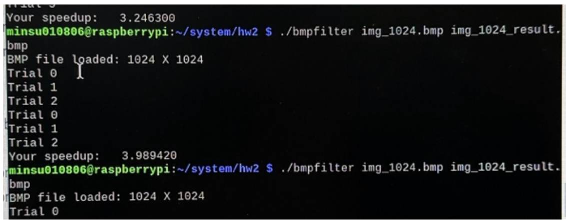
System Programming HW2 report

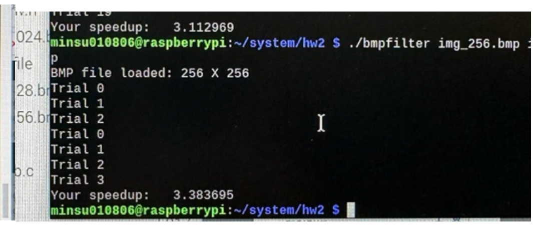
202011194 전민수

1. Implementation results

Image file	original	modified	speedup		Average speed up
img_128.bmp	1	3.2463	3.2463		3.5097144
img_236.bmp	1	3.38369	3.38369		
img_512.bmp	1	3.41295	3.412946		Max speed up
img_768.bmp	1	3.5158	3.515796		3.98984
img_1024.bmp	1	3.98984	3.98984		

*proof





I confirmed that the code runs correctly on a Raspberry Pi environment and produces images as expected.

2. Optimization approach

*I learned and used the CLAMPING technique from open-source filter-related sources..

A. Malloc: In the original code, malloc and free were used for dynamic memory allocation and deallocation in each loop, which I thought caused significant overhead due to interactions with the operating system and memory management system. Therefore, I allocated static variables and wrote the code to reuse these static variables as needed.

B. B. Improved inefficient loop structure: The original code seemed to not properly utilize spatial locality in the calculations within the loop, so I changed the order of the loops.

*After this process, I achieved approximately 1.5 times speedup.

C. Since I thought floating-point operations would have a heavy load, I pre-converted the filter to int for calculations.

*After this process, I achieved approximately 1.7 times speedup.

D. Loop unrolling: It seemed that each pixel operation was processed in a single statement within the loop for all neighboring pixels, so I applied loop unrolling. This aimed to reduce the load of checking loop conditions and branch prediction.

*I wanted to apply this concept learned in class, but I faced practical difficulties, so I searched for and learned filter loop unrolling on the internet.

*After this process, I achieved approximately 2.9 times speedup.

E. Boundary condition handling: I thought it would be efficient to handle the boundary pixels first and then process the pixels not on the boundary.

*After this process, I achieved approximately 3.1 times speedup.

F. Processing 4 pixels at once: By reviewing the code, it seemed inefficient to call the function while x increments by 1, so I hardcoded it solely for speedup.
*After this process, I achieved approximately 3.7 times speedup

Additionally, I achieved max speedup by changing various settings related to cache, memory, and minor settings mentioned in lecture materials.