Homework 2 Report

202111023 김동현

Section 1. Compare with baseline

Resolution	128	256	512	768	1024
Speed up	1.573495	1.191529	1.176223	1.174536	1.805630

The speed-up results of my implementation compared to the baseline are as follows. For image sizes 128 and 1024, the speed-ups are 1.57x and 1.8x, respectively. However, for image sizes 256, 512, and 768, the speed-up is only 1.2x, indicating a less significant improvement.

Section 2. Optimization Approach

1. Reducing Computation Frequency:

To minimize the frequency of computations, I focused on identifying sections of the code where identical calculations were being repeated. By grouping these sections into a single computation, I was able to reduce the redundant computational effort. This approach involved analyzing the algorithm to pinpoint common calculations that could be shared across different parts of the process. By consolidating these computations, the overall workload was significantly decreased, leading to improved performance. This optimization was particularly effective in scenarios where certain operations were repeatedly executed on similar data sets.

2. Sharing Common Subexpressions:

Common subexpression elimination (CSE) was another strategy I employed to enhance performance. This technique involves identifying and eliminating redundant calculations within expressions by reusing the results of previously computed subexpressions. By doing so, the number of operations required to achieve the final result was reduced. This optimization not only streamlined the code but also improved its readability and maintainability. The reduction in redundant calculations contributed to a noticeable increase in the efficiency of the overall process.

3. Row-major Order Memory Access:

I adopted a row-major order memory access pattern to improve memory efficiency. In a row-major order, elements of each row of a matrix are stored in contiguous memory locations. This method aligns with the way data is stored in memory, allowing for more efficient access patterns. By accessing memory in a manner that matches its layout, I was able to increase cache utilization and reduce latency. This optimization was particularly beneficial in reducing the time required for memory accesses, which can be a significant bottleneck in performance-critical applications.

4. Row-wise Convolution Optimization:

For the convolution operation, I implemented a row-wise optimization approach. This involved processing each row of the image sequentially, moving the filter across the columns from left to right. By multiplying each row of the filter by the corresponding image row and adding the result to the appropriate output position, I aimed to maximize cache hits. This approach was expected to enhance performance by ensuring that an entire row of data is loaded into the cache, reducing the need for frequent memory accesses.

However, the performance improvement achieved with this method was limited to a 1.5x speedup. The primary reason for this limited improvement was the necessity to access all three RGB channels' rows. This led to cache misses, as the amount of data being processed exceeded the capacity of the cache. Additionally, the attempt to load too much data at once was not well-suited for the Raspberry Pi's memory capacity. The device's limited memory size could not efficiently handle the large data sets being processed, leading to suboptimal performance.

Despite these challenges, I believe that further optimizations are possible. While time constraints prevented additional improvements, I anticipate that partial restructuring of the code and adjustments to better align with the Raspberry Pi's memory limitations could yield significant performance gains. By tailoring the data processing approach to the device's capabilities, it would be possible to enhance the overall efficiency and achieve better results.