Subject: HDF SDS array access in IDL
Posted by William Clodius on Mon, 26 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

I am confused by some comments in the IDL documentation for HDF file
access e.g., the documentation for HDF_SD_GETDATA

"The HDF_SD_GETDATA procedure retrieves a hyperslab of values from an SD
dataset. By default, the retrieved data is transposed from HDF's column
order format into IDL's row order which is more efficient in IDL. To
retrieve the dataset without this transposition, set the NOREVERSE
keyword."

I believe that IDL, like Fortran, is column major and transposing the
data would be exactly the wrong thing to do for the default. Am I
mistaken or has a misguided C programmer been at work at RSI?

--

William B. Clodius Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2 FAX: (505)-667-3815
PO Box 1663, MS-C323 Group office: (505)-667-5776
Los Alamos, NM 87545 Email: wclodius@lanl.gov

Subject: Re: HDF SDS array access in IDL
Posted by davidf on Tue, 27 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

William Clodius (wclodius@lanl.gov) writes:

> I am confused by some comments in the IDL documentation for HDF file
> access e.g., the documentation for HDF_SD_GETDATA
>
> "The HDF_SD_GETDATA procedure retrieves a hyperslab of values from an SD
> dataset. By default, the retrieved data is transposed from HDF's column
> order format into IDL's row order which is more efficient in IDL. To
> retrieve the dataset without this transposition, set the NOREVERSE
> keyword."
>
> I believe that IDL, like Fortran, is column major and transposing the
> data would be exactly the wrong thing to do for the default. Am I
> mistaken or has a misguided C programmer been at work at RSI?

I think the problem here is that William changes the
"column order" words of the writer to "column major"
and misinterprets what is meant. Since I am *always*
confused about what "column major" means, I'll tell you

what I *do* know. :-)

IDL stores information in a row-order format. That is
to say that in memory row elements are contiguous. HDF
apparently stores information in a column-order format,
so that column elements are contiguous.

Does this matter? Most of the time, not a whit. It has
nothing to do with how columns or rows are specified
in variables, or whether the column notation is first
or second. Where it does matter is if you are doing
something in a loop. If data is stored row-ordered
and you loop over a column index, the loop can be
quite slow, since you have to grab successive chunks
of memory that are not next to each other.

Thus, I think it is a good thing to allow IDL to
do the switch for you. Saves you having to remember
yet one more thing as you work with the data.

Cheers,

David

WARNING: My brain is still several time zones away
from being completely engaged, so my confidence level
in this answer is somewhat lower than my normal 50
percent. :-)

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: HDF SDS array access in IDL
Posted by davidf on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

William Clodius (wclodius@lanl.gov) writes:

> [Long, well-written article snipped.]

I've read William's excellent article three times now and
I'm *still* confused. :-(

> (Note because of the current limitations of C arrays I suspect that
> IDL's arrays are actually implemented as C pointers with indexing
> similar to F2C's.)

Now, hold on there. What the hell does THIS mean!?

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: HDF SDS array access in IDL
Posted by William Clodius on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Dr. G. Scott Lett wrote:
> <snip>
> IDL uses a storage scheme like C, not Fortran. The question of
> column-majority and row-majority was a source of confusion at RSI for some
> time, and this confusion was evident in some of their documentation. Recent
> editions should be better.
> <snip>

Technically what we are talking about is an access scheme not a storeage scheme. Storeage per se is typically defined by the OS/processor, and the language only defines a mapping to that storeage. For multidimensional arrays, Fortran promises that access is most efficient if the left most index varies most rapidly. C promises that access is most efficient if the right most index varies most rapidly. In two dimensional array the first index is commonly termed the column index, the second index the row index. In Fortran, and IDL, varying the column index and keeping the row constant is most efficient, in C varying the row index and keeping the column constant is most efficient. Fortran's access mode is commonly refered to as column major, I have never hear the term row order.

IDL's (5.1) Help is confusing on the issue. For Arrays and Matrices it states

"In a computer, a multidimensional data set ... can be indexed in column-major format, which means that the linear order of the data

elements proceeds from the first element in the first column through the last element in the first column before beginning on the second column, and so on. This is the format used by FORTRAN to index data."

so far correct

"Alternatively, data can be indexed in row-major format, meaning that the linear order of the data elements proceeds from the first element of the first row through the last element of the first row before beginning on the second row, and so on. This is the format used by the C and Pascal computer languages, and is traditionally associated with image processing, because it keeps all the elements of a single image scan line together. Because IDL's origins are in image processing, it indexes data in row-major format."

Almost all languages keep scan lines together and this naming convention has no impact on image processing. They just access them together in different indices, C groups lines by the leftmost index Fortran by the rightmost. C may be more commonly used for image processing for a variety of reasons (more often taught at universities, easier access to hardware, structs have more flexibility than what is available in F77, wide availability of cheap compilers), but this is not one of them.

"NoteMany computer languagesï¿½e.g. C, Pascal, Visual Basic, and Fortanï¿½index 2-dimensional arrays in (row, column) order. IDL indexes arrays in (column, row) order."

Note the above is Humpty Dumpty. The C and Fortran standards do not define which index is associated with the row and which with the column so the definition should follow other conventions. Historically, the convention in column major versus row major is that in column major the first dimension elements are contiguous and in row major the last index elements are contiguous. Under that convention, Fortran and IDL are both column major, C and Pascal are row major. IDL is trying to make up a confusing convention to make itself sound as though it is more like C than Fortran. See for example

http://member.aol.com/CORLISS100/chapter5.html
http://webster.ucr.edu/Page_asm/ArtofAssembly/CH05/CH05-2.ht ml
http://www.cs.uregina.ca/dept/manuals/Manuals/3_0NumRep/3_Nu m_Rep.html

(Note because of the current limitations of C arrays I suspect that IDL's arrays are actually implemented as C pointers with indexing similar to F2C's.)

--

William B. Clodius Phone: (505)-665-9370

Los Alamos Nat. Lab., NIS-2 FAX: (505)-667-3815
PO Box 1663, MS-C323 Group office: (505)-667-5776
Los Alamos, NM 87545 Email: wclodius@lanl.gov

Subject: Re: HDF SDS array access in IDL
Posted by Dr. G. Scott Lett on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

The difference is probably only semantics until you try to make standard
Fortran linear algebra code and IDL play together. (But who would do that?)

> Uh, oh. The symantic confusion is upon us! I suggest we
> all learn the mantra "data is stored in row order". It may
> be our only hope. :-)

Subject: Re: HDF SDS array access in IDL
Posted by davidf on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

William Thompson (thompson@orpheus.nascom.nasa.gov) writes:

> "Dr. G. Scott Lett" <slett@holisticmath.com> writes:
>
>> IDL uses a storage scheme like C, not Fortran. The question of
>> column-majority and row-majority was a source of confusion at RSI for some
>> time, and this confusion was evident in some of their documentation. Recent
>> editions should be better.
>
>
> I completely disagree!!!! IDL uses a storage system like Fortran. The
> leftmost index is the one that changes most rapidly in the stored array.

Uh, oh. The symantic confusion is upon us! I suggest we
all learn the mantra "data is stored in row order". It may
be our only hope. :-)

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155

Subject: Re: HDF SDS array access in IDL
Posted by thompson on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

"Dr. G. Scott Lett" <slett@holisticmath.com> writes:

> IDL uses a storage scheme like C, not Fortran. The question of
> column-majority and row-majority was a source of confusion at RSI for some
> time, and this confusion was evident in some of their documentation. Recent
> editions should be better.

I completely disagree!!!! IDL uses a storage system like Fortran. The leftmost index is the one that changes most rapidly in the stored array. For example, if we execute the IDL commands

```
IDL> a = findgen(3,5)
IDL> print,a
     0.00000     1.00000     2.00000
     3.00000     4.00000     5.00000
     6.00000     7.00000     8.00000
     9.00000     10.0000     11.0000
     12.0000     13.0000     14.0000
IDL> print,a[*]
     0.00000     1.00000     2.00000     3.00000     4.00000     5.00000
     6.00000     7.00000     8.00000     9.00000     10.0000     11.0000
     12.0000     13.0000     14.0000
```

One can see immediately that the first index is the one that changes most rapidly. FORTRAN behaves exactly the same way.

In C this is reversed. I quote from "The C Programming Language, Second Edition" by Brian W. Kernighan and Dennis M. Ritchie.

> ... In C, a two dimensional array is really a one-dimensional array, each of whose elements is an array. Hence subscripts are written as
>
> ```
> daytab[i][j] /* [row][col] */
> ```
>
> rather than
>
> ```
> daytab[i,j] /* WRONG */
> ```
>
> Other than this notational distinction, a two-dimensional array can be treated in much the same way as in other languages. Elements are

stored by rows, so the rightmost subscript, or column, varies fastest
as elements are accessed in storage order.

William Thompson

Subject: Re: HDF SDS array access in IDL
Posted by Dr. G. Scott Lett on Wed, 28 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

David's jetlagged answers are probably worth three of any one else's unafflicted answers. :-) By the way, David, the newsgroup was less lively while you were away.

IDL uses a storage scheme like C, not Fortran. The question of column-majority and row-majority was a source of confusion at RSI for some time, and this confusion was evident in some of their documentation. Recent editions should be better.

Scott

David Fanning wrote in message ...

(good stuff snipped)
>
> Cheers,
>
> David
>
> WARNING: My brain is still several time zones away
> from being completely engaged, so my confidence level
> in this answer is somewhat lower than my normal 50
> percent. :-)
>

Subject: Re: HDF SDS array access in IDL
Posted by thompson on Thu, 29 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

"Dr. G. Scott Lett" <slett@holisticmath.com> writes:

> Ok, ok. Let me try this a different way.

> Standard linear algebra packages written in FORTRAN, such as Linpack,

> followed the _convention_ that general matrices were stored as two
> dimensional arrays, accessed as (row, column). Because there were no matrix
> operations built into old FORTRAN, this was just a convention.

> Fortran now has a number of matrix operations built into the standard
> language. An example is the MATMUL intrinsic function, which follows the
> (row, column) convention. So, now the convention is part of the standard.
> You can multiply an array A of shape (3,4) by an array B of shape (4,3).
> MATMUL(A,B) returns an array of shape (3,3).

> IDL has a number of matrix operations and linear algebra functions. They
> follow the convention of storing matrices as two dimensional arrays in
> (column, row) order. The matrix multiplication operator in IDL can multiply
> an array A of size
> [3,4] and an array B of size [4,3]. A ## B returns an array of size [4,4].

Ah, but there are two different kinds of matrix operations in IDL. You can also use A # B to return an array of size [3,3]. If you use the # operator, then IDL behaves as having matrices in (row,column) order, and the rows and columns are stored as in Fortran. If you use the ## operator, then it behaves as having matrices in (column,row) order, and the rows and columns are stored as in C.

I don't believe the ## operator was even introduced until IDL/v4. An old IDL/v3 manual that I have laying around states that # is the matrix multiplication operator, and there is no mention of ## as an operator. I suspect that ## was added to IDL to make it more C-like. Originally, IDL was written to more closely emulate Fortran.

All the IDL procedures in use here that I'm aware of uses the # operator, and thus follow a (row,column) convention. However, since those matrices are used internally, there's no confusion to anyone who prefers to use the ## operator instead.

Actually, even if one does use ## instead of #, isn't there still a difference in the way IDL indexes elements of a matrix? If you follow the ## convention, then MATRIX(3,5) would be the third column, fifth row. However, in C, wouldn't you write this MATRIX[5][3]?

> This whole question can be academic, or at most cosmetic, unless one does
> things such as linking Fortran linear algebra codes into IDL. ...

It's also vitally important if one is passing data arrays back and forth between IDL and Fortran or C routines. If one has a 1000x300 array in IDL, it's also a 1000x300 array in Fortran, but one had better treat it as a

300x1000 array in C.

William Thompson

Subject: Re: HDF SDS array access in IDL
Posted by davidf on Thu, 29 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

William accidentally sent this article to me instead of
posting it here. I've posted it to the newsgroup with
his permission.

D.

David Fanning wrote:
>
> William Clodius (wclodius@lanl.gov) writes:
>
>> [Long, well-written article snipped.]
>
> I've read William's excellent article three times now and
> I'm *still* confused. :-(
>

I have thought about this overnight and may be able to be clearer. There are four different aspects of arrays, how they are stored in memory, how they are accessed by the language, how matrices are mapped to arrays, and how they are displayed on the screen.

1. Memory storage: Many of the aspects of how they are stored in memory are determined by the OS/processor which provides blocks of memory with optimal alignment properties. Any sensible single processor implementation of arrays will try to lay out all elements of the array in one contiguous block for efficient access. (Many languages restrict this block to the heap, but some allow stack allocation). An array oriented language, e.g., APL, IDL, Algol 68, Fortran 90/95, will also create a small descriptor defining the layout.

2. Array access: Because contiguous elements are usually more efficiently accessed than non-contiguous elements, and most languages want to make it easy for users to implement efficient code, most languages define a mapping of array elements to memory locations in the block. Historically, it is this mapping that is usually described by the terms column major versus row major. I am not certain how this convention started. Under this definition both IDL and Fortran are column major.

3. Matrices are mathematical constructs that can be efficiently implemented as arrays with special operations. The mapping of matrices onto arrays in IDL and Fortran may be different, but I have not looked in detail at this. The interpretation of this mapping depends on whether a vector by default is described as a row or column vector. While using the wrong convention with a given language would be a source of error, using the correct convention should be of comparable efficiency. Note, however, that IDL does not discuss matrices in its discussion of its array naming convention.

4. Array display: Many (most) languages (including C and Fortran) are not concerned with the display of arrays. Such a display for these languages is only defined by external library packages, and can follow the library's arbitrary conventions. A sensible package would map display lines to the most efficient access method, I.e., the last index for C, the first index for Fortran. The IDL language defines the same mapping as an efficient library for Fortran, each line mapping to a different value of the second index.

>> (Note because of the current limitations of C arrays I suspect that
>> IDL's arrays are actually implemented as C pointers with indexing
>> similar to F2C's.)
> <snip>

Forget that part, its not important. For what its worth, I suspect that the main reason that Fortran is not commonly used for image processing is that the language does not define a small integer comparable to the byte.

--

William B. Clodius Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2 FAX: (505)-667-3815
PO Box 1663, MS-C323 Group office: (505)-667-5776
Los Alamos, NM 87545 Email: wclodius@lanl.gov

Subject: Re: HDF SDS array access in IDL
Posted by Liam Gumley on Thu, 29 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

I find it helpful (both in IDL and FORTRAN) to think of 2D *arrays* as having dimensions defined as (NX,NY), where
NX is number of elements in the X direction,
NY is number of elements in the Y direction.

I find this convention to be much easier to understand than (column,row).

Matrices are a different matter of course.....

Liam E. Gumley
Space Science and Engineering Center, UW-Madison
1225 W. Dayton St., Madison WI 53706, USA
Phone (608) 265-5358, Fax (608) 262-5974
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: HDF SDS array access in IDL
Posted by Dr. G. Scott Lett on Thu, 29 Oct 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Ok, ok. Let me try this a different way.

Standard linear algebra packages written in FORTRAN, such as Linpack, followed the _convention_ that general matrices were stored as two dimensional arrays, accessed as (row, column). Because there were no matrix operations built into old FORTRAN, this was just a convention.

Fortran now has a number of matrix operations built into the standard language. An example is the MATMUL intrinsic function, which follows the (row, column) convention. So, now the convention is part of the standard. You can multiply an array A of shape (3,4) by an array B of shape (4,3). MATMUL(A,B) returns an array of shape (3,3).

IDL has a number of matrix operations and linear algebra functions. They follow the convention of storing matrices as two dimensional arrays in (column, row) order. The matrix multiplication operator in IDL can multiply an array A of size
[3,4] and an array B of size [4,3]. A ## B returns an array of size [4,4].

This whole question can be academic, or at most cosmetic, unless one does things such as linking Fortran linear algebra codes into IDL. The difference in storage/access conventions can also have a profound impact on the efficiency of certain algorithms, as David Fanning pointed out. Is an LU decomposition of a matrix faster when the matrix is stored by rows or by columns? Try it and see.
