Subject: Re: Optimization help
Posted by Andrew Cool on Fri, 31 Mar 2006 01:54:07 GMT
View Forum Message <> Reply to Message

Jonathan Greenberg wrote:
> I was wondering if there are any tricks to speeding up the following
> database merging problem:
>
> Say I have a 3 x 10000 array, and I want to find out the row # of the 10,000
> rows matches (if any) an arbitrary 3 x 1 array. I can do this by cycling
> through each column one at a time and doing an intersection (e.g. Where(
> array[0] eq database[0,*]) intersected with where(array[1] eq database[1,*]
> intersected with where(array[2] eq database[2,*])
>
> This seems like a pretty slow approach to doing this, so are there any
> tricks to making this run a lot faster? I'm talking about doing this for an
> image, so the overhead is going to be pretty significant if I can't do any
> matrix tricks and have to look up at pixel one at a time using the above
> method...
>
> --j

For a true colour image, this seems to work just fine in picking out
those pixels that match a particular R,G,B sequence, 25 for the image
I chose as a test.

```
file = 'Bpic.jpg'

read_jpeg,file,image

rgb = [11,12,17]

match = where(image[0,*] EQ rgb[0] AND $
              image[1,*] EQ rgb[1] AND $
              image[2,*] EQ rgb[2])
```

Is that what you're after?

Andrew

Subject: Re: Optimization help
Posted by JD Smith on Fri, 31 Mar 2006 02:03:40 GMT
View Forum Message <> Reply to Message

On Thu, 30 Mar 2006 17:23:30 -0800, Jonathan Greenberg wrote:

> I was wondering if there are any tricks to speeding up the following
> database merging problem:
>
> Say I have a 3 x 10000 array, and I want to find out the row # of the 10,000
> rows matches (if any) an arbitrary 3 x 1 array. I can do this by cycling
> through each column one at a time and doing an intersection (e.g. Where(
> array[0] eq database[0,*]) intersected with where(array[1] eq database[1,*]
> intersected with where(array[2] eq database[2,*])
>
> This seems like a pretty slow approach to doing this, so are there any
> tricks to making this run a lot faster? I'm talking about doing this for an
> image, so the overhead is going to be pretty significant if I can't do any
> matrix tricks and have to look up at pixel one at a time using the above
> method...

REBIN + TOTAL along a dimension does the trick:

```
 match_rows=where(total(database eq rebin(array,3,10000,/SAMPLE),1) eq 3.)
```

What we really need is for ARRAY_EQUAL to allow a DIMENSION keyword, similar to MIN/MAX/MEDIAN/etc. This would be a boolean short-circuiting array equal along any dimension. As soon as it finds an element along a particular dimension that isn't equal, it aborts and moves to the next set. This TOTAL trick is just a cheap way to do that. Anyone at RSI listening?

You could also use MIN to accomplish the same thing:

```
 match_rows=where(min(database eq rebin(array,3,10000,/SAMPLE),DIMENSION=1))
```

which just ensure the minimum along the short dimension is 1 (i.e. all elements match). You could even use PRODUCT:

```
 match_rows=where(product(database eq rebin(array,3,10000,/SAMPLE),1))
```

since for the product to be 1, all must have been 1.

If you are really concerned about speed, you should test these against each other on your hardward/data and see which is faster. For long integers, on my machine, the TOTAL version is actually faster by about 20%. You can gain another 5-10% or so by using the following uglier version:

```
match_rows=where(total(database eq rebin(array,3,10000,/SAMPLE),1,
       /PRESERVE_TYPE) eq 3b)
```

which perform the TOTAL and comparison all in BYTEs, rather than converting to FLOAT (which is unnecessary since the maximum total here is 3). Notice the use of /SAMPLE to slightly speedup REBIN. Omitting it results in about a 5% hit.

By the way, for 3x10000, all of these are very fast, around 1/500 of a second, so it shouldn't give you any trouble for interactive use.

JD
