Subject: FOR loops removal
Posted by loebasboy on Tue, 19 Aug 2008 12:38:50 GMT
View Forum Message <> Reply to Message

Dear all,

Some weeks ago I've noticed that IDL is rather slow with FOR loops the hard way...after reading about it and not believing it, offcourse.
Last week I started removing some FOR loops in my code, apart from some embarasing ones like:

```
sum = 0
FOR i=0,max_y-1 DO BEGIN
  FOR j=0,max-1 DO BEGIN
    sum_int = sum_int + data[i,j]
  ENDFOR
ENDFOR
```

which has a straightforward solution. However I have some other FOR loops which aren't that obvious at all. Like for instance this one:

```
    FOR l = 0, n*2 DO BEGIN
      temp  = 0
      FOR i =0,max_y-1 DO BEGIN
        FOR j=0,max_x-1 DO BEGIN
          jtemp = j + l
          jtemp2 = j + n
          temp = temp + (arr[i,jtemp] * arr [i,jtemp2])
        ENDFOR
      ENDFOR
      output[l] = temp/(max_x*max_y)
    ENDFOR
```

which I could alter, not that straigthforwardly into:

```
    z = size(arr)
    arr = reform(in_arr, z[1]*z[2], /overwrite)
    endt = (max_y*max_x)-1
    FOR l = 0, n*2 DO BEGIN
      temp  = 0
      FOR i=0,endt DO temp = temp + (arr[i+l*max_y] * in_arr [i
+n*max_y])
      output[l] = temp/(max_x*max_y)
    ENDFOR
    in_arr = reform(in_arr, z[1],z[2], /overwrite)
```

where 1 FOR loop is removed. However there is hardly any time profit at all. It is even so that the following code is faster than both,

which is a very straightforward alteration of the first loop:

```
      FOR l = 0, n*2 DO BEGIN
        temp  = 0
        FOR i =0,max_y-1 DO FOR j=0,max_x-1 DO temp = temp + (arr[i,j
j + l] * arr [i, j + n])
        output[l] = temp/(max_x*max_y)
      ENDFOR.
```

With the following variables set and the for loops repeated with another FOR loop of i= 0,10000 (to see a time difference, and in the full program it is repeated about that many times too, but with even larger arrays):

```
  n = 8
  max_x = 5
  max_y = 5
  output = fltarr(2*n+1)
  arr = findgen(interval_y, region) +1
```

I have for the first for loop : 1.6279998 s
the second : 1.6060002 s
the third : 1.2749999 s

I measured the times with SYSTIME, /SECONDS command.

(the full program takes 22,5 h for a standard image, with the alterations I have allready done, I've came up with 18.1 h, which is still 17 h to go to make it workable, i've used the last loop in the above example so far...)

Can anybody tell me why removing one loop doesn't help in this case or what i'm doing wrong?

thnx
Stijn Van der Linden

Subject: Re: FOR loops removal
Posted by David Fanning on Tue, 19 Aug 2008 20:14:46 GMT
View Forum Message <> Reply to Message

Chris writes:

> IDL for loops are slow because the part of IDL
> that interprets your file a fast but crotchety old man who can't hear
> you very well and may not even really be listening. Any time you tell
> him to do something, it takes him a while to interpret what you just

> said - much longer than other, less crotchety men. Once he figures out
> what's going on, however, he's plenty fast (especially if you tell him
> to do something that he was already designed to do, for which he has
> been well optimized). Good vectorization, then, minimizes the number
> of instructions (e.g. iterations in a loop) while maximizing the
> amount of work to do with each instruction.

I think this is a transparent attempt to characterize certain
persons on this newsgroup and I deeply resent it. :-(

Cheers,

David

P.S. Let's just say I would type a smiley face, but my
feelings have been cut too close to the bone. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: FOR loops removal
Posted by loebasboy on Wed, 20 Aug 2008 12:36:31 GMT
View Forum Message <> Reply to Message

On 20 aug, 13:50, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Aug 19, 9:43 am, Wox <nom...@hotmail.com> wrote:
>
>
>
>
>
>> On Tue, 19 Aug 2008 05:38:50 -0700 (PDT), loebasboy
>
>> <stijn....@gmail.com> wrote:
>>> FOR l = 0, n*2 DO BEGIN
>>> temp = 0
>>> FOR i =0,max_y-1 DO BEGIN
>>> FOR j=0,max_x-1 DO BEGIN
>>> jtemp = j + l
>>> jtemp2 = j + n
>>> temp = temp + (arr[i,jtemp] * arr [i,jtemp2])
>>> ENDFOR
>>> ENDFOR
>>> output[l] = temp/(max_x*max_y)

>>>   ENDFOR
>
>> The code below is a start. Does this processing have a name? It feels
>> familiar somehow. Btw, in IDL the first index of an array is the
>> column and the second is the row. So in your case y are the columns
>> and x are the rows. No problem with that off course, just check
>> whether this is how you intended it.
>
>> n = 8
>> max_x = 5
>> max_y = 5
>> output = fltarr(2*n+1)
>> arr = findgen(max_y, 2*n+max_x) +1
>
>> arr2=arr[0:max_y-1,n:max_x-1+n]
>> FOR l = 0, 2*n DO $
>>   output[l] = total(arr[0:max_y-1,l:max_x-1+l]*arr2)
>> output/=max_x*max_y
>
> Following on that last version, I think we can *completely* get rid of
> the loop... though at the expense (as usual) of memory:
>
> n = 8
> max_x = 5
> max_y = 5
> arr = findgen(max_y, 2*n+max_x) +1
>
> max_area = max_x*max_y
> output = total(arr[rebin(lindgen(max_area),max_area,2*n+1) +
>   max_y*rebin(reform(lindgen(2*n+1),1,2*n+1),max_area,2*n+1)] *
>   rebin((arr[*,n:max_x-1+n])[*], max_area,2*n+1), 1) / max_area
>
> Whether that's actually faster will depend on how big max_x, max_y and
> n are, of course... it ends up internally storing a couple of
> max_x*max_y*(2*n+1) arrays, so if that is going to take you into swap
> then you're best off sticking with Wox's version. If that stays in
> physical memory, though, I bet this will win.
>
> -Jeremy.- Tekst uit oorspronkelijk bericht niet weergeven -
>
> - Tekst uit oorspronkelijk bericht weergeven -

Great code, that is another hour of time profit again, and it works.
So vectorization comes down to instead of repeating an action per
element in matrix, putting all elements on the right spot in a matrix
and doing the action on the matrix, right? The only problem is then,
finding out when this is possible, or when it isn't ;).

thank you all, for all the great info allready!

Subject: Re: FOR loops removal
Posted by Jeremy Bailin on Wed, 20 Aug 2008 14:07:33 GMT
View Forum Message <> Reply to Message

> So vectorization comes down to instead of repeating an action per
> element in matrix, putting all elements on the right spot in a matrix
> and doing the action on the matrix, right?

Yup, that sums it up pretty well!

-Jeremy.

Subject: Re: FOR loops removal
Posted by loebasboy on Thu, 21 Aug 2008 07:59:33 GMT
View Forum Message <> Reply to Message

On Aug 20, 4:07 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>> So vectorization comes down to instead of repeating an action per
>> element in matrix, putting all elements on the right spot in a matrix
>> and doing the action on the matrix, right?
>
> Yup, that sums it up pretty well!
>
> -Jeremy.

So I tested the new finetuned program on the standard image and instead of a calculated 15 hour time profit it has become almost 20,5 hour time profit. The program takes now 2.15 hours instead of 22.5 hours. That is a major improvement (< 10x), so thanks for all the info allready. So I started out with even more improvements, I haven't found any vectorisation possibilities yet though. I tried to fasten the following code:

```
n = 20
size = 2*n+1
array = randomn(seed, size)
array[0] = 0
array[5] = 0
array[10] = 0
array[20] = 0
array[size-2] = 0
array[size-1] = 0
```

```
          FOR x = 1, size-2 DO BEGIN
            IF (array[x] EQ 0) THEN BEGIN
              IF ((array[x-1] LE 2) AND (array[x+1] LE 2)) THEN
BEGIN
                array[x] = 2
              ENDIF ELSE BEGIN
                IF ((array[x-1] GE 2) AND (array[x+1] GE 2)) THEN
BEGIN
                  array[x] = -2
                ENDIF
              ENDELSE
            ENDIF
          ENDFOR
```

So I figured that if i use the WHERE function to find where the array equals 0, and then use a FOR loop that only goes trough the indices that the WHERE function has found. So If you consider the WHERE function to be much faster than the FOR loop, you could expect that the second FOR loop would be faster or equally fast than the first FOR loop. The code for the second FOR loop goes like this (some other extra IF functions are needed for special cases like a zero as a first element, last element or no zero at all):

```
       zeroindex = where (array EQ 0,m)
       IF (zeroindex[0] NE -1) THEN BEGIN
         IF (zeroindex[0] EQ 0) THEN k = 1 ELSE k = 0
         IF (zeroindex[m-1] EQ size-1) THEN l = 2 ELSE l = 1
         FOR i= k, m-l DO BEGIN
          IF ((array[zeroindex[i]-1] LE 2) AND (array[zeroindex[i]+1]
LE 2)) THEN BEGIN
            array[zeroindex[i]] = 2
          ENDIF ELSE BEGIN
            IF ((array[zeroindex[i]-1] GE 2) AND (array[zeroindex[i]
+1] GE 2)) THEN BEGIN
              array[zeroindex[i]] = -2
            ENDIF
          ENDELSE
        ENDFOR
```

you could hear me coming from afar ofcourse ;) . The second FOR loop doesn't go faster, at all, with the variables set as above and the two loops repeated for 50000 times. The first loop takes 0.304 s and the second one 0.337 s. Only if the n-value is made larger than 25 the second loop starts to go faster. I checked out profiler to check if the WHERE function makes up for this slowing down this bit of programming and ofcourse it does, the difference in time is 0.033s while the WHERE function takes up 0.066s. So the second loop goes faster but the use of the WHERE function slows the whole program down.

This is some nice checking out ofcourse but it doesn't help me getting
any further. Is there a faster alternative of the WHERE function? Or
did I reach the limit in finetuning here? :)

Subject: Re: FOR loops removal
Posted by Jeremy Bailin on Thu, 21 Aug 2008 13:39:18 GMT
View Forum Message <> Reply to Message

On Aug 21, 3:59 am, loebasboy <stijn....@gmail.com> wrote:
> So I tested the new finetuned program on the standard image and
> instead of a calculated 15 hour time profit it has become almost 20,5
> hour time profit. The program takes now 2.15 hours instead of 22.5
> hours. That is a major improvement (< 10x), so thanks for all the info
> allready. So I started out with even more improvements, I haven't
> found any vectorisation possibilities yet though. I tried to fasten
> the following code:
>
> n = 20
> size = 2*n+1
> array = randomn(seed, size)
> array[0] = 0
> array[5] = 0
> array[10] = 0
> array[20] = 0
> array[size-2] = 0
> array[size-1] = 0
>
> FOR x = 1, size-2 DO BEGIN
> IF (array[x] EQ 0) THEN BEGIN
> IF ((array[x-1] LE 2) AND (array[x+1] LE 2)) THEN
> BEGIN
> array[x] = 2
> ENDIF ELSE BEGIN
> IF ((array[x-1] GE 2) AND (array[x+1] GE 2)) THEN
> BEGIN
> array[x] = -2
> ENDIF
> ENDELSE
> ENDIF
> ENDFOR
>
> So I figured that if i use the WHERE function to find where the array
> equals 0, and then use a FOR loop that only goes trough the indices
> that the WHERE function has found. So If you consider the WHERE
> function to be much faster than the FOR loop, you could expect that
> the second FOR loop would be faster or equally fast than the first FOR
> loop. The code for the second FOR loop goes like this (some other

```
>  extra IF functions are needed for special cases like a zero as a first
>  element, last element or no zero at all):
>
>        zeroindex = where (array EQ 0,m)
>        IF (zeroindex[0] NE -1) THEN BEGIN
>          IF (zeroindex[0] EQ 0) THEN k = 1 ELSE k = 0
>          IF (zeroindex[m-1] EQ size-1) THEN l = 2 ELSE l = 1
>          FOR i= k, m-l DO BEGIN
>           IF ((array[zeroindex[i]-1] LE 2) AND (array[zeroindex[i]+1]
>  LE 2)) THEN BEGIN
>             array[zeroindex[i]] = 2
>           ENDIF ELSE BEGIN
>             IF ((array[zeroindex[i]-1] GE 2) AND (array[zeroindex[i]
>  +1] GE 2)) THEN BEGIN
>               array[zeroindex[i]] = -2
>             ENDIF
>           ENDELSE
>          ENDFOR
>
>  you could hear me coming from afar ofcourse ;) . The second FOR loop
>  doesn't go faster, at all, with the variables set as above and the two
>  loops repeated for 50000 times. The first loop takes 0.304 s and the
>  second one 0.337 s. Only if the n-value is made larger than 25 the
>  second loop starts to go faster. I checked out profiler to check if
>  the WHERE function makes up for this slowing down this bit of
>  programming and ofcourse it does, the difference in time is 0.033s
>  while the WHERE function takes up 0.066s. So the second loop goes
>  faster but the use of the WHERE function slows the whole program down.
>  This is some nice checking out ofcourse but it doesn't help me getting
>  any further. Is there a faster alternative of the WHERE function? Or
>  did I reach the limit in finetuning here? :)

The solution, of course, is to also replace the inner FOR loops with
WHEREs. :-)=

zeroindex = where(array[1:size-2] eq 0, nzero)
if nzero gt 0 then begin
  smallneighbours = where(array[zeroindex-1] le 2 and array[zeroindex
+1] le 2, nsmall)
  if nsmall gt 0 then array[zeroindex[smallneighbours]] = 2
  bigneighbours = where(array[zeroindex-1] le 2 and array[zeroindex+1]
le 2, nbig)
  if nbig gt 0 then array[zeroindex[bigneighbours]] = -2
endif

-Jeremy.
```

Subject: Re: FOR loops removal
Posted by loebasboy on Thu, 21 Aug 2008 14:27:49 GMT
View Forum Message <> Reply to Message

On Aug 21, 3:39 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Aug 21, 3:59 am, loebasboy <stijn....@gmail.com> wrote:
>
>
>
>
>
>
>> So I tested the new finetuned program on the standard image and
>> instead of a calculated 15 hour time profit it has become almost 20,5
>> hour time profit. The program takes now 2.15 hours instead of 22.5
>> hours. That is a major improvement (< 10x), so thanks for all the info
>> allready. So I started out with even more improvements, I haven't
>> found any vectorisation possibilities yet though. I tried to fasten
>> the following code:
>
>> n = 20
>> size = 2*n+1
>> array = randomn(seed, size)
>> array[0] = 0
>> array[5] = 0
>> array[10] = 0
>> array[20] = 0
>> array[size-2] = 0
>> array[size-1] = 0
>
>> FOR x = 1, size-2 DO BEGIN
>> IF (array[x] EQ 0) THEN BEGIN
>> IF ((array[x-1] LE 2) AND (array[x+1] LE 2)) THEN
>> BEGIN
>> array[x] = 2
>> ENDIF ELSE BEGIN
>> IF ((array[x-1] GE 2) AND (array[x+1] GE 2)) THEN
>> BEGIN
>> array[x] = -2
>> ENDIF
>> ENDELSE
>> ENDIF
>> ENDFOR
>
>> So I figured that if i use the WHERE function to find where the array
>> equals 0, and then use a FOR loop that only goes trough the indices
>> that the WHERE function has found. So If you consider the WHERE
>> function to be much faster than the FOR loop, you could expect that
>> the second FOR loop would be faster or equally fast than the first FOR
>> loop. The code for the second FOR loop goes like this (some other

```
>>  extra IF functions are needed for special cases like a zero as a first
>>  element, last element or no zero at all):
>
>>        zeroindex = where (array EQ 0,m)
>>        IF (zeroindex[0] NE -1) THEN BEGIN
>>          IF (zeroindex[0] EQ 0) THEN k = 1 ELSE k = 0
>>          IF (zeroindex[m-1] EQ size-1) THEN l = 2 ELSE l = 1
>>          FOR i= k, m-l DO BEGIN
>>           IF ((array[zeroindex[i]-1] LE 2) AND (array[zeroindex[i]+1]
>>  LE 2)) THEN BEGIN
>>             array[zeroindex[i]] = 2
>>           ENDIF ELSE BEGIN
>>             IF ((array[zeroindex[i]-1] GE 2) AND (array[zeroindex[i]
>>  +1] GE 2)) THEN BEGIN
>>               array[zeroindex[i]] = -2
>>             ENDIF
>>           ENDELSE
>>          ENDFOR
>
>>  you could hear me coming from afar ofcourse ;) . The second FOR loop
>>  doesn't go faster, at all, with the variables set as above and the two
>>  loops repeated for 50000 times. The first loop takes 0.304 s and the
>>  second one 0.337 s. Only if the n-value is made larger than 25 the
>>  second loop starts to go faster. I checked out profiler to check if
>>  the WHERE function makes up for this slowing down this bit of
>>  programming and ofcourse it does, the difference in time is 0.033s
>>  while the WHERE function takes up 0.066s. So the second loop goes
>>  faster but the use of the WHERE function slows the whole program down.
>>  This is some nice checking out ofcourse but it doesn't help me getting
>>  any further. Is there a faster alternative of the WHERE function? Or
>>  did I reach the limit in finetuning here? :)
>
>  The solution, of course, is to also replace the inner FOR loops with
>  WHEREs. :-)=
>
>  zeroindex = where(array[1:size-2] eq 0, nzero)
>  if nzero gt 0 then begin
>    smallneighbours = where(array[zeroindex-1] le 2 and array[zeroindex
>  +1] le 2, nsmall)
>    if nsmall gt 0 then array[zeroindex[smallneighbours]] = 2
>    bigneighbours = where(array[zeroindex-1] le 2 and array[zeroindex+1]
>  le 2, nbig)
>    if nbig gt 0 then array[zeroindex[bigneighbours]] = -2
>  endif
>
>  -Jeremy.- Hide quoted text -
>
>  - Show quoted text -
```

Good code, there was a +1 needed in the zeroindex declaration though.
It doesn't go any faster also, too bad, I guess that the use of the
WHERE function doesn't speed up. But thank you for the suggestion !

Subject: Re: FOR loops removal
Posted by Jeremy Bailin on Thu, 21 Aug 2008 15:40:25 GMT
View Forum Message <> Reply to Message

> Good code, there was a +1 needed in the zeroindex declaration though.

Ah yes, of course. Serves me right for not testing it before posting
it! ;-)

> It doesn't go any faster also, too bad, I guess that the use of the
> WHERE function doesn't speed up. But thank you for the suggestion !

Hmmm, that's too bad. It's possible that going down to two WHEREs over
the full array will be faster than using the first WHERE to thin the
array down. It'll depend on what fraction of the array contains zeros
- if there are lots of zeros, then the first WHERE doesn't help you
that much, whereas if there aren't many then it will help a lot.

Is this part of the code really one of the biggest remaining
bottlenecks? I doubt you'll be able to shave much more off of it.

-Jeremy.

Subject: Re: FOR loops removal
Posted by Chris[6] on Thu, 21 Aug 2008 18:34:06 GMT
View Forum Message <> Reply to Message

What about something like this?

```
s= 2*(array le 2)-1
array = (array ne 0)*array + (array eq 0)*(shift(s,1)+shift(s,-1))
```

You'll have to manually fix the endpoints, but the rest should be
good. I tested this and the first loop on an array of 400,000
elements. The first loop rand in .54 seconds, while the second ran in .
054s.

Also, you should be a little careful when testing whether array equals
zero. If the array isn't an integer type (int, byte, long, etc), then
its possible to have numbers you think ought to be zero but, because

of finite precision arithmetic, have very small nonzero values. If you
suspect that this is happening, you can try a test like abs(array) le
eps, where eps is something like 10^-5 or something big enough to
cover roundoff error but small enough not to treat nonzero data you
care about as zero

chris

Subject: Re: FOR loops removal
Posted by loebasboy on Fri, 22 Aug 2008 09:08:42 GMT
View Forum Message <> Reply to Message

On Aug 21, 8:34 pm, Chris <beaum...@ifa.hawaii.edu> wrote:
> What about something like this?
>
> s= 2*(array le 2)-1
> array = (array ne 0)*array + (array eq 0)*(shift(s,1)+shift(s,-1))
>
> You'll have to manually fix the endpoints, but the rest should be
> good. I tested this and the first loop on an array of 400,000
> elements. The first loop rand in .54 seconds, while the second ran in .
> 054s.
>
> Also, you should be a little careful when testing whether array equals
> zero. If the array isn't an integer type (int, byte, long, etc), then
> its possible to have numbers you think ought to be zero but, because
> of finite precision arithmetic, have very small nonzero values. If you
> suspect that this is happening, you can try a test like abs(array) le
> eps, where eps is something like 10^-5 or something big enough to
> cover roundoff error but small enough not to treat nonzero data you
> care about as zero
>
> chris

Hello, thank you both for the input. Chris, yours doesn't go faster
also despite it is really elegant (and pointing me to the shift
function, which I didnt know exist and can come in handy). The reason
for the lack of speed profit is that I haven't a large array there
that needs to be processed, instead I have many small array that need
to be processed (the whole of my program exist of one BIG for loop)
and there are relatively a lot of zeros in the processed array (which
is why the WHERE function solution doesn't work either)

To answer Jeremy's question, the real bottleneck was the former FOR
loop removal problem, which was part of a function that is repeated
1359520 times for a test image and taking up 39.1 s now and 248.2 s

before. The FOR loop I was asking about now, is a part of a function that is also repeated 1359520 times and takes up 29 s now. The former function cannot be made any faster than it is now (thanks to all of you ;)). So I started working on the simplest FOR loop in the latter function but it didn't work out.

Subject: Re: FOR loops removal
Posted by ben.bighair on Fri, 22 Aug 2008 10:50:36 GMT
View Forum Message <> Reply to Message

On Aug 22, 5:08 am, loebasboy <stijn....@gmail.com> wrote:
> On Aug 21, 8:34 pm, Chris <beaum...@ifa.hawaii.edu> wrote:
>
>
>
>> What about something like this?
>
>> s= 2*(array le 2)-1
>> array = (array ne 0)*array + (array eq 0)*(shift(s,1)+shift(s,-1))
>
>> You'll have to manually fix the endpoints, but the rest should be
>> good. I tested this and the first loop on an array of 400,000
>> elements. The first loop rand in .54 seconds, while the second ran in .
>> 054s.
>
>> Also, you should be a little careful when testing whether array equals
>> zero. If the array isn't an integer type (int, byte, long, etc), then
>> its possible to have numbers you think ought to be zero but, because
>> of finite precision arithmetic, have very small nonzero values. If you
>> suspect that this is happening, you can try a test like abs(array) le
>> eps, where eps is something like 10^-5 or something big enough to
>> cover roundoff error but small enough not to treat nonzero data you
>> care about as zero
>
>> chris
>
> Hello, thank you both for the input. Chris, yours doesn't go faster
> also despite it is really elegant (and pointing me to the shift
> function, which I didnt know exist and can come in handy). The reason
> for the lack of speed profit is that I haven't a large array there
> that needs to be processed, instead I have many small array that need
> to be processed (the whole of my program exist of one BIG for loop)
> and there are relatively a lot of zeros in the processed array (which
> is why the WHERE function solution doesn't work either)
>

Hi,

Given your response to Chris I think the teeeny suggestion I have to share is a moot point. But I'll put it out there because it can be handy. If I understand Chris' code snippet correctly, there can be a spped up by using something like this...

```
mask1 = array ne 0
mask0 = ~mask1
array = mask1*array + mask0*(shift(s,1)+shift(s,-1))
```

It probably is a slim gain but can add up to a lot if you are looping. In my simple minded test below I see better than 2x gain usually.

```
a = RANDOMN(seed, 2000, 2000) > 0.0
b = A GT 0

PRINT, "~B test"
t0 = SYSTIME(/sec)
c = ~B
print, systime(/sec) - t0

PRINT, "B EQ 0 test"
t0 = SYSTIME(/sec)
c = b EQ 0
print, systime(/sec) - t0

IDL> @not_test
~B test
    0.056261063
B EQ 0 test
     0.13446379
```

Cheers,
ben

Subject: Re: FOR loops removal
Posted by Chris[6] on Fri, 22 Aug 2008 18:19:36 GMT
View Forum Message <> Reply to Message

Are you looping 139520 times because you have that many images you are reading from disk? If so, you probably aren't going to be able to do any better. I/O is unavoidably slow. If not, what are you doing that requires so my loops? How quickly do you process one image?

As a general guideline, most image processing on a single image (say a few megapixels) should be on the order of a few seconds once vectorized well. If you have several hundred thousand images, one thing to consider is to merge them into larger files (by making data cubes). This would reduce the number of hard-disk seeks during IO.

Subject: Re: FOR loops removal
Posted by loebasboy on Mon, 25 Aug 2008 08:31:47 GMT
View Forum Message <> Reply to Message

On Aug 22, 8:19 pm, Chris <beaum...@ifa.hawaii.edu> wrote:
> Are you looping 139520 times because you have that many images you are
> reading from disk? If so, you probably aren't going to be able to do
> any better. I/O is unavoidably slow. If not, what are you doing that
> requires so my loops? How quickly do you process one image?
>
> As a general guideline, most image processing on a single image (say a
> few megapixels) should be on the order of a few seconds once
> vectorized well. If you have several hundred thousand images, one
> thing to consider is to merge them into larger files (by making data
> cubes). This would reduce the number of hard-disk seeks during IO.

There is only 1 image that is processed. 1 loop works on a part of the image, that is, several columns and all the rows. When 1 loop is done, all columns are moved up one column spot and at the end of the part a new column is added (from the full image). The first column is then removed from the part. And so the the full image is processed. 600 columns means 600 loops minus the width of the part two times. Also in every loop the process is repeated 4 times.
