Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Wed, 24 Jun 2009 11:19:08 GMT
View Forum Message <> Reply to Message

On Jun 24, 3:54 am, chris <rog...@googlemail.com> wrote:
> Hi,
> i'm looking for a faster approach to do the following:
>
> data=rebin(dist(100),100,100,50,/sample)
> mask=total(data,3) gt 0.
>
> Depending on the size of data and on the hardware of the client the
> code above runs too slow for me. I'd like to find all pixels in a
> hyperspectral application, which bands are not zero. The result should
> be a mask with the same 2D-dimensions as original data for further
> processing.

My first thought was to use ARRAY_EQUAL rather than TOTAL since
ARRAY_EQUAL(im,0) will stop computations as soon as a non-zero element
is found. But as far as I know, ARRAY_EQUAL can only return a
scalar argument, which means that one would have to loop over each
element of the output mask. Ugh.

But if we are forced to use TOTAL, there is no need for a double
precision computation. I found a significant speedup by replacing
the second line with

mask = total(data,3,/integer) GT 0

I also found a smaller speedup with the following code, which is based
on the theory that mathematical operations are always quickest on byte
data.

mask = total(data GT 0,/preserve_type,3) GT 0

-Wayne

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Wed, 24 Jun 2009 11:33:51 GMT
View Forum Message <> Reply to Message

> I also found a smaller speedup with the following code, which is based
> on the theory that mathematical operations are always quickest on byte
> data.
>
> mask = total(data GT 0,/preserve_type,3) GT 0
>

I should add that you shouldn't use the BYTE method if you have more than 255 images in your cube. In any case, I found it wasn't as fast as using TOTAL(/INTEGER). --Wayne

---

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Wed, 24 Jun 2009 11:44:20 GMT
View Forum Message <> Reply to Message

Dear Wayne,
thank you for your quick response and your suggestions. Unfortunately, none of your approaches is faster, only a bit slower. I tried some other methods based on histogram or value_locate or osrt, but each appraoch was much slower then total(dat,3). Any other ideas?

Best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Wed, 24 Jun 2009 11:55:51 GMT
View Forum Message <> Reply to Message

On Jun 24, 7:44 am, chris <rog...@googlemail.com> wrote:
> Dear Wayne,
> thank you for your quick response and your suggestions. Unfortunately,
> none of your approaches is faster, only a bit slower. I tried some
> other methods based on histogram or value_locate or osrt, but each
> appraoch was much slower then total(dat,3). Any other ideas?
>

It might help to know what you are really doing. The example you gave runs in milliseconds. Are you using much larger arrays, or repeating the operation thousands of times? --Wayne

---

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Wed, 24 Jun 2009 12:12:04 GMT
View Forum Message <> Reply to Message

Dear Wayne,
thank you! My code runs for a [1536,231,126] sample also very fast - in 0.13 sec, but it's not for me. At a colleague's PC it lasts minutes without caching the matrix to swap and he asked me for a possible speedup. Unfortunately, I don't know which could be faster than total... I cannot understand, why the fastest method to determine

whether a vector is entirely set to zero is based on total - a sum method... I tried to increase speed by the use of the gpulib, but also got the same effect. Any other ideas?

Best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by Chris[6] on Wed, 24 Jun 2009 13:35:55 GMT
View Forum Message <> Reply to Message

On Jun 24, 2:12 am, chris <rog...@googlemail.com> wrote:
> Dear Wayne,
> thank you! My code runs for a [1536,231,126] sample also very fast -
> in 0.13 sec, but it's not for me. At a colleague's PC it lasts minutes
> without caching the matrix to swap and he asked me for a possible
> speedup. Unfortunately, I don't know which could be faster than
> total... I cannot understand, why the fastest method to determine
> whether a vector is entirely set to zero is based on total - a sum
> method... I tried to increase speed by the use of the gpulib, but
> also got the same effect. Any other ideas?
>
> Best regards
>
> CR

did you try something like max(data, dimen = 3)?

chris

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Wed, 24 Jun 2009 14:19:03 GMT
View Forum Message <> Reply to Message

On Jun 24, 9:35 am, Chris <beaum...@ifa.hawaii.edu> wrote:
>
> did you try something like max(data, dimen = 3)?
>
> chris

This seems like the fastest method so long as the data is always positive (as in the original example) or at least so long as one can rule out the case where all pixels are either negative or zero. --
Wayne

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by Jean H. on Wed, 24 Jun 2009 14:33:41 GMT
View Forum Message <> Reply to Message

wlandsman wrote:
> On Jun 24, 9:35 am, Chris <beaum...@ifa.hawaii.edu> wrote:
>> did you try something like max(data, dimen = 3)?
>>
>> chris
>
> This seems like the fastest method so long as the data is always
> positive (as in the original example) or at least so long as one can
> rule out the case where all pixels are either negative or zero. --
> Wayne

look at the min also...

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Wed, 24 Jun 2009 15:38:00 GMT
View Forum Message <> Reply to Message

Min and Max approach is two times slower in my case, so this doesn't
seem to be a solution. Any other ideas?

Thanks and best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Wed, 24 Jun 2009 16:34:34 GMT
View Forum Message <> Reply to Message

On Jun 24, 11:38 am, chris <rog...@googlemail.com> wrote:
> Min and Max approach is two times slower in my case, so this doesn't
> seem to be a solution. Any other ideas?
>

Be sure to calculate min and max at the same time, e.g.
mask1 = max(data,dimen=3,min=mask)
mask = (mask or mask1) NE 0

But it seems that the best performance is hardware dependent.
Below are the repeatable times in seconds I get for the different
methods for a 1536 x 231 x 126 array on different systems.

{ x86_64 linux unix linux 7.0

TOTAL 0.26
TOTAL(/INTEGER) 0.28
TOTAL(byte) 0.17
MINMAX 0.25

(x_86_64 darwin unix Mac OS X 7.06)
TOTAL 0.24
TOTAL(/INTEGER) 0.16
TOTAL(byte) 0.22
MINMAX 0.24

Since you are getting the best times for the first (TOTAL()) method, I suspect your hardware is optimized for floating point calculations.
If you were to code it in C (i.e. not worry about loops) the quickest method should be some variant of ARRAY_EQUAL
where you stop the comparisons once you find a non-zero element in a band. But until ARRAY_EQUAL gets a dimension keyword like MIN and MAX I don't think any other IDL method is going to be much faster.
--Wayne

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Wed, 24 Jun 2009 20:51:28 GMT
View Forum Message <> Reply to Message

Dear Wayne,
thank you again for your response. I got for Windows Vista x64 with 2x2.5 GHz Intel DualCore and 8 GB Ram within:
IDL 6.4 x64
- maxmin: 0.25 s
- total(/integer): 0.19 s
- total : 0.13 s

So, maybe it doesnt matter, but the total method seeme to beat each other approach...:(

Any further ideas?

Best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by Jeremy Bailin on Wed, 24 Jun 2009 20:53:15 GMT
View Forum Message <> Reply to Message

On Jun 24, 12:34 pm, wlandsman <wlands...@gmail.com> wrote:
> On Jun 24, 11:38 am, chris <rog...@googlemail.com> wrote:
>
>> Min and Max approach is two times slower in my case, so this doesn't
>> seem to be a solution. Any other ideas?
>
> Be sure to calculate min and max at the same time, e.g.
> mask1 = max(data,dimen=3,min=mask)
> mask = (mask or mask1) NE 0
>
> But it seems that the best performance is hardware dependent.
> Below are the repeatable times in seconds I get for the different
> methods for a 1536 x 231 x 126 array on different systems.
>
> { x86_64 linux unix linux 7.0
> TOTAL 0.26
> TOTAL(/INTEGER) 0.28
> TOTAL(byte) 0.17
> MINMAX 0.25
>
> (x_86_64 darwin unix Mac OS X 7.06)
> TOTAL 0.24
> TOTAL(/INTEGER) 0.16
> TOTAL(byte) 0.22
> MINMAX 0.24
>
> Since you are getting the best times for the first (TOTAL()) method, I
> suspect your hardware is optimized for floating point calculations.
> If you were to code it in C (i.e. not worry about loops) the quickest
> method should be some variant of ARRAY_EQUAL
> where you stop the comparisons once you find a non-zero element in a
> band. But until ARRAY_EQUAL gets a dimension keyword like MIN and
> MAX I don't think any other IDL method is going to be much faster.
> --Wayne

How about using product? It should be well-optimized for the cases of multiplying-by-one and multiplying-by-zero:

mask = ~product(data gt 0, 3, /preserve_type)

-Jeremy.

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by Jeremy Bailin on Wed, 24 Jun 2009 21:04:37 GMT
View Forum Message <> Reply to Message

On Jun 24, 4:53 pm, Jeremy Bailin <astroco...@gmail.com> wrote:

> On Jun 24, 12:34 pm, wlandsman <wlands...@gmail.com> wrote:
>
>
>
>> On Jun 24, 11:38 am, chris <rog...@googlemail.com> wrote:
>
>>> Min and Max approach is two times slower in my case, so this doesn't
>>> seem to be a solution. Any other ideas?
>
>> Be sure to calculate min and max at the same time, e.g.
>> mask1 = max(data,dimen=3,min=mask)
>> mask = (mask or mask1) NE 0
>
>> But it seems that the best performance is hardware dependent.
>> Below are the repeatable times in seconds I get for the different
>> methods for a 1536 x 231 x 126 array on different systems.
>
>> { x86_64 linux unix linux 7.0
>> TOTAL 0.26
>> TOTAL(/INTEGER) 0.28
>> TOTAL(byte) 0.17
>> MINMAX 0.25
>
>> (x_86_64 darwin unix Mac OS X 7.06)
>> TOTAL 0.24
>> TOTAL(/INTEGER) 0.16
>> TOTAL(byte) 0.22
>> MINMAX 0.24
>
>> Since you are getting the best times for the first (TOTAL()) method, I
>> suspect your hardware is optimized for floating point calculations.
>> If you were to code it in C (i.e. not worry about loops) the quickest
>> method should be some variant of ARRAY_EQUAL
>> where you stop the comparisons once you find a non-zero element in a
>> band. But until ARRAY_EQUAL gets a dimension keyword like MIN and
>> MAX I don't think any other IDL method is going to be much faster.
>> --Wayne
>
> How about using product? It should be well-optimized for the cases of
> multiplying-by-one and multiplying-by-zero:
>
> mask = ~product(data gt 0, 3, /preserve_type)
>
> -Jeremy.

Oops, that should of course read:

mask = ~product(data eq 0, 3, /preserve_type)

-Jeremy.

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Wed, 24 Jun 2009 22:30:38 GMT
View Forum Message <> Reply to Message

Dear Jeremy,
I tested it, but your approach is in my specific case also slower.
Maybe, there is no solution... :(

Best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by Chris[6] on Thu, 25 Jun 2009 00:46:15 GMT
View Forum Message <> Reply to Message

On Jun 24, 12:30 pm, chris <rog...@googlemail.com> wrote:
> Dear Jeremy,
> I tested it, but your approach is in my specific case also slower.
> Maybe, there is no solution... :(
>
> Best regards
>
> CR

maybe a faster computer?!

chris

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Thu, 25 Jun 2009 06:58:54 GMT
View Forum Message <> Reply to Message

> maybe a faster computer?!
>
> chris

Dear Chris
it's me, Chris :) Back to topic: That may be, but nevertheless there
are routines missing, which can compare a 3D matrix with a 1D vector
simply and fast. Any other ideas?

Best regards

CR

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by wlandsman on Thu, 25 Jun 2009 12:10:51 GMT
View Forum Message <> Reply to Message

>
>> chris
>
> Dear Chris
> it's me, Chris :) Back to topic: That may be, but nevertheless there
> are routines missing, which can compare a 3D matrix with a 1D vector
> simply and fast. Any other ideas?
>

Check your assumptions. The ideas suggested are all completely vectorized IDL code and have the same speed to an order of magnitude. They have some unnecessary computations because every pixel in each band is checked even if one already knows the column contains non-zero elements. This should slow the routines down by about a factor of two from an ideal method.

But if the existing code takes less than a second on all the machines tested, and takes several minutes on your user's machine, then I would take a very close look as to whether it is really this section of code that is causing the problem. --Wayne

---

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by JDS on Thu, 25 Jun 2009 15:22:11 GMT
View Forum Message <> Reply to Message

I agree with all the assessments thus far. These methods are within a factor of 2 or 3 of the best IDL-native vectorized result (and very likely a factor of 10-30 off the compiled C result). As for this calculation taking "minutes", this sounds suspiciously like running out of memory and hitting the disk. That would be unusual given the ~180MB data size here, but perhaps other processes or parts of the routine are taxing memory, or it's a very old machine with <<1GB of RAM. I'd look to this issue first. Here, no matter the algorithm, it runs in a fraction of a second.

That concern aside, there is another approach -- one you will rarely

find me recommending. If you happen to know that null bands are going to be found very rarely, a thinned WHERE loop can actually outperform the native vector operation:

```
s=size(data,/DIMENSIONS)
chnk=s[0]*s[1]
zeroes=lindgen(chnk)
for i=0L,s[2]-1 do begin
  z=where(data[zeroes+i*chnk] eq 0.,cnt)
  if cnt eq 0 then break
  zeroes=zeroes[z]
endfor
```

Here I'm operating on an array of the size mentioned above:

```
data=randomu(sd,1536,231,126)
data[where(data lt .9)]=0.
```

By tuning the ".9" factor, you can arrange for as many null bands as you want.

When only a few bands are null in a given data cube, this is roughly 2.5x faster for me. When it's very rare to have *any* null bands, this method can be *much* faster: 20-30x. The reason is clear: it takes only a few iterations to prove the absence of nulls in that case, and index thinning proceeds rapidly

But here's the catch (isn't there always a catch?). If null bands are present at a frequency of even 1 in 200 or more, this loop method becomes slower than TOTAL. In the worst case (all bands null), it's about 6x slower (all on my dual-core machine, YMMV). So, as is usual with these things, the answer to "which method is faster for my data" is: "it depends on your data."

You might also notice this is a reasonable study case for the recently debated issue of "when are for loops *not* evil". Since in each iteration, a large number of elements are being compared, the looping overhead is not severe. You'll also notice this illustrates the method of "compute your own index vector and re-use." Had we used IDL's native array range operator [x:y] or [*,*,z], this most certainly would have spoiled the time savings.

One other point worth mentioning: if your data cubes are "skinny and tall", with the third dimension long compared to the others, this loop method will perform even better. For instance, using a similarly sized data cube, but much taller:

```
data=randomu(sd,153,231,1260)
```

data[where(data lt .998)]=0.

I find speed parity between TOTAL and the thinned WHERE loop occurs when 8% of the bands are null.

JD

Subject: Re: Faster approach for total(data,dimension) possible?
Posted by rogass on Fri, 26 Jun 2009 14:44:55 GMT
View Forum Message <> Reply to Message

Dear all,
thank you for your suggestions. I will pick up JD's approach - like several times :). For a very large data set (eg. 5000x5000x2000) I will generally broke it up into chunks. Then within the chunks, I will generate a mask for the first channel where the pixels have zero values assuming they are zero-channel representatives. This mask will be used with the total approach and the ~mask with JD's For-Loop-approach. After this I will merge both results - result1+result2 eq 2. This may be faster for a very large data set. Additionally, the presence of a working gpulib will be tested and used to increase the computation speed.

All the best

CR
