Subject: Efficient IDL programming
Posted by dean on Thu, 02 Dec 1993 23:08:18 GMT
View Forum Message <> Reply to Message

I would like to thank everbody who responded to my request for help in "extracting bits from bytes". Below is a test PRO that I made to read in the graphic file. It reads, converts, expands, enhances, and reverses my file (from (512,64) to (512,512)) in about 30 seconds.

I started with DEC2BIN.PRO posted by Bill Thompson. This worked, but it took awhile to go thru 32,768 calculations. Both Chris Chase and Dr. Marty Ryba suggested "masks" which speed things up considerably.

I just wanted to check to see if anyone would know if I can illiminated the FOR DO BEGIN loops to make this PRO a little more efficient.

Thanks again guys,

Kelly Dean

```
==========================================================
==================
pro test
        head = bytarr(56)
        premature_EOF = 1
        ON_IOERROR, SHORT_GRF
;
;       Read "in house" graphic file.
;
        OPENR, unit, 'dtopo:gms512.grf', /GET_LUN
        READU, unit, head
        chead = STRING(head)
        ck_imx = STRMID(chead,0,6)
;
;       Verify that it is an IMX graphic file before proceeding
;
        IF ( ck_imx EQ '%IMAGE' ) THEN BEGIN $
                head_lgth = STRMID(chead,28,7)
                IF ( head_lgth GT 56 ) THEN BEGIN $
                        rem_head = bytarr(head_lgth-56)
                        READU, unit, rem_head
                ENDIF
                xsize = strmid(chead,36,6)
                ysize = strmid(chead,43,6)
                imxgrf = bytarr(xsize,ysize/8)
                graphic = bytarr(xsize,ysize)
                readu, unit, imxgrf
                premature_EOF = 0
```

```
            SHORT_GRF: IF premature_EOF THEN PRINT, 'Short graphic'
            close, unit
;
;       Create mask.
;
            tmp = lindgen(8)
            mask = 2L^tmp
;
;       Expand array(xsize,ysize/8) to array(xsize,ysize).
;
            yyy = 0
            FOR y = 0,(ysize/8)-1 DO BEGIN
            xxx = 0
            FOR x = 0,xsize-1 DO BEGIN
              IF ( xxx EQ xsize ) THEN BEGIN $
                  xxx = 0
                  yyy = yyy + 1
              ENDIF
;       Perform the conversion ( Dr. Marty Ryba (MIT) suggestion).
              graphic(xxx,yyy) = (imxgrf(x,y) and mask) ne 0
              xxx = xxx + 8
            ENDFOR
            yyy = yyy + 1
            ENDFOR
;
;       Enhance value so you can see it and reverse image, then display.
;
            graphic(Where(graphic EQ 001b )) = 244b
            graphic = rotate(graphic,7) ; Transpose 270 deg, (Xo,-Yo)
            tv, graphic
        ENDIF
END
```

Subject: Re: Efficient IDL programming (use outer product)
Posted by chase on Mon, 06 Dec 1993 17:51:47 GMT

View Forum Message <> Reply to Message

>>>> > "dean" == dean <dean@phobos.cira.colostate.edu> writes:

dean>   I just wanted to check to see if anyone would know if I can
dean> illiminated the FOR DO BEGIN loops to make this PRO a little
dean> more efficient.

dean>   Thanks again guys,

dean>   Kelly Dean

[Kelly wants to efficiently decode a "packed" bit array back into an "unpacked" bit array.]

Here is something you can try.

For efficiency, what you need is a general outer product that can use any binary operator, e.g., AND. IDL's outer product "#" uses only multiplication (see comments at end). Fortunately, you can decode your bytes into the corresponding bit patterns using multiplication.

Suppose A=bytarr(512,64) contains your data. Then you can obtain your bit patterns as such:

```
mask = 2B^indgen(8)

B = bytarr(512,512)

A = transpose(A) ; Put the 64 bytes along the rows.

B(*) = (byte(mask#A(*))/128B)(*)
```

Here is a smaller example that takes an array of 16 rows of 2 bytes each and decodes the bits into a 16 by 16 byte array:

```
IDL> mask = 2^indgen(8)
IDL> z=(bindgen(2)+1)#(bindgen(16)+3)
IDL> help,z
Z               LONG      = Array(2, 16)
IDL> b=bytarr(16,16)
IDL> b(*) = (byte(mask#z(*))/128b)(*)
;; Unfortunately "#" always seems to convert the types of its operands
;; to LONG before performing the outer product (I would call this
;; unexpected behavior).  Hence the byte() conversion _before_ the
;; division.
IDL> help,b
B               BYTE      = Array(16, 16)
IDL> print,b
  0   0   0   0   0   0   1   1     0   0   0   0   0   1   1   0
  0   0   0   0   0   1   0   0     0   0   0   0   1   0   0   0
  0   0   0   0   0   1   0   1     0   0   0   0   1   0   1   0
  0   0   0   0   0   1   1   0     0   0   0   0   1   1   0   0
  0   0   0   0   0   1   1   1     0   0   0   0   1   1   1   0
  0   0   0   0   1   0   0   0     0   0   0   1   0   0   0   0
  0   0   0   0   1   0   0   1     0   0   0   1   0   0   1   0
  0   0   0   0   1   0   1   0     0   0   0   1   0   1   0   0
  0   0   0   0   1   0   1   1     0   0   0   1   0   1   1   0
  0   0   0   0   1   1   0   0     0   0   0   1   1   0   0   0
```

```
  0  0  0  0  1  1  0  1    0  0  0  1  1  0  1  0
  0  0  0  0  1  1  1  0    0  0  0  1  1  1  0  0
  0  0  0  0  1  1  1  1    0  0  0  1  1  1  1  0
  0  0  0  1  0  0  0  0    0  0  1  0  0  0  0  0
  0  0  0  1  0  0  0  1    0  0  1  0  0  0  1  0
  0  0  0  1  0  0  1  0    0  0  1  0  0  1  0  0
;; Space inserted
IDL> print,z
       3       6
       4       8
       5      10
       6      12
       7      14
       8      16
       9      18
      10      20
      11      22
      12      24
      13      26
      14      28
      15      30
      16      32
      17      34
      18      36
```

If the bits need to go in the opposite order (LSB first), just reverse the mask array.

IDL Comments/musing/wishful thinking:

I would like to see two `APL'-like operators in IDL for dealing with vectors and matrices:
1) outer products - using a given binary operator.
2) reduction - apply a scalar valued function along one dimension of an array (works like TOTAL function when using the dimension parameter). For example, return the maximum of each row of a maxtrix.

It seems that I was constantly implementing these types of operations on arrays using FOR loops, especially reduction. (The FOR loops can make execution _slow_). These can be implemented as functions using CALL_FUNCTION for reduction and EXECUTE for outer products. However, these implementations are not nearly efficient as builtin implementations could be.

A general outer product could be added to IDL by a simple addition to

the syntax similar to the MATLAB "./" operator for element by element division. For example, IDL could use the "#" as prefix notation to an operator, e.g.:

```
x #+ y
x #and y
x #< y
```

where "#bop" between two vectors means perform an outer product using `bop' instead of multiplication.

I suppose this is just syntactic sugar and not necessary.

NOTE: If anyone is interested in my OUTER and REDUCE IDL functions implementing outer products and reduction just ask and I will email them to you.

Chris

P.S. I am interested in comments/bugs with idl.el and idl-shell.el. Send them my way.

--

==========
Bldg 24-E188
The Applied Physics Laboratory
The Johns Hopkins University
(301)953-6000 x8529
chris_chase@jhuapl.edu