A General-Purpose and Configurable Planar Data Processor for Energy-Efficient Pooling Computation

Lunshuai Pan
SEU-FEI Nano-Pico Center, Key Laboratory of MEMS of Ministry of Education, School of Electronic Science and Engineering, Southeast University, Nanjing, 210096, People's Republic of China.
Nanjing, China
ls.pan@siat.ac.cn

Peng Xue
Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
Shenzhen, China
1481225406@qq.com

Hongxing Li
SEU-FEI Nano-Pico Center, Key Laboratory of MEMS of Ministry of Education, School of Electronic Science and Engineering, Southeast University, Nanjing, 210096, People's Republic of China.
Nanjing, China
1757740270@qq.com

Litao Sun*(corresponding author)
SEU-FEI Nano-Pico Center, Key Laboratory of MEMS of Ministry of Education, School of Electronic Science and Engineering, Southeast University, Nanjing, 210096, People's Republic of China.)
Nanjing, China
slt@seu.edu.cn

Mingqiang Huang*(corresponding author)
Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
Shenzhen, China
mq.huang2@siat.ac.cn

***Abstract*—Convolutional Neural Networks (CNN) have been widely used in artificial intelligence applications. A typical CNN contains both convolution and pooling layer, in which the convolution is to detect local conjunctions of features and the pooling is to merge similar patterns into one. It is necessary to make pooling operation, which plays a great role in CNN. Up to now, there have been numerous researches on CNN accelerators, however, most of the previous works are only focused on the acceleration of convolution layers, and the specific studies on pooling units are still lacking. Besides, the existing pooling designs are usually constrained by either the poor flexibility or the low energy/area efficiency. In this work, we propose a general purpose and energy-efficient planar data processor to support the pooling operation from different CNN structure. By using the configurable data path control method, the processor is able to support universal pooling operation with arbitrary input feature shape and arbitrary pooling kernel/stride/padding size. Besides, the processor exhibits high efficiency with hardware utilization ratio near 100% during operation, indicating good performance of the design. Most importantly, it is energy-efficient that exhibits 86%-off on power consumption and 62%-off on area utilization when compared with the separate pooling module of NVDLA (NVIDIA Deep Learning Accelerator), thus is particularly suitable for the resource-limited edge intelligent devices.**

Keywords—Energy-Efficient; CNN accelerator; Pooling

I. INTRODUCTION

Deep convolutional neural networks (CNN) have demonstrated significant success on many artificial intelligence tasks, an emerging tendency and practical application of which is to put intelligence at the edge[1], such as the smart phones, drones, and AIot devices. However, the edge device usually operates at resource and power limited environment, where high performance yet low energy dissipation are strongly desired. For this reason, the traditional computation platform of central processing unit (CPU) or graphics processing unit (GPU) are both incapable because they suffers either low throughput or low energy-efficiency. Instead, it brings opportunities of energy-efficient specific CNN hardware accelerator for deep learning inference.

In a high performance CNN network, there exists both convolution layers and pool layers: the role of convolution is to detect local conjunctions of features from previous layer, and that of pooling is to merge semantically similar features into one . Though the specific CNN accelerators such as TPU[2], NVDLA[3], Eyeriss[4] and DianNao[5] have attracted great attentions, whereas, these designs are usually focused on the convolution operation, and the hardware implementation of pooling has rarely been addressed. The key challenges in pooling accelerator relies on the two points.

Firstly, flexibility and universality are important for the CNN hardware accelerator. Because the popular CNN network structure varies from year to year, and the pooling operation also varies from network to network. For example, the classical VGG-16 network contains only max-pooling operation, meanwhile the ResNet network needs both max-pooling and average-pooling. Besides, the pooling parameters of VGG-16 and ResNet18 are different. As a result, few previous works can support both type of networks, not to mention the support of more networks.

Secondly, energy-efficient hardware design is highly desired in edge computing. As is known, there is an obvious trade-off between simplicity and generality in hardware circuit design. Take the famous CNN hardware accelerator, NVIDIA Deep Learning Accelerator (NVDLA) as an example, the planar-data-processor of which is highly configurable and supports different pool group sizes with three pooling functions: maximum-pooling, minimum-pooling, average-pooling. However, the circuit area and power consumption of such module is dramatically large, which goes against the energy-efficient computing at the edge. What's more, the NVDLA is still not that flexible, for example, the pooling kernel has to be fixed as a square window (namely K_x=K_y), and the strides at the x-direction

This work was supported by Guangdong Science and Technology Innovation Committee 2019A1515111142, Shenzhen Science and Technology Innovation Committee JCYJ20200109115210307.

and the y-direction are also compulsively to be equal (namely S_x=S_y).

In this work, we present a general purpose and energy-efficient planar data processor for the pooling operations.The pooling operations are decomposed into various sub-tasks, and all of the parameters in control module are pre-calculated and sent to the configurable register array afterwards. The processor has been synthesized in typical ASIC[6] flow \cite{ASIC}, and the PPA (performance, power, area) are all much better than that of separate pooling module of NVDLA.

II. Overview of the Proposed Method

Fig1 shows the typical working flow of proposed general-purpose pooling computation unit. The pooling IP (Intellectual Property) is designed in modular architecture so that it can be easily integrated with other CNN modules. During operation, the pooling unit will be connected to a control CPU and out-chip DDR SDRAM (Double-Date-Rate Synchronous-Dynamic-Random-Access-Memory) with Advanced eXtensible Interface (AXI) protocol for the parameters configuration and data storage/transmission, respectively, which is commonly seen in CNN accelerators.

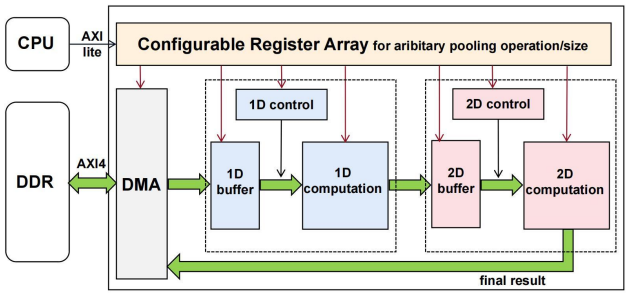

Fig.1. System framework of the proposed planar data processor for pooling.

Inside of the pooling unit, there exists three sub-modules, namely the operation control module of Configurable-Register-Array (CRA), read/write DMA (Direct Memory Access) module, and the computation module. CRA receives parameters from CPU and controls the hardware working flow. DMA transfers the streaming data between the out-chip memory (DDR) and the on-chip buffer. The computation unit decomposes the pooling operation into 1D calculation and 2D calculation with a pipelined dataflow.

The key contributions of this work rely on two points. Firstly, how the CRA unit is designed to support arbitrary type of pooling. Secondly, how the computation unit is designed to save the circuit area, especially to save the the on-chip buffer size thus increasing energy-efficiency.

III. CRA Unit for Pooling Configuration

A. Analysis on Pooling

Fig.2 exhibits typical 2×2 max-pooling operation. The input feature is mapped in 3D matrix with a shape of [Channel][Height][Width], where the input height is set to be 4 and input width is 8 as an example. The pooling filter size is 2×2, and the stride size is also 2×2. During pooling, the 2×2 filter window will move over the initial inputs planarly with 2×2 stepping size, thus generating much smaller output feature map with 2×4 pixels at each channel.

For the input/output channel depth, it can be seen that there is no interaction between different channels, therefore it is particularly suitable for parallel computation along the channel depth ' s dimension. In Fig.2, the exampled parallelism is T = 4, which means the input/output feature maps will be expanded into 4D matrix with a shape of [Channel/4][Height][Width][4].

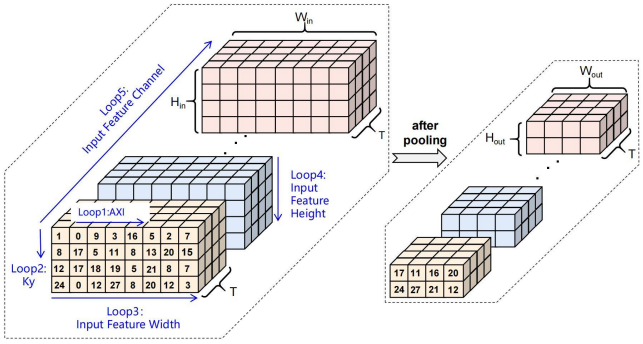

Fig.2. Feature data cube before and after pooling, in which the five-stage loops represent the DMA transmission strategy.

B. Analysis on DMA

In order to support the arbitrary pooling operation meanwhile decrease the on chip buffer size, AXI burst transmission should be fully utilized. The transferred input/output feature data must be carefully mapped in the DDR memory with continuous and incremental address. In consideration of the shape of input feature map [Channel/T][Height][Width][T], the burst transmission is suggested to transfer data exactly along the width direction. Besides, the AXI burst transmission length (denote as axi_burst_len) should not be too large (<64), so that it can quickly jump to the next address to realize universal pooling operation with arbitrary input feature shape and arbitrary pooling kernel/stride/padding size.

According to the above discussion, the data transmission in DMA can be divided into five-stage loops. As presented in Fig.3, Loop1 represents the high-performance AXI burst transmission along the width direction. Loop2 is the count of K_y in pooling-kernel height direction. Loop3 repeats the transfer until the whole row of data is processed. The loop1 to loop3 is to generate a complete row of output feature map in width direction. Loop4 and loop5 is for the output feature height and channel depth direction, respectively.

C. Configuration for Arbitrary Pooling Settings

The analysis of arbitrary pooling will start from the on-chip buffer size. Suppose there is an on-chip buffer with size of axi_burst_len, then outputs can be easily calculated according to the definition of pooling:

$$W_{out} = (W_{in} + 2 \times P_x - K_x)/S_x + 1 \quad (1)$$

$$H_{out} = (H_{in} + 2 \times P_y - K_y)/S_y + 1 \quad (2)$$

$$CH_{out} = CH_{in} \quad (3)$$

$$max\_in\_number = (axi\_burst\_len - K_x)//S_x + 1 \quad (4)$$

where the symbol of "//" is floor division. With the value of max_out_numbers, the required input pixels can be then deduced by

$$max\_in\_number = (max\_out\_numbers - 1) \times S_x + K_x - P_x \quad (5)$$

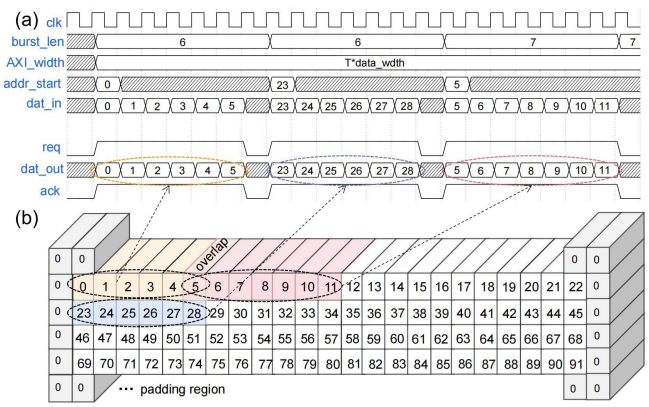

Fig.3. Dataflow in pooling. (a) Timing diagram of specific data flow in DMA burst transmission.(b) Input feature map with the shape of H_{in}=4, W_{in}=23, Kernel=3, Stride=2, Padding=1.

Since it is floor division in equation (4), the max_in_number will always be smaller than axi_burst_len, therefore the on-chip buffer is large enough for the input. Besides, the padding region should be carefully considered because not all of transmission requires padding.

Fig.3 shows an example of the working flow with arbitrary parameters setting, in which W_{in} = 23, H_{in} = 4, K_x = 3, S_x = 2, P_x = 1, axi_burst_len = 8. According to equation (4), the max_out_number is calculated to be 3, then the input pixel numbers can be deduced using equation (5). Specifically, the first and second group of input feature data contains 6 elements of "0/1/2/3/4/5" and "23/24/25/26/27/28" in the first and second row, respectively. But the third group of input feature data contains 7 elements of "5/6/7/8/9/10/11" in the first row again. It should be noted that the original buffer size is defined to be equal to the burst length (here is 8), however, only 6 or 7 elements are used in the example. This is a compromise design in consideration of the arbitrary and unknown size of inputs, which is acceptable.

By using the configurable data path control method, the proposed design is able to support any shape of pooling kernel. The only cost for its high flexibility is the increasing of registers (it needs about 20 numbers of registers), which is almost negligible. With the optimized design, the on-chip buffer can be decreased down to only ~kb level, exhibiting hundreds of enhancement on area efficiency.

IV. Computation Unit for Pooling Operation

In this section, we will tend to explain how the data is calculated in the planar data processor. Fig.4 shows the dataflow in X-direction and Y-direction.

A. X-direction 1D Calculation

The X-direction calculation contains 1D buffer, FSM control module and computation module. The 1D buffer is to store the initial input feature data from out-chip memory (DDR). To increase the efficiency, typical ping-pong buffers are used here, and each buffer will store only one row of input data.

The 1D FSM module is designed to transfer the buffered data to the 1D calculation module. Since it contains only one row of data in each ping-pong buffer, the task of 1D FSM

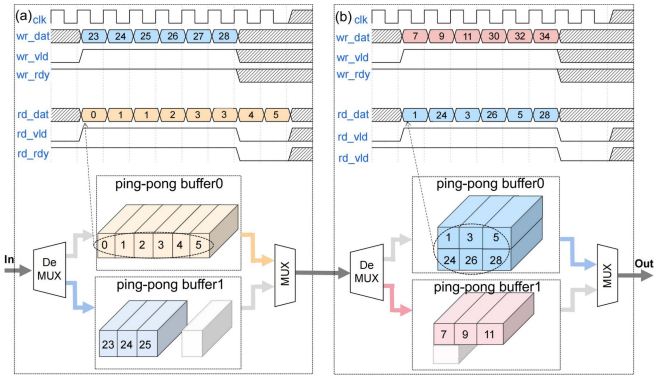

Fig.4. Dataflow in X-direction and Y-direction computation.

will simply be the calculation of segmentation. The number of segmentation means how many output pixels it will generate, which has been pre-configured and denoted as the first/middle/last_width_out.

Once padding is taken in consideration, the segmentation can be then calculated in two steps. Firstly, judge whether padding is needed, in which both left padding and right padding are included. Secondly, make the segmentation according to its output pixels and pooling kernel size/stride.

The 1D calculation module is to generate the 1D outputs. Similar to the FSM module, it also needs to judge whether padding is needed and what is the correct segmentation. If the input is the first element, it will be stored in the MUX and waiting for the second input. If not, the new input data and the stored temporary input value will compute the pooling operation until the last one comes.

B. Y-direction 2D Calculation

The 2D computation module is the similar to that of 1D. Generally, the 2D buffer stores the temporary results from 1D calculation module, then the 2D FSM module transfers the 2D buffer data to the calculation module, finally the 2D calculation module generates the final result. The difference is, the 1D ping-pong buffer contains one-dimensional vector input data (only one row of data), whereas the 2D buffer contains two-dimensional tensor input data (row number is K_y, and column number is the width output).

To process the two-dimensional tensor data, larger on-chip buffer is used, and the 2D buffer size is set to be N*axi_burst_len, with the restricted condition of

$$N \times axi\_burst\_len > K_y \times current\_width\_out \quad (6)$$

Since the current output width is constrained by

$$current\_width\_out = (axi\_burst\_len - K_x)//S_x + 1 \quad (7)$$

The number of N=2 will satisfy most of the pooling cases. If not, one can increase the 2D buffer size or deduce the input feature pixels from the 2D buffer instead. The x-direction calculation has been completed, thus the two ping-pong buffers owns the same length of input data in this stage. Besides, since all of the required data have been stored in buffer, the FSM module can easily control the complete y-direction calculation for the final output results.

TABLE I Simulation results on various pooling operations. (The efficiency larger than 100% comes from the skip of zero-padding region.)

Pooling models		*CH*	W_{in}	H_{in}	K_x	K_y	S_x	S_y	P_x	P_y	H_{out}	W_{out}	*Theoretical clk1*	*Practical clk2*	*Hardware efficiency*
A	max-pooling in Resnet18	64	112	112	3	3	2	2	1	1	56	56	112896	111552	101.2%
B	avg-pooling in Resnet18	512	7	7	7	7	1	1	0	0	1	1	1568	2048	76.6%
C	max-pooling in VGG16	64	112	112	2	2	2	2	0	0	56	56	50176	56498	88.8%
D	Example in this work	32	23	23	3	3	2	2	1	1	12	12	2592	2379	108.9%
E	arbitrary parameter	64	57	43	4	3	3	2	2	1	20	22	21120	19516	108.2%

V. Experiment Results

A. Supporting of the General-purpose Pooling Computation

Table 1 shows the results on various pooling layers, in which the AXI burst length is set to be axi_burst_len = 16, and parallelism is T = 16. It can be seen that the proposed processor supports the arbitrary parameters pooling operations.

The total operation numbers in pooling is calculated by $OPs = K_x \times K_y \times H_{out} \times W_{out} \times CH$. The required theoretical clock cycle clk1 is predicated to be clk1 = OPs/T, and the practical clock cycle clk2 is generated from the testbench (register of the "end" signal). Table 1 also shows the hardware efficiency on different pooling operations. It is clear to see that the hardware utilization ratio are all larger than 76%, and in some cases it is more than 100%. This is because the zero-padding region has been skipped in our design, thus the practical clock cycle is smaller than its theoretical operation numbers.

With a larger burst length, the hardware efficiency will be higher. However, the size of on-chip memory will also be larger. In consideration of both the efficiency and buffer size, the burst length is suggested to be 16 for the practical application.

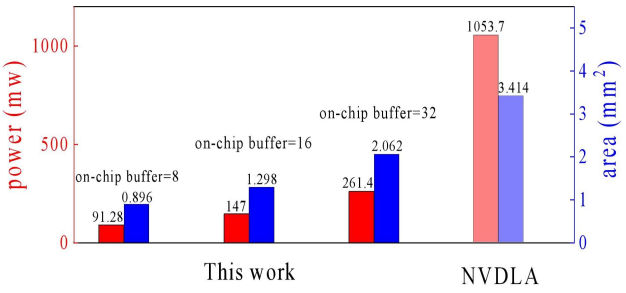

Fig.5. Comparison of power and area between this work and NVDLA.

B. PPA Evaluation

The aforementioned designs in RTL are then synthesized by Synopsys Design Compiler under 90 nm process to obtain the performance, power and area (PPA) values. The famous NVDLA is evaluated with exactly the same constraints.

At a same data-bit-width of 16bit and parallelism of 16, the pooling computation unit in NVDLA consumes 3.414 mm^2 area and 1054 mW power consumption, while our design is 1.298 mm^2 and 147mW, respectively, corresponding to 62%-off on area utilization and 86%-off on power consumption (fig5). Besides, our design shows much better performance on higher clock frequency. The WNS (worst negative slack) is always positive under 400MHz clock, which is several times better than that of NVDLA.

VI. Conclusion

In summary, we propose a general-purpose and configurable planar data processor for the energy-efficient pooling computation. With the developed configurable register array and corresponding configurable data path control method, the processor is able to support universal pooling operation with arbitrary input feature shape and arbitrary pooling kernel/stride/padding size, and it exhibits high efficiency with near 100% hardware utilization ratio. Besides, the relationship between hardware efficiency and hardware cost (mainly buffer size) has been systematically analyzed. The on-chip buffer can be down to only ~kb level, meanwhile maintaining high performance and high flexibility. Finally, the RTL is synthesized in ASIC flow at 90 nm process node. Compared with the famous NVDLA, the proposed design shows 86%-off on power consumption and 62%-off on area utilization, indicating great potential for the application in energy-efficient edge-computing.

References

[1] C. You, K. Huang, H. Chae and B. -H. Kim, "Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading," in IEEE Transactions on Wireless Communications, vol. 16, no. 3, pp. 1397-1411, March 2017, doi: 10.1109/TWC.2016.2633522.

[2] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of the 44th annual international symposium on computer architecture. 2017: 1-12.

[3] Zhou G, Zhou J, Lin H. Research on NVIDIA deep learning accelerator[C]//2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID). IEEE, 2018: 192-195.

[4] Chen Y H, Yang T J, Emer J, et al. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019, 9(2): 292-308.

[5] Chen T, Du Z, Sun N, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284.

[6] Beerel P A, Dimou G D, Lines A M. Proteus: An ASIC flow for GHz asynchronous designs[J]. IEEE Design & Test of Computers, 2011, 28(5): 36-51.