Subject: Can my big for loop exploit array operations?
Posted by JTMHD on Tue, 20 Oct 2015 12:35:07 GMT
View Forum Message <> Reply to Message

Hi,

I'm writing a little post-processing procedure and need help avoiding a hefty for loop, which will typically require 1000^3 iterations.

Its purpose is to take data from a simulation, which expresses three components of a vector field at a different position s.t.

x - component is defined on cell boundary in x-direction, cell center in other directions (xb,yc,zc(
y - component is defined on cell boundary in y-direction, cell center in other directions (xc,yb,zc)
z - component is defined on cell boundary inz-direction, cell center in other directions (xc,yc,zb)

and express the field as defined in the cell center for all components (xc,yc,zc).

Rather than using interpolation, I think the best way will be to simply average the values across the cell face to head off errors down the line.

The problem is im not sure how to avoid a hefty for loop which goes through all elements taking the desired average as follows....

```
FUNCTION reformat_bcomponents_1,t

data = getdata(t,/BX,/BY,/BZ,/GRID)

bx =  DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ
E(data.z,/N_ELEMENTS))
by =  DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ
E(data.z,/N_ELEMENTS))
bz =  DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ
E(data.z,/N_ELEMENTS))

FOR iz = 0,SIZE(data.z,/N_ELEMENTS) -1 DO BEGIN
FOR iy = 0,SIZE(data.y,/N_ELEMENTS) -1 DO BEGIN
FOR ix = 0,SIZE(data.x,/N_ELEMENTS) -1 DO BEGIN

  bx[ix,iy,iz] = MEAN( [ data.bx[ix,iy,iz], data.bx[ix+1,iy,iz] ] ,/DOUBLE)
  by[ix,iy,iz] = MEAN( [ data.by[ix,iy,iz], data.by[ix,iy+1,iz] ] ,/DOUBLE)
  bz[ix,iy,iz] = MEAN( [ data.bz[ix,iy,iz], data.bz[ix,iy,iz+1] ] ,/DOUBLE)

ENDFOR
ENDFOR
ENDFOR
```

mag_str = CREATE_STRUCT('bxp',bx,'byp',by,'bzp',bz)

RETURN,mag_str
END

If anyone can point me in the direction of how to do this exploiting array operations I'd be grateful.

Thanks in advance,

Jonathan

---

Subject: Re: Can my big for loop exploit array operations?
Posted by Dick Jackson on Tue, 20 Oct 2015 15:51:01 GMT
View Forum Message <> Reply to Message

---

JTMHD wrote on 2015-10-20 5:35am:> Hi,
>
> I'm writing a little post-processing procedure and need help avoiding a hefty for loop, which will typically require 1000^3 iterations.
>
>
> Its purpose is to take data from a simulation, which expresses three components of a vector field at a different position s.t.
>
> x - component is defined on cell boundary in x-direction, cell center in other directions (xb,yc,zc(
> y - component is defined on cell boundary in y-direction, cell center in other directions (xc,yb,zc)
> z - component is defined on cell boundary inz-direction, cell center in other directions (xc,yc,zb)
>
> and express the field as defined in the cell center for all components (xc,yc,zc).
>
> Rather than using interpolation, I think the best way will be to simply average the values across the cell face to head off errors down the line.
>
> The problem is im not sure how to avoid a hefty for loop which goes through all elements taking the desired average as follows....
>
>
> FUNCTION reformat_bcomponents_1,t
>
> data = getdata(t,/BX,/BY,/BZ,/GRID)
>
> bx = DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ
E(data.z,/N_ELEMENTS))
> by = DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ
E(data.z,/N_ELEMENTS))
> bz = DBLARR(SIZE(data.x,/N_ELEMENTS),SIZE(data.y,/N_ELEMENTS),SIZ

E(data.z,/N_ELEMENTS))
>
> FOR iz = 0,SIZE(data.z,/N_ELEMENTS) -1 DO BEGIN
> FOR iy = 0,SIZE(data.y,/N_ELEMENTS) -1 DO BEGIN
> FOR ix = 0,SIZE(data.x,/N_ELEMENTS) -1 DO BEGIN
>
>    bx[ix,iy,iz] = MEAN( [ data.bx[ix,iy,iz], data.bx[ix+1,iy,iz] ] ,/DOUBLE)
>    by[ix,iy,iz] = MEAN( [ data.by[ix,iy,iz], data.by[ix,iy+1,iz] ] ,/DOUBLE)
>    bz[ix,iy,iz] = MEAN( [ data.bz[ix,iy,iz], data.bz[ix,iy,iz+1] ] ,/DOUBLE)
>
> ENDFOR
> ENDFOR
> ENDFOR
>
> mag_str = CREATE_STRUCT('bxp',bx,'byp',by,'bzp',bz)
>
> RETURN,mag_str
> END
>
> If anyone can point me in the direction of how to do this exploiting array operations I'd be grateful.
>
> Thanks in advance,
>
> Jonathan

Hi Jonathan,

I love a well-written question! I hope I've understood you correctly.

I think the middle 12 lines of code can be reduced to three (and execution time likely reduced by a factor of 10 or 20):

```
FUNCTION reformat_bcomponents_1,t

data = getdata(t,/BX,/BY,/BZ,/GRID)

bx = (data.bx[0:-2, *, *] + data.bx[1:*, *, *]) / 2
by = (data.by[*, 0:-2, *] + data.by[*, 1:*, *]) / 2
bz = (data.bz[*, *, 0:-2] + data.bz[*, *, 1:*]) / 2

mag_str = CREATE_STRUCT('bxp',bx,'byp',by,'bzp',bz)

RETURN,mag_str
END
```

If you like, this may be even faster, partly because IDL needs to do less indexing (expanding the "*"'s behind the scenes). Also, peak memory usage will be lower, for the same reason:

-----
Aside:
Making 2-element kernels for convolution in a 3-D array is tricky in IDL. REPLICATE() may indeed create an array with "1" as a trailing dimension, but when you actually look at it (or pass it into a function call), it collapses. (something to do with quantum physics, I think :-)...

```
IDL> help,replicate(0.5, [2, 1, 1])
<Expression>    FLOAT     = Array[2]
IDL> help,replicate(0.5, [1, 2, 1])
<Expression>    FLOAT     = Array[1, 2]  ; This one would cause CONVOL to fail, below
IDL> help,replicate(0.5, [1, 1, 2])
<Expression>    FLOAT     = Array[1, 1, 2]
```

I'll cast them all to three dimensions for consistency.
-----

So, the three lines above become these six:

```
bx = CONVOL(data.bx, REFORM(REPLICATE(0.5, [2, 1, 1]), [2, 1, 1]))
bx = bx[1:*, *, *]
by = CONVOL(data.by, REFORM(REPLICATE(0.5, [1, 2, 1]), [1, 2, 1]))
by = by[*, 1:*, *]
bz = CONVOL(data.bz, REFORM(REPLICATE(0.5, [1, 1, 2]), [1, 1, 2]))
bz = bz[*, *, 1:*]
```

These timings and memory metrics are typical for my experiments with arrays of size 100^3. You may need some special handling if you have 1000^3, but this work can be easily divided into slabs of, say, 100 columns/rows/planes at a time. (sorry for line wrapping):

```
IDL> help,/memory
heap memory used:   61573317, max:   61591749, gets: 466522324, frees: 466429302
                                 ^^^^^^^^
                                 ("max:" my base memory usage)

IDL> tic & bx = (data.bx[0:-2, *, *] + data.bx[1:*, *, *]) / 2 & toc & help,/memory
% Time elapsed: 0.20199990 seconds.
heap memory used:   61573305, max:   85333785, gets: 466522291, frees: 466429269
                                 ^^^^^^^^
                                 ("max:" peak memory used for (a+b)/2 method)

IDL> tic & bx2 = CONVOL(data.bx, REPLICATE(0.5, [2, 1, 1])) & bx2 = bx2[1:*,*,*] & toc &
help,/memory
% Time elapsed: 0.16899991 seconds.
heap memory used:   61573189, max:   77574061, gets: 466522197, frees: 466429175
                                 ^^^^^^^^
                                 ("max:" peak memory used for CONVOL method)
```

And yes, these give the same results: :-)

```
IDL> array_equal(bx,bx2)
   1
```

Hope this helps!

--

Cheers,
-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada
www.d-jackson.com

## Subject: Re: Can my big for loop exploit array operations?
Posted by JTMHD on Wed, 21 Oct 2015 12:21:01 GMT

View Forum Message <> Reply to Message

Hi Dick,

For the test data (256^3) I saw a fantastic speed-up using both methods you suggested - apparently as good as 35x faster for the first method (using the negative indexing) and about 60x faster with the convolution.

I was expecting a solution in the spirit of your first answer, exploiting the array indices. It hadn't occurred to me at all to use convolution and was immensely helpful to see how this could be used going forward. I'm now dealing with such huge data that I can't afford to be a lazy programmer so its good to see these different approaches (in the past I've not had to worry about optimization so much!)

Thank you for your help,

Jonathan