Subject: Re: Arrays suck. Loops rock.
Posted by JD Smith on Fri, 04 Mar 2005 20:59:11 GMT
View Forum Message <> Reply to Message

On Fri, 04 Mar 2005 13:22:22 -0500, Benjamin Hornberger wrote:

> If you're sick of discussions about loops, better skip to the next
> message. If not, read on.
>
> I'm disappointed. I've been told loops suck and arrays rock in IDL. I've
> read the dimension juggling tutorial
> (http://www.dfanning.com/tips/rebin_magic.html) and was striving hard to
> eliminate all loops from my code. Until today. When suddenly my code
> with two FOR loops ran twice as fast as loopless. One loop was right in
> between.

Well, as the perpetrators of the tongue-in-cheek emodiment-of-pure-evil FOR loop characterization, I'll add this qualifier: arrays are great, until you run out of memory for them. Then they are paged to disk, and take 1000x as long to access.

Thought experiment. Total up the first 20 billion prime numbers.

ARRAY method: allocate array of 20 billion long-long(-long-long) integers, pre-compute primes and fill into array, TOTAL.

LOOP method: compute primes in sequence, increment running total as you go.

Can you see why the former is doomed to fail?

There are classes of algorithms where you trade memory size for compute speed. The typical tradeoff occurs at a much larger memory size in IDL than it does in many other languages, because IDL is so fast at manipulation huge piles of array data, and so relatively slow at looping through things.

Then, there are other problems where your memory size is fixed (e.g. you have a real data cube of several GB). This is a crucial distinction. In the second case, your data size is dictated. You must try to cut it up into big, but manageable chunks and operate on it.

In your case, you're trying to create a few floating array of size 300x300x10x20, which should easily fit in memory. However, in none of your cases are you accessing a single index at a time, so they are all "array based" at some level. Even nloop=2 is hammering away at sub-images of size 300*300 per iteration.

Why don't you try adding another case:

```
    else if nloops eq 4 then begin
      FOR j=0, n_data-1 DO  $
        for k=0,n_fast_pixels-1 do $
          for l=0,n_slow_pixels-1 do $
            displays[k, l, i] += linear_combinations[j, i] * $
                                 (image_data)[k, l, j]
```

to see what happens when you iterate over each and every member of the array:

```
IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=0
took        1.1159251 sec
IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=1
took       0.96256900 sec
IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=2
took        2.0543392 sec
IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=4
took        27.267108 sec
```

Ouch! As Craig has said many times, if you can keep the amount of work you are doing per iteration high, you won't feel the looping "penalty" much, and can actually keep up good speed. As soon as the looping penalty is comparable to the work you are doing per iteration (or much larger, as in the last example), you're out of luck. The fastest case for me, NLOOPS=1, still uses REFORM/REBIN.

JD

Subject: Re: Arrays suck. Loops rock.
Posted by Y.T. on Tue, 08 Mar 2005 19:51:22 GMT
View Forum Message <> Reply to Message

> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=0
> took   1.1159251 sec
> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=1
> took   0.96256900 sec
> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=2
> took   2.0543392 sec
> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=4
> took   27.267108 sec
>

I'd like to add to this that I sure hope you guys didn't just run each

call once -- for then the results you get are utterly arbitrary. At the
level of a sizable fraction of a second the OS can throw stuff at you
for no better reason than that it is tuesday. On your first call, IDL
has to allocate a lot of memory, for example, which it may or may not
free up and thus it may or may not be faster on the second call. Any
memory allocation may or may not proceed swiftly depending on how much
cleanup the OS has to do before it can hand IDL the requisite amount of
memory. And so forth.

Thus for any kind of timing comparisons you'll always want to re-run
the routine in question at least a couple times (ideally something like
1000 times from a loop and then compute the average). Otherwise you're
measuring OS latency because of a big TCP packet that happened to hit
your ethernet port just that moment (or similar things) rather than the
actual speed of the algorithm.

(I honestly fail to understand the provided example and am not at all
sure that the two "rebins" are necessary. You are aware that 'rebin'
performs interpolations, right?)

cordially

Y.T.

--
Remove YourClothes before you email me.

Subject: Re: Arrays suck. Loops rock.
Posted by Benjamin Hornberger on Tue, 08 Mar 2005 20:12:56 GMT
View Forum Message <> Reply to Message

Y.T. wrote:
> (I honestly fail to understand the provided example and am not at all
> sure that the two "rebins" are necessary. You are aware that 'rebin'
> performs interpolations, right?)

The rebin / reform steps are necessary if, e.g., you want to multiply a
3-d array [m, n, k] by a 2-d array [m, n] "layer by layer" in one array
operation rather than looping over k. Rebin won't interpolate in this
case, because you rebin from size 1 to size k (which just clones a value
k times). See the Dimension Juggling Tutorial at

http://www.dfanning.com/tips/rebin_magic.html

Some of the reform steps are necessary because of IDL's terrible "feature" of cutting of trailing dimensions of size 1.

Thanks for the input! (I did run it multiple times, with similar results, but I see that I should probably shut down as much other activity on the computer as I can while timing these calculations.)

Benjamin

Subject: Re: Arrays suck. Loops rock.
Posted by JD Smith on Tue, 15 Mar 2005 00:46:16 GMT
View Forum Message <> Reply to Message

On Tue, 08 Mar 2005 11:51:22 -0800, Y.T. wrote:

>
>> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=0
>> took 1.1159251 sec
>> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=1
>> took 0.96256900 sec
>> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=2
>> took 2.0543392 sec
>> IDL> test_loop,findgen(300,300,10),findgen(10,20),disps,nloops=4
>> took 27.267108 sec
>>
>
>
> I'd like to add to this that I sure hope you guys didn't just run each
> call once -- for then the results you get are utterly arbitrary. At the
> level of a sizable fraction of a second the OS can throw stuff at you
> for no better reason than that it is tuesday. On your first call, IDL
> has to allocate a lot of memory, for example, which it may or may not
> free up and thus it may or may not be faster on the second call. Any
> memory allocation may or may not proceed swiftly depending on how much
> cleanup the OS has to do before it can hand IDL the requisite amount of
> memory. And so forth.
>
> Thus for any kind of timing comparisons you'll always want to re-run
> the routine in question at least a couple times (ideally something like
> 1000 times from a loop and then compute the average). Otherwise you're
> measuring OS latency because of a big TCP packet that happened to hit
> your ethernet port just that moment (or similar things) rather than the
> actual speed of the algorithm.

But of course. In addition to that caveat, the major warning that any given timing comparison may be completely invalid on another system should

be made.

> (I honestly fail to understand the provided example and am not at all
> sure that the two "rebins" are necessary. You are aware that 'rebin'
> performs interpolations, right?)

Here REBIN is used in a bit of a non-intuitive way, simply to replicate data in arrays along additional dimensions. Used with /SAMPLE (and in recent versions of IDL, by default), it never tries to perform any sort of interpolation, despite that being its primary use. This functionality could probably be split out into another more aptly-named routine at a huge gain in readability. But then people might discover how to do it by reading the manuals, and that would be worrisome.

JD
