Assignment 2 Report

202011135 이가현

1. Implementation Results

	img_128.bmp	img_256.bmp	img_512.bmp	img_768.bmp	img_1024.bmp
Average (10 times)	2.3085986	2.3101781	2.339659	2.3243313	2.6605752
Best	2.419507	2.436093	2.500952	2.444482	2.81055

2. Optimization Approach

- **Eliminate Dynamic Memory Allocation**

  Removed the malloc and free calls within the loop to reduce memory allocation overhead

- **Merging convolution() and filter_optimized()**

  Integrating the convolution logic to reduce the overhead of multiple function calls

- **Integer conversion for filter values**

  Filter values were converted from floating-point to integer values, improving performance by leveraging faster integer arithmetic. Also, bit-shifting was used to handle scaling efficiently

- **Loop Unrolling**

  Processing was optimized using loop unrolling, handling 4 pixels at a time instead of one.

  Loop unrolling was tested with different configurations to identify the optimal balance between performance and complexity. The performance improvements for unrolling 2 pixels, 4 pixels, and 8 pixels at a time were evaluated. Unrolling 2 pixels at a time resulted in a speedup factor of 2.028642, and 8 pixels at a time resulted in 1.476248. Therefore, unrolling 4 pixels at a time provided the best balance

- **Reduced Multiplications**

  Multiplications were minimized by pre-computing values that were used repeatedly within loops. The use of addition and bitwise operations further decreased the number of muiltiplications.

- **Share Common Subexpressions**

  Common subexpressions were identified and calculated once, then reused in multiple places within the code.

- **Removing Aliasing**

  The restrict keyword was used to inform the compiler that the pointers used in the function do not overlap. This allowed the compiler to optimize memory access patterns more effectively, reducing potential bottlenecks caused by memory aliasing.

- **Variable Type Optimization**

  Variables originally declared as double, such as r, g, b, were converted to int types. This change leveraged the faster integer arithmetic operations, resulting in improved performance due to reduced computational complexity associated with floating-point operations.