Subject: Re: Philosophy of for loops
Posted by Struan Gray on Wed, 30 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Wayne, landsman@my-deja.com writes:

> inarr= randomn(seed, 3, 2048,2048)
> outarr = fltarr(2048,2048,/nozero)
>
> (1) for j=0,2047 do for i=0,2047 do outarr[i,j] = median(inarr[*,i,j])
>
> (2) for j=0,2047 do begin
> for i=0,2047 do begin
> outarr[i,j] = median(inarr[*,i,j])
> endfor
> endfor
>
> Form (1) is slightly faster, but the calculation
> cannot be interrupted with a Control-C. Also,
> it is my impression that the speed difference
> is less than it used to be, and that form (2)
> is now better optimized.

On my machine (a Mac G3 powerbook, IDL 5.3) (1) is slightly faster for small arrays but the difference is insignificant by the time you are up to 2048x2048.

> (I also assume that the two FOR loops are unavoidable
> here, but I would be delighted to be proved wrong.)

I often take data which consists of 1D spectra on a 2D spatial grid, ending up with arrays which are (Nspecpoints, Ngridx, Ngridy) in dimension. Often I want to do some processsing operation on all the individual spectra, and it helps a lot to 'unwrap' the array so that you do one loop instead of two nested ones. In your case the code would look like this:

```
npoints = 2048
inarr = reform(inarr, 3, npoints*npoints, /overwrite)
outarr = fltarr(npoints*npoints, /nozero)
for i=0L, long(npoints)*npoints - 1 do $
  outarr[i] = median(inarr[*,i])
inarr = reform(inarr, 3, npoints, npoints, /overwrite)
outarr = reform(outarr, npoints, npoints, /overwrite)
```

For the sorts of arrays I use (100, 200, 200) this is quite a bit faster, but interestingly enough by the time you get up to arrays like

yours the speed advantage has gone. Watch out for overflow on your loop indices.

In this particular case you can actually work without loops altogether:

```
inarr = reform(inarr, 3*npoints*npoints, /overwrite)
outarr = reform(median(inarr, 3), 3, npoints, npoints, /overwrite)
outarr = reform((temporary(outarr))[1,*,*])
inarr = reform(inarr, 3, npoints, npoints, /overwrite)
```

The median filter creates a whole load of 'wrong' elements, but we can ignore them and eliminating the loop speeds the whole thing up so much that it's worth the overhead to calculate them. This version was substantially faster than the other three at all array sizes.

Both of these techniques require the 'interesting' dimension to be first, but sandwiching the code with a ROTATE or TRANSPOSE will do that without a punitive overhead so they can be made quite general.

I apologise to J.D. for not fitting HISTOGRAM in there somewhere.

Struan
