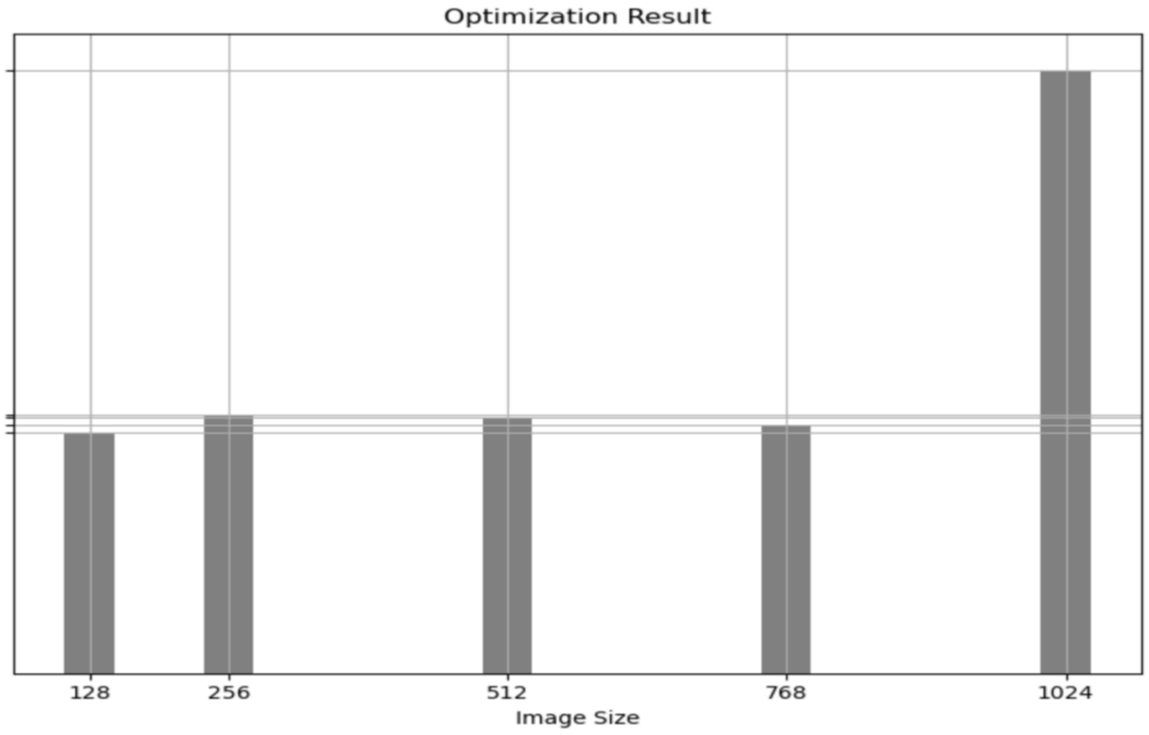
HW2 CODE OPTIMIZATION REPORT

202011203 정은호

SECTION 1. IMPLEMENTATION RESULTS

The following graph and table show the speedup of optimized filter compared to the baseline filter, on various images of size 128x128, 256x256, 512x512, 768x768, and 1024x1024. The x axis represents image size, and the y axis represents speedup.

Image Size	128*128	256*256	512*512	768*768	1024*1024
Speedup	2.4141	2.4453	2.4404	2.4269	3.0479



SECTION2. OPTIMIZATION APPROACH (comparison is done with edge_filter)

Approach 1: No unnecessary malloc or memset

I refactored the baseline code by moving the malloc and free functions outside the loop, thus eliminating unnecessary dynamic allocation costs per loop iteration. Additionally, I removed redundant memset function calls. (This could be thought as another type of Code Motion, explained in Code motion)

Image Size	128*128	256*256	512*512	768*768	1024*1024
Speedup_before	1.00	1.00	1.00	1.00	1.00
Speedup_after	1.15	1.14	1.13	1.13	1.18

Approach 2: Reduction in Strength

Recognizing that multiplication is more costly than addition, I optimized the index calculation by introducing variables. This adjustment replaces unnecessary multiplications with addition operations.

Speedup_before	1.15	1.14	1.13	1.13	1.18
Speedup_after	1.20	1.16	1.19	1.18	1.20

Approach 3: Code Motion

By using code motion technique, I eliminated unnecessary repeated memory access or multiplication operations. If index does not change within a single loop, it is inefficient to keep accessing memory using the same index. It is rather wiser to get the value in the index, and never change the value within the same loop.

Speedup before	1.20	1.16	1.19	1.18	1.20
Speedup after	1.45	1.44	1.44	1.44	1.73

Approach 4: Reduce overhead for function calls

I integrated the convolution function into the filter function to eliminate the overhead of unnecessary stack creation and deletion.

Speedup before	1.45	1.44	1.44	1.44	1.73
Speedup after	1.55	1.55	1.56	1.57	1.87

Approach 5: Direct Memory Access

I changed from using the Pixel struct to using a uint8_t array to directly access and store values in memory. This approach eliminates unnecessary struct copying, which could eventually lead to improvement in performance.

Loop unrolling w/o Direct Memory Access	2.40	2.51	2.41	2.48	2.92
Loop unrolling W Direct Memory Access	2.42	2.46	2.42	2.44	2.98

Approach 6: Loop unrolling

Loop unrolling is a technique which introduces ILP(instruction-level parallelism). By duplicating the body in a loop, we can exploit parallelism and thereby reduce the number of loop control instructions, and conditional branch. I initially attempted loop unrolling with a factor of 2 and subsequently increased it to 4, 8, and 16 and so on to compare performance. It was observed that significant performance gains ceased after reaching a factor of 8.

Speedup before	1.55	1.55	1.56	1.57	1.87
Speedup after	2.40	2.51	2.41	2.48	2.92

-------------------Failed Approaches----------------

Failed Approach 1: Fixed-point Arithmetic

When I tried fixed-point arithmetic, the speedup with edge_filter was almost 3.3. However, the computation result was different as I tried with boxblur_filter. Since fixed point arithmetic use limited number of digits, there must have been loss in precision as I used fixed-point Arithmetic.

Failed Approach 2: creation of Unit24 Struct

I tried to group the uint8 variables R, G, and B for unified management, aiming to minimize memory access. However, this did not work as well as expected. I expect that the reason for this is because structs often introduce padding to ensure that the data is aligned. This padding could have lead to inefficiencies in memory usage and access.

Other Failed Approaches: Usage of inline function had less effect than approach 4, and blocked matrix multiplication method did no good for increasing the speedup.

---------------------Conclusion---------------------

Various techniques were combined to increase the speedup of convolution filter code. While some techniques were not that effective, ‘loop unrolling’ technique showed best performance!