Subject: Re: Avoid loop in matrix operation
Posted by greg michael on Thu, 06 Jul 2006 10:49:35 GMT
View Forum Message <> Reply to Message

How about:

mean_topo=total(H,3)/(size(H))[3]

regards,
Greg

Subject: Re: Avoid loop in matrix operation
Posted by L. Testut on Thu, 06 Jul 2006 10:56:17 GMT
View Forum Message <> Reply to Message

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 ?

curvature=curv(H)

...hum maybe my question is stupid ! it depends on the way I write my function ?

Thanks again,
Cheers
Laurent

Subject: Re: Avoid loop in matrix operation
Posted by greg michael on Thu, 06 Jul 2006 11:55:53 GMT
View Forum Message <> Reply to Message

The problem is that MEAN always returns a scalar, so no clever manipulations will ever get an array out of it. I can only see that you

could write a routine to apply an arbitrary such function in the way
you want... or you include it in your curv function if you only need it
once.

regards,
Greg

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

Subject: Re: Avoid loop in matrix operation
Posted by greg michael on Thu, 06 Jul 2006 21:01:35 GMT
View Forum Message <> Reply to Message

Sounds good to me. It's the kind of thing you would expect IDL *ought* to be able to do...

Subject: Re: Avoid loop in matrix operation
Posted by George N. White III on Fri, 07 Jul 2006 19:09:07 GMT
View Forum Message <> Reply to Message

On Thu, 6 Jul 2006, JD Smith wrote:

> 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.

Other matrix languages have had this for years -- how far behind can IDL stay without going backwards?

People following this thread might want to look at Matlab's recent implementation of new functions to vectorize operations over heterogeneous arrays and structures, described in
< http://www.mathworks.com/company/newsletters/digest/2006/mar /vector.html>

--
George N. White III <aa056@chebucto.ns.ca>

Subject: Re: Avoid loop in matrix operation
Posted by JD Smith on Fri, 07 Jul 2006 20:46:31 GMT
View Forum Message <> Reply to Message

On Fri, 07 Jul 2006 16:09:07 -0300, George N. White III wrote:

> On Thu, 6 Jul 2006, JD Smith wrote:
>
>> [quoted text muted]
>
> Other matrix languages have had this for years -- how far behind can IDL
> stay without going backwards?
>
> People following this thread might want to look at Matlab's recent
> implementation of new functions to vectorize operations over heterogeneous
> arrays and structures, described in
> < http://www.mathworks.com/company/newsletters/digest/2006/mar /vector.html>

Say what you will about IDL's arrays, but they are fast, I think a
good bit faster than any comparable solution among 4G languages. I do
have to say, this "cell array" concept mentioned is really quite nice.
It's essentially syntatic sugar for PTRARR, hiding the pointer
creation and de-referencing bit, with the ability to thread over
entire such arrays at once. IDL has nothing like it.

BTW, these new MATLAB functions don't allow you to collapse over array
dimensions, just operate on all elements of an array, or all fields of
a structure. IDL can already do the former (since it can dereference
arbitrary fields of arrays of structures), but not the latter (though
with .(i) notation it wouldn't be hard to generalize). I'm not sure
if Matlab has anything like the generic array threading of APL or J.

JD