit Linux)

I find it odd that slowness is a concern, but I suppose
there are many more people using FINDGEN than function
graphics. :-)

Cheers,

David

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

Subject: Re: strange behaviour of bytscl by large arrays
Posted by lecacheux.alain on Tue, 24 Apr 2012 20:03:48 GMT
View Forum Message <> Reply to Message

On 24 avr, 19:30, fawltylangu...@gmail.com wrote:
> On Monday, April 23, 2012 10:22:08 PM UTC+2, Chris Torrence wrote:
>> 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
>
> I could not reproduce this 4x slowdown. The integer counter + conversion method is only 30% slower in the following C test program (Intel Core i5-2500, 64 bit Linux):
>
> #include <time.h>
> #include <stdio.h>
> #include <stdlib.h>
>
> double timediff(struct timeval* tv1, struct timeval* tv2)
> {
> return tv2->tv_sec-tv1->tv_sec+(tv2->tv_usec-tv1->tv_usec)*1e-6;
>
> }
>
> int main()
> {
> int n=1000000000, j;
> float* x=malloc(n*sizeof(float));
> float f;
> struct timeval tv1, tv2;
>
> gettimeofday(&tv1, NULL);

>   for (j=0; j<n; j++) x[j]=j;
>   gettimeofday(&tv2, NULL);
>   printf("integer counter: %lf %f\n", timediff(&tv1, &tv2), x[n-1]);
>
>   gettimeofday(&tv1, NULL);
>   f=0.0;
>   for (j=0; j<n; j++) x[j]=f++;
>   gettimeofday(&tv2, NULL);
>   printf("float counter: %lf %f\n", timediff(&tv1, &tv2), x[n-1]);
>
> }
>
> Also, IDL help says:
>
> The FINDGEN function creates a floating-point array of the specified dimensions. Each element of the array is set to the value of its one-dimensional subscript.
>
> So it should be equivalent to float(lindgen()), as one-dimensional subscript is an integer.
>
> But I don't want to convince you, I can accept that it is a feature :-)
>
> regards,
> Lajos
>
>

By using the IDL profiler with :
  l = lindgen(100000)
  f = findgen(100000)
  fl = float(l)
I get:
  findgen -> 0.805 s.
  lindgen -> 0.894 s.
  float   -> 0.209 s.
showing that FPU addition is faster than CPU's one, and type
conversion is a relatively slow process.
alain.

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Karl[1] on Wed, 25 Apr 2012 17:53:26 GMT
View Forum Message <> Reply to Message

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/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

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/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 responsability of the user which should know that

"findgen" can be only used up to about 16 10^6, and "dindgen" must be
used beyond. It would be certainly useful to have this reminder in the
documentation.
alain.

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Kenneth P. Bowman on Wed, 25 Apr 2012 21:11:43 GMT
View Forum Message <> Reply to Message

In article < 29512254.739.1335282014771.JavaMail.geo-discussion-forums@yn y11 >,
 Chris Torrence <gorthmog@gmail.com> wrote:

> 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

Since this turns out to be a floating-point precision issue, does DINDGEN
use a long64 counter?

And more importantly, could this possibly be documented in the manuals
for the sake of future generations?

I know it is not the IDL way to document implementation details, but sometimes
they are important when trying to understand how things work or why they
don't.

Ken Bowman

Subject: Re: strange behaviour of bytscl by large arrays
Posted by David Fanning on Wed, 25 Apr 2012 21:37:39 GMT
View Forum Message <> Reply to Message

Kenneth P. Bowman writes:

> And more importantly, could this possibly be documented in the manuals
> for the sake of future generations?

I'm not sure future generations will be *able* to read.
I know the current generation can't spell, given the
myriad examples in my local newspaper. But, just in case,
I'll write an article about this in the next day or two,
when all the facts have trickled in.

Cheers,

David

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

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Lajos Foldy on Thu, 26 Apr 2012 14:59:13 GMT
View Forum Message <> Reply to Message

On Wednesday, April 25, 2012 11:11:43 PM UTC+2, Kenneth P. Bowman wrote:

> Since this turns out to be a floating-point precision issue, does DINDGEN
> use a long64 counter?

I think this is not a precision issue. Float can represent numbers up to 10^38 with a relative error of 10^-7. For huge values FINDGEN() creates indices with much bigger errors and this is the consequence of the current implementation, not the nature of floating point representation.

Double has a relative error of 10^-16 so DINDGEN does not need an integer counter.

regards,
Lajos

Subject: Re: strange behaviour of bytscl by large arrays
Posted by lecacheux.alain on Thu, 26 Apr 2012 16:00:23 GMT
View Forum Message <> Reply to Message

On 26 avr, 16:59, fawltylangu...@gmail.com wrote:
> I think this is not a precision issue. Float can represent numbers up to 10^38 with a relative error of 10^-7. For huge values FINDGEN() creates indices with much bigger errors and this is the consequence of the current implementation, not the nature of floating point representation.
>

This is a precision issue, not *relative* but *absolute*. For n
expressed as a floating point number and larger than its precision
inverse, n+1 is no longer discernible from n. As you can see:

IDL> print,float(10L^8+indgen(10)),FORMAT='(10Z8)'
 5F5E100 5F5E100 5F5E100 5F5E100 5F5E100 5F5E108 5F5E108 5F5E108
5F5E108 5F5E108
IDL> print,double(10L^8+indgen(10)),FORMAT='(10Z8)'
 5F5E100 5F5E101 5F5E102 5F5E103 5F5E104 5F5E105 5F5E106 5F5E107
5F5E108 5F5E109

Here 10L^8 is larger than 2/(machar()).eps = 16777216, and smaller
than 2/(machar(/DOUBLE)).eps) (about 9e15).
Creating a floating point ramp beyond 16777216 is formally possible,
but is no sense since distinct values will be more and more spaced.

alx.

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Karl[1] on Thu, 26 Apr 2012 16:29:00 GMT
View Forum Message <> Reply to Message

On Wednesday, April 25, 2012 1:24:15 PM UTC-6, alx wrote:
> 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/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 responsability of the user which should know that
> "findgen" can be only used up to about 16 10^6, and "dindgen" must be
> used beyond. It would be certainly useful to have this reminder in the
> documentation.
> alain.

Right, if the user does not want any skips past the 1/eps mark, they should use dindgen.

And I agree that the usefulness of the findgen output with the expected skips past the 1/eps mark is dubious. One example that comes to mind is a dense graph or chart in that range where the skips would be hard to notice. The argument is that having the skips is better than having the values be "stuck" at 1/eps, which would lead to a constant value in that part of the graph in my

example. I admit this is all pretty weak; I just think that getting the result of the function closer to the right answer within machine limitations is slightly better.

Yes, the performance problem is in the conversion. I was suggesting:

```
for float f = 0.0 to min(16777215.0, n)
  *pFltVector++ = f
  f += 1.0
endfor

if n >= 16777216
  for long i = 16777216 to n
    *pFltVector++ = (float)i
    i += 1
  endfor
endif
```

which preserves the performance in the most often used cases.

Karl

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Lajos Foldy on Thu, 26 Apr 2012 16:51:20 GMT
View Forum Message <> Reply to Message

On Thursday, April 26, 2012 6:00:23 PM UTC+2, alx wrote:
> On 26 avr, 16:59, fawltylangu...@gmail.com wrote:
>> I think this is not a precision issue. Float can represent numbers up to 10^38 with a relative error of 10^-7. For huge values FINDGEN() creates indices with much bigger errors and this is the consequence of the current implementation, not the nature of floating point representation.
>>
>
> This is a precision issue, not *relative* but *absolute*. For n
> expressed as a floating point number and larger than its precision
> inverse, n+1 is no longer discernible from n. As you can see:
>
> IDL> print,float(10L^8+indgen(10)),FORMAT='(10Z8)'
> 5F5E100 5F5E100 5F5E100 5F5E100 5F5E100 5F5E108 5F5E108 5F5E108
> 5F5E108 5F5E108
> IDL> print,double(10L^8+indgen(10)),FORMAT='(10Z8)'
> 5F5E100 5F5E101 5F5E102 5F5E103 5F5E104 5F5E105 5F5E106 5F5E107
> 5F5E108 5F5E109
>
> Here 10L^8 is larger than 2/(machar()).eps = 16777216, and smaller
> than 2/(machar(/DOUBLE)).eps) (about 9e15).
> Creating a floating point ramp beyond 16777216 is formally possible,
> but is no sense since distinct values will be more and more spaced.

>
> alx.

I know all that. But when I write float(10l^8) I expect a floating point number with a relative error of 10^-7, a number in the range [10l^8-10, 10l^8+10]. FINDGEN()'s value is not in this range, it is far from it.

But as others wrote, the real solution is to mark FINDGEN in the docs as undefined/unsuitable for values greater than 16777216. Probably FINDGEN should print a warning, too.

regards,
Lajos

Subject: Re: strange behaviour of bytscl by large arrays
Posted by chris_torrence@NOSPAM on Fri, 04 May 2012 15:46:39 GMT
View Forum Message <> Reply to Message

Hi all,

I've added a fix for this in IDL Next (not 8.2). If you ask for values larger than 16777215, then it switches to using a 64-bit integer for the counting, and then casts those values to floats. You will still get duplicate numbers and discontinuities (impossible to fix that), but at least it won't get stuck on a single number. I did the same thing for double, in case you have any desire to allocate 9007199254740992 elements (that's 8 petabytes).

I also added a START keyword to all of the *INDGEN routines and MAKE_ARRAY, that allows you to specify a starting index. I needed this for testing purposes so I didn't have to keep creating huge arrays, and it seemed like a generally useful feature. It's much faster to specify START than to add an offset to the indgen after you've created it.

Cheers,
Chris
ExelisVIS

Subject: Re: strange behaviour of bytscl by large arrays
Posted by ben.bighair on Fri, 04 May 2012 16:04:14 GMT
View Forum Message <> Reply to Message

Hi,

On Friday, May 4, 2012 11:46:39 AM UTC-4, Chris Torrence wrote:

> I also added a START keyword to all of the *INDGEN routines and MAKE_ARRAY, that allows you to specify a starting index. I needed this for testing purposes so I didn't have to keep creating huge arrays, and it seemed like a generally useful feature. It's much faster to specify START than

to add an offset to the indgen after you've created it.

Yeah! That's great. I wonder if you would consider adding something like R's "seq" function. In IDL I imagine it my be something like this...

result = seq(from = x, to = y, by = z, length = l, type = "t")

I find it incredibly useful - I'm sure others would find it indispensable, too. You're so close with the new START keyword (from) and the desired output length (length).

Cheers,
Ben

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Craig Markwardt on Sat, 05 May 2012 05:27:06 GMT
View Forum Message <> Reply to Message

On Friday, May 4, 2012 11:46:39 AM UTC-4, Chris Torrence wrote:
> Hi all,
>
> I've added a fix for this in IDL Next (not 8.2). If you ask for values larger than 16777215, then it switches to using a 64-bit integer for the counting, and then casts those values to floats. You will still get duplicate numbers and discontinuities (impossible to fix that), but at least it won't get stuck on a single number. I did the same thing for double, in case you have any desire to allocate 9007199254740992 elements (that's 8 petabytes).
>
> I also added a START keyword to all of the *INDGEN routines and MAKE_ARRAY, that allows you to specify a starting index. I needed this for testing purposes so I didn't have to keep creating huge arrays, and it seemed like a generally useful feature. It's much faster to specify START than to add an offset to the indgen after you've created it.

Cool, that's pretty useful.
Craig

Subject: Re: strange behaviour of bytscl by large arrays
Posted by Lajos Foldy on Sat, 05 May 2012 11:39:52 GMT
View Forum Message <> Reply to Message

On Friday, May 4, 2012 5:46:39 PM UTC+2, Chris Torrence wrote:
> Hi all,
>
> I've added a fix for this in IDL Next (not 8.2). If you ask for values larger than 16777215, then it switches to using a 64-bit integer for the counting, and then casts those values to floats. You will still get duplicate numbers and discontinuities (impossible to fix that), but at least it won't get stuck on a single number. I did the same thing for double, in case you have any desire to allocate

9007199254740992 elements (that's 8 petabytes).
>
> I also added a START keyword to all of the *INDGEN routines and MAKE_ARRAY, that allows you to specify a starting index. I needed this for testing purposes so I didn't have to keep creating huge arrays, and it seemed like a generally useful feature. It's much faster to specify START than to add an offset to the indgen after you've created it.
>
> Cheers,
> Chris
> ExelisVIS

Thanks.

May I suggest SEED or ROOT instead of START? START will break 'make_array(5, /st)' :-)

regards,
Lajos
