Subject: Re: For loops vs. matrix operations
Posted by Craig Markwardt on Wed, 17 Dec 2003 22:57:52 GMT
View Forum Message <> Reply to Message

"Jonathan Greenberg" <greenberg@ucdavis.edu> writes:

```
> I know some matrix programs perform better if you do straigh matrix math vs.
> a for-next loop -- is idl this way?  E.g. is:
>
> array=intarr(10000)
> for i=0,(10000-1) do begin
>     array[i]=array[i]+1
> endfor
>
> MUCH slower than:
>
> array=intarr(10000)
> array=array+1
>
> ?
>
> I'm trying to figure out how much time I should be using rewriting some code
> to optimize the algorithm, which is why I'm asking (the code is more complex
> than above, obviously, but I did notice I could "matricize" some of the code
> in places)...
```

The simplest answer is... optimize the slowest parts. To be a little more specific, the slowest parts are usually the innermost loops, which in your case above *is* the loop. If you can find obvious things like the one you listed above, then definitely do it.

One nice feature of IDL which I didn't know about until recently is PROFILER. While it doesn't give a line-by-line breakdown of execution time, it does give a function-by-function one. If you have more than a few routines, PROFILER should be able to tell you where to start optimizing first.

Happy optimizing!
Craig

--
--- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
--- --------------

Subject: Re: For loops vs. matrix operations
Posted by mperrin+news on Wed, 17 Dec 2003 23:05:08 GMT
View Forum Message <> Reply to Message

Jonathan Greenberg <greenberg@ucdavis.edu> wrote:

```
> I know some matrix programs perform better if you do straigh matrix math vs.
> a for-next loop -- is idl this way?  E.g. is:
>
> array=intarr(10000)
> for i=0,(10000-1) do begin
>     array[i]=array[i]+1
> endfor
>
> MUCH slower than:
>
> array=intarr(10000)
> array=array+1
>
> ?
```

Yes, the for loop version will be *vastly* slower. This is because IDL
makes a seperate trip through the parse/interpret cycle for every pass
through the for loop, greatly increasing the overhead.

> I'm trying to figure out how much time I should be using rewriting some code
> to optimize the algorithm, which is why I'm asking (the code is more complex
> than above, obviously, but I did notice I could "matricize" some of the code
> in places)...

Matricize as much as you possibly can!

 - Marshall

Subject: Re: For loops vs. matrix operations
Posted by Wonko[3] on Wed, 17 Dec 2003 23:24:00 GMT
View Forum Message <> Reply to Message

greenberg@ucdavis.edu (Jonathan Greenberg) wrote:

> I know some matrix programs perform better if you do straigh matrix math
> vs. a for-next loop -- is idl this way? E.g. is:

```
> array=intarr(10000)
> for i=0,(10000-1) do begin
>     array[i]=array[i]+1
> endfor
```

> MUCH slower than:

> array=intarr(10000)
> array=array+1

Not only MUCH, but **MUCH** slower, at least.

Even faster is this: aray = temporary(array) + 1
This avoids duplicating the a variable first, saving time and memory.

But why don't you try it yourself?

```
IDL> a = intarr( 10000000L )
IDL> t=systime(1) & for i = 0L, 10000000-1 do a[i]=a[i]+1 & print, systime(1)-t
      4.2659999
IDL> t=systime(1) & a=a+1 & print, systime(1)-t
      0.18999994
IDL> t=systime(1) & a=temporary(a)+1 & print, systime(1)-t
      0.040000081
```

> I'm trying to figure out how much time I should be using rewriting some
> code to optimize the algorithm, which is why I'm asking (the code is more
> complex than above, obviously, but I did notice I could "matricize" some
> of the code in places)...

Matricyzation should always save time, especially if you have small
inner loops. I also think this makes the code more readable and
universal.

Alex
--
Alex Schuster   Wonko@wonkology.org   PGP Key available
alex@pet.mpin-koeln.mpg.de

Subject: Re: For loops vs. matrix operations
Posted by James Kuyper on Wed, 17 Dec 2003 23:52:51 GMT

View Forum Message <> Reply to Message

Alex Schuster wrote:
...
> Matricyzation should always save time, especially if you have small
> inner loops. I also think this makes the code more readable and
> universal.

Usually, yes, but some of the things you have to do in IDL to get
reasonable speed by avoiding the use of loops are extremely un-readable.
I think most of the arcane uses of HISTOGRAM, for instance, fall into

this category.

Subject: Re: For loops vs. matrix operations
Posted by JD Smith on Thu, 18 Dec 2003 01:31:22 GMT
View Forum Message <> Reply to Message

On Wed, 17 Dec 2003 16:52:51 -0700, James Kuyper wrote:

> Alex Schuster wrote:
> ...
>> Matricyzation should always save time, especially if you have small
>> inner loops. I also think this makes the code more readable and
>> universal.
>
> Usually, yes, but some of the things you have to do in IDL to get
> reasonable speed by avoiding the use of loops are extremely un-readable.
> I think most of the arcane uses of HISTOGRAM, for instance, fall into
> this category.

As one of the purveyors of arcane HISTOGRAM usage, I have to agree.
There are some problems that have clear solutions with HISTOGRAM, even
many funky-looking REVERSE_INDICES things, but lots of operations
would be clearer with a plain old loop.

This got me thinking about FOR loops in IDL: their speed penalty, as
has been mentioned, is a direct result of the highly convenient IDL
interpreter. For each statement in each trip through a FOR loop, IDL
goes through a very large and costly internal interpreter loop which
provides all sorts of whiz-bang conveniences, like parsing execute
statements, responding to interrupts and errors, and who know what
else. In fact, this penalty is not really intrinsic to a FOR loop; it
just represents the finite amount of time it takes to interpret any
single IDL statment. In fact, if I wrote a very long procedure like:

```
 a[0]=a[0]+1
 a[1]=a[1]+1
 a[2]=a[2]+1
 ...
 a[999999]=a[999999]+1
```

it would also run very slowly, since each lines suffers the
"interpreter penalty" -- in fact, except for the long time it takes to
read in and compile a file of 1 million lines, the executing takes
exactly the same amount of time (about .7s on my machine) as the
equivalent for-loop. So perhaps we should call it the "interpreter
penalty" instead of the "for loop penalty". But what if you don't
need all the whiz-bang conveniences of the interpreter for each and

every command in a long loop? What if, instead, you could request IDL
to shunt your calculation into a tight, optimized "side-loop" that
comes with a set of restrictions, e.g. no EXECUTE, non-interruptible,
etc. It could look like:

```
for i=0L,999999L do begin
  .compile_opt TIGHTLOOP
  a[i]=a[i]+1
endfor
```

In theory, you *should* be able to save on the penalty of interpreting
that one line 1 million times, since it's the same line each time.
And then I asked myself, why can't IDL just recognize loops which are
amenable to TIGHTLOOP'ing, and perform that optimization
automatically? Perhaps you couldn't approach the speed of a loop at
the machine level (i.e. written in C), but you might be able to shave
a significant amount off the large penalty. Of course, I'm not privy
to the internals of IDL's coding, so this is all speculation, but
perhaps there's a way for us to have our cake and eat it too.

JD

Subject: Re: For loops vs. matrix operations
Posted by Kenneth P. Bowman on Thu, 18 Dec 2003 02:59:13 GMT
View Forum Message <> Reply to Message

In article <Z%3Eb.41137$jo.28094@newssvr29.news.prodigy.com>,
 "Jonathan Greenberg" <greenberg@ucdavis.edu> wrote:

```
>  array=intarr(10000)
>  for i=0,(10000-1) do begin
>     array[i]=array[i]+1
>  endfor
>
>  MUCH slower than:
>
>  array=intarr(10000)
>  array=array+1
```

Try timing it and see :-)

```
n     = 1000000
array = LONARR(n)

time0 = SYSTIME(/SECONDS)
FOR i = 0, n-1 DO array[i] = array[i] + 1
```

```
time0 = SYSTIME(/SECONDS) - time0

time1 = SYSTIME(/SECONDS)
array = array + 1
time1 = SYSTIME(/SECONDS) - time1

PRINT, 'Speed-up : ', time0/time1



IDL> @time1
Speed-up :       42.071883
```

Subject: Re: For loops vs. matrix operations
Posted by marc schellens[1] on Thu, 18 Dec 2003 07:25:24 GMT

View Forum Message <> Reply to Message

Marshall Perrin wrote:
> Jonathan Greenberg <greenberg@ucdavis.edu> wrote:
>
>> I know some matrix programs perform better if you do straigh matrix math vs.
>> a for-next loop -- is idl this way? E.g. is:
>>
>> array=intarr(10000)
>> for i=0,(10000-1) do begin
>>   array[i]=array[i]+1
>> endfor
>>
>> MUCH slower than:
>>
>> array=intarr(10000)
>> array=array+1
>>
>> ?
>
>
> Yes, the for loop version will be *vastly* slower. This is because IDL
> makes a seperate trip through the parse/interpret cycle for every pass
> through the for loop, greatly increasing the overhead.

Interpret cycle only. Parsing is only done once.

>> I'm trying to figure out how much time I should be using rewriting some code
>> to optimize the algorithm, which is why I'm asking (the code is more complex
>> than above, obviously, but I did notice I could "matricize" some of the code
>> in places)...

>
>
> Matricize as much as you possibly can!

True always and anyway.

marc

Subject: Re: For loops vs. matrix operations
Posted by David Fanning on Thu, 18 Dec 2003 12:55:24 GMT
View Forum Message <> Reply to Message

Alex Schuster writes:

> Matricyzation should always save time, especially if you have small
> inner loops. I also think this makes the code more readable and
> universal.

Indeed. Witness any of JD's one line wonders. *Very simple* to read.

Cheers,

David

P.S. Let's just say *understanding* is a bit slower in coming
for me. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http:/www.dfanning.com/
Phone: 970-221-0438, IDL Book Orders: 1-888-461-0155