# System Programming: HW2 Report

202011176 이채원

1. Implementation Results

| image | speedup |
|---|---|
| img_128.bmp | 1.261070 |
| img_256.bmp | 1.228202 |
| img_512.bmp | 1.243268 |
| img_768.bmp | 1.278762 |
| img_1024.bmp | 1.408624 |

2. Optimization Approach

2.1 Memory Access Pattern Optimization

For optimization, the first step was to change the memory access pattern. To improve cache efficiency by accessing memory row by row, the order of the double loop was changed. Previously, a column-major access pattern was used, which resulted in frequent cache misses. By changing to a row-major access pattern, the cache hit rate was increased. As a result, speedup was increased to about 1.1.

2.2 Dynamic Memory Allocation Removal

Next, dynamic memory allocation (‘malloc’) and deallocation(‘free’) were removed, and the ‘Pixel’ structure was allocated directly on the stack to reduce overhead. Dynamic memory allocation takes considerable time for memory management, and removing it improved performance.

2.3 Loop Unrolling

Additionally, within the double loops, the inner loop was unrolled four times to reduce the loop control overhead. Loop unrolling reduces the number of iterations, minimizing loop control instructions and improving the efficiency of the CPU pipeline. This resulted in a speed of approximately 1.3 based on the ‘img_1024.bmp’.

2.4 Boundary Check Minimization

Finally, boundary condition checks were minimized to reduce unnecessary operations. Conditions such as ‘x < width’, ‘x + 1 < width’, ‘x + 2 < width’, ‘x + 3 < width’ were added to call the convolution functions only when necessary, avoiding redundant processing for all pixels. This optimization ensured that convolution was invoked only when needed, thereby enhancing performance. As a result, speedup of approximately 1.4 was achieved based on ‘img_1024.bmp’.