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
```