System Programming HW2 Report

202211069 류운비

Speedup Result

img_128.bmp	img_256.bmp	img_512.bmp	img_768.bmp	img_1024.bmp
2.64x	2.66x	2.65x	2.59x	3.22x

Optimization Approach

Strategy 1: Optimizing Memory Allocation

filter_optimized() is allocating memory repeatedly when updating every each pixels. I thought there is no need to do that in every pixel, just updating values in already allocated memory. The ‘malloc’ came out ahead from 2 for() loops, ‘free’ came out back to 2 for() loops.

This procedure lead speed up 1x → 1.23x (rough average) on img_1024.bmp

Strategy 2: Optimizing Exception Handling

At ‘static Pixel convolution()’, there is code that checking the index of pixels if it becomes less than 0 or larger than width or height. But for x, there is no need that checking in 2nd loop. Replacing it before y loop starts will enhance performance.

This procedure lead speed up 1.23x → 1.25x (rough average) on img_1024.bmp

Strategy 3: Optimizing Cache Use

In the lecture, there was the comparison about changing the loading sequence of matrix. The loading method which is friendly for the cache size will enhance the speed. In the ‘filter_optimized()’, I changed to use the matrix data with primarily fixing y then using x. Also, the accessing method on ‘static Pixel convolution’ has changed.

This procedure lead speed up 1.25x → 1.585x (rough average) on img_1024.bmp

Strategy 4: Optimizing Repeated Same Calculation

Looking at ‘static Pixel convolution()’, we can check the same calculations are repeated. For example, ‘(x+dx)+(y+dy)*width’ and ‘(dx+1)+(dy+1)*3’. We don’t need to calculate in same time. Furthermore, we just need to add 1 on ‘(dx+1)+(dy+1)*3’ which started from 0, also ‘x+dx’ and ‘y+dy’ can be optimized. At ‘filter_optimized()’, also ‘x+y*width’ can be optimized similarly.

This procedure lead speed up 1.585x → 1.97x (rough average) on img_1024.bmp

Strategy 5

We know that the size of filter is 3x3, the loop in ‘static Pixel convolution()’ will be repeated in 9times.

For the efficiency, the loop unrolling can be adopted. I implemented the code that if we are not accessing to out of range of input (i.e. x, y are not less than 0 or larger than width-1 or height-1), unrolled loop will be used for efficient calculations. The loop in 'filter_optimized()' can also be unlopped. But as it is not available to implement all the iterations, I modified x to increase in 4 at once. Following the test, changing the y increasement didn't affect on the processing time.

This procedure lead speed up 1.97x → 2.6x (rough average) on img_1024.bmp

Strategy 6

Instead of allocating memory, we can update output[o] as stack variables. Now, we are now able to update r, g, b at once.

This procedure lead speed up 2.6x → 2.75x (rough average) on img_1024.bmp

Strategy 7: Failed

To make the code friendly for the cache, I thought that if we access given bmp image array in the sequence of small sized block because in the 'convolution()', it will access near data array position while using dx and dy. I primarily blocked into 8x8 size. But the speed changed 2.75x -> 2.7 (decreased). 4x4 size also decreased (2.6x). I think that this change isn't necessary because prior method is already accessing sequentially, the effect of blocking method might be less than effect of sequential access. This idea may improve the performance if the filter size is bigger than 3x3.

Strategy 8

In the 'static Pixel convolution()' part, 'r', 'g', 'b' has been defined as double. But we don't need to use double for the precision because the checker in the main tolerates very small differences. So, I changed it r, g, b into float which uses 4B while double uses 8B. It gave significant improvement on speed.

This procedure lead speed up 2.75x → 2.88x (rough average) on img_1024.bmp

Strategy 9

In the 'static Pixel convolution()' part, each r, g, b are checked if it is larger than 255 or smaller than 0. The code has been made to check if it is larger than 255 even if it is set to 0 in previous line. I changed this algorithm to manually make p.r(or p.g, p.b) to (unsigned char)0; if they are smaller than 0, 255 as same algorithm. If it involves no case, p.r(or p.g, p.b) to (unsigned char)r;. It enables to avoid modifying r(or g, b) Also memset() doesn't necessary so that I deleted.

This procedure lead speed up 2.88x → 3.22x (rough average) on img_1024.bmp