ESL HW1 Report

111064511 陳煜清

Link to the GITHUB repository

https://github.com/YuChin-Chen/ESL_hw1/tree/master

Introduction

In this homework, we implemented a median and mean filter in SystemC module. This filter applies median filter to the input image first and then mean filter after that. If we apply median filter to the input image and store the temporary result to a memory and then read the temporary result from the memory and apply mean filter again, it will be very wasting for hardware cost since we need another memory to store the temporary result. Hence, our implementation does not use a memory to store the temporary result. In addition, there are two parts in our implantation. In the first part, we do not add any buffer into the filter. Due to this reason, we can see the filter repeat taking same data from the memory and this is not efficiency. To improve the efficiency, we add buffer into the filter in second part and we can see that the number of pixel transfer becomes to one fifth. The details of implementation will be shown in the following section.

Implementation

Figure 1 shows the sequence for processing each pixel.

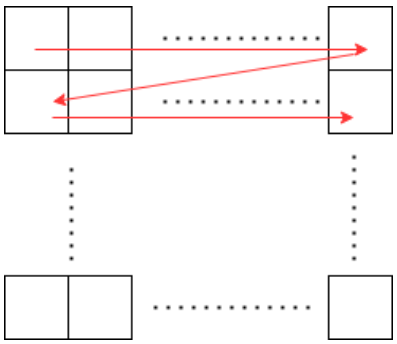

Figure 1.

Part 1

Assume the filter is processing pixel A, the filter will compute the result of median filter for A and its neighbors first, and then applies mean filter to them. After that, the filter moves on to processing next pixel and does not buffer any data computed before.

Part 2

I add two types of buffers in my design.

- Input data buffer (9 pixels)

- Median filter buffer (9 pixels)

There are two situations.

First situation, the pixel we are processing is at the head of row. Assume the filter is processing pixel A, the filter will compute the result of median filter for A and its neighbors first by the sequence shown in figure 2. When applying median filter to pixel a, the filter stores all of the input data into input data buffer so that it does not need to fetch the same data when processing pixel b. When the filter is processing pixel b, it only fetches three new data which are not used when processing pixel a. At the same time, the filter removes the data which are not needed when processing pixel c from input data buffer and also put the new data into the buffer. It follows the same logic to processing remain pixels. All the results of median filter are also stored in the median filter buffer during the above process.

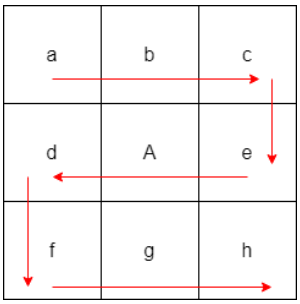

Figure 2.

Second situation, the pixel we are processing is not at the head of row. Some results of median filter have already in the median filter buffer as shown in figure 3.

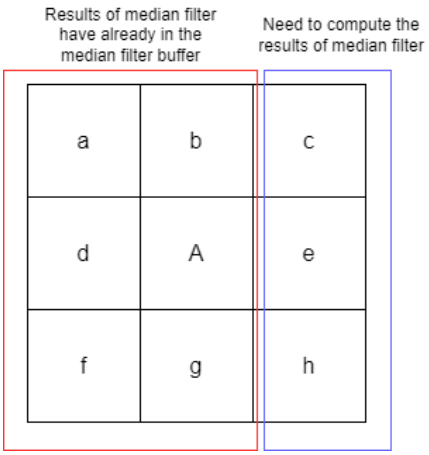

Figure 3.

We compute the result of pixel c after median filter at first. At this time, we need fetch c and all of its neighbors from memory and stored them into the input data buffer. After we get the result of median filter for pixel c, the filter store it into the median filter buffer and remove the data does not needed to use. Then the filter moves on to get the result of median filter of next pixel e and then pixel h.

I add one control signal into my filter to indicate the first situation.

Features

- Two types of data buffer in my design.
- Additional control signal to indicate the start of one row.

Result

(a) Original images.

(b) Images after adding noise.

(c) Images after doing filtering.

Image	Without buffers	With buffers
Lena	5308416	987648
Peppers	21233664	3941376

(d) Number of pixel transfer

Discussion and Conclusion

- Part 1 needs 81 memory transfers for each pixel.
- Part 2 needs 33 memory transfers for each pixel in situation one and needs 15 memory transfers in situation two.
- Lenna image has 256 pixels in one row.
  - Part 1 needs $81 \times 256 = 20736$ memory transfers in one row.
  - Part 2 needs $15 \times 255 + 33 = 3858$ memory transfers in one row.
  - Part 2 has $\frac{20736-3858}{20736} \times 100\,\% = 81.39\,\%$ memory transfer reduction of part 1.
- Peppers image has 512 pixels in one row.
  - Part 1 needs $81 \times 512 = 41472$ memory transfers in one row.
  - Part 2 needs $15 \times 511 + 33 = 7698$ memory transfers in one row.
  - Part 2 has $\frac{41472-7698}{41472} \times 100\,\% = 81.44\,\%$ reduction transfer reduction of part 1.

I think my design on situation 1 is too complicated and it does not get much benefit because situation 1 is not the major part during the filtering process. This complicated design consume more hardware resource for control unit but it is not worth.