Assignment 2: Efficient Program Implementation

202011047 SeungTae Kim

First Section

The following table shows the SpeedUp values according to each image size. Among these, 1024 Size's Speedup is the maximum value obtained after repeated trials, and the average is around 2.9.

Image Size	1024	768	512	256	128
SpeedUp	3.104100	2.572371	2.479298	2.486958	2.465993

Second Section

Optimization Strategies Applied

1.Removal and Integration of Functions – Observed Increase in speedup : around 0.2

Calling functions itself incurs costs, so I adjusted the filter_optimized function to simultaneously perform the role of convolution. This showed some effect.

2.Removing Unnecessary Procedure Calls – Observed Increase in speedup : around 0.3

Functions like malloc and free can cause overhead. In the previous code, malloc and free were repeatedly called within loops, which I believed would cause significant performance degradation. Therefore, I removed malloc and free, and directly inserted the calculated values into the output array to eliminate overhead. In this process, the convolution function was almost integrated into filter_optimized.

3.Code Motion– Observed Increase in speedup : around 0.3

This involves moving computations performed inside a loop to the outside. Here, I moved the calculation of output to outside the loop. By summing into Row_output and immediately calculating within the height loop, the results for the inner loop (width) were calculated, significantly reducing repetitive calculations. Initially, I planned to move it completely outside, but encountered errors and structural difficulties, so I settled on this configuration.

4.Strength Reduction – Observed Increase in speedup : around 0.5

During the calculation of r, g, b, I minimized the operations as much as possible. By using row0, row1, and row2, I reduced the repetitive calculations, and accordingly restructured the calculation codes. Although the number of variables increased due to unavoidable initialization, the speed improved.

5.Sharing of Common Subexpressions– Observed Increase in speedup : around 0.2

When restructuring the calculation codes during strength reduction, I stored all the repeatedly used calculation results in variables to prevent additional computations, resulting in better performance.

6.Case Division– Observed Increase in speedup : around 0.4

I confirmed that dividing cases and calculating accordingly showed better performance than integrating conditions into a single calculation. Therefore, I divided all cases using if statements, so that only specific calculations were performed when conditions were met. The conditions were very simple, so I expected faster results.

Failed Attempts

Loop Unrolling, Separate Accumulators, Reassociation

I attempted Loop Unrolling and Reassociation, but they produced strange results. Although I saw repeated trials and execution, speedup measurements were not possible, leading me to conclude that interaction with main was impossible. Even after repeated attempts using different methods, the same unresponsive results were observed, so I gave up. I also tried Separate Accumulator, but possibly due to incorrect coding, it did not result in speedup and instead showed a decrease. It was prone to errors even with minor mistakes, so I did not include it as an optimization technique.

** I have tried coding using block matrices, but the maximum result was only around 2.8. I realized that combining the above optimization techniques into a single code is more effective than using block matrices alone.