Subject: Re: Larger arrays or more dimensions?
Posted by David Higgins on Sun, 29 Nov 2009 23:45:34 GMT
View Forum Message <> Reply to Message

On 26 Nov, 16:40, pp <pp.pente...@gmail.com> wrote:
> On Nov 26, 11:49 am, David Higgins <higgins.da...@gmail.com> wrote:
>
>
>
>
>
>> I have a large data set which can be organised into sub categories,
>> where in each sub category an experimental variable was different.
>
>> (For anyone who cares, it's MRI data which are multiple echoes, in
>> multiple phases, in multiple dynamics, through multiple RF channels,
>> over multiple signal averages.)
>
>> The data is essentially a 3D set, repeated over these various degrees
>> of freedom ("phases", "dynamics", etc).
>
>> Here's my question. In terms of memory management, and program speed,
>> would it be better to have a 3D array, and extend the 3rd dimension
>> over and over for all these degrees of freedom, or would it be better
>> to use a multiple dimension array?
>
>> (I shall be preforming FFTs on the data.)
>
>> Thanks
>> Dave Higgins
>
> Probably both ways could be equally efficient (or equally inefficient,
> depending on how you do it), depending on how you organize your
> operations on the arrays, and over which slices each operation
> depends.
>
> That said, if your problem's "units" are 3D arrays, and you have a set
> of those where each 3D array depends on some degrees of freedom, it
> would more naturally fit with the use of more dimensions. It would
> make the code easier to write and understand, since it would make
> explicit how variations in these parameters map to different 3D
> arrays. You would not need to keep track yourself of the indexes on
> the 3rd dimension that map to each parameter value.
>
> You can make use of the significant conveniences IDL provides for
> multiple dimensions (relating to the other recent topic, one of IDL's
> greatest strengths): use of 1D indexes in multiple dimension arrays,
> array_indices to convert between them, the semantics of vector

> indices, several useful functions for those operations (reform, rebin,
> replicate, value_locate, where, histogram) and the ease of writing
> routines that can handle inputs and outputs of variable dimensions.
>
> For instance, I have used these features to write a routine that
> interpolates over an arbitrary number of dimensions of an array with
> an arbitrary (usually larger) number of dimensions, keeping the other
> dimensions unaltered.
>
> Two points to keep in mind:
>
> 1) The number of dimensions is limited to 8.
>
> 2) As your units seem to be 3D arrays, it will probably be better to
> keep those 3 dimensions the leftmost. That way, each unit will be
> stored contiguously, which will make accessing a unit much more
> efficient and simpler to code and read. Also, that way routines that
> expect to be given a single one of these units can be given a direct
> slice and will work just as if they had been given a 3D array (because
> trailing dimensions of length 1 are ignored).

Thanks for the detailed reply. This info is going to help. I have a
related question, regarding reading in the file. I'm using
READ_BINARY, and then making another - sorted - copy of the data using
header data, with regards to our discussion above about dimensions.
But this seems very memory intensive; the data are unpredictable and
the data set might by large. I was wondering about ASSOC but it seems
like the data needs to be already in large chunks (it is not). I've
come across SHMMAP but it seems like a dark art for IDL gurus only. So
I'm back to READ_BINARY, and I'm thinking of writing my own template
structure from header data (BINARY_TEMPLATE would be no good since I
need the program to run without user interaction). Then, I think, I'll
have only one copy of the data on the go, straight into a sorted
format.
Does this seem like the right strategy?
Thanks
Dave Higgins