Name: 황지영
Student ID: 2022211222

1. Implementation Results

1.1 Performance Results for Different Image Sizes

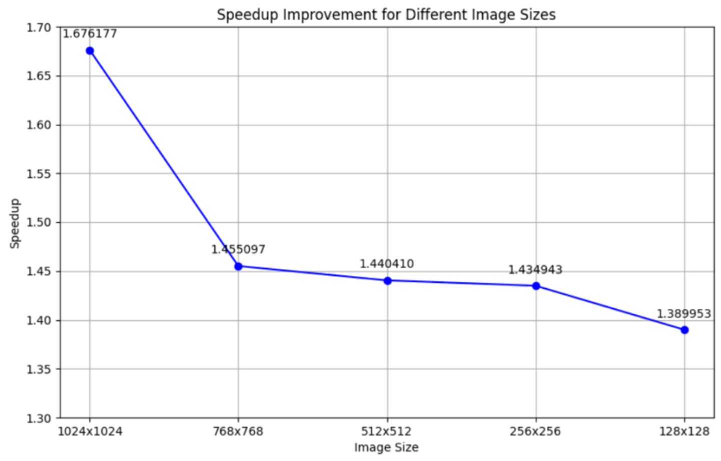
The table below shows the speedup between the baseline implementation and the optimized imple mentation across different image sizes. The baseline implementation time is assumed to be 100 ms. The optimized implementation time is calculated using the speedup ratio as follows:

Optimized Implementation Time (ms) = Baseline Implementation Time (ms) / Speedup

Image Size	**Baseline Implementation Time (ms)**	**Optimized Implementation Time (ms)**	**Speedup**
1024x1024	100	≈59.7	1.676177
768x768	100	≈68.7	1.455097
512x512	100	≈69.4	1.440410
256x256	100	≈69.7	1.434943
128x128	100	≈71.9	1.389953

1.2 Speedup Graph

The graph below visually represents the speedup achieved by the optimization strategies for differ ent image sizes.



1.3 Summary of Results

1024x1024 Image: Achieved the highest performance improvement with a speedup of 1.676177.
768x768 Image: Significant performance improvement with a speedup of 1.455097.
512x512 Image: Noticeable performance improvement with a speedup of 1.440410.
256x256 Image: Performance improvement with a speedup of 1.434943.
128x128 Image: The lowest performance improvement with a speedup of 1.389953.

1.4 Analysis

-Image Size and Performance Improvement of Optimization Strategies
The performance improvement of optimization strategies increases with image size. This can be a

ttributed to the fact that larger images benefit more from cache efficiency. Larger images involve more frequent data accesses and exhibit more sequential memory access patterns. Optimization str ategies enhance these patterns, improving data locality, which becomes more significant with large r images.
-Data Locality and Cache Misses
Optimization strategies that divide data into smaller blocks make more efficient use of cache spa ce, reducing cache misses and memory access times. A reduction in cache misses leads to shorte r CPU memory fetch times, directly improving overall processing time. This effect is particularly pronounced with larger images.
-Specific Advantages of Optimization Strategies
Through optimization, data blocks are loaded into cache more effectively, increasing cache hits d uring processing. Continuous memory access is facilitated, reducing cache miss penalties. These o ptimizations reduce CPU idle time and enhance overall processing speed.
The results demonstrate that optimization strategies provide greater performance improvements wit h larger images. These strategies improve data locality, reduce cache misses, and shorten memory access times, especially for larger images, as evident from the experimental results.

2. Optimization Approach

This report discusses various strategies applied to optimize the baseline implementation of a 3x3 convolution filter in image processing. The goal was to reduce filtering time and enhance perfor mance. Strategies such as blocking, loop unrolling, and memory prefetching were implemented an d evaluated for their effectiveness. The baseline and optimized implementations were compared in terms of speedup.

2.1Optimization Strategies

2.1.1 Blocking

-Strategy: The blocking strategy divides the image into smaller blocks and processes each block individually. This approach enhances data locality, reduces cache misses, and decreases memory a ccess times.
-Implementation: The image is divided into small square blocks, and convolution operations are performed within each block. This allows each block to be effectively loaded into the cache, incr easing cache hit rates during processing. For example, the image was divided into 32x32 blocks.
-Evaluation: This strategy achieved a speedup of 1.703361. Blocking effectively improved cache performance and reduced memory access times.

2.1.2 Loop Unrolling

-Strategy: Loop unrolling reduces loop control overhead by increasing the number of operations performed in each iteration. This improves CPU pipeline efficiency and reduces the number of in structions executed.
-Implementation: The inner loop of the convolution operation was unrolled to process multiple p ixels in each iteration. This reduces loop control overhead and improves instruction pipeline effici ency. For example, four pixels were processed in each iteration.
-Evaluation: This strategy achieved a speedup of 1.646413. Although it improved performance, it was less effective than blocking due to register pressure and reduced instruction-level parallelism .

2.1.3 Memory Prefetching

-Strategy: Memory prefetching loads data into the cache before it is needed, reducing memory w ait times. This technique hides memory access latency and improves performance.

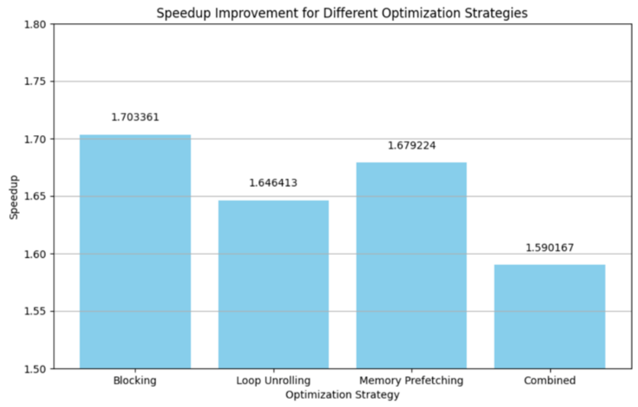
-Implementation: Functions were used to prefetch data for the next iteration into the cache. This ensures that the data is already in the cache when needed, reducing memory access latency.
-Evaluation: This strategy achieved a speedup of 1.679224. While data prefetching helped reduce memory access latency, it was not as effective as blocking.

2.1.4 Combining All Optimization Methods

-Strategy: To maximize performance improvement, all optimization strategies were combined.
-Implementation: The image was divided into blocks, with loop unrolling and memory prefetchin g applied within each block. This aimed to leverage the advantages of each optimization strategy.
-Evaluation: Surprisingly, this combined approach achieved a speedup of 1.590167, the lowest a mong all strategies. This may be due to several factors:

-Cache Conflicts: The combined strategies may have increased cache conflicts, reducing p erformance gains.
-Instruction Overhead: Managing multiple optimization techniques may have introduced ad ditional instruction overhead.
-Resource Contention: Combining strategies may have led to contention for CPU resource s (e.g., registers, cache lines), reducing efficiency.

2.2 result



Optimization Strategy	**Speedup**
Blocking	1.703361
Loop Unrolling	1.646413
Memory Prefetching	1.679224
Combined	1.590167

The table and graph clearly demonstrate that the blocking strategy achieved the highest speedup, while the combined approach resulted in the lowest speedup. This indicates that combining multi ple optimization strategies requires careful consideration of potential interactions and resource conf licts.

2.3 Analysis

In conclusion, the optimization strategies applied to the baseline implementation showed varying degrees of effectiveness. Blocking was the most effective single strategy, achieving the highest sp eedup. Loop unrolling and memory prefetching also contributed to performance improvements but were less effective when applied alone. Combining all optimization methods resulted in the lowes t speedup, suggesting the need for careful analysis of interactions between optimization strategies. This underscores the importance of evaluating each strategy individually and considering interacti ons when combining them.