Subject: a=a(*,*,[4,1,2,3,0]) efficiency
Posted by Ray on Tue, 14 Jul 1998 07:00:00 GMT
View Forum Message <> Reply to Message

I am wondering about the efficiency of the following

```
; read data from file into a which is an integer array 128x128x5
; open, ..., read a, ... close,...

; reorder data
a=a(*,*,[4,1,2,3,0])
```

Does IDL make a temporary copy of a when size of the left
hand side (a) is the same as the right hand side a(*,*,[4,1,2,3,0]) ?
If so, is there a better way to reorder my data? In my application
the last dimension of a is typically much greater than 5 (e.g. 300).

Ray Muzic
rfm2@po.cwru.edu

Subject: Re: a=a(*,*,[4,1,2,3,0]) efficiency
Posted by Ray on Thu, 16 Jul 1998 07:00:00 GMT
View Forum Message <> Reply to Message

A few suggestions were proposed in response to my original posting. One
suggestion was using a c-routine. To maintin maximum portability, I
prefer staying with straight IDL.

Overall, all suggestions that I evaluated required about the same memory
usage and cpu time. However, I devised a method that swaps individual
(*,*,i) elements (See Method 4 at the bottom of this message). This
significantly reduced memory usage but also significantly increased cpu
time.

One detail that became obvious as the result of experimentation is that
if
I say a[*,*,v] = ... (i.e. put the subscripted expression on the left of
the equal sign),
then IDL does not make a temporary variable. Consider the following

```
IDL> a=indgen(2,3,4)
IDL> v=3-indgen(4)
IDL> print,a
       0       1
       2       3
       4       5

       6       7
       8       9
      10      11

      12      13
      14      15
      16      17

      18      19
      20      21
      22      23
IDL> a=a(*,*,v)
IDL> print,a
      18      19
      20      21
      22      23

      12      13
      14      15
      16      17

       6       7
       8       9
      10      11

       0       1
       2       3
       4       5
```

This is the result is that a is rearranged as I expected.

Now consider a[*,*,v]=a. If a temporary variable is created, then the rearranged value should be the same as the above result. This is not the case

```
IDL> a=indgen(2,3,4)
IDL> v=3-indgen(4)
IDL> a[*,*,v]=a
IDL> print,a
       0       1
       2       3
       4       5

       6       7
       8       9
      10      11

       6       7
       8       9
      10      11
```

```
     0       1
     2       3
     4       5
```

Also, if I try to force IDL to use a temporary variable, I see

```
IDL> a=indgen(2,3,4)
IDL> v=3-indgen(4)
IDL> a[*,*,v]=temporary(a)
% Variable is undefined: A.
% Execution halted at:  $MAIN
```

For those interested about additional information about my application:
The array contains a 3D medical image set with the 3 dimensions
corresponding to
spatial coordinates x, y, and z. In some acquisition modes the data is
saved to file out of order with respect to the z dimension. Thus, for
volumetric visualization, I want to rearrange the data . (This, because
of my application, one of the proposed a solutions-- was accessing the
data theough pointers and rearranging the data by swapping pointers--is
not too palatable.)

```
---------------------IDL CODE FOLLOWS-------------------
pro fliptime

v=199-indgen(200)
a=indgen(128,128,200)
tic=systime(1)
a=a[*,*,v]
print,'Method 1 ',systime(1)-tic

b=indgen(128,128,200)
tic=systime(1)
b=b[*,*,v]
print,'Method 2 ',systime(1)-tic
if (total(a ne b) gt 0) then print, 'Method 2 did not yield correct
result'

b=indgen(128,128,200)
tic=systime(1)
b=b[*,*,v]
print,'Method 3 ',systime(1)-tic
if (total(a ne b) gt 0) then print, 'Method 3 did not yield correct
result'

b=indgen(128,128,200)
tic=systime(1)
b=(temporary(b))[*,*,v]
```

```
print,'Method 4 ',systime(1)-tic
if (total(a ne b) gt 0) then print, 'Method 4 did not yield correct
result'

b=indgen(128,128,200)
tic=systime(1)
; no error checking!  assumes all indicies appear exactly once in v
idx=indgen((size(b))(3))  ; used to keep track of original indicies
for i=0,(n_elements(v)-1) do begin
  w=where(idx eq v(i))
  if i ne w(0) then begin  ; swap w and i
    tmp=b[*,*,i]
    b[*,*,i]=b[*,*,w(0)]
    b[*,*,w(0)]=tmp
    tmp=idx[i]    ; record swap
    idx[i]=w(0)
    idx[w(0)]=tmp
  endif
endfor
print,'Method 5 ',systime(1)-tic
if (total(a ne b) gt 0) then print, 'Method 5 did not yield correct
result'
end
```

Ray wrote:

```
>  I am wondering about the efficiency of the following
>
>  ; read data from file into a which is an integer array 128x128x5
>  ; open, ..., read a, ... close,...
>
>  ; reorder data
>  a=a(*,*,[4,1,2,3,0])
>
>  Does IDL make a temporary copy of a when size of the left
>  hand side (a) is the same as the right hand side a(*,*,[4,1,2,3,0]) ?
>  If so, is there a better way to reorder my data?  In my application
>  the last dimension of a is typically much greater than 5 (e.g. 300).
>
>  Ray Muzic
>  rfm2@po.cwru.edu
```

Subject: Re: a=a(*,*,[4,1,2,3,0]) efficiency
Posted by davis on Fri, 17 Jul 1998 07:00:00 GMT

View Forum Message <> Reply to Message

On 15 Jul 1998 01:30:30 +0200, David Kastrup
<dak@mailhost.neuroinformatik.ruhr-uni-bochum.de>
wrote:
> temporary in the first place. How about
>
> a = (temporary(a))[*,*,[4,1,2,3,0]]

I have no knowledge of the internals of IDL, but I do not think that the use of `temporary' will help.  I am guessing that `temporary' simply does the following:

1. Push value of `a' onto the stack.  This results in the reference count to array attached to `a' being increased by 1.

2. Free `a' and undefine the variable.  This has the effect of decrementing the reference count of array attached to `a' by 1.

The net result is that the ownership of the array attached to `a' will have changed from `a' to the stack. Now consider:

```
  a = a[*,*,[4,1,2,3,0]]
```

This will probably do the following:

1. Push value of `a' onto stack. Reference count of array increased by 1.

2. Retrieve array from stack.

3. Create a new array that is a copy of the array on the stack but with elements interchanged. Push result onto stack with a reference count of 1.

4. Free array popped from stack. This reduces the reference count of array attached to `a' by 1.

5. Assign the value of array on stack to `a'.  First free the array attached to `a', reducing the reference count by 1.

6. Then remove the new array from the stack and assign it to `a'. The reference count of this array is still 1.

In both cases, at some instant, the original array and its ``interchanged'' copy will both exist. All `temporary' does is move step 5 to between steps 1 and 2.

I imagine that `temporary' is really only useful in more complex expressions, e.g., consider

```
a = (a + b) + c
```

which consists of 3 arrays `a', `b', and `c'.  During the evaluation of the RHS of this statement, 2 extra arrays will be created: (a+b) and the result (a+b)+c.  Thus at some point, 5 arrays will exist. Just prior to the assignment to `a', the temporary arrat (a+b) will be freed. Now consider:

```
a = (temporary(a) + b) + c
```

After the evaluation of (temporary(a)+b), only 3 arrays will exist: (a+b), b, and c. Then when (a+b) is added to `c', another array will be created raising the total number needed to 4.

Again, this is pure speculation and I may be totally wrong. But I cannot thing of another way to implement this.

--John