Subject: Please help me avoid loops and conditionals
Posted by pford on Tue, 09 Sep 2003 16:44:34 GMT
View Forum Message <> Reply to Message

Greetings,

I use IDL just frequently enough to know that my ingrained programming style is sub optimal for IDL but not frequently enough to clearly see how to improve it. What I want to do is have one object inside the other, in this case two concentric ellipses, and fill them with different values. This function will be invoked thousands of times if not more, so any small improvement here will show significant..

In the example below I see 3 items that sould slow this down, the array declaration, the loops and the conditional statements. (Note: I have not tried running this yet since I don't currently have access to the machine with IDL, so there are likely typo bugs, etc.)

```
function elp2, a, b, box_dim, vval, e_a,e_b, I_ratio
x_box = box_dim/2
box = intarr(box_dim,box_dim)
o_val = fix(vval / I_ratio)
v = fix(vval)
for i = 0, box_dim-1 do begin
    for j = 0, box_dim-1 do begin
        x = float(i - x_box)
        y = float(j - x_box)
        if( ((x/(a+e_a))^2 + (y/(b+e_b))^2) LE 1.0) then $
  if( ((x/a)^2 + (y/b)^2) LE 1.0) then box(i,j) = o_val $
  else box(i,j) = v
 endfor; j = 0, box_dim-1 do
endfor; i = 0, box_dim-1 do
return, box
end
```

So how do I go about converting this into a Boolean matrix operation that avoids all of this? Would it be faster to create a mask array such as:

```
 x = float((indgen(box_dim) ï¿½ box_dim/2) # replicate(1, box_dim))
 y = (transpose(x) / b)^2
 x = (x / a)^2
       mask  =  where((x + y) LE 1.0)
```

?

Thanks

Subject: Re: Please help me avoid loops and conditionals
Posted by Chris Lee on Wed, 10 Sep 2003 14:15:23 GMT
View Forum Message <> Reply to Message

In article <c857619b.0309090844.540303e@posting.google.com>, "Patrick Ford" <pford@bcm.tmc.edu> wrote:

> Greetings,
> I use IDL just frequently enough to know that my ingrained programming
> style is sub optimal for IDL but not frequently enough to clearly see
> how to improve it. What I want to do is have one object inside the
> other, in this case two concentric ellipses, and fill them with
> different values. This function will be invoked thousands of times if
> not more, so any small improvement here will show significant.. In the
> example below I see 3 items that sould slow this down, the array
> declaration, the loops and the conditional statements. (Note: I have not
> tried running this yet since I don't currently have access to the
> machine with IDL, so there are likely typo bugs, etc.) function elp2,
> a, b, box_dim, vval, e_a,e_b, I_ratio x_box = box_dim/2
> box = intarr(box_dim,box_dim)
> o_val = fix(vval / I_ratio)
> v = fix(vval)
> for i = 0, box_dim-1 do begin
> for j = 0, box_dim-1 do begin
> x = float(i - x_box)
> y = float(j - x_box)
> if(((x/(a+e_a))^2 + (y/(b+e_b))^2) LE 1.0) then $
> if(((x/a)^2 + (y/b)^2) LE 1.0) then box(i,j) = o_val $ else box(i,j)
> = v
> endfor; j = 0, box_dim-1 do
> endfor; i = 0, box_dim-1 do
> return, box
> end
> So how do I go about converting this into a Boolean matrix operation
> that avoids all of this? Would it be faster to create a mask array such
> as:
>
> x = float((indgen(box_dim) ï¿½ box_dim/2) # replicate(1, box_dim)) y =
> (transpose(x) / b)^2
> x = (x / a)^2
> mask = where((x + y) LE 1.0)
> ?
> Thanks

Hi,
I did answer this earlier, I think it may have been sent as an email instead though. This is the abbreviated version.

The function elp4, below, speeds up the elp2 function you give, I tested it on a few numbers, which were in the email (which I'm not sure got past my mail server, given my email address).

Roughly, I tested elp2 and elp4 for 100 iterations of a box_dim=[10,100,250,1000] case, the elp2 took about as long doing the box_dim=100 case as the elp4 did on the 1000 case (25 seconds).

There is another method where you can recycle the w1 and w2 arrays between each call to elp4, but for this to be valid, the box_dim, a, b, e_a and e_b variables must be constant between calls (so the masks remain valid). In this case, 100 iterations of box_dim=1000 took 1.7 seconds (compared to 25 seconds for elp4 and >60 (?) for elp2).

Reusing the box_array (saving on mallocs) didn't have that much effect, only 0.3 seconds on any case (which is admittedly 20% for the fastest version but 1% for the slower).

I hope Patrick got the email with the other functions in it.

Chris.

```
function elp4, a,b, box_dim,vval,e_a,e_b, I_ratio
x_box=box_dim/2
o_val=fix(vval/I_ratio)
v=fix(vval)

box=make_array(dimension=[box_dim,box_dim],value=0)

x=(findgen(box_dim)-x_box) # replicate(1,box_dim)
y=transpose(x)

w1=where((x/(a+e_a))^2 + (y/(b+e_b))^2 le 1.0,c1)
if(c1 gt 0) then w2=where(((x/a)^2 + (y/b)^2)[w1] le 1.0,c2)

if(c1 gt 0) then box[w1]=v
if(c2 gt 0) then box[w1[w2]]=o_val

return, box

end
```

Subject: Re: Please help me avoid loops and conditionals
Posted by David Fanning on Wed, 10 Sep 2003 17:06:43 GMT
View Forum Message <> Reply to Message

Patrick Ford writes:

> The second is related to the same program. I want to take a string of
> 4 contiguous bytes and convert them into a long unsigned or signed
> integer. I remember doing this in Fortran, Pascal and C with some
> variations on a theme of having variables of type byte and long point
> to the same memory location. Is there some clever way of doing the
> same in IDL? My current klugy method is based on the following:
>
> R = ary(p) + (16 * ary(p+1)) + (16^2 * ary(p+2)) + (16^3 * ary(p+3))
>
> Where ary is a byte array.

This is what the LONG and ULONG functions are used for.
The first positional parameter is the byte array you want
to read from and the second positional parameter is
the offset into the array.

```
  number = Long(ary, 16) ; skip first 16 bytes, read next four.
```

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Please help me avoid loops and conditionals
Posted by pford on Sat, 13 Sep 2003 20:58:54 GMT
View Forum Message <> Reply to Message

"Chris Lee" <cl@127.0.0.1> wrote in message
news:<20030910.151522.638422090.14392@buckley.atm.ox.ac.uk>...
> In article <c857619b.0309090844.540303e@posting.google.com>, "Patrick
> Ford" <pford@bcm.tmc.edu> wrote:
>
>
>> Greetings,
--Deleted--
>
> Hi,
> I did answer this earlier, I think it may have been sent as an email
> instead though. This is the abbreviated version.
>

> The function elp4, below, speeds up the elp2 function you give, I tested
> it on a few numbers, which were in the email (which I'm not sure got past
> my mail server, given my email address).
>
> Roughly, I tested elp2 and elp4 for 100 iterations of a
> box_dim=[10,100,250,1000] case, the elp2 took about as long doing the
> box_dim=100 case as the elp4 did on the 1000 case (25 seconds).
>
> There is another method where you can recycle the w1 and w2 arrays
> between each call to elp4, but for this to be valid, the box_dim, a, b,
> e_a and e_b variables must be constant between calls (so the masks remain
> valid). In this case, 100 iterations of box_dim=1000 took 1.7 seconds
> (compared to 25 seconds for elp4 and >60 (?) for elp2).
>
> Reusing the box_array (saving on mallocs) didn't have that much effect,
> only 0.3 seconds on any case (which is admittedly 20% for the fastest
> version but 1% for the slower).
>
> I hope Patrick got the email with the other functions in it.
>
> Chris.
> ----
>
> function elp4, a,b, box_dim,vval,e_a,e_b, l_ratio
> x_box=box_dim/2
> o_val=fix(vval/l_ratio)
> v=fix(vval)
>
> box=make_array(dimension=[box_dim,box_dim],value=0)
>
> x=(findgen(box_dim)-x_box) # replicate(1,box_dim)
> y=transpose(x)
>
> w1=where((x/(a+e_a))^2 + (y/(b+e_b))^2 le 1.0,c1)
> if(c1 gt 0) then w2=where(((x/a)^2 + (y/b)^2)[w1] le 1.0,c2)
>
> if(c1 gt 0) then box[w1]=v
> if(c2 gt 0) then box[w1[w2]]=o_val
>
> return, box
>
> end

First, thank you to all who responded.

Chris, I did not get your email.

I fiddled around with this trying to do this and I came up with the

following:

```
pro test

a_d = 6
b_d = 4
a = 15
b = 21
box_dim = 64
box = intarr(box_dim,box_dim)
bxd2 = box_dim*2
view = intarr(bxd2,bxd2)

  x = float((indgen(box_dim)  - box_dim/2) # replicate(1, box_dim))
  x1 = x
  y = (transpose(x) / b)^2
  x = (x / a)^2

  y1 = (transpose(x1) / (b+b_d))^2
  x1 = (x1 / (a_d+a))^2

  r0 = x + y
  r1 = x1 + y1

  mask0 = where(r0 LE 1.0)
  mask1 = where(r1  LE 1.0)
  mask2 = where( (r1 LE 1.0) AND NOT(r0 LE 1.0))

  box[mask1] = 255
  box[mask0] = 128
  view[0:box_dim-1, 0:box_dim-1] = box

  box[mask2] = 255
  box[mask0] = 128
  view[box_dim:bxd2-1, box_dim:bxd2-1] = box
  tv, view

  end
```

Notice no loops or if statements.(yea!) Now the next question is it more efficient to use the mask1 method where I double assign or mask2 where I expand a logical expression?
Any significant time improvement in changing NOT(r0 LE 1.0) to (r0 GT 1.0)?

I was trying to eliminate the values in mask1 that corresponded to mask0, but at this point I don't see how to do that efficiently. My attempts using the where command were a disaster.

Subject: Re: Please help me avoid loops and conditionals
Posted by Chris Lee on Fri, 19 Sep 2003 09:40:46 GMT
View Forum Message <> Reply to Message

In article <c857619b.0309131258.541e73de@posting.google.com>, "Patrick Ford" <pford@bcm.tmc.edu> wrote:

> "Chris Lee" <cl@127.0.0.1> wrote in message
>> <big snip>
...

> First, thank you to all who responded. Chris, I did not get your email.

Damn :) if you really want to make the function as fast as possible and know that you will resuse the mask, pass the mask into the function as an optional keyword, then check for the value of the keyword, if the masks are available, resuse them, this gets you a huge increase in speed but you need to be careful when changing the values of the ellipse (i.e the mask).

> I fiddled around with this trying to do this and I came up with the
> following:
> pro test
> a_d = 6
> b_d = 4
> a = 15
> b = 21
> box_dim = 64
> box = intarr(box_dim,box_dim)
> bxd2 = box_dim*2
> view = intarr(bxd2,bxd2)
>
> x = float((indgen(box_dim) - box_dim/2) # replicate(1, box_dim)) x1 =
> x
> y = (transpose(x) / b)^2
> x = (x / a)^2
>
> y1 = (transpose(x1) / (b+b_d))^2
> x1 = (x1 / (a_d+a))^2
>
> r0 = x + y
> r1 = x1 + y1
>
> mask0 = where(r0 LE 1.0)
> mask1 = where(r1 LE 1.0)
> mask2 = where((r1 LE 1.0) AND NOT(r0 LE 1.0))
>

>   box[mask1] = 255
>   box[mask0] = 128
>   view[0:box_dim-1, 0:box_dim-1] = box
>
>   box[mask2] = 255
>   box[mask0] = 128
>   view[box_dim:bxd2-1, box_dim:bxd2-1] = box tv, view
>
>   end
> Notice no loops or if statements.(yea!) Now the next question is it
> more efficient to use the mask1 method where I double assign or mask2
> where I expand a logical expression?

My guess would be that they are both the same, when I tried a similar
thing I had

mask2= where(r0[mask1] gt 1.0)
or something , you might not want to use that without checking
it... also, count your where results to make sure you have valid masks if
you do that. (I think the mask1 and mask2 would need swapping).

This only produced a 0.5 second improvement in 30 seconds, but gets
better as the outer mask gets smaller and the array size gets larger (you
only look at the previously masked data with this)

> Any significant time improvement in changing NOT(r0 LE 1.0) to (r0 GT
> 1.0)?

I doubt it, if there is an improvement it would be small, my assembly knowledge
is 'decaying' but I think there is a GT and LT in assembly so you may get a
2x speedup for this line only, but's it's O(n) so why worry :) Unrolling
that 3 layer FOR loop you wrote last year will save you much more time :)

> I was trying to eliminate the values in mask1 that corresponded to
> mask0, but at this point I don't see how to do that efficiently. My
> attempts using the where command were a disaster

The following pseudo-code snippet (from elp4) is how I did the mask
within a mask thing

w1=where(outer mask le 1.0,c1)
 if(c1 gt 0) then w2=where((inner mask of subset)[w1] le 1.0,c2)

 if(c1 gt 0) then box[w1]=v
 if(c2 gt 0) then box[w1[w2]]=o_val
                              ^^^^^^^^
                             the 'magic'

If I understand what you wrote, you want to remove anything inside the outer mask if it's inside the inner mask, David Fanning and others have written functions which can tell you which values are in one array and not another. Using the where command as you did for mask2 should work (it does, right?) but you are repeating the same logical expression twice (not exactly a bottleneck though), have a look on David's website or search the newsgroup for the "A and not B" thing when both are arrays.

You might get a speed up from that, but I'm not convinced of it, if your doing some complicated 3d polygon drawing, then yes, but if you plotting/contouring, IDL probably doesn't care if you put a 255 in a box or a 0.
