Name: 정유진 (Eugene Jeong)

Student ID: 202211181

Optimization Approach

Optimizing Memory Access Patterns

One key factor in optimizing the performance of the convolution function is the improvement of the memory access pattern. In the default implementation, memory access was sporadic and inefficient. To address this, the memory access was made more sequential and cache-friendly. By processing multiple pixels (8 in this case) in a single loop iteration, the overhead of loop control was reduced, and spatial locality was improved.

Cache Utilization Strategy

Effective use of the CPU cache is also important. The optimized function processes pixels in blocks, ensuring that once a cache line is loaded, it is fully utilized before moving on. This approach minimizes cache misses and exploits the temporal and spatial locality of references. By accessing adjacent pixels consecutively, the power of the cache is maximized.

Instruction Count Reduction Measures

The number of executed instructions was reduced by simplifying calculations and minimizing redundant operations. Integer arithmetic was used instead of floating-point operations to significantly reduce the number of instructions. Filter values were precomputed and stored as integers to enable faster arithmetic operations. Bit-shift operations were used to handle scaling, replacing division to further improve performance.

Result

Bmp 128: 2.198811

Bmp 768: 1.946885

Bmp 512: 1.904467

Bmp 256: 1.840630

Bmp 128: 1.822907

Explaining Failed Strategies

Multithreading

The introduction of multithreading to parallelize the convolution operation on multiple cores was considered. However, the overhead of thread management and synchronization offset the potential speedup benefit, especially for small image sizes.

SIMD Instructions

The use of SIMD (single instruction, multiple data) instructions, such as ARM NEON, was also considered. While SIMD can provide significant performance gains, it was excluded from the constraints of this challenge to ensure a fair comparison.

Conclusion

The optimized convolution implementation achieved significant speedup over the baseline implementation by focusing on memory access patterns, cache utilization, and reducing the number of executed instructions. While some strategies did not show the expected improvement, the overall optimization process resulted in measurable performance gains. Future work will explore hardware-specific optimizations and additional algorithmic improvements to further improve performance.