HW2_report

Student ID: 202011032

Name: 김민관

1. Implementation Result

As a result of implementing the code, the following speedup was obtained.

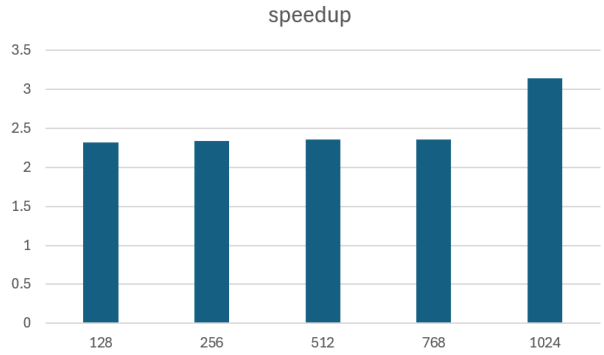

Figure 1 Compare speedups by size

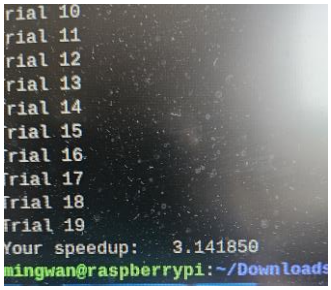

Figure 2 maximum speedup (1024)

2. Optimization approach

    1. Remove memory allocation and release

    Instead of allocating and releasing dynamic memory every time each pixel was processed, we changed it to return the Pixel structure directly. This improves performance by eliminating memory allocation and release overhead.

    2. Improved cache efficiency

    The order of internal iterations was changed to row-first to increase the cache hit rate. This reduced memory access time and improved overall performance.

    3. Remove unnecessary memset

    The memset function calls used to initialize Pixel were removed. This improves performance by eliminating unnecessary initialization tasks.

    4. Remove duplicate calculations

    Instead of repeatedly calculating x + dx and y + dy, we made them reuse the values once calculated. This reduces unnecessary computation and improves overall performance.

5. Replace multiplication with addition

We pre-calculated the starting index of the y row using the row_offset variable, and used the addition operation instead of multiplication in the inner loop. This has improved the operation speed by replacing the multiplication operation with the addition operation.

6. Pre-calculate row offset

In the convolution function, y_offset and yy_offset were computed in advance to reduce the in-loop multiplication operation. Multiplication was avoided when calculating the filter index by pre-calculating the filter_base_index.

7. Minimized Multiplication Operations

When calculating filter_index and pixel_index, both multiplication operations were replaced by addition operations. This minimizes the number of multiplication operations to improve performance.

8. Move pre-calculated values out of the loop

We pre-calculated the width_minus_1 and height_minus_1 variables to reduce the width-1 and height-1 operations inside the loop.

9. Remove duplicate calculations

The convolution function did not use the y_offset variable, so it was removed. Variables for filter index calculation and pixel index calculation were moved out of the loop.

10. Add Type Casting

The input, output, and filter arrays were cast in Pixel* and float* types, respectively, so that they could be used without warning.

11. Optimizing access to memory

In the convolution and filter_optimized functions, row and column pointers were calculated in advance and changed to be used. This can reduce memory access time.

12. Minimize range check

Unnecessary operations were eliminated by minimizing range checks inside repetitive statements.

13. Loop Unrolling

We applied loop unrolling by increasing the x-loop by 4 in the filter_optimized function. This allows you to process four pixels at the same time.

14. Extract Common Partial Expression

Clamping operations of r1, g1, and b1 and r2, g2, and b2 were extracted as common partial expressions. This eliminated redundant operations and improved performance.

15. parallel processing

The process of calculating R, G, and B values for each pixel was parallelized.
This can reduce computational time.

These various optimization techniques have been applied to significantly improve code performance. In particular, techniques such as memory management, operation minimization, cache efficiency improvement, and parallel processing have reduced overall execution time. Through this optimization, performance improvement can be expected when processing large images.