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
