System Programming

Report

2024.06.16

202211165 Jaeeun Jang

DGIST
대구경북과학기술원

Optimization Strategy

[Performance]

Image size	Speed up (times)
128	1.818058
256	1.789109
512	1.951765
768	1.913505
1024	1.998755

[Optimization Approaches]

1. Loop Unrolling and Rearrangement

- Loop Unrolling: Reduced loop overhead by processing multiple iterations within one loop cycle. This approach aimed to minimize the overhead associated with loop control and enhance the efficiency of memory accesses.

- Loop Rearrangement: Reorganized loops to maximize data locality and reduce data dependencies. By rearranging loops, we aimed to improve cache utilization and optimize memory access patterns.

  ➔ About 1.4 speed up

2. Memory Access Optimization

- Optimized Data Structures: Restructured data structures to promote contiguous memory access and leverage cache efficiency. This optimization aimed to minimize cache misses and enhance overall performance, changing Column-wise to Row-wise.

- Cache Utilization: Optimized data access patterns to improve cache line efficiency. This involved accessing data in a manner that aligns with the cache architecture, reducing latency and improving throughput.

  ➔ About 1.6 speed up

3. Precomputed Values

- Constant Precomputation: Precomputed constant values such as convolution filter coefficients. This strategy aimed to eliminate repetitive computations during runtime, reducing processing time and enhancing overall efficiency.

  ➔ About 1.65 speed up

4. Function Call Reduction

- Elimination of Redundant Functions: Consolidated the convolution logic directly into the filtering function. By removing separate function calls, we minimized function call overhead and streamlined the execution flow, leading to improved performance.

  ➔ About 1.8 speed up

[More Detailed Implementation]

- Merging Convolution into Filtering Function: Combined the original convolution function into the filter_optimized function. This integration eliminated the overhead of calling separate functions and reduced memory access redundancies by directly manipulating pixel data.

- Optimized Loop Structures: Modified loop structures to iterate over a reduced range (-1 to 1 for both x and y offsets) while ensuring boundary conditions were checked efficiently. This reduced unnecessary computations and improved the efficiency of the filtering operation.

- Direct Pixel Access: Accessed pixel data directly using array indexing and minimized pointer dereferencing. This

approach enhanced memory access efficiency and reduced the number of memory read/write operations, contributing to overall performance gains.

➔ I believe that the speed could be further improved through data type conversion or simplifying the for loop. I'm disappointed that I couldn't increase it much.