202111152 이하임

1. Implementation results

bmp size					Statistics (Speed up)			
128	**256**	**512**	**768**	**1024**	**Max**	**Average**	**SD**	**Median**
2.9306	2.9073	2.9027	2.9179	3.3577	***3.3577***	3.0032	0.39685	2.9179

The results were obtained using the default filter(edge_filter). I conducted tests on various image sizes ranging from 128x128 to 1024x1024, yielding a maximum value of 3.357658 and an average value of 3.003222. It was confirmed that the speed-up for the 1024x1024 image was generally higher.

filter	bmp size							
	128	**256**	**512**	**768**	**1024**	**Max**	**Average**	**Median**
edge	2.9306	2.9073	2.9027	2.9179	3.3577	***3.3577***	3.0032	2.9179
identity	2.6057	2.6414	2.6509	2.6425	3.0281	3.0281	2.6425	2.6426
sharpen	2.6492	2.6555	2.6597	2.6769	3.0873	3.0873	2.7457	2.6597
gaussian	2.6772	2.6986	2.6462	2.6502	3.1436	3.1436	2.7632	2.6772

The results obtained by running with the given filter are as shown above. The speed improvement was confirmed to be approximately threefold.

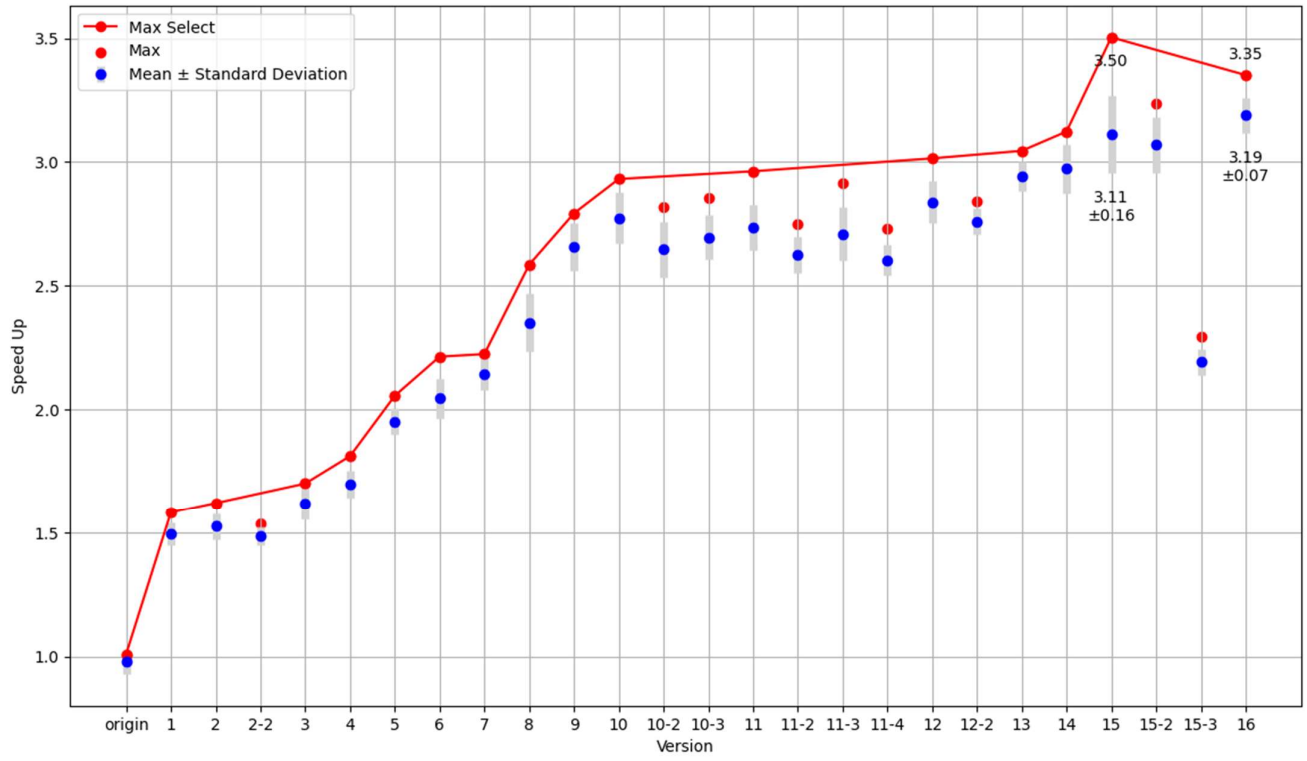
2. Optimization approach

10 times of ‘speed up’ measurements were conducted on a 1024x1024 bmp file, and the results were as follows. Including the original, there are a total of 26 versions, and version numbers were assigned based on performance improvements over previous versions. Unselected models were assigned numbers such as -2 and -3.

Version	optimization approaches	Statistics (Speed up)		
		Max	**Average**	**SD**
Origin		1.009045	0.980195	0.049906
1	Improve cache locality	1.58132	1.49473	0.04408
2	Inline the convolution function	1.61997	1.52437	0.05326
2-2	Unroll loops by a factor of 4	1.5371	1.48531	0.036
3	Unroll the for loop as much as possible	1.69964	1.6173	0.0638
4	Float to int to avoid floating-point operations.	1.81087	1.69634	0.05616
5	Improve cache locality & Reduce condition checks	2.05517	1.95102	0.052
6	Optimize the final assignment part.	2.21329	2.04367	0.08049
7	Unroll loops by a factor of 4.	2.22362	2.14385	0.06594
8	Unroll the loop for dx (-1 ~ 1)	2.58485	2.35076	0.11748
9	Unroll the loop for dy (-1 ~ 1)	2.79264	2.65756	0.09476
10	Remove redundant declarations	2.93138	2.7734	0.1026
10-2	Perform operations on the same variable consecutively.	2.8179	2.64658	0.11266
10-3	Simplify bitwise operations	2.85599	2.6956	0.08831
11	code motion (move x and y calculations outside the loop)	2.96269	2.73439	0.09318
11-2	code motion (move array index calculations)	2.74709	2.62536	0.07507
11-3	Explicitly write out the filter loop	2.91415	2.70876	0.10826
11-4	Convert the filter array to a pointer	2.73255	2.60307	0.06135

12	Explicitly write out the filter index	3.01484	2.83744	0.08548
12-2	Access using pointers	2.83923	2.75975	0.05315
13	Simplify operations using the ++ operator	3.04534	2.94181	0.05803
14	Remove one condition inside the for loop using break.	3.12332	2.97254	0.1006
15	Eliminate one internal loop	***3.50432***	3.11151	0.15636
15-2	Change values to arrays	3.23435	3.07004	0.11274
15-3	Use calloc for zero padding	2.29224	2.19168	0.05243
16	Reduce condition checks	3.35163	***3.18968***	***0.07125***

After 16 updates, the 15th model exhibited the highest max value; however, considering the average value and standard deviation, the 16th model was deemed the best.



The graph above includes the results of all attempts. It shows the average, maximum values, and error bars for each model, with the values for the two best models indicated.

Optimization Approaches Summary	**Times** (Select Model)
Loop Unrolling and Optimization	5
Cache Locality and Condition Checks	3
Inlining and Function Optimization	1
Data Handling and Conversion	2
Code Motion	1
Simplifying and Optimizing Operations	4
Total	16

The summary of these attempts is as follows: Initial optimizations, such as improving cache locality and inlining functions, demonstrated significant improvements. Subsequently, unrolling the for loops yielded substantial performance gains.