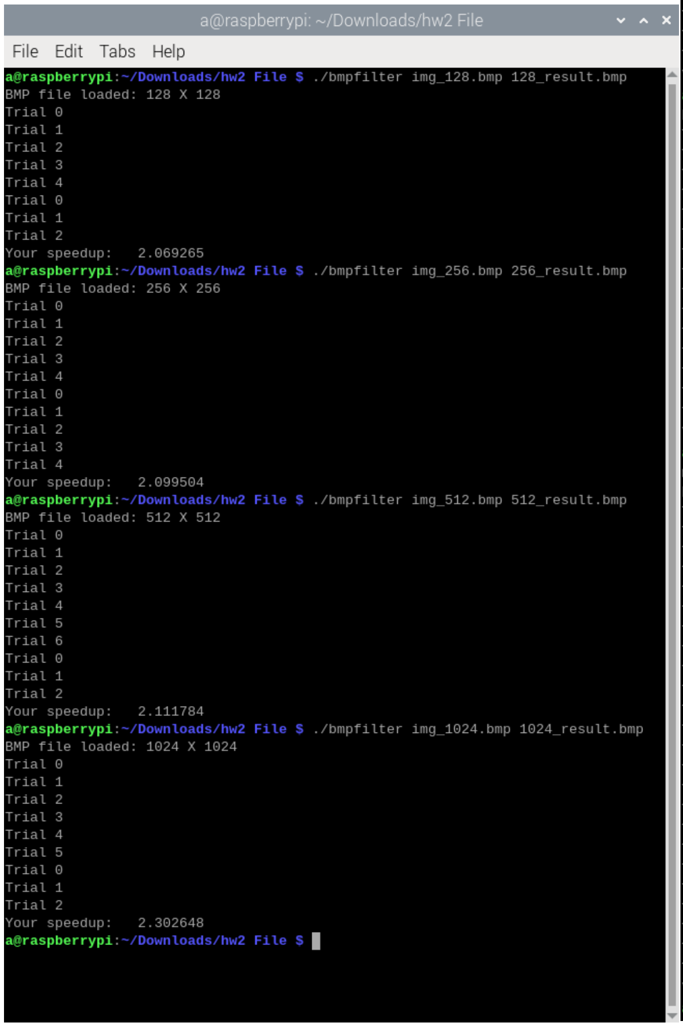
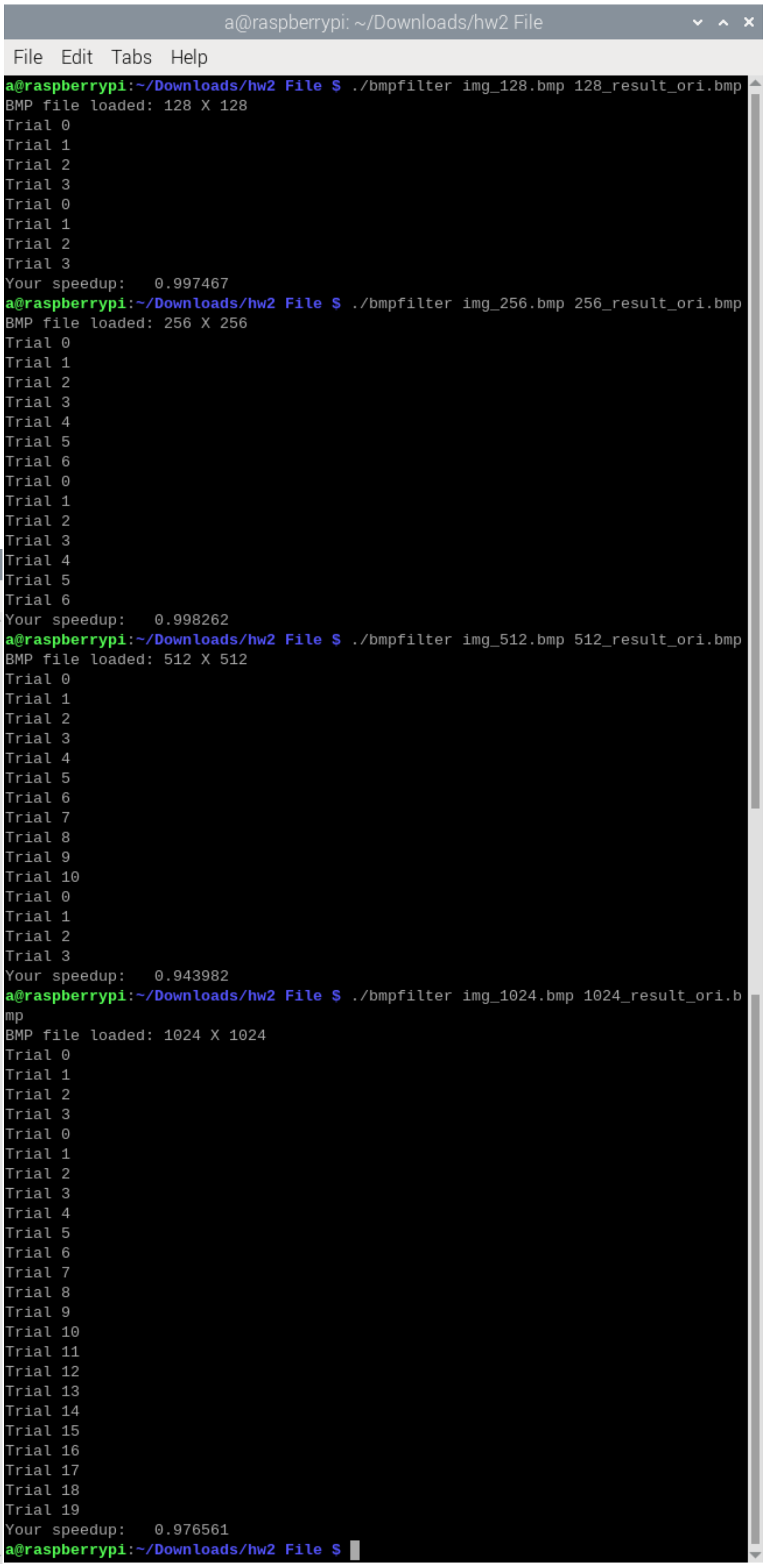
Assignment #2

201811058 박건희

Implementation Result:



```
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_128.bmp 128_result.bmp
BMP file loaded: 128 X 128
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 0
Trial 1
Trial 2
Your speedup:   2.069265
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_256.bmp 256_result.bmp
BMP file loaded: 256 X 256
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Your speedup:   2.099504
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_512.bmp 512_result.bmp
BMP file loaded: 512 X 512
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 0
Trial 1
Trial 2
Your speedup:   2.111784
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_1024.bmp 1024_result.bmp
BMP file loaded: 1024 X 1024
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 0
Trial 1
Trial 2
Your speedup:   2.302648
a@raspberrypi:~/Downloads/hw2 File $ 
```



```
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_128.bmp 128_result_ori.bmp
BMP file loaded: 128 X 128
Trial 0
Trial 1
Trial 2
Trial 3
Trial 0
Trial 1
Trial 2
Trial 3
Your speedup:   0.997467
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_256.bmp 256_result_ori.bmp
BMP file loaded: 256 X 256
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Your speedup:   0.998262
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_512.bmp 512_result_ori.bmp
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 9
Trial 10
Trial 0
Trial 1
Trial 2
Trial 3
Your speedup:   0.943982
a@raspberrypi:~/Downloads/hw2 File $ ./bmpfilter img_1024.bmp 1024_result_ori.b
mp
BMP file loaded: 1024 X 1024
Trial 0
Trial 1
Trial 2
Trial 3
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 7
Trial 8
Trial 9
Trial 10
Trial 11
Trial 12
Trial 13
Trial 14
Trial 15
Trial 16
Trial 17
Trial 18
Trial 19
Your speedup:   0.976561
a@raspberrypi:~/Downloads/hw2 File $ 
```

Img_128 optimized

Img_128 original

Img_256 optimized

Img_256 original

Img_512 optimized

Img_512 original

Img_1024 optimized

Img_1024 original

The first two images show the result panel from raspberry pi. The left one is with optimized code by me, and the right one is with given code. As shown, optimized code is relatively faster. Next eight images show the results of actual image file run by code, ordered by optimized code, and original code, respectively. 4 sets of all images are shown. The image below is a chart of the actual improvement rate. For calculating speedup, it is calculated by time taken by baseline implementation divided by optimized implementation, thus higher value of speedup means better performance in speed. All 4 results have showing more than 40% improvement in speed, which seems to be a substantial progress.

	128px	256px	512px	1024px
Original	0.997467	0.998262	0.943982	0.976561
Optimized	2.069265	2.099504	2.111784	2.302648
Improvement Rate	48.20%	47.55%	44.70%	42.41%

Optimization Approach:

1. **Memory access optimization**
   a. **Strategy:** Before the convolution operation, load the neighborhood of a 3x3 pixel into local variables. Then, do the convolution operation on these local variables, rather than directly accessing the memory.
   b. **Result:** The optimized code runs faster than the baseline by lowering the amount of memory accesses and increasing cache use. Around 20% improvement was made here.
2. **Avoiding dynamic memory allocation**
   a. **Strategy:** Rather than dynamically allocating memory for each pixel, pre-allocate the output buffer before beginning the convolution process and send the results directly to the buffer.
   b. **Result:** Writing the results directly to the buffer, which has already been pre-allocated rather than dynamically allocating memory for every pixel, improving speed of total calculations. Around 30% improvement was made here.
3. **Loop unrolling**
   a. **Strategy:** Conduct several operations inside a single loop iteration by unrolling the inner loops of the convolution process to decrease the number of iterations.
   b. **Result:** In this case, the 3x3 convolution's limited number of iterations did not result in a dramatic speedup.
4. **Parallelizing independent operations**
   a. **Strategy:** Find and streamline separate processes that are running concurrently without adding further overhead.
   b. **Result:** Not much was improved, since the compiler runs the program efficiently.