Lab 2-Systolic Array Matrix Multiplication

An Lihua
2022131001
Shanghaitech University, SIST
anlh2022@shanghaitech.edu.cn

***Abstract*—Systolic array is a concurrently popular concept in acceration of matrix multiplication (MAC). This lab aims to design a matrix multiplication module based on systolic array, and apply it to 2D convolution using GEMM.**

In this lab, I have completed the im2col, PE, and systolic_array modules and passed all the test cases.

***Index Terms*—MAC, systolic array, im2col, GEMM**

I. Introduction

Matrix multiplication is a popular kernel in high-performance scientific computing workloads. NVIDIA now build GPU hardware that excels at the task of performing matrix multiplication. In contemporary usage, matrix multiplication hardware has even made it into the core of the Google Tensor Processing Unit (TPU). This lab's objective is to design a matrix multiplication module based on systolic array, and apply it to 2D convolution using GEMM.

The top diagram of this system is given in Fig. 1. It is mainly consists of memory, im2col, systolic array and two buffers. The inputs of the system are image and weights (convolution kernels), which are stored in the memory. The im2col module is used to do conversion on the image in order to do convolutions using GEMM, a proper organization of read and write address is needed. After that, the systolic_array module instantiate a weight-stationary systolic array composed by PE (Processing Elements), it receives the activation X from X_buffer and output results Y to Y_buffer. The PE array in systolic module is designed in parallel, so shift registers FIFO are essential. The results in Y_buffer will also be send to the memory.

All the input and output data are stored in a 32-bits memory, the base address of each parameter is given in Table I. Note that the image, weights and outputs matrix are stored in row-major order, while the im2col results are stored in column-major order.

TABLE I
Base addresses of each parameter.

Name	Value	Description
IMG_BASE	0x00000000	image start address
WEIGHT_BASE	0x00001000	weights start address
IM2COL_BASE	0x00002000	im2col start address
OUTPUT_BASE	0x00003000	output start address

II. im2col + GEMM

Fig. 2 is an example of im2col operation. When performing a 2D convolution, data in the convolution window is stored discontinously in memory, which is not efficient for computation. The im2col operation expand each convolution window on the image into a column of matrix X, while keeping the operation of 2D convolution is equivalent to the mutiplication of matrix W and matrix X, which has accerated the computing speed and reduce memory access time.

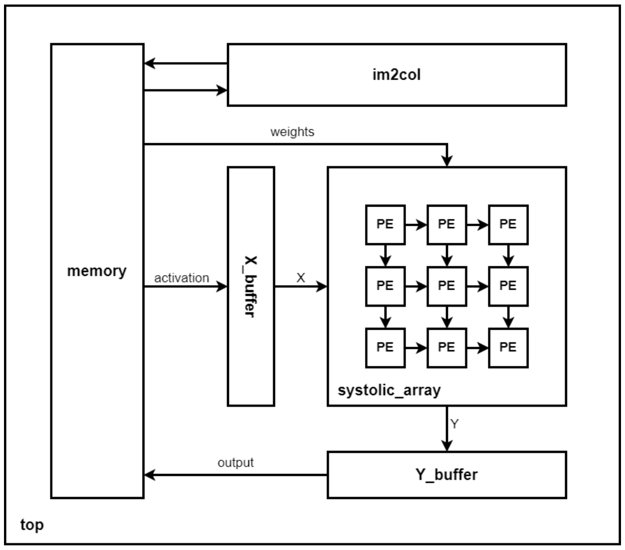


Fig. 1. System top diagram.

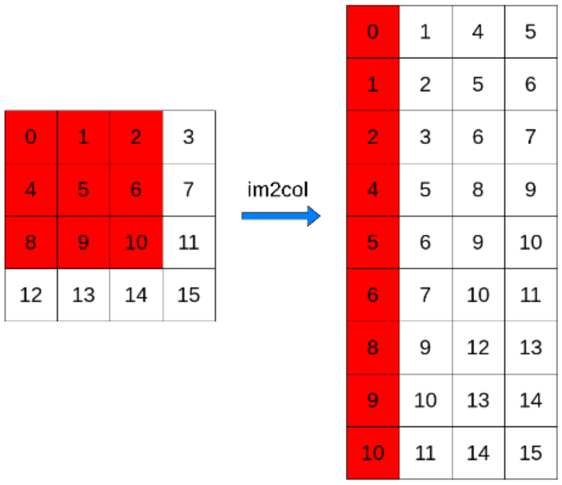


Fig. 2. im2col operation example.

However, in this system design, padding with zero is under consideration, a proper organization of the read/write address is important. In this lab, I use the concept of absolute address and relative address to represent the read address, while write address is simply given in order.

Fig. 3 shows basic concept of im2col design. For example, a 3x3 FILTER will go through a 4x4 IMAGE with 0-padding. The base address represent the location of the center pixel of FILTER, is can only reach the blue region in IMAGE. The filter counter represents the relative address of read data. Then the absolute addresss of read data can be get according to the filter counter and base address, the mapping of read address is shown in Fig. 4.

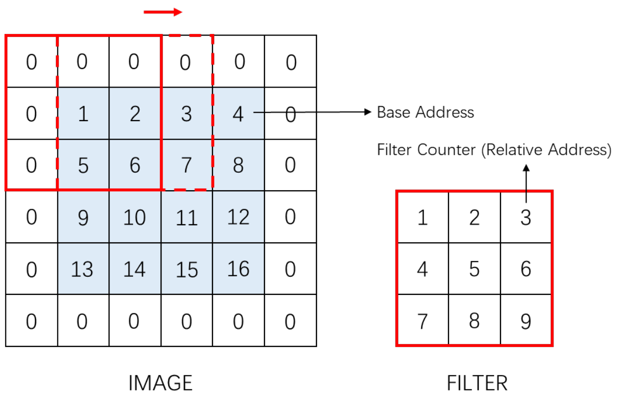
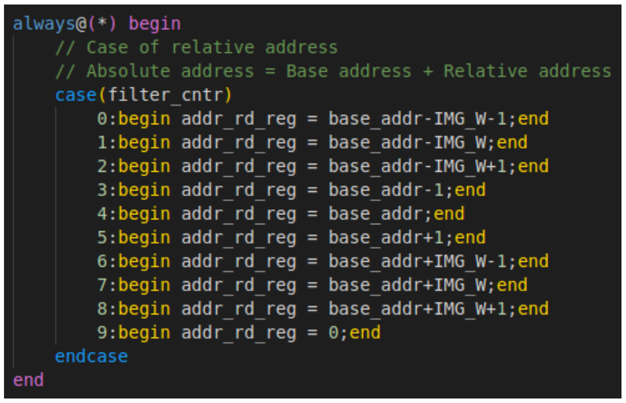


Fig. 3. Diagram of address organization.



```
always@(*) begin
    // Case of relative address
    // Absolute address = Base address + Relative address
    case(filter_cntr)
        0:begin addr_rd_reg = base_addr-IMG_W-1;end
        1:begin addr_rd_reg = base_addr-IMG_W;end
        2:begin addr_rd_reg = base_addr-IMG_W+1;end
        3:begin addr_rd_reg = base_addr-1;end
        4:begin addr_rd_reg = base_addr;end
        5:begin addr_rd_reg = base_addr+1;end
        6:begin addr_rd_reg = base_addr+IMG_W-1;end
        7:begin addr_rd_reg = base_addr+IMG_W;end
        8:begin addr_rd_reg = base_addr+IMG_W+1;end
        9:begin addr_rd_reg = 0;end
    endcase
end
```

Fig. 4. Code of read address.

For the write data, when filter is in the IMAGE blue region, like the green window in Fig. 5, there is no need to output zeros. However, there are two special cases that need to output zero: (1) when filter is at the corner of IMAGE; (2) when filter is at the side of IMGAE.

For example, when the filter is at the left-up corner (red window) of IMAGE in Fig. 5, the write data should be zero when the filter counter is 1, 2, 3, 4 and 7, while the filter is at right side (yellow window) in Fig. 5, the write data should be zero when counter is 3, 6, 9. Table II shows how to determine write data with padding, it gives the critical filter counter to output **ZERO** or **Read**.

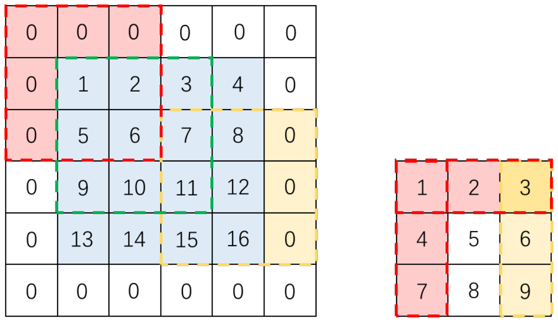


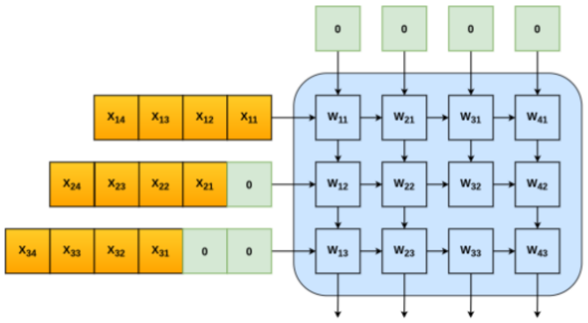
Fig. 5. Special cases.

TABLE II
DETERMINE WRITE DATA.

Location	Write = Read	Write = 0
Left-Up Corner	5, 6, 8, 9	-
Right-Up Corner	4, 5, 7, 8	-
Left-Down Corner	2, 3, 5, 6	-
Right-Down Corner	1, 2, 4, 5	-
Up Side	-	1, 2, 3
Left Side	-	1, 4, 7
Right Side	-	3, 6, 9
Down Side	-	7, 8, 9

III. Weight Stationary Systolic Array

The basic component in a systolic is the Processing Element (PE). As shown in Fig. 6, the input activation is streamed from left side with proper offset, which will be completed by adding shift registers FIFO to realize delay before input. After that, the shifted output will be aviliable at the bottom side.



0 0 0 0
X_{14} X_{13} X_{12} X_{11}
X_{24} X_{23} X_{22} X_{21} 0
X_{34} X_{33} X_{32} X_{31} 0 0
W_{11} W_{21} W_{31} W_{41}
W_{12} W_{22} W_{32} W_{42}
W_{13} W_{23} W_{33} W_{43}

Fig. 6. Weight stationary systolic array.

Next, I am going to describe my design of PE unit and the weight stationary systolic array with pipline in detail.

A. Processing Element (PE)

PE module is quite simple to realize. Like the diagram shown in Fig. 7, for a weight stationary systolic array, each PE stores a constant weight W and multiply it with input X_in then add to Y_in on every clock cycle, then set the new result to Y_out, which is the outoput of concurrent PE cell. At the same time, the output Y_out and X_out of current PE cell will be passed into the neighbor PE for next clock cycle's computation. This module is sequential, all the outputs are reset when the *rst* signal is pulled up.

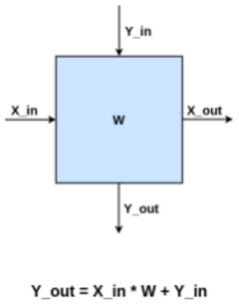


Fig. 7. PE unit.

B. Systolic Array

For the convience of connection with PE array, the input X and weight W are read and then store in new memory, which are KxN X_reg and (K+1)xN W_reg respectively.

It is worth to notice that X and Y need some shift registers FIFO to control the cycle of data arrival to get the correct result. FIFO is a cascade of a series of registers, which accepts one data in each cycle and simultaneously outputs one data. By adjusting the depth of FIFO can control the number of cycles data takes from entering to reading/writing out. The diagram of FIFO before input X is shown in Fig. 8, FIFO before output Y is similar.

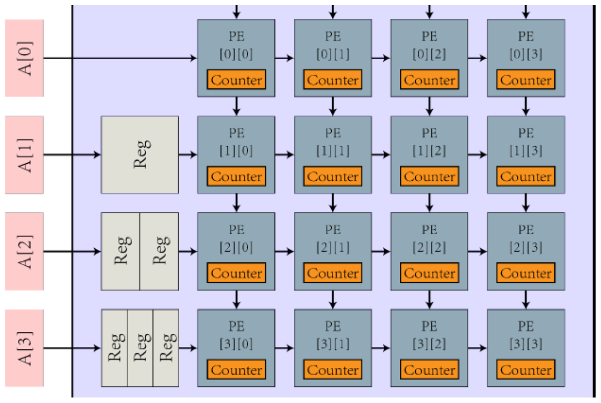


Fig. 8. Diagram of FIFO before input X.

Next is to generate a KxN PE array, the connection between each two adjacent PE is also shown in Fig. 8.

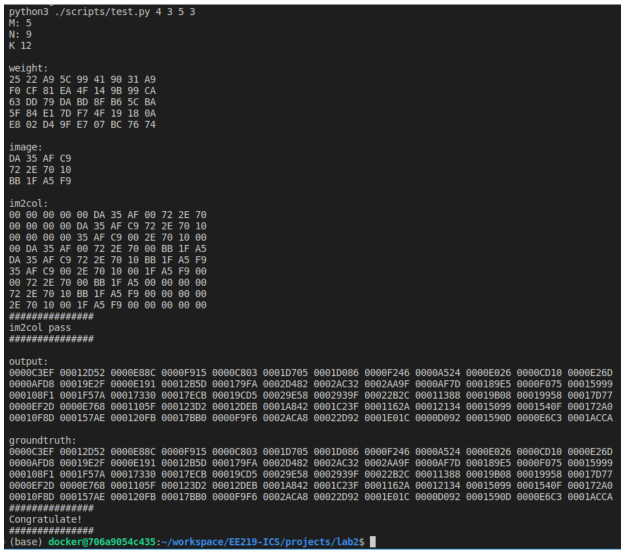
Output Y is also stored temporary in a Kx(N+1) memory. The additional one column is for the last output from last PE row, the same reason for the additional one row for input X.

C. valid & down

Given a counter *m_cntr* to record the clock cycles. When the counter reach N+K+1, it means the first sets of inputs have gone through the KxN PE array, so the signal **valid** can be pulled up and the first output Y can be get. When the counter reach N+K+M+1, it means all the M columns of input X have been sent into the systolic array and all the results can be output, so the signal **down** can be pulled up.

IV. Simulation Results

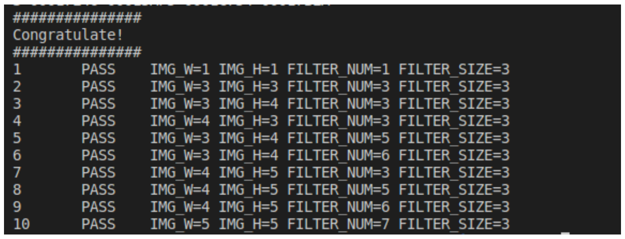
Fig. 9 is the single test result, both im2col and systolic array are correct!



```
python3 ./scripts/test.py 4 3 5 3
M: 5
N: 9
K 12

weight:
25 22 A9 5C 99 41 90 31 A9
F0 CF 81 EA 4F 14 9B 99 CA
63 DD 79 DA BD 8F B6 5C BA
5F 84 E1 7D F7 4F 19 18 0A
E8 02 D4 9F E7 07 BC 76 74

image:
DA 35 AF C9
72 2E 70 10
BB 1F A5 F9

im2col:
00 00 00 00 00 DA 35 AF 00 72 2E 70
00 00 00 00 DA 35 AF C9 72 2E 70 10
00 00 00 00 35 AF C9 00 2E 70 10 00
00 DA 35 AF 00 72 2E 70 00 BB 1F A5
DA 35 AF C9 72 2E 70 10 BB 1F A5 F9
35 AF C9 00 2E 70 10 00 1F A5 F9 00
00 72 2E 70 00 BB 1F A5 00 00 00 00
72 2E 70 10 BB 1F A5 F9 00 00 00 00
2E 70 10 00 1F A5 F9 00 00 00 00 00
###############
im2col pass
###############

output:
0000C3EF 00012D52 0000E88C 0000F915 0000C803 0001D705 0001D086 0000F246 0000A524 0000E026 0000CD10 0000E26D
0000AFD8 00019E2F 0000E191 00012B5D 000179FA 0002D482 0002AC32 0002AA9F 0000AF7D 000189E5 0000F075 00015999
000108F1 0001F57A 00017330 00017ECB 00019CD5 00029E58 0002939F 00022B2C 00011388 00019B08 00019958 00017D77
0000EF2D 0000E768 0001105F 000123D2 00012DEB 0001A842 0001C23F 0001162A 00012134 00015099 0001540F 000172A0
00010F8D 000157AE 000120FB 00017BB0 0000F9F6 0002ACA8 00022D92 0001E01C 0000D092 0001590D 0000E6C3 0001ACCA

groundtruth:
0000C3EF 00012D52 0000E88C 0000F915 0000C803 0001D705 0001D086 0000F246 0000A524 0000E026 0000CD10 0000E26D
0000AFD8 00019E2F 0000E191 00012B5D 000179FA 0002D482 0002AC32 0002AA9F 0000AF7D 000189E5 0000F075 00015999
000108F1 0001F57A 00017330 00017ECB 00019CD5 00029E58 0002939F 00022B2C 00011388 00019B08 00019958 00017D77
0000EF2D 0000E768 0001105F 000123D2 00012DEB 0001A842 0001C23F 0001162A 00012134 00015099 0001540F 000172A0
00010F8D 000157AE 000120FB 00017BB0 0000F9F6 0002ACA8 00022D92 0001E01C 0000D092 0001590D 0000E6C3 0001ACCA
###############
Congratulate!
###############
(base) docker@706a9054c435:~/workspace/EE219-ICS/projects/lab2$
```

Fig. 9. Test results.

Fig. 10 is the result of 10 continuous tests, and the printed information shows that all the test cases are passed!



```
###############
Congratulate!
###############
1       PASS    IMG_W=1 IMG_H=1 FILTER_NUM=1 FILTER_SIZE=3
2       PASS    IMG_W=3 IMG_H=3 FILTER_NUM=3 FILTER_SIZE=3
3       PASS    IMG_W=3 IMG_H=4 FILTER_NUM=3 FILTER_SIZE=3
4       PASS    IMG_W=4 IMG_H=3 FILTER_NUM=3 FILTER_SIZE=3
5       PASS    IMG_W=3 IMG_H=4 FILTER_NUM=5 FILTER_SIZE=3
6       PASS    IMG_W=3 IMG_H=4 FILTER_NUM=6 FILTER_SIZE=3
7       PASS    IMG_W=4 IMG_H=5 FILTER_NUM=3 FILTER_SIZE=3
8       PASS    IMG_W=4 IMG_H=5 FILTER_NUM=5 FILTER_SIZE=3
9       PASS    IMG_W=4 IMG_H=5 FILTER_NUM=6 FILTER_SIZE=3
10      PASS    IMG_W=5 IMG_H=5 FILTER_NUM=7 FILTER_SIZE=3
```

Fig. 10. Test for 10 continuous cases.

V. Conclusion

In this lab, I have completed the im2col, PE, and systolic_array modules and passed all the test cases. I use the concept of absolute address and relative address to control the read address for im2col module. In systolic_array module, I realize the PE unit and connected all the PE array correctly. FIFO shift registers are also implemented to control the cycle of data flow.

References

[1] Siting, Liu. EE219: ICS, 2022-Fall
[2] Yajun, Ha. EE116: FPGA-based Hardware System Design, 2020