
Subject: Re: Array Concatenation Optimization

Posted by [Peter Mason](#) on Tue, 08 Feb 2005 22:18:24 GMT

[View Forum Message](#) <> [Reply to Message](#)

Ken Mankoff wrote:

```
> I've read JD's tutorial
> [http://www.dfanning.com/tips/array_concatenation.html] a few times,
> but cannot get my array juggling as fast as I would like it...
>
> I'm converting an 8 bit image to 24 bit RGB. Why? Well, long story,
> but I am stuck in the Z buffer and need to make nice anti-aliased
> color images. So everything is 3x as large&thick, converted to RGB,
> and then rebined...
>
> The conversion to RGB is currently one of the bottlenecks in my
> code, and I would like to speed it up. I am using this function:
>
> function toRGB, r,g,b, image
> s = size(image,/dim)
> image_rgb = bytarr( 3, s[0], s[1] )
> image_rgb[0, *, *] = r[image]
> image_rgb[1, *, *] = g[image]
> image_rgb[2, *, *] = b[image]
> return, image_rgb
> end
>
> I can cut the execution time in half if I change the entire function
> to this one line:
>
> return, [[[r[image]]],[[g[image]]],[[b[image]]]]]
>
> But now it is [n,m,3], and the WRITE_PNG procedure needs it to be
> [3,n,m], and wrapping a TRANSPOSE() around that 1 line makes it go
> from 2x as fast to 7x as slow as the original function.
>
> I don't get any improvement if I change the * to explicit ranges,
> although I remember reading that this should make array insertions
> faster....
>
> image_rgb[0, 0:s[0]-1, 0:s[1]-1] = r[image]
>
> Another trick I have read about is to use the TEMPORARY() function,
> but I don't think it is applicable in this case.
>
> Any other suggestions?
>
> Thanks,
>
```

> Ken Mankoff
> <http://edgcm.org/>
> <http://spacebit.dyndns.org/>

Interesting. What platform are you using? I ran a little test program (included below) on an "un-hyperthreaded" P4 Windows2000 laptop. I got the following times for N=2048 (a square, 2048*2048 image):

Method 1 (traditional): 0.30 (IDL6.1), 0.30 (IDL6.0), 0.74 (IDL5.5)
Method 2 (transpose): 0.26 (IDL6.1), 0.27 (IDL6.0), 0.43 (IDL5.5)
Method 3 (insertion): 0.57 (IDL6.1), 0.57 (IDL6.0), 0.76 (IDL5.5)
Method 4 (steam power): 6.21 (IDL6.1), 6.40 (IDL6.0), 6.64 (IDL5.5)

I must admit that I was surprised by these results. I would have thought that method 3 would be much faster than method 1. So much for the "never use an * subscript on the LHS of an assignment if you can help it" rule that I cherish. I can't explain it. Evidently RSI has put some work into this since IDL5.5, though.

Curiously, the transpose method performed the best on my platform. I included method 4 because it *might* perform better than the others if you were to code something like it in C.

Cheers
Peter Mason

```
;Test program
pro qad_rgbload
;
n=2048L
r=bindgen(256)
g=r+64b
b=g+64b
img=bytsc1(dist(n,n))
rgb=bytarr(3,n,n)
;
; Method 1
t0=systime(1)
rgb[0,*]=r[img]
rgb[1,*]=g[img]
rgb[2,*]=b[img]
print,systime(1)-t0
;
; Method 2
t0=systime(1)
rgb[0]=transpose( [[[r[img]]],[[g[img]]],[[b[img]]]], [2,0,1] )
print,systime(1)-t0
;
```

```

; Method 3
img=reform(img,1,n,n,/overw)
t0=systime(1)
rgb[0,0,0]=r[img]
rgb[1,0,0]=g[img]
rgb[2,0,0]=b[img]
print,systime(1)-t0
;
; Method 4
img=reform(img,n,n,/overw)
t0=systime(1)
n1=n-1L
for i=0L,n1 do begin
  for j=0L,n1 do begin
    k=img[i,j]
    rgb[0,i,j]=r[k]
    rgb[1,i,j]=g[k]
    rgb[2,i,j]=b[k]
  endfor
endfor
print,systime(1)-t0
return
end

```

Subject: Re: Array Concatenation Optimization
 Posted by [KM](#) on Tue, 08 Feb 2005 23:18:51 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, 8 Feb 2005, Peter Mason wrote:

```

> Interesting. What platform are you using? I ran a little test
> program (included below) on an "un-hyperthreaded" P4 Windows2000
> laptop. I got the following times for N=2048 (a square, 2048*2048
> image):
> Method 1 (traditional): 0.30 (IDL6.1), 0.30 (IDL6.0), 0.74 (IDL5.5)
> Method 2 (transpose): 0.26 (IDL6.1), 0.27 (IDL6.0), 0.43 (IDL5.5)
> Method 3 (insertion): 0.57 (IDL6.1), 0.57 (IDL6.0), 0.76 (IDL5.5)
> Method 4 (steam power): 6.21 (IDL6.1), 6.40 (IDL6.0), 6.64 (IDL5.5)

```

I have two OS X boxes

- 1) 1 GHz G4 IDL 5.6 (slower than any of yours)
- 2) 2.5 GHz dual G5 IDL 6.0.3 (faster than all of yours)

My image size varies (depends on what the user drags the window to), but it is up to 4800x2400. It gets larger, but at that point the user can deal with a lag while I generate the image. Sometimes it is as small as 1800x1800.

Computer 1 (about 1/2 the speed if your IDL5.5):

Method 1: 0.78

Method 2: 0.96

Method 3: 1.28

Method 4: 18.217

Method 5: 0.34 (fast!)

Computer 2 (a bit faster than your IDL6.1):

Method 1: 0.25

Method 2: 0.19

Method 3: 0.35

Method 4: 7.40

Method 5: 0.07 (fast!)

FYI, I added method 5 which is my one-liner:

```
t0 = systime(1)
rgb = [[[r[img]]],[[g[img]]],[[b[img]]]]
print, systime(1)-t0
```

> Curiously, the transpose method performed the best on my platform. You taught me something about transpose. I just did a transpose() and it takes about 3 times as long as your transpose(...[2,0,1]). That subscript vector takes it from the slowest method to what is so far usually the fastest... Thank you muchly!

-k.

<http://edgcm.org/>

<http://spacebit.dyndns.org/>
