2019 IEEE 19th International Conference on Communication Technology

A High-Performance Systolic Array Accelerator Dedicated for CNN

Jing Shen, Haoqi Ren, Zhifeng Zhang, Jun Wu*, Wenqi Pan, Zhenyu Jiang
College of Electronics and Information Engineering, Tongji University, 201804
Shanghai, China
e-mail: {1732983, renhaoqi, zhangzf, wujun, 1810866, 1832925}@tongji.edu.cn

***Abstract*—The rapid development of artificial intelligence has made the convolutional neural network (CNN) more important. The traditional computing architecture based on CPU can't meet the requirements of the practical applications. Therefore, the development of a new hardware computing platform for CNN becomes more urgent. This paper proposes a systolic array accelerator dedicated to CNN. As CNN model requires a lot of simple logic operations, we optimize the convolution calculation module with the systolic multiply-accumulate (MAC) array. We design three kinds of convolution calculation mappings, which can deal with convolution with different sizes. Our accelerator realizes efficient reuse of local storage area data, which reduces data movement and improves computing performance. To balance storage bandwidth and computational speed, the convolutional layer is subdivided into granular tasks and executed to mask the time of accessing external storage. This accelerator also supports Winograd convolution of 3x3 weight kernels. The heterogeneous system consisting of the accelerator and the self-developed digital signal processor SWIFT (SWIFT DSP) is verified on the FPGA platform. The experimental results show that our accelerator outperforms traditional accelerators under the same condition.**

Keywords-CNN; DNN; parallel computer architecture; accelerator

I. INTRODUCTION

With the widespread use of deep learning in many fields, the underlying model-convolutional neural network (CNN) has also received increasing attention. However, as the depth of the network continues increasing, the scale of data movement and calculation continues to expanding, and the computational workload exponentially increases. However, the traditional processors are not designed for calculating the workload of neural networks, which are the basis of many applications. Therefore, new architectures are required to meet the growing computing demands.

Accelerators for specific CNNs (such as AlexNet, ZynqNet) are not in common use and therefore not suitable for mass production. Since the TPU is not for sale, only GPU and FPGA are available to be the training platform for the neural network, whose power consumption is large and the price is high. Since GPU is not a specially designed AI chip, only a small part of its function is suitable for deep machine learning. Besides, GPU's future target will still not be AI either. FPGAs have too much unnecessary flexibility to support common functions, and are less efficient in power efficiency and area efficiency for deep neural network (DNN) calculations. The market demands and application prospects of deep learning accelerators for DNN are very broad.

NVIDIA proposed a AI chip architecture--NVIDIA Deep Learning Accelerator (NVDLA). The NVDLA is a free and open architecture that promotes a standard way to design deep learning inference accelerators. With its modular architecture, NVDLA is scalable, highly configurable, and designed to simplify integration and portability. It supports a wide range of performance levels and readily scales for applications ranging from smaller, cost-sensitive Internet of Things (IoT) devices to larger performance oriented IoT devices. The convolution module uses three parallel convolution calculation mapping modes (DC, WG and IMG) to optimize the analysis and design the convolution pipeline, to realize the local storage area. Efficient reuse reduces the number of data movements and increases the degree of parallelism in multiple dimensions in the calculation.

Google proposed the second generation tensor processing unit (TPU) in 2017 [1]. Compared to GPU, TPU use low-precision (8-bit) calculations to reduce the number of transistors used in each step. Reduced accuracy has little effect on the accuracy of deep learning, but it can significantly reduce power consumption and speed up operations. At the same time, the TPU uses a systolic array to optimize matrix multiplication and convolution operations and reduce I/O operations. In addition, the TPU uses larger on-chip memory to reduce access to DRAM for greater performance. The key of the systolic array is that the data flows in the array of processing element. It reduces the memory accesses. Another advantage of the systolic array is that, with the same number of multipliers, the bandwidth of the accelerator is much smaller.

In 2017, SenseTime proposed an FPGA-based fast Winograd algorithm [2]. The key of the Winograd algorithm is to replace the multiplication with more additions.

In this paper, we propose a high-performance systolic array accelerator (HSA) dedicated to convolutional networks. In order to simultaneously solve the limited bandwidth in the embedded applications and support Winograd optimization, we combine systolic array (the TPU solution), broadcast array (the NVDLA solution) and regroup multipliers into a 4x4 multiply-accumulate (MAC) cell array. Each MAC cell contains 32 8-bit multipliers and outputs only one partial sum. The accelerator supports 8-bit and 16-bit fixed-point operations and 16-bit floating-point operations, as well as Winograd convolution for 3x3 weight kernels.

978-1-7281-0535-2/19/$31.00 ©2019 IEEE

SWIFT DSP is a self-developed digital signal processor [3]. SWIFT consists of a nine-stage pipelined processor core, 128KB of on-chip instruction memory and 1MB of data memory, debug interface and basic peripherals. SWIFT supports 32-bit fixed-point arithmetic and 4-way 40-bit, 8-way 20-bit or 16-way 10-bit vector operations. The SWIFT DSP is manufactured by SMIC's 65-nanometer process. The chip area is 16.6mm^2, the measured main clock frequency reaches 400MHz. The fixed-point operation peak is 6400MMACs, and the unit power consumption is 0.6mW/MHz. The SWIFT DSP also has a complete tool chain including the simulator, the assembler, the compiler and the debugger. We use SWIFT DSP and Xilinx Vertex7 to build the verification environment. SWIFT DSP acts as the host and calls the HSA to perform convolution operations to verify the performance of the HSA.

The structure of this paper is as follows: Section II proposes the novel systolic array accelerator dedicated to CNN, describing the overall framework and analyzing the theoretical performance superiority. Section III is about the specific implementation of HSA's key modules, including systolic MAC, convolution DMA module and accumulator module, and gives the heterogeneous SoC system architecture composed of HSA and SWIFT DSP. Section IV shows the experiment results and Section V draws the conclusion.

II. THE OVERVIEW OF HSA

In this section, we present the structure of the HSA and describe the function of each module. Further, we analyze the parallelism of the CNN, thereby theoretically analyze the performance improvement of the HSA.

A. The HSA Architecture

The proposed high-performance systolic array accelerator dedicated to CNN can handle different sizes of data with various weights and support Winograd optimized calculations. As shown in Figure 1, the HSA contains a convolution pipeline and a glue module.

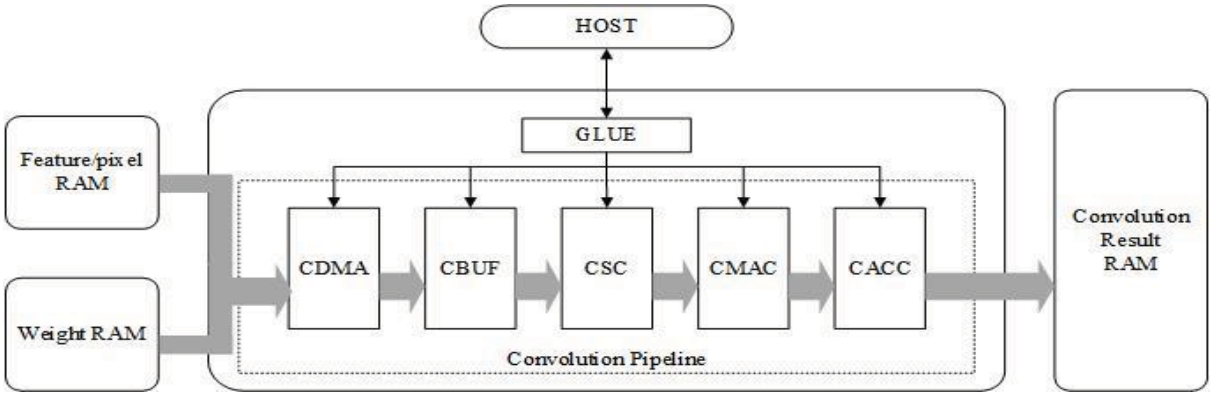

Figure 1. The architecture of the HSA

The glue module communicates with the host processor and sends the configuration information to convolution pipeline. The host processor offloads the entire CNN calculation to the accelerator. Especially, the host transfers the input image and the detailed CNN structure to the accelerator (the size of data and weight, convolution mode, address etc.). The glue module sends configuration command to each sub-function module of the convolution pipeline, and then the accelerator starts to work. After the convolution calculation is completed, the glue module sends an interrupt to the host, and the accelerator completes the work.

The convolution pipeline includes five modules: CDMA, CBUF, CSC, CMAC and CACC. Each module contains a register file for storing configuration information. The functions of each module are as follows:

- CDMA: processing the weights and data in a certain order (including floating-point fixed-point processing and Winograd pre-processing, if necessary) and then temporarily stores the results in CBUF in a specific order.
- CBUF: saving the processed weights and data arranged according to the rules.
- CSC: sending the weights and data in CBUF to the mac array for calculation in a certain order.
- CMAC: MAC array module, doing multiply-accumulate operations, outputs the result partial sum and sends it to CACC.
- CACC: accumulator module, accumulating the output of the CMAC module until the final result is obtained and outputs the result to the destination address.

B. Parallelism in CNN

The CNN forward propagation task can be parallelized in several ways [4]. In a multi-layer CNN, due to the feed-forward nature of forward propagation, the data dependencies between successive layers preclude parallel execution of all layers of the CNN. Therefore, task-parallelism across layers is limited and it is also more difficult to be exploited.

Operator-level (fine-grained) parallelism: Consider one image-kernel convolution that convolves an image with I_r rows and I_c columns and a kernel with W_r rows and W_c columns. Each output pixel requires W_r*W_c multiply-

accumulations all of which can be performed in parallel. The output pixels themselves are all independent. Practical considerations, such as available memory bandwidth, hardware computation resources, and power considerations, do limit the degree of fine-grained parallelism we can exploit in hardware. For example, it is not practical to simultaneously perform all the sub-convolutions in parallel due to the excessive memory bandwidth which is necessary to bring in almost the entire image in one cycle.

Coarse-grain parallelism: If a CNN layer has n input images and m outputs, then all m*n image-kernel convolutions can, in theory, be performed in parallel. However, for typical m and n, provisioning enough memory bandwidth to keep m*n convolvers busy is impractical.

With a smaller number of convolvers, we can extract parallelism in two ways: inter-output and intra-output. We can parallelize the computation of a single output image since it is the sum of n input-kernel convolutions. We refer this as intra-output parallelism. Also, multiple output images can be computed in parallel, and we refer this as inter output parallelism. The key observation is that different layers in a CNN network exhibit vastly different amount of intra-output and inter-output parallelism.

C. *Theoretical Analysis*

As shown in Figure 2, NVDLA has 8 mac cells. Each mac cell can do 128 8-bit mac operations. Only one partial sum is output, and there is no data interaction between mac cells. To solve this problem, we introduce the idea of systolic arrays: re-grouping multipliers, which is that each MAC cell can do 32 8-bit MAC operations and output one partial sum; MAC cell is organized into 4x4 array, each cell has its own weight, data flows by columns and partial sum flows by rows. In the extreme case, which means that there is no buffer for the accelerator, HSA can reduce the number of accesses by 75% compared to the normal convolution.

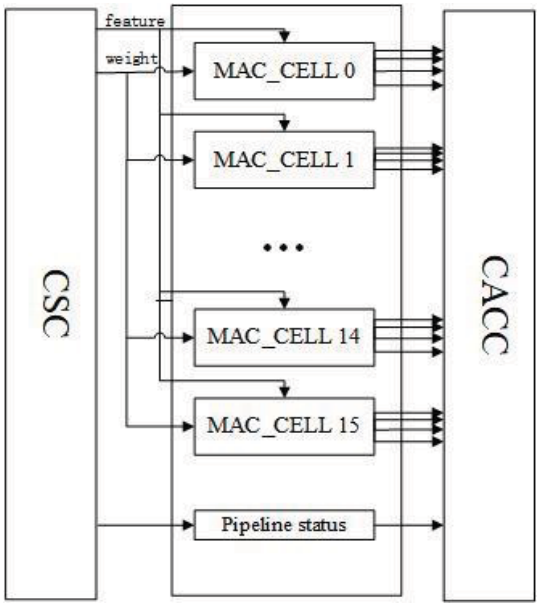

Figure 2. Module CMAC in NVDLA

The following pseudo-code represents the mapping method of the convolution calculation in out accelerator.

The two inputs involved in the convolution are image and filter, and the output involved in the convolution is the convolution result.

- Image: The dimension is C*H*W, where C is the channel, also called depth, H and W are the width and height of the image, respectively.
- Filter: The dimension is K*M*M, where K is the kernels, M is the width and the height, respectively.
- Result: The dimension is K*H*W, where K is the channel, H and W are the width and height, respectively.

Algorithm 1 HSA Convolution

Input: image, fliter
Output: result

```
for cntW in 1...W/subW do
  for cntH in 1...H/subH do
    for cntK in 1...K/subK do
      for x in 1...M do
        for y in 1...M do
          for k in 1...subK do
            for cntC in 1...C/subC do
              for w in 1...subW do
                for h in 1...subH do
                  for c in 1...subC do
                    curW = cntW*subW + w;
                    curH = cntH*subH + h;
                    curC = cntC*subC + c;
                    result(curW, curH, k) +=
                      image(curW, curH, curC)*filter(x, y, k);
                  end for
                end for
              end for
            end for
          end for
        end for
      end for
    end for
  end for
end for
```

III. THE IMPLEMENT OF HSA

In this section, we present a detailed structure and implementation of the CMAC, CDMA and CACC. Each module can accept two sets of configuration information to mask the time of the host's configuration for the accelerator. We also mount the HSA on the SWIFT DSP to form a heterogeneous system.

A. *Specific Description of Key Modules*

1) *Convolution DMA module*

The CDMA includes a set of configuration registers, a shared buffer, a precision conversion sub-module, and four function sub-modules: CDMA_WT, CDMA_DC, CDMA_WG and CDMA_IMG. In this module, we fix the point of external data and weights, do Winograd pre-processing (optional) and re-arrange data. Then we cache the data and weights in CBUF. According to the rules of data rearrangement, the Convolution DMA module can be further

divided into four blocks: CDMA_WT, CDMA_DC, CDMA_WG and CDMA_IMG.

We illustrate the workflow of Convolution DMA with CDMA_DC (DMA in direct convolution mode).

- Check status of convolution buffer for enough free space;
- Generate read transactions;
- Cache feature data in shared buffer (precision processing);
- Load data from shared buffer and perform fixed-point processing and rearrangement;
- Generate write transactions;
- Update the status of convolution buffer.

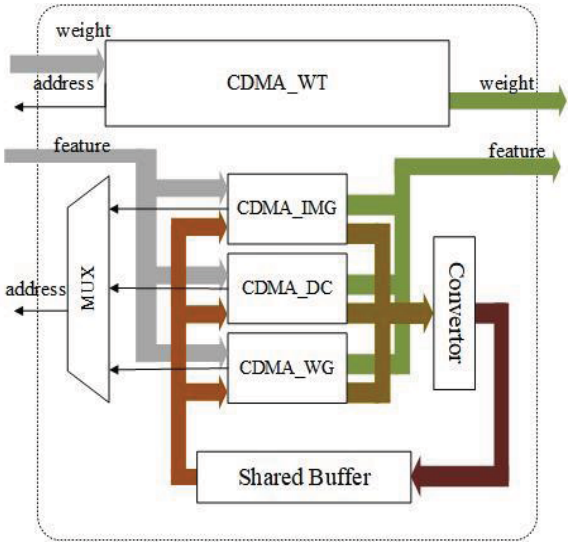

Figure 3. The architecture of CDMA

The CDMA_IMG process is the same as CDMA_DC. CDMA_WG only has one more step for Winograd pre-processing. CDMA_WT is similar too. But the rules for CDMA_WT, CDMA_WG, CDMA_DC and CDMA_IMG rearrangement are different. The three modules: CDMA_WG, CDMA_IMG and CDMA_DC, can't be used at the same time. The weight DMA and the feature DMA can work simultaneously.

2) CMAC module

The CMAC module consists of a set of configuration register and a MAC core. The MAC core is a systolic array of 4x4 mac cells. Each cell contains 32 8-bit multipliers, which can do 32 8-bit multiply-accumulate operations or 16 16-bit multiply-accumulate operations. Each column in the mac cell systolic array shares the same weight, and the feature flows along the rows, partial sums flow along the columns. The accelerator supports 16-bit floating-point, 8-bit and 16-bit fixed-point multiplication. CSC sends the weights (if the weights need to be updated) and the features to the MAC array. After passing through the systolic array and the addition tree, four sets of partial sum results are obtained and sent to the accumulator.

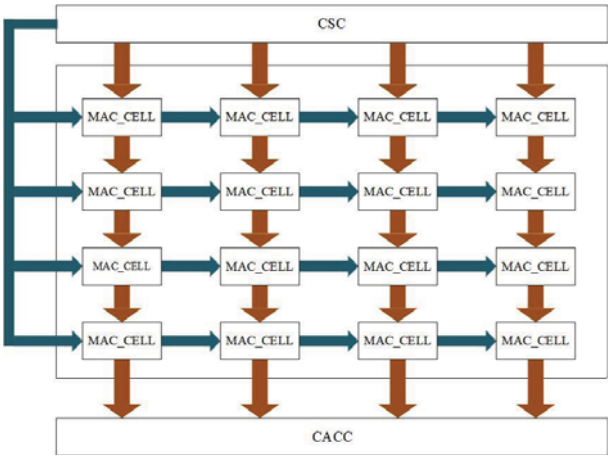

Figure 4. The architecture of CMAC

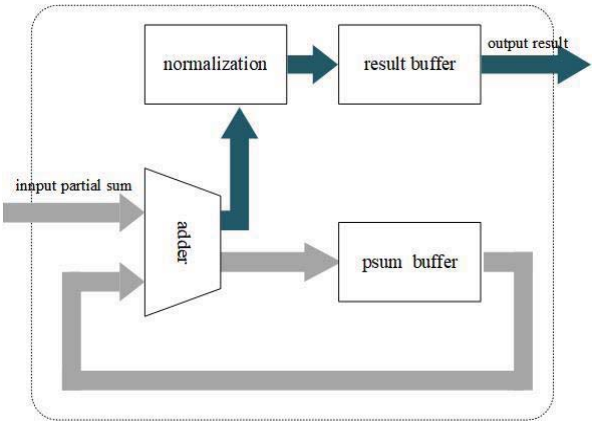

Figure 5. The architecture of CACC

3) CACC Module

The accumulator module contains a set of configuration registers, a buffer for partial sums, accumulation sub-module and a buffer for result. The partial sums of the systolic array are sent to the accumulator (ie, the results of the atomic operation), and then accumulates (ie, the stripe operation) with the partial sums (ie, the results of stripe operation) temporarily stored in the accumulation module until the final result is obtained (ie, the results of the channel operation). The final result is normalized and placed in the result buffer of the accumulator, and then transferred to the external memory.

B. A heterogeneous Systems for HSA and DSP

SWIFT DSP is a digital signal processor with independent intellectual property. As the HSA's computational core and data scheduling center, has a high throughout Very Long Instruction Word (VLIW) instruction set architecture and supports Single Instruction Multiple Data (SIMD) data path. The VLIW instruction supports 4 sub-instructions, which means one instruction can implement 4 sub-instructions concurrent processing. SWIFT's SIMD supports a single instruction to process 4 sets of 40 bits of data, 8 sets of 20 bits of data, or 16 sets of 10 bits of data.

SWIFT adopts the Harvard-architecture with a 128KB on-chip instruction memory and a 1MB on-chip data memory providing wide memory bandwidth [5]. The DMA module on SWIFT supports unchained continuous transmission and chained intermittent transmission. In Burst transmission mode, a data block of 256 bytes can be transmitted in 17 clocks. The VLIW, SIMD, large amount of on-chip memory, high-speed bus and optimized high-speed DMA make SWIFT have high bandwidth and are suitable for parallel computation and transfer of a large amount of video streams [6]. Compared with ASIC's design flow, which involves the complicated RTL design and corresponding huge number of backend work to achieve the final hardware layout design, coding with high level language greatly simplifies the design process.

The glue module of the HSA is configured to accept the accelerator configuration information from the DSP. When the accelerator completes a convolution operation, an interrupt signal is sent to the DSP to indicate that the operation is completed. Since SWIFT DSP uses the wishbone bus protocol and the accelerator uses the AXI4 protocol, so the glue module is also responsible for bridging the two bus protocol to enable the system to operate normally.

IV. Experiment and Result

We use the Xilinx Virtex-7 FPGA platform to implement heterogeneous systems of HSA and SWIFT DSP. Vertex 7 has up to 2M logic slices to meet system design needs. The hardware development environment used in this experiment is ISE 14.7. With the same number of multipliers, the same feature and weight (the size of feature is 16x16x64 and the size of weight is 3x3x128) and the same clock frequency, we test the NVDLA convolution pipeline and HSA, and the results are shown in Figure 6. It can be seen that under the same condition, the HSA gets the slightly better performance than the NVDLA solution.

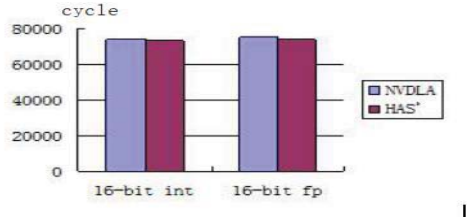

Figure 6. Performance comparison of the HSA and NVDLA convolution pipeline.

V. Conclusion

This paper proposes a systolic array accelerator dedicated to CNN. First, we analyze the shortcoming of the existing work. Second, to address this problem, we combine the idea of systolic array and broadcast array. We re-organize the MAC array and theoretically analyze the performance of the accelerator. We give the specific structure of the accelerator, and constitute a heterogeneous system with the SWIFT DSP and accelerator. Finally, we evaluate the system's performance. The experimental results show that the proposed design has improvement under the same condition when accelerating convolutional neural networks in resources and power-constrained platforms.

Acknowledgment

The authors thank the editors and the anonymous reviewers for their invaluable comments to help to improve the quality of this paper. This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61831018, 61571329 and 61631017, and Guangdong Province Key Research and Development Program Major Science and Technology Projects under Grant 2018B010115002.

References

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers et al., "In-datacenter performance analysis of a tensor processing unit," in 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2017, pp. 1–12.

[2] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," in 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017, pp. 101–108.

[3] H. Ren, Z. Zhang, and J. Wu, "Swift: A computationally-intensive dsp architecture for communication applications," Mobile Networks and Applications, vol. 21, no. 6, pp. 974–982, 2016.

[4] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," ACM SIGRCH Computer Architecture News, vol. 38, no. 3, pp. 247–257, 2010.

[5] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: energy-efficient hardware accelerators for machine learning," Communications of the ACM, vol. 59, no. 11, pp. 105–112, 2016.

[6] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," arXiv preprint arXiv:1804.06826, 2018.

[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energyefficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138, 2016.

[8] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2017, pp. 553–564.

[9] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in Proceedings of the 25th edition on Great Lakes Symposium on VLSI. ACM, 2015, pp. 199–204.