System Programming: Assignment 2. Efficient Program Implementation

Submitter: 장지원 (202211167)

1. Implementation Results

I am using four optimization methods to increase convolutional operation speed. The best result is achieved when the loop unrolling at img_1024. It '3.21' times better than base case.

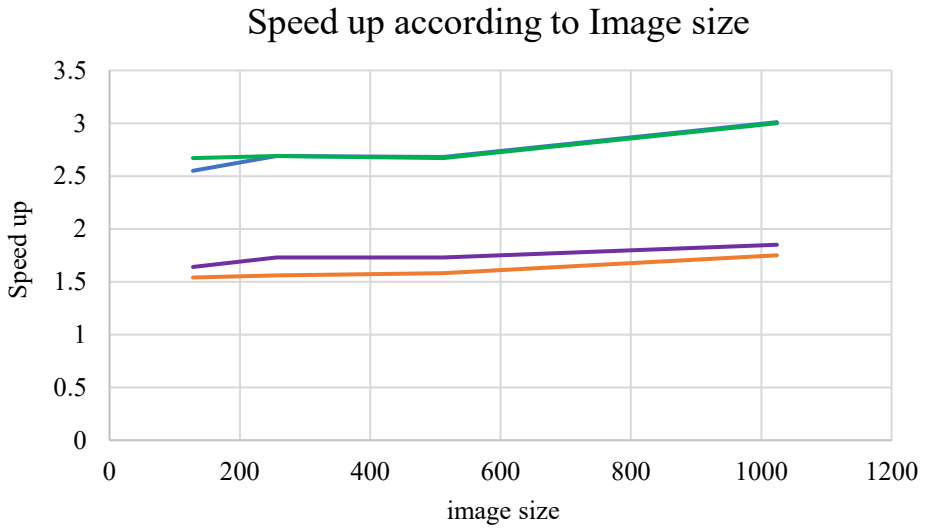

Figure 1. *Speed-up according to image size. The orange line represents basic optimization (i.e., code motion, Reduction in Strength), the purple line represents Function Inline optimization, the blue line represents Loop Unrolling optimization, and the green line represents the result when Zero-Padding(to remove conditional branch) is applied.*

2. Optimization Approaches

Using Optimization: The first optimization is **basic** (i.e. code motion, Reduction in strength). This optimization is effective in reducing multiplication operations. Also, **dynamic programming** can help reduce multiplication operations too. I used these two methods to create lightweight operations. Actually, In my code, there is only a multiplication operation in *input_image*filter.* Most of the other operations are addition operations. The second optimization is the **inline function**. The base case code given that called the 'convolution' function. If we including convolution in the base, we can yield slightly higher accuracy. The third optimization is the **Loop Unrolling**. It is a strong optimization method. This can effectively reduce the number of operators. I applied unrolling factor 3 in *for (int dx = -1; dx < 2; ++dx)*, *for (int dy = -1; dy < 2; ++dy)*, and applied 2 in *for (x = 0; x < width; x++)*. Especially in *for (int dx = -1; dx < 2; ++dx)*, *for (int dy = -1; dy < 2; ++dy)*, all operations Can be exchanged into constant(e.g. i*2 for 3 can be exchanged into 2,4,6). This significantly reduces code execution time, as demonstrated in Figure 1. The last optimization method is reducing the branch. The conditional branch is used because it can disregard the filter beyond the boundary. It should be done at run-time. Because the conditional branch cannot be predicted at compile time. This means that it can be hard to process the instructions (with conditional branch) in parallel. It implies that SS(Super Scaler) cannot enhance its performance on conditional branches. Therefore, reducing the conditional branch will help improve the performance of SS. We can achieve these in two ways. The First is to **limit the for-loop** to prevent it from crossing

the image boundary. This is good for speed up and intuitive. However, it may not produce the correct output. So, I used the last metric. This is **zero-padding**. Zero-padding is the technique that fills the boundary with zeros. However, we can find there is no significant improvement with adding zero-padding in Figure 1. Presumably, this is because the cost of zero-padding is similar to the cost of using a conditional branch. I think that is why there has not been an improvement.

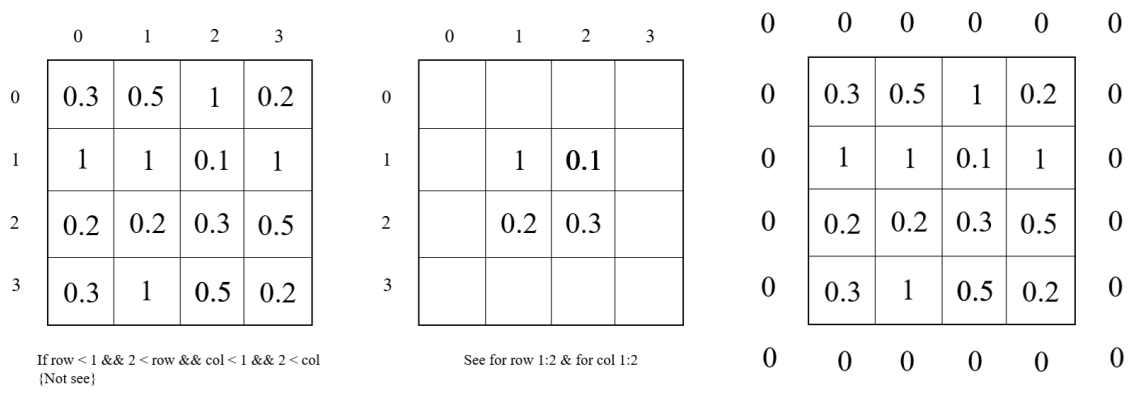

Figure 2. *Three methods of disregard the filter beyond the boundary. On the left is the method using a conditional branch, in the middle is the method of limitation the for-loop, and on the right is zero-padding.*