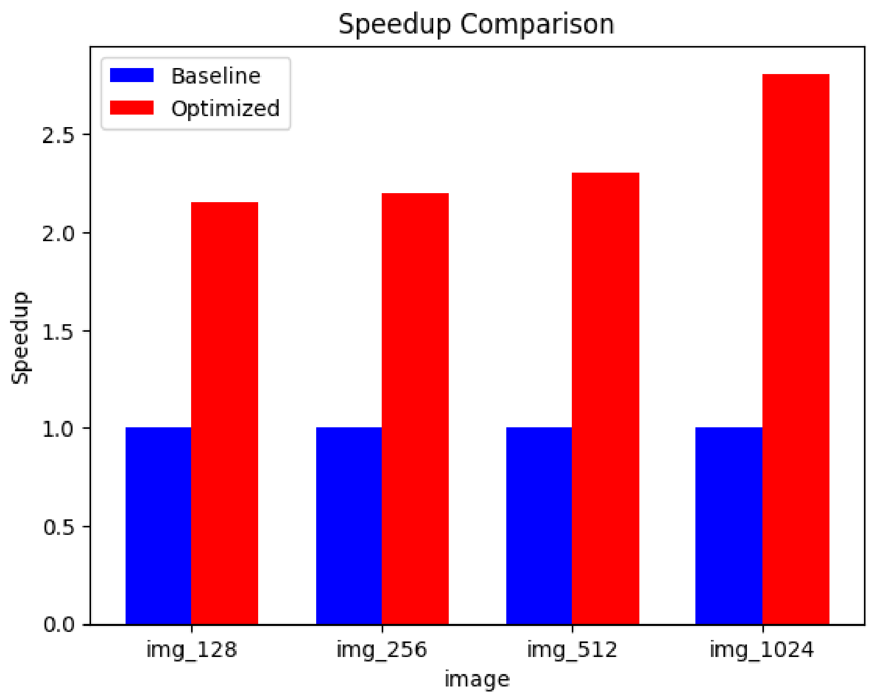
System Programming Report

202011162 이웅기

Section I. Implementation Results

In the order of image sizes 128, 256, 512, and 1024, each speedup showed 2.15, 2.2, 2.3, and 2.81 times better performance than baseline. When measuring the performance, it was confirmed that a small error occurred for each measurement, and after five executions, it was set as an average value. At this time, the speedup value derived after the first compilation comes out as a very large value corresponding to 4 to 5, but afterwards, it was confirmed that it converged within a similar range, and the value after the first compilation was excluded and the value was calculated. In addition, strange result is not included.
Below is the graph showing results.



Section II. Optimization Approaches

First of all, the optimization strategies I used are mainly as follows.
Loop unrolling, Code motion, Reducing Procedure Calls, Data locality.
Of course, in addition, the location change of parentheses, memory repetitive access avoidance, and additional operation orientation were used in the task.
Now I'll discuss about how much performance has improved with each strategy.

1) Loop unrolling

When processing the value for each pixel, there was a cost that had to go around the for statement 16 times, and at this time, it was confirmed that there was no data dependency in the for statement, and 16 pixels were processed at the same time by unrolling the internal loop.

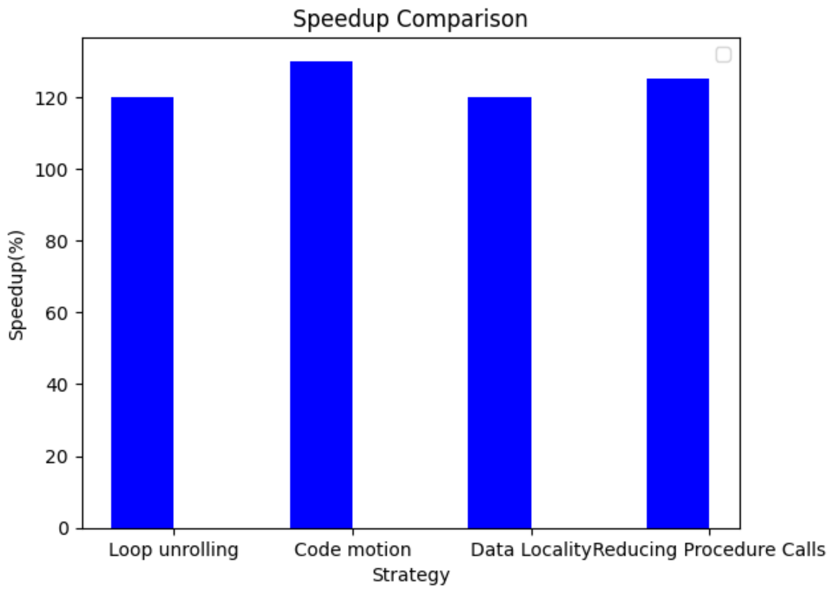
2) Code motion

Code motion has been used in many parts. In particular, since multiplication has a larger time cost than other operations, it is not calculated every time you go around the for statement, which is independent, and define the multiplication operation in advance to reduce the cost. In addition, at this time, even when multiplying, it was changed to addition. As an example, instead of multiplying every time, it was optimized by adding a width value every time iteration was rotated after previously defining it as a specific value.

3) Data Locality
In order to reduce cache miss, it was considered to process as many pixels at a time in consideration of the size of the cache block, and thus it was confirmed that the cache block size of the operating system was 64B. Accordingly, in order to improve Data Locality, the block size was designated as 32, and the corresponding size was processed at a time.

4) Reducing Procedure Calls
Considering the size of the cache block, many forns were created in the process of processing data by 16 pixel units at a time, and the convolution process was applied by directly putting it into the filter_optimized function because it was thought that the cost was high to continue to call the convolution function in the process of loop unloading.

Below is how much speedup is achieved by each strategy.



In addition to the above strategies, there are areas where performance has improved through memory access restrictions, addition operation orientation, and pointer use, so it is difficult to determine the superiority of each strategy when considered independent, and it is thought that each strategy interacted well.

***5) Strategies that failed to improve performance**
When calculating in pixel units or accessing the cache, we tried to use temporal locality by accessing it outside the for statement in advance through warm-up, but the performance was rather reduced by 10%. In addition, we tried to change the multiplication operation to shift operation for Reducing Strength, but the result showed that the image was broken due to the lack of accuracy. In addition, we tried to prevent the for statement from rotating several times by efficiently arranging the conditional statement, but the strategy could not be used because it confirmed that the image was broken as a result of the same.