Subject: Re: FOR loops and efficiency
Posted by JDS on Tue, 26 May 2009 21:51:06 GMT
View Forum Message <> Reply to Message

> I still stand by my rule of thumb. The problem with FOR loops is the
> amount of time spent doing loop overhead stuff. If you run your loop
> but *take all the calculations out*, and the total execution time is
> not perceptible, then you probably won't gain by optimizing/
> vectorizing.

I find that analysis lacking for a few reasons. Consider this example:

```
IDL> a=randomu(sd,10000000L)
IDL> t=systime(1) & b=total(a,/CUMULATIVE) & print,systime(1)-t
     0.066554070
IDL> t=systime(1) & for i=0L,10000000L-2 do a[i+1]+=a[i] &
print,systime(1)-t
      2.2091250
IDL> print,array_equal(a,b)
   1
```

The non-loop version was >30x faster. So by your rule of thumb, our simple loop with its small calculation must be totally dominated by loop overhead. Let's check that:

```
IDL> t=systime(1) & for i=0L,10000000L-2 do begin & end & print,systime
(1)-t
      0.12700295
```

Not so much. The loop overhead contributed only roughly 5% of the total computation time, and yet the simple call to TOTAL blew it out of the water. Why?

Loop overhead is one reason to avoid FOR loops with high iteration count, but it is by no means not the *only* reason. Element by element memory access is less efficient, multiple processor cores cannot typically be used in the loop statement, and no doubt many other code/data fetch efficiency factors come into play. That's why there is no simple answer to "when is a FOR loop a problem". The only real answer is "when non-FOR loop methods can be found, are faster, and you need the extra speed." My rough rule of thumb: if you are accessing many hundreds or thousands of elements in each iteration, you are probably not being impacted by efficiency issues, though you will lose potential multicore speedup.

Sometimes you encounter a problem which just requires a loop. IDL *really needs* the ability to easily call out to C code which can be

automatically compiled (ideally by a tool distributed with IDL itself) into efficient form, and which can be integrated directly into the session. MATLAB has had most of this capability for some time. There are hacks using MAKE_DLL, but they are tricky, depend on user compiler installation, and are consequently rarely used.

JD
