Subject: Re: Avoid loop in matrix operation
Posted by JD Smith on Thu, 06 Jul 2006 19:58:47 GMT
View Forum Message <> Reply to Message

On Thu, 06 Jul 2006 03:56:17 -0700, L. Testut wrote:

>
> greg michael wrote:
>> How about:
>>
>> mean_topo=total(H,3)/(size(H))[3]
>>
>> regards,
>> Greg
>
> Thanks Greg,
> That exactly what I want for this case. But in a case where the function
> is not as simple as the MEAN function. Let's say that I want to compute
> the curvature of the DEM, with my own function CURV(H), is it possible to
> make it as simple as for the mean ?

Only if your CURV() function uses built-in primitives which can
operate on entire arrays at once, including
TOTAL/PRODUCT/MEDIAN/MIN/MAX/CONVOL and others, along with arithmetic,
array multiplication, and similar "array-aware" operators. There is no
"generic threading" operator which allows you to apply a generic function
along a specific dimension or set of dimensions of an array.

In fact, now that I think of it, why *isn't* there such a generic
array threading function? The main concern with looping in IDL is
that the loop itself imposes a significant overhead per iteration.
This overhead can be greater (far greater in some cases) than the
actual work you are trying to do.

So how about a new function, called APPLY/COLLAPSE/REDUCE/whatever,
which takes an array, a generic function (which accepts an array or
vector argument and returns an array/vector/scalar with one fewer
dimensions), and a dimension over which to apply that function. Then
it runs through in a *tight* loop, applying that function and
collating the results into a return array, with a minimum of loop
overhead.

You still have function call overhead, and all the memory gymnastics
required to accumulate results, but at least you're not paying for
full trips around the interpreter loop, checking for widget events,
keyboard input, and all manner of other baloney that you don't really
need. This could speed up typical cases significantly, and more
importantly open the door for a new set of inter-operable, user-coded

array operations to proliferate.

I had once advocated a new language semantic for a "fast loop", but I think an array-based construct like REDUCE could get you 90% of the way there without any new language elements.

JD