System programing Assignment 2 report

201911012 공영재

- Implementation results

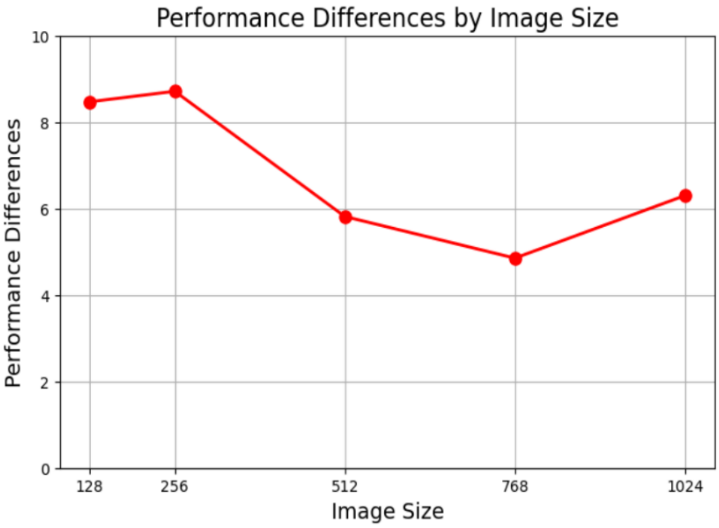
Image size 128: 8.469134

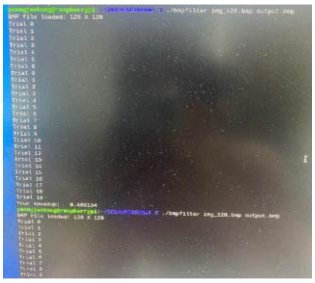
Image size 256: 8.712017

Image size 512: 5.814758

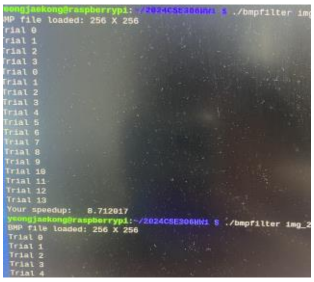
Image size 768: 4.852342

Image size 1024: 6.300120

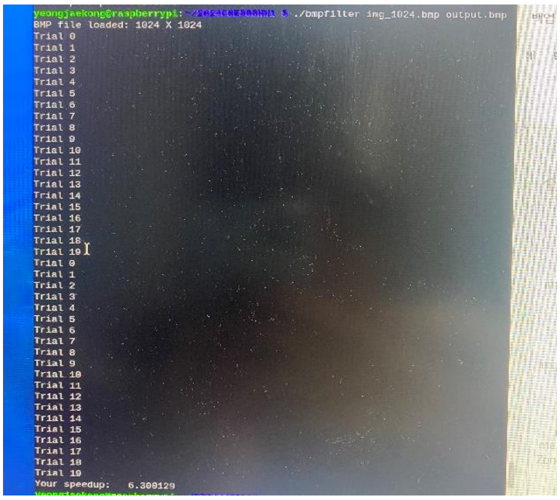




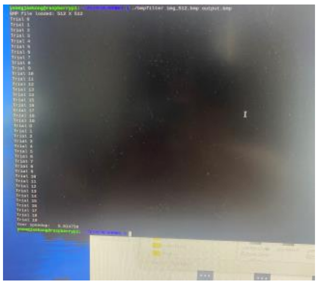
```
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_128.bmp output.bmp
BMP file loaded: 128 X 128
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 7
Trial 8
Trial 9
Trial 10
Trial 11
Trial 12
Trial 13
Trial 14
Trial 15
Trial 16
Trial 17
Trial 18
Trial 19
Your speedup:   8.469134
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_128.bmp output.bmp
BMP file loaded: 128 X 128
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 7
Trial 8
Trial 9
```



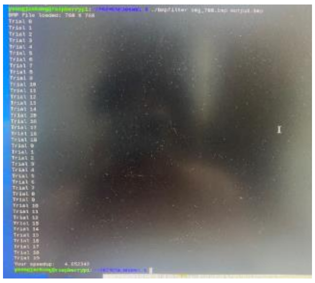
```
eongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img
MP file loaded: 256 X 256
rial 0
rial 1
rial 2
rial 3
rial 0
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6
Trial 7
Trial 8
Trial 9
Trial 10
Trial 11
Trial 12
Trial 13
Your speedup:   8.712017
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_2
BMP file loaded: 256 X 256
Trial 0
Trial 1
Trial 2
Trial 3
Trial 4
```



```
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_1024.bmp output.bmp
BMP file loaded: 1024 X 1024
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
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:   6.300129
```



```
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_512.bmp output.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 11
Trial 12
Trial 13
Trial 14
Trial 15
Trial 16
Trial 17
Trial 18
Trial 19
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:   5.814758
yeongjaekong@raspberrypi:~/2024CSE306HW1 $
```



```
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ ./bmpfilter img_768.bmp output.bmp
BMP file loaded: 768 X 768
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
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:   4.852342
yeongjaekong@raspberrypi:~/2024CSE306HW1 $
```

Optimization approach

The challenge is to optimize the program that processes images by applying 3x3 filters to images in BMP formats. As a key technology for optimization, Loop Unrolling was used to reduce loop overhead, and it also minimized conditional statements to prevent branch prediction failure. In addition, fast memory access is possible by utilizing pointer operation, and finally, a method of increasing cache utilization by using cache-friendly memory access patterns was used. The Inline function was tried and failed. The content below is an analysis of it. (p.s. There are finely modified codes in the middle, so the degree of improvement is not continuous. The overall result was calculated by Raspberry Pi, but the degree of speedup by optimization method was calculated by local environment WSL.)

1. Loop Unrolling:
In convolution(), the for statement was turned by 3 spaces for each x,y, and when it unrolled, it was possible to confirm the speed improvement. If the for statement is not large, it was changed to unroll them. Through this, it was intended to reduce loop overhead and improve ILP.

2. Minimize conditional statements:
When I first wrote the code, I used a lot of conditional statements inside the convolution function, but when I reduced the conditional statements, I checked that the speed was increasing and corrected the code in the direction of not using it as much as possible. In the convolution, the conditional statements used in the three situations of x-1, x, and x+1 were changed to be used only once. Loop rolling and conditional states were mixed and coded, so the combined speedup=1.0 -> 2.05~

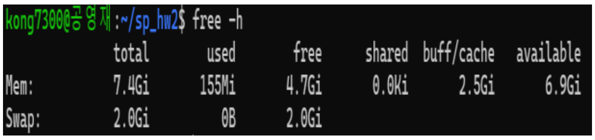
3. Pointer operation:
When the existing constant operation was changed to a pointer operation, the speed was increased, and all possible operations were changed to operations using pointers. speedup=2.2~ -> 2.6~

4.Cache-friendly memory access patterns: Used with loop unrolling, we designed the convolution in filter_optimized() to operate four times, +1 in the x-direction, inducing it to perform cache-friendly on the code. speedup=2.0~ -> 2.1~

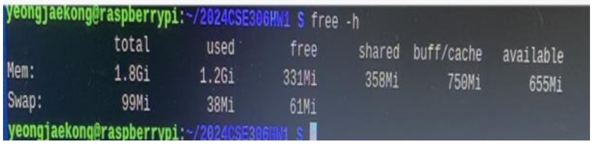
- Failure case

I thought that if I used an Inline function in convolution(), the speed would increase, but I couldn't feel the difference than I thought. In particular, in the case of the Raspberry Pi environment, the speed even increased when inline was removed. This may be due to the nature of the in-line function. In-line functions operate by copying and inserting the function code itself instead of calling the function. This reduces function call overhead, but increases the code size. It was inferred that in an environment with small memory, the problem of insufficient memory due to an increase in code size could have been a bigger bottleneck than the function call overhead.

In addition, as a strange point, when experimenting with WSL in a local PC environment, there was a problem that the deviation became too large when it was executed on Raspberry Pi. Compared to local PCs, Raspberry Pi uses a low-power ARM-based CPU, and the cache memory and RAM size are small. A small cache memory slows down the CPU because it has to access memory frequently. Also, a small RAM size can cause swapping due to insufficient memory. As a result, the code worked well on local PCs thanks to the large cache and RAM, but on Raspberry Pi, I think the limited resources caused performance bottlenecks and increased the deviation. Below is the memory difference between two.



```
kong7300@공영재:~/sp_hw2$ free -h
               total        used        free      shared  buff/cache   available
Mem:           7.4Gi       155Mi       4.7Gi       0.0Ki       2.5Gi       6.9Gi
Swap:          2.0Gi          0B       2.0Gi
```



```
yeongjaekong@raspberrypi:~/2024CSE306HW1 $ free -h
               total        used        free      shared  buff/cache   available
Mem:           1.8Gi       1.2Gi       331Mi       358Mi       750Mi       655Mi
Swap:           99Mi        38Mi        61Mi
yeongjaekong@raspberrypi:~/2024CSE306HW1 $
```