Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by MichaelT on Mon, 07 Apr 2008 16:19:34 GMT
View Forum Message <> Reply to Message

You can do it without WHERE and EXTRACT. As long as your the number of surrounding elements and your 2d Byte array is not too large.
Otherwise you may get an "unable to allocate memory: to make array" message. Basically, arrays of indexes are generated so that you can directly compare your original array with its 5x5 = 25 shifted versions. The positions at the border are not considered.

==========code begins==========

```
;Generate a byte array with random numbers, ranging from 0 to 9
nx = 300
ny = 200
b = byte(randomu(s, nx, ny) * 10)

;Define your search "radius". Your search area is 5x5 so the search
radius (sr) is 2
sr = 2

;width and height of your search area is (sr * 2 + 1)
nsr = (sr * 2 + 1)

;Generate an array containing the index deviations from your central
index (for x and y indexes)
;Example for x
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;y is simply the transposed of x

vx = (LINDGEN(nsr) - sr) # (LONARR(nsr) + 1l)
vy = Transpose(vx)

;Reform the (5x5) array so that it becomes a vector of length (25)
vx = reform(vx, nsr^2)
vy = reform(vy, nsr^2)

;Now replicate this for each element of your byte array (omitting the
sr=2 positions at each border)
vxs = replicate({a: vx}, nx - 2*sr, ny - 2*sr)
vx = vxs.a

vys = replicate({a: vy}, nx - 2*sr, ny - 2*sr)
```

```
vy = vys.a

;Now the x- and y-indexes of your byte array are generated, again
omitting the positions at the border.
;So it starts at position sr=2 and runs through position nx-1-
sr=nx-1-2
ix = (lonarr(ny - 2*sr) + 1l) # (lindgen(nx - 2*sr) + sr)
iy = (lindgen(ny - 2*sr) + sr) # (lonarr(nx - 2*sr) + 1l)

;Replicate this as often as there are elements in your 5x5 window =
nsr^2 = 25.
ixs = replicate({a: ix}, nsr^2)
iys = replicate({a: iy}, nsr^2)

;Transpose the array so that it has the same dimensions as vx and vy
ix = transpose(ixs.a)
iy = transpose(iys.a)

;Now the shifted positions are generated simply by adding the index
deviations to the index numbers
ixv = ix + vx
iyv = iy + vy

;b[ixv, iyv] eq b[ix, iy] results in an array containing 1 where a
shifted position is equal to the central position otherwise 0.
;This is summed over your 25 shifted positions: total(result, 1)
;In the end you have to substract 1 from each element as the central
position is compared to itself as well and contributes to the sum.

bn = total(b[ixv, iyv] eq b[ix, iy], 1) - 1l

;Location [0, 0] in the bn-array then corresponds to the value for [2,
2] in the b-array, due to the border problem

;Print example to check:
print, bn[0, 0]
print, b[0: 4, 0: 4]

end
==========code end=======================
```

I hope it will work in your case and should be quicker than a loop.

Michael

Subject: Re: Avoiding FOR loops (version googleplex.infinity)

You can do it without WHERE and EXTRACT. Basically, you have to generate arrays of indexes for the original array and, in your case, its 25 shifted versions. Then you can directly compare your central position values to the surrounding values. If your search area is large and/or your byte array, you may get the "unable to allocate memory: to make array" message by IDL.

==========code begin==========

```
;Generate a byte array with random numbers, ranging from 0 to 9
nx = 300
ny = 200
b = byte(randomu(s, nx, ny) * 10)

;Define your search "radius". Your search area is 5x5 so the search
radius (sr) is 2
sr = 2

;width and height of your search area is (sr * 2 + 1)
nsr = (sr * 2 + 1)

;Generate an array containing the index deviations from your central
index (for x and y indexes)
;Example for x
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;-2, -1, 0, 1, 2
;y is simply the transposed of x

vx = (LINDGEN(nsr) - sr) # (LONARR(nsr) + 1l)
vy = Transpose(vx)

;Reform the (5x5) array so that it becomes a vector of length (25)
vx = reform(vx, nsr^2)
vy = reform(vy, nsr^2)

;Now replicate this for each element of your byte array (omitting the
sr=2 positions at each border)
vxs = replicate({a: vx}, nx - 2*sr, ny - 2*sr)
vx = vxs.a

vys = replicate({a: vy}, nx - 2*sr, ny - 2*sr)
vy = vys.a
```

```
;Now the x- and y-indexes of your byte array are generated, again
omitting the positions at the border.
;So it starts at position sr=2 and runs through position nx-1-
sr=nx-1-2
ix = (lonarr(ny - 2*sr) + 1l) # (lindgen(nx - 2*sr) + sr)
iy = (lindgen(ny - 2*sr) + sr) # (lonarr(nx - 2*sr) + 1l)

;Replicate this as often as there are elements in your 5x5 window =
nsr^2 = 25.
ixs = replicate({a: ix}, nsr^2)
iys = replicate({a: iy}, nsr^2)

;Transpose the array so that it has the same dimensions as vx and vy
ix = transpose(ixs.a)
iy = transpose(iys.a)

;Now the shifted positions are generated simply by adding the index
deviations to the index numbers
ixv = ix + vx
iyv = iy + vy

;b[ixv, iyv] eq b[ix, iy] results in an array containing 1 where a
shifted position is equal to the central position otherwise 0.
;This is summed over your 25 shifted positions: total(result, 1)
;In the end you have to substract 1 from each element as the central
position is compared to itself as well and contributes to the sum.

bn = total(b[ixv, iyv] eq b[ix, iy], 1) - 1l

;Location [0, 0] in the bn-array then corresponds to the value for [2,
2] in the b-array, due to the border problem

;Print example to check:
print, bn[0, 0]
print, b[0: 4, 0: 4]

end

==========code end==========
```

I hope it is quicker than your loop.

Michael

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by MichaelT on Mon, 07 Apr 2008 16:27:11 GMT

View Forum Message <> Reply to Message

Oops, now there are line breaks where they should not be and the comments are no comments any more. So be careful when copying the code!

Michael

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by Gaurav on Tue, 08 Apr 2008 05:31:09 GMT
View Forum Message <> Reply to Message

Phew, and here I was thinking that this will turn out one of those too simple problems that elicit single statement responses.

I also received a reply from Jim Pendleton. I shall try and implement the two methods and compare their times-taken.

Thanks for saving my nails. ;-)

Gaurav

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by Gaurav on Tue, 08 Apr 2008 09:41:04 GMT
View Forum Message <> Reply to Message

Unfortunately, Michael's method fails for my array which is quite large (it being an image). Some of the images are pretty massive (at around 10k by 10k pixels).

I am still tring to make the best out of Jim's method which somehow makes use of Convolution and that is supposed to be good. I would love to hear more responses.

Thank you,

Gaurav

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by nathan12343 on Tue, 08 Apr 2008 21:33:45 GMT
View Forum Message <> Reply to Message

On Apr 8, 3:41 am, Gaurav <selfishgau...@gmail.com> wrote:
> Unfortunately, Michael's method fails for my array which is quite

> large (it being an image). Some of the images are pretty massive (at
> around 10k by 10k pixels).
>
> I am still tring to make the best out of Jim's method which somehow
> makes use of Convolution and that is supposed to be good. I would love
> to hear more responses.
>
> Thank you,
>
> Gaurav

You can still use Michael's method, you just need to break up the task into smaller chunks which will fit into memory. You could try doing it row by row in the image (with a for loop), which should still be pretty quick.

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by Tom McGlynn on Wed, 09 Apr 2008 15:18:15 GMT
View Forum Message <> Reply to Message

On Apr 8, 5:41 am, Gaurav <selfishgau...@gmail.com> wrote:
> Unfortunately, Michael's method fails for my array which is quite
> large (it being an image). Some of the images are pretty massive (at
> around 10k by 10k pixels).
>
> I am still tring to make the best out of Jim's method which somehow
> makes use of Convolution and that is supposed to be good. I would love
> to hear more responses.
>
> Thank you,
>
> Gaurav

Dear Guarav,

One thing you might keep in mind is that there is nothing wrong or necessarily inefficent using for loops. You just need to make sure that you are doing enough within each loop to amortize the cost of interpreting the statements. E.g., for the very large images you are dealing with you might try something as simple as the little (largely untested) function below. I doubt it's optimal, but it's reasonably fast for 10Kx10k images (~20 seconds for a 5x5 box). I suspect that a cleverer algorithm would be able to use the memory cache more efficiently, but with 100 million comparisons per loop iteration for a 10K image, the loop overhead is not going to be an issue!

```
   Regards,
   Tom McGlynn

function comparebox, input, boxsize

  sz = size(input)
  if sz[0] ne 2 then begin
    print,'Input not 2-D array'
    return, 0
  endif

  nx = sz[1]
  ny = sz[2]

  if nx lt boxsize or ny lt boxsize then begin
    print,'Box too large for input'
  endif

  output = intarr(nx,ny)

  offset = boxsize/2
  for i=-offset,offset do begin
    mnx = 0      > (-i)
    mxx = (nx-1) < (nx-1-i)
    for j=-offset,offset do begin
      mny = 0      > (-j)
      mxy = (ny-1) < (ny-1-j)

      if (i ne 0 or j ne 0) then begin
          output(mnx:mxx,mny:mxy) = output(mnx:mxx,mny:mxy)+ $
             (input(mnx:mxx,mny:mxy) eq input(mnx+i:mxx+i, mny+j:mxy
+j))
      endif
    endfor
  endfor
  return, output
end
```

Subject: Re: Avoiding FOR loops (version googleplex.infinity)
Posted by Gaurav on Thu, 10 Apr 2008 06:13:54 GMT
View Forum Message <> Reply to Message

Dear All,
Over the past two days I was able to think of an algorithm and it
solved my problem. And now, looking at Tom's solution above, it
appears that it is almost the same thing that I came up with.

What I did was to pad up my image with kernelSize/2 pixels all around and then compare it with temporary arrays shifted all around the kernel by turn. Wherever the shifted array was equal to the padded array, I added one in the final, output array. Run this loop for the kernel size and you are done. Now it is up to you guys to decide if it is exactly the same as Tom's or there is some difference and as to which would be faster. Here follows my code:

```
##################### Code begin ###################################
dims = size(img,/dimensions) ; getting the dimensions of input image
output_img = bytarr(dims(0), dims(1))

kernelSize = 5 ;define the kernel size (here, 5)
padSize = FLOOR(kernelSize /2.0) ;calculate the padding to generate
around the original image
paddedImg = bytarr(dims(0)+2*padSize, dims(1)+2*padSize) ;initialize
the padded image
paddedImg(padSize, padSize) = img ;define the padded image
tempImg = bytarr(dims(0)+2*padSize, dims(1)+2*padSize) ;will contain
the padded, final output

for x = -padSize, padSize do begin
 for y =  -padSize, padSize do begin
 if x eq 0 and y eq 0 then continue ;we do not want to compare with
the element itself
 ;the x, y loops shift the whole image array around the kernel and
compares the shifted image with the original
 indices = WHERE(paddedImg eq SHIFT(paddedImg, x, y)) ;
 tempImg(indices) = finalImg(indices) + 1 ;increment '1' wherever an
eqality is found
 endfor
endfor
output_img = EXTRAC(tempImg, padSize, padSize, dims(0),
dims(1)) ;extract the output image from the padded output image.
################### end of code/algorithm ######################
```

In my case it works wonderfully well and speeds things up by a factor of a few hundred times. Any further optimization would be highly appreciated.

I guess, I ought to go for a manicure for my gnawed nails now.

Cheers,
Gaurav