202011011 구본근

1. Implementation results

image	speedup
img_128	2.415
img_256	2.455
img_512	2.544
img_768	2.460
img_1024	3.071

There was a significant difference in computational load at 1024, and the difference in speed became less noticeable as the number of pixels decreased. However, at 768, the difference in speed was less than at 512, which might be due to it not being a multiple of 2.

2. Optimization approach

1. First, the image was divided into blocks of a size that could fit into the cache. The block size was defined as 128, as increasing it to 256 or higher did not significantly change the results compared to 128.

The speedup without tiling was relatively low at 1.1712.

2. The provided skeleton code had the outer loop on x and the inner loop on y, which is a column-wise approach and not optimal. This was changed to a row-wise approach.

- Using the column-wise approach, the speed was recorded at 2.12.

3. Set the filter as a local variable for easy access.

- Without this adjustment, the speed was 2.39, lower than when it was handled.

4. Simplification of boundary conditions

- The original code checked boundary conditions for each dx, dy, but now it first checks the value of the center pixel, and if dx and dy conditions cannot reach the boundary, it skips the check and proceeds directly to the convolution operation.

- Without this adjustment, a speedup of 1.412 was achieved.

5. Elimination of unnecessary memory allocation

- Operations that could be sufficiently handled locally, like memset and malloc, were unnecessarily allocating memory, which was removed. This resulted in a speed improvement of 1.8.

6. Combining functions into one

- Since calling functions also consumes time, the process of combining two functions into one was performed. It did not have a significant impact but achieved an improvement of 0.2.

Failed strategy

1. Using fmin and fmax instead of 6 if statements

- Attempting to use the well-optimized fmin and fmax functions instead of 6 conditional statements actually reduced the speed.

2. Converting filter and pixel to int

- Considering that integer operations are faster than float or double, the filter and pixel were scaled and then converted to integers, with the intention to later divide them back. This caused significant errors with filters like Gaussian and average, and converting the pixel to int also took a lot of time and caused errors.

3. Unrolling

- Attempted unrolling by skipping every 2 x values, but due to time constraints, proper boundary conditions were not set, resulting in many errors compared to the Ground Truth. This is left for future study.