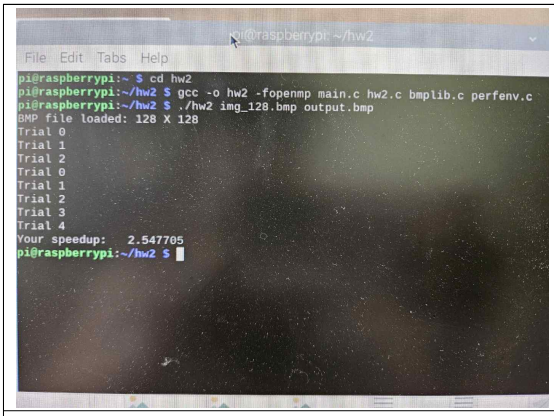
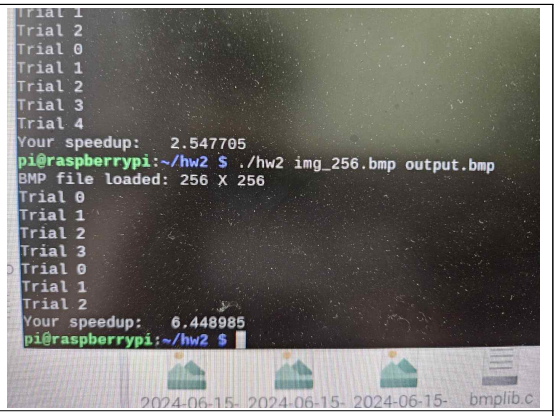
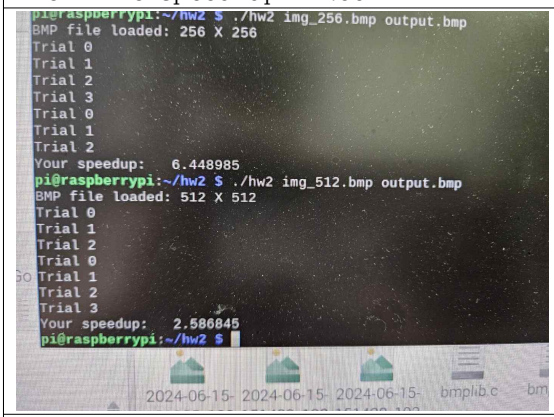
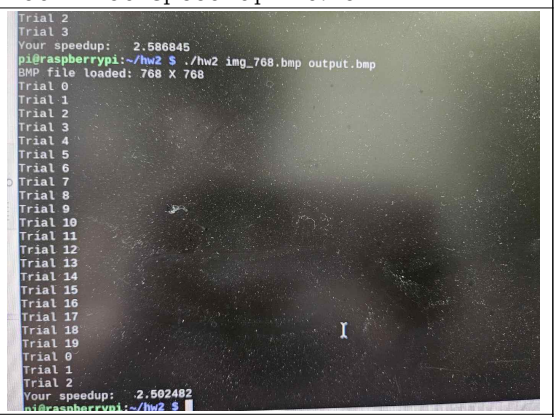
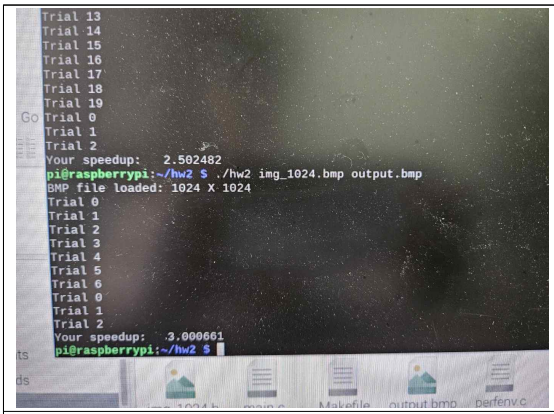
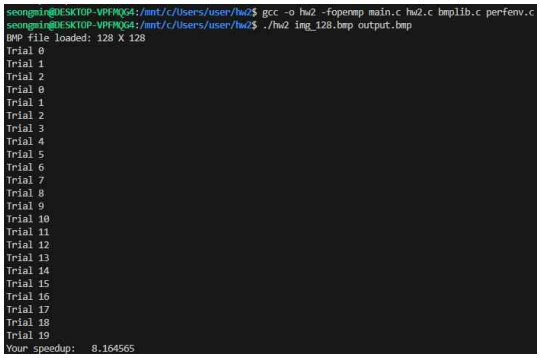
System programming report

202111179 Ju Seong Min

1. Implementation result

First, when the existing hw2.c code was executed for each bmp size, the speedup value was identified. As a result of the execution, it showed an average speedup of about 1.00. And after the code optimization, the code was first executed on the desktop in the **Wsl environment. It showed 8.16** times the speedup performance based on 128 x 128 images, and ran with the corresponding code on Raspberry Pi. The execution time for each image is shown in the table below. It showed an average speed-up performance of **3.42 times in five images.**

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>128 x 128 speed up : 2.55</td><td>256 x 256 speed up : 6.45</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>512 x 512 speed up : 2.59</td><td>768 x 768 speed up : 2.50</td></tr>
<tr><td></td><td></td></tr>
</table>

	
1024 x 1024 speed up : 3.00	Wsl 기준 128 x 128 speed up : 8.16

2. Optimization approach

1. Removing Unnecessary Memory Allocation

In the case of the original code, memory is dynamically allocated and released for each pixel, which is a costly task, so minimizing this can greatly improve the performance. Therefore, the memory was not dynamically allocated to each pixel, but the result was directly allocated to the output image. This improved the performance by reducing the overhead of memory allocation and release.

2. Loop Order for Memory Access

The original code loops x first and y, but I looped y first and then x. Sequential memory access increases cache utilization, allows faster access, and uses locality to access continuous memo locations, enabling efficient processing.

3. Using Static Allocation for Pixel

Static memory allocation is faster than dynamic memory allocation. In other words, the performance was increased by reducing the overhead of dynamic memory management by using static allocation unlike the previous one.