Subject: Re: Cumulative max() in *arbitrary* dimension?
Posted by JDS on Fri, 09 Mar 2012 16:58:44 GMT
View Forum Message <> Reply to Message

On Thursday, March 8, 2012 7:17:29 PM UTC-5, Heinz Stege wrote:
> On Thu, 8 Mar 2012 10:33:39 -0800 (PST), JDS wrote:
>
>> I've since tuned this up a bit more, saving 1/2 of the index computation
>> during each step of the loop by incrementing a running index array. It's
>> now (rather remarkably) >5x faster than MAX(DIMENSION=3) for me with
>> large 3D arrays. And of course it gives all the intermediate cumulative
>> max values.
>>
> The loop can be tuned up even more. Replacing the array of indices by
> two scalars for the subscript range makes the loop faster and also
> saves memory. I replaced the following 2 lines of your code
>   inds=lindgen(off)
>   for i=1,s[d]-1 do a[i*off]=a[inds]>a[(inds+=off)]
> by the following 3 lines:
>   i1=0
>   i2=off-1
>   for i=1,s[d]-1 do a[i*off]=a[i1:i2]>a[(i1+=off):(i2+=off)]
>
> In my examples max_cumulative is about 2 to 3 times faster than
> before:
> ~2.5 times for a 60x400x3000 byte array
> ~3.1 times for a 60x400x300 byte array
> ~2.1 times for a 60x40x3000 byte array
>
> Heinz

Hey, Heinz... very cool. This flies in the face of the general notion that "having IDL compute index arrays in loops is wasteful." This is likely because you are required to update the index set during each iteration, so you may as well let IDL do this internally. In other typical cases, you are asking IDL to repeat the calculation of an identical index loop that you can simply pre-cache for a large savings.

I replaced yours instead with the rather similar:

```
  for i=1,s[d]-1 do a[i*off]=a[(i-1)*off:i*off-1]>a[i*off:(i+1)*off-1]
```

and got about 2.5x speedup for the final dimension. For non-final dimensions, the speedup is much less, since TRANSPOSE imposes a reasonably large overhead. Since this challenged my first "rule of thumb", I decided to check the next one: that TRANSPOSE and then in-order operation saves time over indexing out of memory order. That one holds for non-final dimensions, by at least a factor of 2.

BTW, it's now *15x* faster than MAX(DIMENSION=3), for the reasons Lajos mentions.

Thanks for your thoughts.

JD