Subject: Re: Maximum value array resampling
Posted by JD Smith on Mon, 08 Aug 2005 17:48:59 GMT
View Forum Message <> Reply to Message

On Fri, 05 Aug 2005 19:57:55 -0400, Richard G. French wrote:

>
>>
>> For arbitrary images with both dimensions even:
>>
>> d=size(x,/DIMENSIONS) & nx=d[0]/2 & ny=d[1]/2
>> y=transpose(max(reform(transpose(reform(x,2,nx,2*ny),[0,2,1]), $
>> 4,ny,nx),DIMENSION=1))
>>
>> How does it work? It juggles dimensions so that the indices of all the
>> 2x2 sub-arrays are next to each other in memory, and then uses
>> max(/DIMENSION) to collapse over them. The inner call to REFORM puts
>> them adjacent to each other, but in the wrong dimension, then TRANSPOSE
>> makes them adjacent in the fast-changing dimension. The rest is
>> straightforward.
>>
>> There may be a quicker way with only one call to TRANSPOSE, but I
>> couldn't find it (anyone?). Also, if you don't care to keep X, throw a
>> couple of /OVERWRITE keywords for both REFORM statements, to save some
>> memory and time.
>>
>> JD
>>
>
> On a time test, I found that this clever approach takes 18 sec for a 5000 x
> 4000 image, compared to the inelegant routine I posted that takes 4 seconds
> to compute the indices of each element and then 7 seconds for the resampling
> of each image. For lots of images, the speed gain is about 2.5x compared to
> this routine. Now we just need to find a way to speed up the clever routine
> and we'll be all set! I had not known about the /DIMENSION keyword to the
> MAX() function - thanks for the lead on that! Are you sure there isn't a way
> to use HISTOGRAM to do this?

I find just the opposite: your loop+REBIN method is much slower using
5000x4000 long integer arrays:

IDL> .run /home/jdsmith/idl/test/max_local.pro
% Compiled module: $MAIN$.
no loop [5000,4000]: 3.8933
French index loop [5000,4000]: 33.1060
modified index loop [5000,4000]: 1.0784

I.e. yours is about 8-10x slower on this size image! All of this of course depends on memory (1GB here). I suspect your multiple REBIN'ing of those large images is to blame. That said, I tried with a much smaller image, but the results were similar:

```
IDL> .run /home/jdsmith/idl/test/max_local.pro
% Compiled module: $MAIN$.
no loop [1000,1000]: 0.0975
French index loop [1000,1000]: 0.5524
modified index loop [1000,1000]: 0.0589
```

So, what is the modified index loop which beats both of ours? Your method inspired this significant simplification:

```
d=size(x,/DIMENSIONS) & nx=d[0] & ny=d[1]
nx2=nx/2 & ny2=ny/2

inds=rebin(lindgen(nx2)*2L,nx2,ny2,/SAMPLE)+ $
    rebin(transpose(lindgen(ny2)*2L*nx),nx2,ny2,/SAMPLE)

xmax=x[inds]
offsets=[0L,1L,nx,nx+1L]
for i=1,3 do xmax >= x[inds+offsets[i]]
```

So, at least in this case, the fastest (and definitely the most straightforward to understand) method involves a loop! It can even be extended to nxm (instead of 2x2) local box sizes quite straightforwardly. Heresy, you say? As many have pointed out, if you keep the amount of work per loop iteration large, you don't feel IDL's loop penalty. Indexing and then comparison operating on images of size 2500x2000 definitely counts as a "large amount of work". The other take away point: relative algorithm speed can depend sensitively on your memory and other local environment. The only sure way to pick the speed winner is to test various options on your data with your equipment.

JD