Subject: Re: about replicate_inplace
Posted by Mark Hadfield on Tue, 22 Jul 2003 21:46:05 GMT
View Forum Message <> Reply to Message

Xiaoying Jin wrote:
> I am really puzzled by the IDL usage of memory.

Me too.

> 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.

Interesting. On my system (Pentium 4 2.67 GHz with 1 GB RAM) I get
similar results, though the difference is not so marked. eg with nreg =
10000, my times are 0.22 s & 0.34 s respectively. (I have to run these
tests a couple of times to get stable results.)

However creating the array with REPLICATE takes 0.37 s, ie.
significantly longer than BYTARR and slightly less than REPLICATE_INPLACE.

> 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?

Well, based on my experience, REPLICATE_INPLACE *does* take (slightly)
less time than REPLICATE.

> Any suggestion?

For most purposes, just use the approach that gives the most
straightforward, readable code. Where performance or memory usage
becomes an issue, test different approaches.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

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

Subject: Re: about replicate_inplace
Posted by R.G. Stockwell on Tue, 22 Jul 2003 23:42:31 GMT

View Forum Message <> Reply to Message

"JD Smith" <jdsmith@as.arizona.edu> wrote in message news:pan.2003.07.22.22.25.44.654998.25152@as.arizona.edu...

```
>  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).

to populate with an arbitary value, check out

```
onebyte = 1b
T = systime(1)
b = bytarr(nreg, nreg)+onebyte
print, systime(1) - T, ' seconds'

>     0.029999971 seconds
```

compared to the above code example:

```
      0.13000000 seconds, BYTARR and [*]
     0.050000072 seconds, BYTARR and REPLICATE_INPLACE
     0.039999962 seconds, REPLICATE_INPLACE
     0.040000081 seconds, MAKE_ARRAY
```

Also, kinda interesting

```
T = systime(1)
b = b*zero+onebyte
print, systime(1) - T, ' seconds, *0 + 1'
```

gives us:

```
   0.039999962 seconds, *0 + 1
```

which is the same as replciate_inplace and make_array.

just thought it was kinda interesting......

cheers,
bob

Subject: Re: about replicate_inplace
Posted by Richard Younger on Wed, 23 Jul 2003 17:57:15 GMT
View Forum Message <> Reply to Message

Also, there's the /NOZERO keyword to all the array creation routines that
neglects to zero out the data.

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

Using /NOZERO seems to speed BYTARR and REPLICATE_INPLACE things up to the
speed of MAKE_ARRAY, but it's still not quite as good as addition.

With nreg=3000,

```
   0.37500000 seconds, BYTARR and [*]
   0.35899997 seconds, BYTARR/nozero and [*]
   0.14100003 seconds, BYTARR and REPLICATE_INPLACE
   0.12500000 seconds, BYTARR/nozero and REPLICATE_INPLACE
   0.10899997 seconds, REPLICATE_INPLACE
   0.12500000 seconds, MAKE_ARRAY
  0.093999982 seconds, onebyte+BYTARR
```

Best,
Rich

--
Richard Younger
(de-spammed return address)

Subject: Re: about replicate_inplace
Posted by xje4e on Wed, 23 Jul 2003 18:04:43 GMT
View Forum Message <> Reply to Message

"Richard Younger" <younger@ll.MIT.edu> wrote in message
news:3F1ECC7B.B3E675BA@spaMIT.edu...
>   0.37500000 seconds, BYTARR and [*]
>   0.35899997 seconds, BYTARR/nozero and [*]

```
>        0.14100003 seconds, BYTARR and REPLICATE_INPLACE
>        0.12500000 seconds, BYTARR/nozero and REPLICATE_INPLACE
>        0.10899997 seconds, REPLICATE_INPLACE
>        0.12500000 seconds, MAKE_ARRAY
>       0.093999982 seconds, onebyte+BYTARR
```

From these results, it seems that replicate_inplace is the quickest way to assign particular values to an array. But to have an array having values 0, BYTARR is the quickest way according to my test.

Best regards,

Xiaoying