System Programming HW2

202011012 권용석

I. Implementation Result

	Try 1	Try 2	Try 3	Average
Img_128	2.123375	2.188282	2.17527	2.162309
Img_256	2.203346	2.130625	2.131894	2.155288
Img_512	2.320558	2.139664	2.14213	2.200784
Img_768	2.138866	2.138646	2.21283	2.163447
Img_1024	2.517364	2.497793	2.481216	2.498791

We see a roughly 2.1 to 2.5x speedup. The speedup tends to be higher as the size of the image increases.

II. Optimization Approach

I used four main strategies:

1. Memory Allocation Optimization:

Remove dynamic memory allocation (malloc and free) and directly calculate and allocate the result for each pixel. Previously, we used to store the value of convolution, copy it to an array in output, and free dynamic memory, but now we store the calculation result directly in output. This reduces the time spent allocating and freeing memory and improves performance. This resulted in a speedup of about 1.3x based on img_1024.

2. Separate Boundary Processing:

We perform boundary processing using separate loops for each side (top, bottom, left, right). This eliminates the iteration of checking boundary conditions and only performs boundary checks when necessary. Previously, the convolution would check if each pixel is an edge or not through an if statement, but since images usually have more interior pixels than edges, we decided to separate the edge and interior parts. This resulted in a speedup of about 1.4x based on img_1024.

3. Unlooping:

We unloop dx and dy and process each filter element explicitly. This removes the loop overhead and optimizes performance by directly accessing each filter element. This resulted in a speedup of about 1.8x based on img_1024.

4. Combining Two or More If Statements:

Since there were two separates if statements for one condition, we optimized the operation by replacing them with a single if statement. This resulted in a speedup of about 1.2x for img_1024.

By combining all these strategies, we were able to achieve the speedup shown in the table above.