Subject: Timing results - matrix multiply vs. indexing
Posted by landers on Thu, 30 Jun 1994 14:33:33 GMT
View Forum Message <> Reply to Message

Hi gang,

About a week or two ago we were discussing ways to make a 2-D array from a 1-D array (which represented constant rows or columns). To summarize, the two (primary) ways to do this are:

```
    m = n_elements( array1 )

    array2 = replicate( 1.0, n ) # array1
or
    array2 = array1( lindgen( n,m ) / n ) )
```

I made the statement that the array indexing (lindgen) method should be faster because it didn't involve matrix multiplication. Wrong! I had not tested this, just assumed. Well, Dan Carr of RSI _did_ test it, and e-mailed me his results and test program. I tweaked up his test program and played with it some more.

Turns out that the replicate & matrix multiply method is generally faster. Depending on what data types you are dealing with, ahd how exactly you do it, the `#' method is somewhere between 130% and 900% as fast as the array indexing (lindgen) method.

Other interesting results of these tests:

- The indexing method is reasonably constant in speed for all data types.

- The matrix multiply opreator (at least when doing outer products like this) is evidently highly optimized for FLOATs. It is roughly twice as fast at multiplying integer types. Also, it seems that all integer types are multiplied as LONGs, based on the times as well as the fact that `#' always returns a LONG for integer multiplies.

I have some results for a single test, below. I am comparing the indexing method with various forms of the matrix multiply method. I did 4 kinds of #'s. Two of these attempt to preserve the data type of the result, and two don't care:

- like data types multiplied, then re-assigned to an array of that type, like this:

```
 result = intarr( n,m )
 result(*) = replicate( 1, n ) # array1
```

- Multiply by a replicated FLOAT, and then re-assign to proper type:
 result = intarr(n,m)
 result(*) = replicate(1.0, n) # array1

- like data types multiplied, the type of the result is whatever it is
  (will be at least a LONG):
 result = replicate(1, n) # array1

- Multiply by a replicated FLOAT, the type of the result is whatever it
  is (will be at least a FLOAT):
 result = replicate(1.0, n) # array1

Here's the results for an arbitrary case I ran in PV-WAVE on my Sparc10. Dan got similar results with IDL on an SGI Indigo. The times are in seconds (wall time) for a loop of a bunch of operations.

```
    BYTE(i)                    9.9
    BYTE(*) =       1B # BYTE      7.5
    BYTE(*) =      1.0 # BYTE      6.0
       LONG =       1B # BYTE      2.9
      FLOAT =      1.0 # BYTE      1.3
     INT(i)                   9.9
     INT(*) =        1 # INT       7.7
     INT(*) =      1.0 # INT       6.3
       LONG =        1 # INT       2.9
      FLOAT =      1.0 # INT       1.2
    LONG(i)                   10.0
    LONG(*) =       1L # LONG      7.3
    LONG(*) =      1.0 # LONG      6.4
       LONG =       1L # LONG      2.9
      FLOAT =      1.0 # LONG      1.2
   FLOAT(i)                   10.0
   FLOAT(*) =      1.0 # FLOAT     5.6
   FLOAT(*) =      1.0 # FLOAT     5.6
      FLOAT =      1.0 # FLOAT     1.2
      FLOAT =      1.0 # FLOAT     1.2
  DOUBLE(i)                  10.4
  DOUBLE(*) =      1.0D # DOUBLE    6.7
  DOUBLE(*) =       1.0 # DOUBLE    6.8
     DOUBLE =      1.0D # DOUBLE    1.6
     DOUBLE =       1.0 # DOUBLE    1.6
 COMPLEX(i)                  10.3
 COMPLEX(*) = (1.0, 0.0) # COMPLEX   7.5
 COMPLEX(*) =       1.0 # COMPLEX   7.6
    COMPLEX = (1.0, 0.0) # COMPLEX   2.3
    COMPLEX =       1.0 # COMPLEX   2.3
```

I conclude from this that:

- If you're interested in speed, you should use:
 result = replicate(1.0, n) # array1

- This method is also easier (for me) to remember and get right.

- If you're interested in preserving data type, then do the above, and
  re-assign it to another array of the same type. Unless you don't care
  about speed, and then the indexing method _might_ make for `better
  reading' code.

- You still have no option but indexing with lindgen for STRINGs and
  STRUCTs.

- Your mileage may vary. Tax, title, and license fees not included. Never
  make a purchase without first reading a detailed prospectus. Always read,
  understand, and follow the instruction manual that comes with your power
  tools. And remember, there is no more important safety rule than to wear
  these: your safety glasses.

The test code is included below.

Cheers,
;Dave

----------begin----------
```
pro test_arr

; array sizes and loop repetitions
; I'm using odd sizes to ensure maximum memory-boundary hassles.  May
; not matter, but who cares.

Size1 = 213L
Size2 = 135L
Reps =107

types = [ 'BYTE','INT','LONG','FLOAT','DOUBLE','COMPLEX' ]
type1 = [ '1B','1','1L','1.0','1.0D','(1.0, 0.0)' ]

print
print, Size2, Size1, Size2, reps, FORMAT="('Converting Array(',i3,') to Array(',i3,',',i3,')         (',i4,'
reps)')"
print
for j = 0, 5 do begin

; use random numbers to make sure we prevent any 'easy' floating
; multiplies - make sure the mantisa is filled up.  Maybe this doesn't
; matter, but whatever.
```

```
case j of
   0: arr1 = bytscl( randomu(s,Size2) )   ; byte
   1: arr1 = fix( randomu(s,Size2) * Size2 )   ; int
   2: arr1 = long( randomu(s,Size2) * Size2 )   ; long
   3: arr1 = randomu(s,Size2)     ; float
   4: arr1 = double( randomu(s,Size2) )/ randomu(s,Size2) ; double
   5: arr1 = complex( randomu(s,Size2), randomu(s,Size2) ) ; complex
endcase

; *** Test the lindgen method.

t2 = Systime(1)
for i=0, Reps do begin
 arr2 = arr1(lindgen(Size1,Size2) / Size1)
endfor
t2 = systime(1) - t2

; *** Test the replicate method - same type #, any type result.

one = arr1(0)
one(0) = 1
t3 = Systime(1)
for i=0, Reps do begin
 arr3 = replicate(one, Size1) # arr1
endfor
t3 = systime(1) - t3

; *** Test the replicate method - same type #, same type result.

one = arr1(0)
one(0) = 1
t4 = Systime(1)
for i=0, Reps do begin
 arr4 = replicate(arr1(0), Size1, Size2 )
 arr4(*) = replicate(one, Size1) # arr1
endfor
t4 = systime(1) - t4

; *** Test the replicate method - FLOAT #, any type result.

one = 1.0
t5 = Systime(1)
for i=0, Reps do begin
 arr5 = replicate(one, Size1) # arr1
endfor
t5 = systime(1) - t5
```

```
; *** Test the replicate method - FLOAT #, same type result.

one = 1.0
t6 = Systime(1)
for i=0, Reps do begin
 arr6 = replicate(arr1(0), Size1, Size2 )
 arr6(*) = replicate(one, Size1) # arr1
endfor
t6 = systime(1) - t6

; gather up array types for printing the results

s1 = size(arr1)
s3 = size(arr3)
s4 = size(arr4)
s5 = size(arr5)
s6 = size(arr6)
a1 = types( s1( s1(0)+1) -1 )
b1 = type1( s1( s1(0)+1) -1 )
a3 = types( s3( s3(0)+1) -1 )
a4 = types( s4( s4(0)+1) -1 )
a5 = types( s5( s5(0)+1) -1 )
a6 = types( s6( s6(0)+1) -1 )

; print results

formi="(a8,'(i)',T35,f6.1)"
formm="(a11,' = ',a10,' # ',A,T35,f6.1)"

print, a1, t2, FORM=formi
print, a4+'(*)',    b1,   a1, t4, FORM=formm
print, a6+'(*)', '1.0', a1, t6, FORM=formm
print, a3,          b1,   a1, t3, FORM=formm
print, a5,       '1.0', a1, t5, FORM=formm

endfor

end
```