Subject: Re: about replicate_inplace
Posted by JD Smith on Tue, 22 Jul 2003 22:25:44 GMT
View Forum Message <> Reply to Message

On Tue, 22 Jul 2003 13:37:15 -0700, Xiaoying Jin wrote:

```
> Hi, there,
>
> I am really puzzled by the IDL usage of memory. Here is an example:
>
> T = systime(1)
> b = bytarr(nreg, nreg)
> print, systime(1) - T, ' seconds.'
>
> T = systime(1)
> replicate_inplace, b, 0
> print, systime(1) - T, ' seconds.'
>
> In the example, the first part is to create an array with values 0. The
> second part is to assign each element in the array the value 0. The
> running time for the first part is 0.12sec, while the running time for
> the second part is 0.37sec.
>
> My question is: since the first part needs to allocate the memory and
> assign values to elements, it should take longer time than the second
> part. In IDL help, it says: "REPLICATE_INPLACE can be faster and use
> less memory than the IDL function REPLICATE or the IDL array notation
> for large arrays that already exist. " However, why the first part takes
> less time?
>
> If this is really the case in IDL, then it's better to allocate a new
> array than to use the current array. It's really strange for me.
>
```

If you want that array to be full of zeroes, then it looks like yes,
but I submit this for your inspection:

```
nreg=2000
T = systime(1)
b = bytarr(nreg, nreg)
b[*]=1b
print, systime(1) - T, ' seconds, BYTARR and [*]'

T = systime(1)
b = bytarr(nreg, nreg)
replicate_inplace, b, 1b
print, systime(1) - T, ' seconds, BYTARR and REPLICATE_INPLACE'
```

```
T = systime(1)
replicate_inplace, b, 1b
print, systime(1) - T, ' seconds, REPLICATE_INPLACE'

T = systime(1)
b = make_array(/BYTE,nreg,nreg,VALUE=1b)
print, systime(1) - T, ' seconds, MAKE_ARRAY'

      0.15847003 seconds, BYTARR and [*]
     0.027868986 seconds, BYTARR and REPLICATE_INPLACE
     0.016924024 seconds, REPLICATE_INPLACE
     0.021242023 seconds, MAKE_ARRAY
```

Looks like REPLICATE_INPLACE is the fastest way to populate an
existing array with 1's (or some arbitrary value, for that matter).
If you have to make the array in the first place, MAKE_ARRAY is your
best bet. Using [*] or other lhs index range notations is, as has
been discussed here before, a real performance penalty, since it
implies first creating an implicit array of indices (which in this
case just span from [0 - (2000^2-1)]). This can get you into memory
trouble too, since it amounts to temporary doubling the memory usage
for a (numerical, anyway) array.

JD