Subject: Re: how to handle 3-D data
Posted by thompson on Thu, 26 Jan 1995 19:13:10 GMT
View Forum Message <> Reply to Message

mombasa@kronos.arc.nasa.gov (Tarang Kumar Patel) writes:

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

>> daffer@primenet.com writes:

>>> In <3fa92o$4ve@mojo.eng.umd.edu>, surinder@eng.umd.edu (Surinder P. Singh) writes:
>>>>
>>>>
>>>>
>>>> I get data out put from
>>>> FORTRAN 77, a function
>>>> F(i,j,k) .
>>>>
>>>> What is the best way to write this
>>>> data and read by idl ?
>>>> Can i increse speed of reading
>>>> and writing by unformatted data ?
>>>>
>>> Definately use unformated data. To write a four byte float in full precision as
>>> formatted data (i.e. text) takes 8 or [9 bytes] (7 digits + decimal
>>> [+ minus sign]), four of five bytes more than in binary representation.

>>> Just remember that IDL and Fortran think of arrays different ways. I forget
>>> exactly the way you say this, but I think the expression is that Fortran is row
>>> major and IDL is column major. If you don't impose some output
>>> format by means of a do loop around the output statment or an implied
>>> do loop in the format statement, then if the array that is written out has
>>> dimensions 3 X 4 X 5 than it should be read in as 5 X 4 X 3.

>> That's wrong, if one is speaking about Interactive Data Language from Research
>> Systems Incorporated. Perhaps the author was thinking about Interface
>> Definition Language--I don't know anything about that. However, both RSI's IDL
>> and FORTRAN address arrays in the same way. I don't remember the terminology
>> either, but in both IDL and Fortran the first dimension in the array is the one
>> that changes first--i.e. the element (2,1) follows immediately after the
>> element (1,1). Both behave in exactly the opposite way from C.

> Yes, the storage is same for IDL and FORTRAN i.e 1st DIMENSION varying
> fastest. However IDL arrays differ in that the 1st dimension is the COLUMN
> index and not the ROW index.

> for example

> a =indgen(2,3) ; 2 columns, 3 rows
> print,a
> 0 1
> 2 3
> 4 5
> I
> where a(0,0) = 0, a(0,1) = 2, a(0,2) = 4 and so on

> FORTRAN and C adhere to MATRIX notation. Thus in FORTRAN this would appear
> as a(1,1) = 0 a(2,1) = 2, and in C a[0][0]=1, a[1][0] = 2
> The fact that FORTRAN stores an array in ,e,ory as row major has nothing
> to do with the way a user address's the array, thats really code efficiency
> issue. Though as far as storage is concerned IDL and FORTRAN are the same i.e
> 1st DIMENSION varying the fastest, however the 1st dimension has different
> meanings

> So storage wise the elements of the array would appear in a consecutive order
> as follows
>
> Thus for the above example 0,1,2,3,4,5 are stored in that sequence
> In IDL notation a(0,0), a(1,0), a(0,1) note the 1st dimension is the COLUMN

> In FORTRAN notation a(1,1), a(2,1), a(3,1) " 1st " " " ROW
> and thus this would be 0,2,4,1,3,5
>
> In C notation a[0][0],a[0][1],a[1][0],a[1][1] " 1st " " " ROW
> and thus this would be in memory as 0,1,2,3,4,5

> So you see this really confuses matters. The situation is made worse by RSI's
> manual claiming that IDL and FORTRAN ordering are a like i.e row major.
> Well IDL is not row major in the classic notation of MATRICES.

You must be using a different version of Fortran than I'm using. I put
together the following short test program in both Fortran and IDL

Fortran:

```
      program test
      dimension a(5,3)
       open(unit=1,file='test.dat',form='formatted',readonly,status ='old')
      read (1,*) a
      read (*,*) i,j
      write (*,*) a(i,j)
      end
```

IDL:

```
      pro test
      a = fltarr(5,3)
```

```
openr,1,'test.dat'
readf,1,a
read,i,j
print,a(i-1,j-1)
end
```

test.dat

```
 1       20      300     4000    50000
 11      12      13      14      15
 111     222     333     444     555
```

Running this on a VAX/VMS computer, I get the same results from both the VMS and Unix versions no matter what indices I type in. Both IDL and any version of Fortran I've ever run into work exactly the same way. They're both the opposite of the way C treats arrays.

Bill Thompson
