Subject: Optimizing a lookup table
Posted by ianpaul.freeley on Sat, 03 Mar 2007 01:15:14 GMT
View Forum Message <> Reply to Message

I would like to write a function that uses a look-up table. Now the easiest (coding-wise), is to just save the look-up table and have the function restore it. However, I plan to be calling this function in a loop, so this would result in numerous unnecessary disk-reads. Anyway, what is the most efficient way to get a lookup table into an IDL function?

My thoughts so far:
1) Just print the look-up table to the screen, copy/paste into the function and add some brackets and commas, presto, variable is in the function and will stay loaded as long as the function is compiled. This will work for my present problem, but would be unwieldy for really large look-up tables and I worry about double-precision getting truncated on the print.
2) have a procedure that reads in the table and puts it in a common block, then just start my function with an if-statement to see if the common block exists and if not, call the reading procedure. My question here is, how can I check to see if a common block has already been created? I know I could call the common block maker outside the loop, but that seems lame and makes it more complicated if I want to share the code.

Unless someone comes up with something really witty, I'll just use option 1. Just seems like this would be a common problem that someone's solved before.

cheers,
IP Freeley

Subject: Re: Optimizing a lookup table
Posted by Paul Van Delst[1] on Mon, 05 Mar 2007 22:29:37 GMT
View Forum Message <> Reply to Message

ianpaul.freeley@gmail.com wrote:
> I would like to write a function that uses a look-up table. Now the
> easiest (coding-wise), is to just save the look-up table and have the
> function restore it. However, I plan to be calling this function in a
> loop, so this would result in numerous unnecessary disk-reads.
> Anyway, what is the most efficient way to get a lookup table into an
> IDL function?
>
> My thoughts so far:
> 1) Just print the look-up table to the screen, copy/paste into the

> function and add some brackets and commas, presto, variable is in the
> function and will stay loaded as long as the function is compiled.
> This will work for my present problem, but would be unwieldy for
> really large look-up tables and I worry about double-precision getting
> truncated on the print.
> 2) have a procedure that reads in the table and puts it in a common
> block, then just start my function with an if-statement to see if the
> common block exists and if not, call the reading procedure. My
> question here is, how can I check to see if a common block has already
> been created? I know I could call the common block maker outside the
> loop, but that seems lame and makes it more complicated if I want to
> share the code.

How about

3) Have an initialisation procedure that reads in the table and puts it in its own structure. The LUT file should contain all the LUT dimensions to which the LUT structure is allocated during the initialisation. Your data is then in a nice neat bundle that can be either a) passed to whatever procedure as required or b) stuck in a common block. Whether you choose (a) or (b) depends on a bunch of stuff like size, frequency of use, the overall design of your application, etc.

I do this all the time and updating the LUT is a simple act of replacing a file. Very easy.

> Unless someone comes up with something really witty, I'll just use
> option 1.

Kludgy and not very extensible. What happens when you want to use a bunch of different numbers? When I receive code like that I return it with a "please fix it" note.

> Just seems like this would be a common problem that
> someone's solved before.

Yep. See #3 above. :o)

> cheers,
> IP Freeley

Lucky you.

cheers,

paulv

--
Paul van Delst Ride lots.
CIMSS @ NOAA/NCEP/EMC Eddy Merckx