System_Programming assignment2

202011168 이재현

1. Implementation results

Increasing times is the maximum speed up value when measured 5 times. The average speed up is approximately 0.2 less than the maximum speed up.

FileName	img_128.bmp	img_256.bmp	img_512.bmp	img_768.bmp	img_1024.bmp
identity_filter	3.0990	3.1625	3.2050	3.2009	3.7569
edge_filter	2.9392	3.1150	2.9477	2.9860	3.5067
sharpen_filter	3.0360	3.1015	3.3640	3.1847	3.6046
boxblur_filter	3.0917	3.1563	3.1987	3.3414	**3.7925**
gaussian_filter	3.0867	3.2593	3.2597	3.2258	3.7402

The degree of performance improvement tends to increase as the image size increases. You can see that the number gets bigger because the amount of computation increases. When img_1024.bmp was applied to boxblur_filter, the highest value, 3.7925, came out. This may be because the coefficients of the filters are all the same. The lowest value is 2.9392 when edge_filter is applied to img_128.bmp.

2. Optimization approach

(1) Memory Allocation remove

removed dynamic memory allocation in the filter_optimized function to reduce execution time. This significantly improved performance.

(2) memset remove

removing unnecessary memset calls to avoid unneeded memory initialization and improve performance.

When applying the first two methods, the speed increased by about 1.5. It was confirmed to be an effective method.

(3) inline function

Defined the clamp function as an inline function to reduce function call overhead. Inline functions are executed directly in place, improving performance. but Did not see any performance improvement. I think because there weren't that many calls.

(4) loop unrolling

Applied loop unrolling to reduce the number of iterations within loops. This code uses 2-way loop unrolling to process two pixels at once. This reduces loop control overhead and increases CPU instruction-level parallelism. Through loop unrolling, a speed improvement of more than 3 was confirmed for the first time. It is believed to be the best optimization method.

(5) precomputing

Precomputed indices for neighboring pixels within the loop to improve data locality. This makes memory access patterns more predictable, enhancing cache performance.

(6) Data types

Double type takes up more memory space. Avoid using it as much as possible.

(7) Integration of convolution function and filter_optimized function (presured call)
By integrating the two functions, convolution is not called, and performance is improved by allowing the operation to be performed directly within the code. This reduces presured calls.

(8) accumulators
We created float r, g, and b variables to temporarily store the calculation results, and stored them in pixels after all calculations were completed. This improves performance by reducing the number of memory accesses.

(9) Using macros
Using macros instead of functions for min and max is faster than calling functions. However, no actual performance improvement was confirmed.

# Unfortunately, after method 5, too many methods were added and removed, so it was not possible to determine whether each method was independently effective.