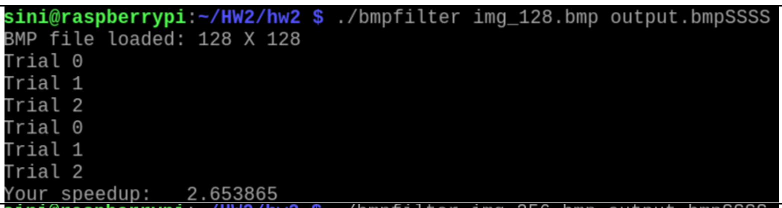
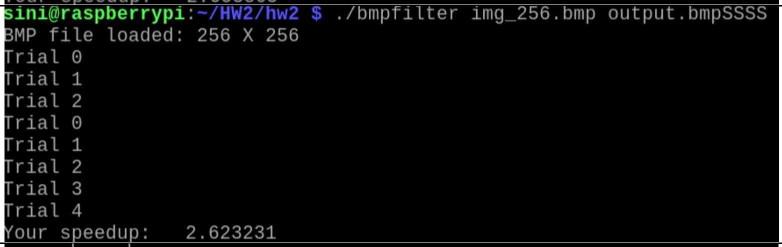
2024 System Programming HW2 Report

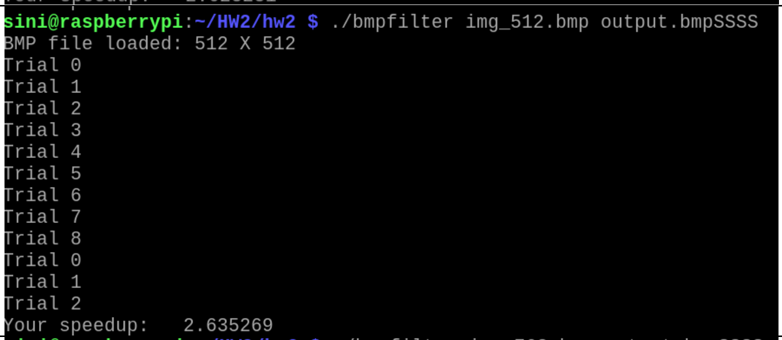
202211130 윤신이

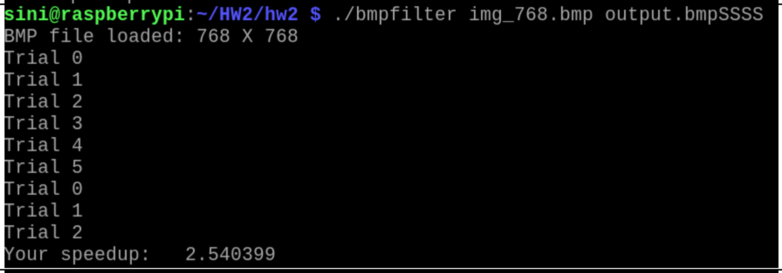
[Section 1 – Implementation Results]

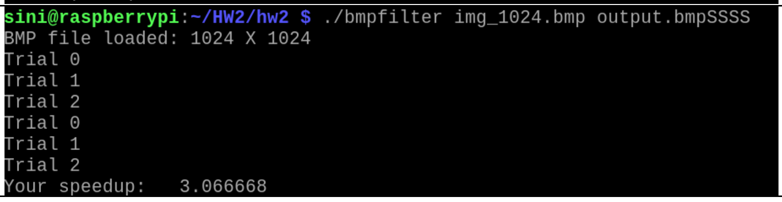
BMP	Result	Speedup

128 bmp	 `sini@raspberrypi:~/HW2/hw2 $ ./bmpfilter img_128.bmp output.bmpSSSS`
	`BMP file loaded: 128 X 128`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Your speedup:   2.653865`
		2.65

256 bmp	 `sini@raspberrypi:~/HW2/hw2 $ ./bmpfilter img_256.bmp output.bmpSSSS`
	`BMP file loaded: 256 X 256`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 3`
	`Trial 4`
	`Your speedup:   2.623231`
		2.62

512 bmp	 `sini@raspberrypi:~/HW2/hw2 $ ./bmpfilter img_512.bmp output.bmpSSSS`
	`BMP file loaded: 512 X 512`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 3`
	`Trial 4`
	`Trial 5`
	`Trial 6`
	`Trial 7`
	`Trial 8`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Your speedup:   2.635269`
		2.64

768 bmp	 `sini@raspberrypi:~/HW2/hw2 $ ./bmpfilter img_768.bmp output.bmpSSSS`
	`BMP file loaded: 768 X 768`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 3`
	`Trial 4`
	`Trial 5`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Your speedup:   2.540399`
		2.54

1024 bmp	 `sini@raspberrypi:~/HW2/hw2 $ ./bmpfilter img_1024.bmp output.bmpSSSS`
	`BMP file loaded: 1024 X 1024`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Trial 0`
	`Trial 1`
	`Trial 2`
	`Your speedup:   3.066668`
		3.07

[Section 2 – Optimization Approach]

In the submitted code, the for-loop in the `convolution` function, which has a known number of iterations, was unrolled and written explicitly. Additionally, variables that were previously accessed using indices were modified to use pointers, and frequently used values were assigned to variables to prevent redundant calculations. The data types of `r`, `g`, and `b` were changed from `double` to `float`.

In the `filter_optimized` function, complex processes such as redundant calculations and memory allocation using `malloc` were replaced with pointer-based operations. The order of the for-loops was also modified to consider memory and cache access patterns.

Furthermore, an attempt was made to embed the entire `convolution` function within the `filter_optimized` function to eliminate function call overhead. While this approach resulted in a significant speedup in the WSL environment, it did not yield substantial improvements on the Raspberry Pi platform.

The strategy that I found most effective in improving speedup was simplifying or eliminating for-loops. Significant speedup improvements were observed when the for-loop in the `convolution` function was unrolled and when the complex for-loop processes in the `filter_optimized` function were optimized.

[Overall Implementation Summary]

1. Loop Unrolling: The loop in the convolution function was explicitly written out.

2. Pointer Utilization: Variables previously accessed via indices were modified to use pointers.

3. Redundant Calculation Elimination: Frequently used values were assigned to variables to avoid redundant calculations.

4. Data Type Optimization: The data types of r, g, and b were changed from double to float.

5. Memory and Cache Optimization: The order of the for-loops in filter_optimized was adjusted to optimize memory and cache access patterns.

6. Function Call Elimination: Although not present in the submitted code, the integration of the convolution function into the filter_optimized function to avoid function call overhead was considered and implemented during the project. This approach proved to be more effective in the WSL environment than on the Raspberry Pi.