Subject: counting bits
Posted by Joe Foose on Mon, 17 Feb 2003 19:50:26 GMT
View Forum Message <> Reply to Message

In IDL, does anybody have any ideas of how to efficiently count all the bits
that are set in an array of unsigned long integers? I'm not concerened with
which bits are set, but just how many. thanks, joe.

Subject: Re: counting bits
Posted by Dick Jackson on Thu, 20 Feb 2003 22:53:24 GMT
View Forum Message <> Reply to Message

"JD Smith" <jdsmith@as.arizona.edu> wrote in message
news:pan.2003.02.20.15.43.26.137656.2731@as.arizona.edu...
> On Wed, 19 Feb 2003 08:47:31 -0700, Dick Jackson wrote:
>
>> IDL> CountingBits
>> IShft-AND-lookup method: 2.063 seconds. tot =   46137328
>> Byte-lookup method: 0.691 seconds. tot =   46137292
>>
>> Uh-oh... I set the Total calls to have /Double and then we both get
the
>> same (I hope correct) answer:
>>
>> IDL> CountingBits
>> IShft-AND-lookup method: 2.063 seconds. tot =   46137344
>> Byte-lookup method: 0.671 seconds. tot =   46137344
>
> That's very strange. Here's what I get for my four independent
> methods without any /DOUBLE:
>
>   3.4187140
>   46137344
>   6.9928349
>   46137344
>   1.2564960
>   46137344
>   1.2767580
>   46137344

In case anyone's still following this exercise, we found that if 'bits'
is a byte array (not integer) my floating-point problem goes away, and
makes it faster! Adding this after 'bits' is defined will do it:

bits = Byte(bits)

... and gives another nice speedup:

```
IDL> CountingBits
IShft-AND-lookup method:     1.883 seconds.
tot =     46137344
Byte-lookup method:     0.541 seconds.
tot =     46137344
```

Cheers,
--
-Dick

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

Subject: Re: counting bits
Posted by condor on Tue, 25 Feb 2003 23:17:05 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> wrote in message news:<pan.2003.02.20.15.43.26.137656.2731@as.arizona.edu>...

> One thing I did notice when creating "random" arrays:
>
> IDL> print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) mod 2 eq 1),$
>   '% odd'
>
> Try this a few times. That lowest bit just does not get set. Some
> floating-point representation expert must have an explanation.

Dunno that this needs an expert: give a /double to the call to rendomu and it works as expected -- otherwise randomu will return a float array, floats have 4 byte representation and thus the graininess at which floats can be represented cannot possibly be better than 1 bit in 32 (and in reality it's a good bit less).

In other words: you're multiplying floats 0<f<1 with 2.^31 which means for them to be distinguishable in the last bit the original floats would have had to have a spacing of 1/2^30 :

```
m = machar()
print,m.eps
1.19209e-07
print,1/(2^31.)
4.65661e-10
```

So you have numbers that are at most about 10^7 apart from each other
(the machine precision) and you multiply them with almost 10^10 and
thus will not get numbers that are 'one' apart from each other.

You want weird? Check for all the bits OTHER than the last one:

```
print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
  2ul eq 2ul),'% set'

print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
  4ul eq 4ul),'% set'

print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
  8ul eq 8ul),'% set'
```

etc ...

Subject: Re: counting bits
Posted by JD Smith on Wed, 26 Feb 2003 18:29:52 GMT
View Forum Message <> Reply to Message

On Tue, 25 Feb 2003 16:17:05 -0700, Big Bird wrote:

> JD Smith <jdsmith@as.arizona.edu> wrote in message
> news:<pan.2003.02.20.15.43.26.137656.2731@as.arizona.edu>...
>
>> One thing I did notice when creating "random" arrays:
>>
>> IDL> print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) mod 2
>> eq 1),$
>>   '% odd'
>>
>> Try this a few times. That lowest bit just does not get set. Some
>> floating-point representation expert must have an explanation.
>
>
> Dunno that this needs an expert: give a /double to the call to rendomu
> and it works as expected -- otherwise randomu will return a float array,
> floats have 4 byte representation and thus the graininess at which
> floats can be represented cannot possibly be better than 1 bit in 32
> (and in reality it's a good bit less).
>
> In other words: you're multiplying floats 0<f<1 with 2.^31 which means
> for them to be distinguishable in the last bit the original floats would
> have had to have a spacing of 1/2^30 :
>
> m = machar()

>   print,m.eps
>   1.19209e-07
>   print,1/(2^31.)
>   4.65661e-10
>
> So you have numbers that are at most about 10^7 apart from each other
> (the machine precision) and you multiply them with almost 10^10 and thus
> will not get numbers that are 'one' apart from each other.
>
> You want weird? Check for all the bits OTHER than the last one:
>
> print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
>   2ul eq 2ul),'% set'
>
> print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
>   4ul eq 4ul),'% set'
>
> print,FORMAT='(F5.2,A)',total(ulong(randomu(sd,100)*2.^31) and $
>   8ul eq 8ul),'% set'
>
> etc ...

I think you meant to include the "and" inside the total() call. And
yes, it is bizarre:

```
IDL> r=ulong(randomu(sd,100)*2.^31) & for i=0,31 do print,FORMAT='(I2,": ",I2,A)',i,total((r AND
ulong(2.D^i)) ne 0UL),'% set'
 0:  0% set
 1:  0% set
 2:  1% set
 3:  1% set
 4:  9% set
 5: 17% set
 6: 27% set
 7: 59% set
 8: 44% set
 9: 50% set
10: 46% set
11: 57% set
12: 50% set
13: 55% set
14: 51% set
15: 48% set
16: 56% set
17: 51% set
18: 52% set
19: 43% set
20: 46% set
```

21: 44% set
22: 35% set
23: 52% set
24: 47% set
25: 51% set
26: 44% set
27: 51% set
28: 46% set
29: 53% set
30: 45% set
31: 0% set

I guess I was looking not for an explanation of why the bits can't be evenly populated (which is obvious), but why *in particular* the lowest bits seem consistently poorly populated. I performed a very long run also:

```
IDL> r=ulong(randomu(sd,10000000)*2.^31) & for i=0,31 do print,FORMAT='(I2,":
",F5.2,A)',i,total((r AND ulong(2.D^i)) ne 0UL)/100000.,'% set'
 0:  0.39% set
 1:  0.78% set
 2:  1.56% set
 3:  3.13% set
 4:  6.25% set
 5: 12.50% set
 6: 25.00% set
 7: 50.02% set
 8: 50.01% set
 9: 49.98% set
10: 50.02% set
11: 50.02% set
12: 50.00% set
13: 50.01% set
14: 50.01% set
15: 50.00% set
16: 50.00% set
17: 50.04% set
18: 50.00% set
19: 50.00% set
20: 49.97% set
21: 50.02% set
22: 50.03% set
23: 50.02% set
24: 50.01% set
25: 50.03% set
26: 50.01% set
27: 50.00% set
28: 49.99% set
```

29: 49.98% set
30: 50.00% set
31: 0.00% set

So it's not a low-number statistics problem. You'll notice a *very* curious pattern emerges.

JD

Subject: Re: counting bits
Posted by thompson on Wed, 26 Feb 2003 20:15:28 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> writes:

> IDL> r=ulong(randomu(sd,100)*2.^31) & for i=0,31 do print,FORMAT='(I2,": ",I2,A)',i,total((r AND ulong(2.D^i)) ne 0UL),'% set'
> 0: 0% set
> 1: 0% set
> 2: 1% set
> 3: 1% set
> 4: 9% set
> 5: 17% set
> 6: 27% set
> 7: 59% set
> 8: 44% set
> 9: 50% set
> 10: 46% set
> 11: 57% set
> 12: 50% set
> 13: 55% set
> 14: 51% set
> 15: 48% set
> 16: 56% set
> 17: 51% set
> 18: 52% set
> 19: 43% set
> 20: 46% set
> 21: 44% set
> 22: 35% set
> 23: 52% set
> 24: 47% set
> 25: 51% set
> 26: 44% set
> 27: 51% set
> 28: 46% set
> 29: 53% set

> 30: 45% set
> 31: 0% set

That's pretty simple to explain. Floating point numbers are stored with a mantissa and an exponent, both stored within the same 4 byte word. A few of the bits are devoted to the exponent, and the rest are devoted to the mantissa. When you call

 randomu(sd,100)

you generate a bunch of numbers which mostly have the same exponent bits, because all the numbers are of the same order of magnitude, while the mantissa bits are generally 50% on or 50% off. When you then multiply this by 2.^31 and convert it into a long integer, you're primarily sampling the mantissa bits.
In fact, the only reason why the last few bits are sometimes set at all is that some of the random numbers are close to zero, and thus end up with different exponents.

You can see this by looking directly at the bits of the original floating point numbers.

```
IDL> r=ulong(randomn(sd,100),0,100) & for i=0,31 do print,FORMAT='(I2,": ",I2,A)',i,total((r AND ulong(2.D^i)) ne 0UL),'% set'
 0: 58% set
 1: 49% set
 2: 48% set
 3: 48% set
 4: 49% set
 5: 49% set
 6: 47% set
 7: 42% set
 8: 53% set
 9: 51% set
10: 55% set
11: 46% set
12: 44% set
13: 48% set
14: 50% set
15: 59% set
16: 40% set
17: 52% set
18: 53% set
19: 57% set
20: 48% set
21: 41% set
22: 43% set
23: 43% set
24: 75% set
```

25: 86% set
26: 96% set
27: 96% set
28: 96% set
29: 96% set
30: 4% set
31: 53% set

See how the mantissa is stored in the lower bits, and there's very little variation in the uppermost bits where the mantissa is stored?

Bill Thompson