Energy efficiency optimization of FPGA-based CNN accelerators with full data reuse and VFS

Weixiong Jiang, Heng Yu, Xinzhe Liu, Yajun Ha

University of Nottingham
UK | CHINA | MALAYSIA

Faculty of Science and Engineering, University of Nottingham Ningbo China, 199 Taikang East Road, Ningbo, 315100, Zhejiang, China.

First published 2019

University of Nottingham
UK | CHINA | MALAYSIA

Energy Efficiency Optimization of FPGA-based CNN Accelerators with Full Data Reuse and VFS

Weixiong Jiang[1], Heng Yu[2], Xinzhe Liu[1], Yajun Ha[1]
[1]School of Information and Science Technology, Shanghaitech University, Shanghai, China
[2]School of Computer Science, University of Nottingham Ningbo China, Ningbo, China
Email:{jiangwx, liuxz, hayj}@shanghaitech.edu.cn, heng.yu@nottingham.edu.cn

***Abstract*—While FPGA has been recognized as a promising platform to accelerate Convolutional Neural Networks (CNNs) in embedded computing given its high flexibility and power efficiency, two challenges still have to be addressed to enhance its applicability on the edge-computing paradigm. First, the power and performance of the CNN accelerator are still bounded by memory throughput, and a CNN-customized architecture is desirable to fully utilize the on-chip storage. Second, power optimization algorithms are insufficiently explored on CNN-targeted platforms. In this paper, we design a novel FPGA-based CNN accelerator architecture that makes full use of the on-chip storage resources leveraging data reuse and loop unrolling strategies. We also present an efficient FPGA-based voltage and frequency scaling (VFS) system that enables VFS of the CNN accelerator for power optimization. We devise a VFS policy that fully exploits the power efficiency potential of the FPGA. Experiment results show up to 40% energy can be saved with our VFS platform and policy.**

I. Introduction

Recently, FPGA has become a promising platform for edge-computing given its energy efficiency compared to GPU and high flexibility compared to ASICs [1]–[3]. Convolutional Neural Network (CNN), as an essential component to realize computational intelligence, has been increasingly seen implemented on FPGA platforms. For instance, a programmable CNN accelerator architecture together with data quantization strategy and compilation tool is raised in [4]. In [5], the authors quantitatively analyze and optimize the design objectives of the CNN accelerators based on multiple design variables.

Two challenges still have to be addressed beyond existing FPGA-based CNN works to further enhance its applicability on the edge-computing scenarios. First, the performance is bounded by memory accessing, thus customized mechanisms are necessary to reduce CNN accelerator's dependence on memory bandwidth [6]. Second, although extensive studies have been targeting system-level power optimizing strategies such as Voltage and Frequency Scaling (VFS) [7]–[9], the models used are largely theoretical. VFS methodologies based on measured system values have less been reported; nonetheless, devising a model-free VFS approach is exceptionally valuable given the complexity of obtaining per-platform power models.

In our work, we introduce a novel FPGA-based CNN accelerating system that addresses both of the above mentioned challenges. Specifically, we design a novel CNN accelerator exploring data resue and loop unrolling. We also devise an FPGA-based VFS system that enables the VFS functionality of the CNN accelerator. Our major contributions include the following:

- We introduce a data reuse strategy and the corresponding multi-level storage and multi-level ping-pong architecture. With our data reuse strategy, there is no repeated data access.
- We present an FPGA-based VFS platform with high resolution and flexibility as well as low area overhead and scaling time.
- We propose a VFS policy based on the measured system metrics, while previous works on VFS use idealized models.

TABLE I
Definitions of CNN dimensional parameters.

Params	Shorthand	Params	Shorthand
Output channel width	M	Input channel width	N
Output row no.	R	Output column no.	C
Kernel size	K	Stride	S
Tile o/p channel width	T_m	Tile i/p channel width	T_n
Tile row no.	T_r	Tile column no.	T_c

- We implement the proposed CNN accelerator architecture on our proposed VFS platform, and achieve advantageous power and performance compared to the state-of-the-art approaches.

The rest of the paper is organized as follows. Section II presents the architecture of the CNN accelerator. Section III describes our VFS solution on FPGA and presents the VFS policy. Section IV provides experimental results, and Section V concludes the paper.

II. CNN Accelerator Design

In this section, we introduce the design considerations of the proposed data reuse and loop unrolling methodologies, which is followed by presenting the overall accelerator architecture design.

A. Data Reuse

In order to reduce the memory latency, the feature maps and the weights need to be loaded into programmable logic (PL) before computing. A usual approach is to divide the entire feature map into several tiles, given limited memory resources. The entire feature map is computed vertically first and then horizontally in $\frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c}$ times successively, as shown in Fig.1, where the notations are defined in Table I. The amount of data transfer for computing each tile is given below:

$$\gamma_{in} = T_n \times T_r \times T_c \times S \times S \tag{1}$$

$$\gamma_{weight} = T_m \times T_n \times K \times K \tag{2}$$

$$\gamma_{out} = T_m \times T_r \times T_c \tag{3}$$

The following equations give the repeat coefficient of data access for the input feature map, weight, and output feature map respectively:

$$\beta_{in} = \frac{M}{T_m}, \beta_{weight} = \frac{R}{T_r} \times \frac{C}{T_c}, \beta_{out} = \frac{N}{T_n} \tag{4}$$

As shown in Eqn.4, the feature map tiling leads to a significant increase in data accesses. However, the data accesses can be minimized by designing an optimized data reuse strategy. For example, if the feature map shown in Fig.1 is computed in the order of $1 \rightarrow 4 \rightarrow 7 \rightarrow 10 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 11 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 12$, the weight can be reused but the output feature map have to be repeatedly

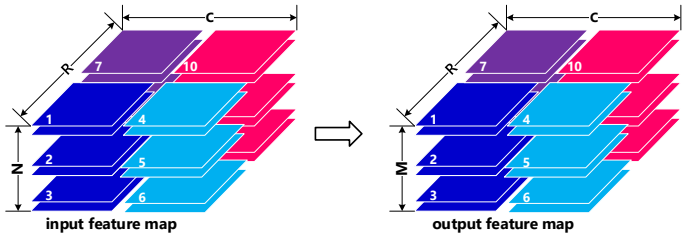

Fig. 1. Feature map tiling

Fig. 2. Two loop unrolling approaches

accessed by $\frac{N}{Tn}$ times. If all the output feature maps are stored on the chip, the data access can be further decreased at the expense of higher memory resource utilization. In order to minimize data accesses and make full use of the on chip memory resource, we develop a novel data reuse strategy where the entire input feature maps are stored on chip and the weights for each output feature map are loaded before the computation starts. Only after one output feature map is computed completely, then the next iteration begins. With this method, there is no repeated data access and β_{in}, β_{out} and β_{weight} are all reduced to 1. The pseudo code is as follows:

```
L1: for(M_x=0; M_x<M/T_m; M_x++){
L2:   Load_Weight(Tm*N*K*K);
L3:   for(Cx=0; Cx<C/Tc; Cx++){
L4:     for(Rx=0; Rx<R/Tr; Rx++){
L5:       for(Nx=0; Nx<N/Tn; Nx++){
L6:         Convolution();}
L7:       Store_Output();}}}
```

B. Loop Unrolling

The convolution operation has a variety of parallel features, and there are mainly two ways to unroll it. The first type of unrolling is carried out intra layer called fine-grained parallelism as shown in the left side of Fig.2, where all the calculations in the same convolution kernel are carried out simultaneously. The parallelism is $K \times K$. The second unrolling strategy, called coarse-grained parallelism, is carried out inter layers and each of the output neuron corresponding to T_n convolution windows that perform MAC operations. The computations of the convolution window in different input feature maps but in the same location can be executed in parallel. For example, as shown in the right side of Fig.2, the 3 neurons in dark blue are in the same location of its convolution window and they are computed simultaneously. Meanwhile, neurons in the same location but different output feature maps are connected to the same convolution window, so T_m output neurons in the same location can also be calculated in parallel, and the parallelism is $T_n \times T_m$. The coarse-grained parallelism can be adapted to any convolution layer so we choose the coarse-grained unrolling strategy in this work.

C. Accelerator Architecture

Fig.3 shows the architecture of our proposed accelerator. The feature maps of each layer except the first and the last ones are stored

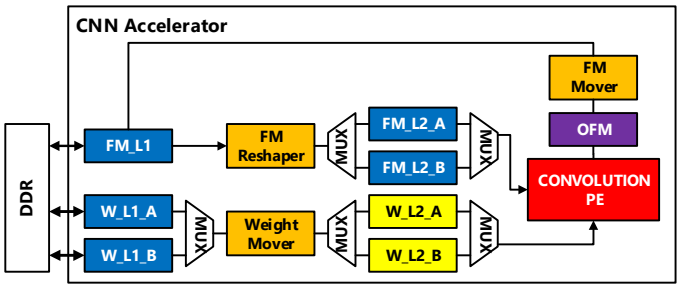

Fig. 3. The architecture of the proposed CNN accelerator

in **FM_L1** to minimize the data access. The feature maps are tiled by the **FM Reshaper** and then stored in **FM_L2s** before being sent to the **CONVOLUTION Processing Engine (CPE)**. **W_L1s** store the weights needed by each output feature map while **W_L2s** store the weights needed by each convolution tile. Thus the size of **W_L1s** is $T_m \times N \times K \times K$, while that of **W_L2s** is $T_m \times T_n \times K \times K$. The dual-level weight buffer architecture is employed in that **W_L1s** have to store large amounts of data, and should be implemented by BRAM. **W_L2s** have to provide $T_m \times T_n$ weights each cycle to guarantee the computation not be blocked, and should be implemented by LUTRAM. In addition, we double the buffers in a ping-pong fashion to further overlap the computation and data transfer/reshape. The **CPE** receives data from **W_L2s** and **FM_L2s**, and stores the results in output feature map. When the entire convolution layer is computed, the output feature maps are sent back to **FM_L1**.

III. VFS Platform and Policy

Fig.4 shows the architecture of our CNN-VFS system. The FS module is a clock generator that provides clock signals for the CNN accelerator and its DMA. The VS module, implemented by a power management IC (PMIC), controls and monitors switching regulators. Users can scale frequency from 20MHz to 400MHz at the step size of 1MHz in 3μs and scale voltage from 650mV to 850mV at the step size of 10mV in 2ms. In this section, we introduce how to build the VFS platform and formulate the power optimizing problem, then we provide our VFS policy under the measured metrics of the VFS platform.

A. Voltage and Frequency Scaling Platform

The state-of-the-art FPGA boards usually use power regulators and a PMBus-compliant system controller to supply core and auxiliary voltages. The PMIC is connected to the FPGA via PMBus, a protocol for power management which can be regarded as a subset of I2C, and controls several switching regulators to supply power for different components on FPGA. Users can scale the voltage and monitor both voltage and current by sending standard PMBus commands. We choose mixed mode clock manager (MMCM) as the clock generator in our design whose output frequency can be calculated by the following equation:

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O} \tag{5}$$

where M, D, and O are the configurations for the MMCM and can be changed through the AXI4-Lite interface available in the MMCM blocks, and new frequencies can be generated at run-time.

B. Power Optimization Problem Formulation

For realistic applications, there are many occasions where the accelerator can meet the performance requirements without running

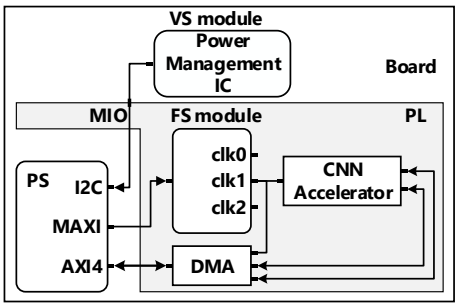

Fig. 4. System architecture of CNN accelerator enabled with VFS.

at full speed. There are basically two strategies under this context, one is to set the accelerator's running frequency to a low value thus it takes the accelerator longer time to finish task and the idle time is short, while the other is to set the accelerator's running frequency to a high value thus the accelerator's active time is short and the idle time is long. To better illustrate the problem, we define E_T, t_T, t_S, t_I, T_A, P_S, P_I, P_A, and $P_{Average}$, where E, P, t, T, S, I and A stand for Energy, Power, time, Total, Scaling, Idle and Active respectively. Their relationship is shown in the following equations:

$$t_T = t_I + t_S + t_A \tag{6}$$

$$E_T = t_I \times P_I + t_S \times P_S + t_A \times P_A \tag{7}$$

$$P_{Average} = E_T / t_T \tag{8}$$

The total time t_T is fixed and determined by the actual need, and t_S is the time required to scale voltage and frequency and is determined by the VFS platform. The t_A is the time for the CNN accelerator to process one frame of an image and is determined by its running frequency for a specific CNN accelerator. P_I is the power consumption when the accelerator is idle, P_S is the average power during the voltage and frequency scaling period, and P_A is the power consumption when the accelerator is running. P_I, P_S, P_A are determined by both frequency and supply voltage. The target is to find the frequency and voltage combination that gives a minimal $P_{Average}$.

C. VFS Policy

We can reduce the supply voltage to some extent if the accelerator is not running at its highest frequency, and we call the lowest supply voltage that guarantees the normal operation of the accelerator the *optimal voltage*. The optimal voltage is mainly determined by the accelerator's running frequency and the device parameters. Thus the power optimization problem is further simplified to find the optimal running frequency to minimize $P_{Average}$. What's more, the supply voltage of the PL can be scaled to the minimum that the configuration will not be lost. Therefore, the workflow for each cycle is as follows: the accelerator is set to the running frequency and optimal voltage first and then process one frame of an image. Once the computation is finished the clock frequency and supply voltage is scaled to the idle frequency and voltage to save static power. The frequency scaling time can be neglected but that of voltage cannot. If there is not enough time to scale the voltage, only the frequency is scaled to a minimum. **Algorithm 1** shows the detailed steps to calculate $P_{Average}$ at each running frequency with our policy. After that, we choose the running frequency with a minimum $P_{Average}$ as the solution.

Algorithm 1 Average Power Calculation at a Specific Frequency

Input: $Frequency$
Output: $P_{Average}$
  if $t_T - t_A \geq t_S$ **then**
    $t_I = t_T - t_A - t_S$
    $E_T = t_I \times P_{Imin} + t_S \times \frac{P_{IminF@OV} + P_{min}}{2} + t_A \times P_A$
  else
    $t_I = t_T - t_A$
    $E_T = t_I \times P_{IminF@OV} + t_A \times P_A$
  end if
  $P_{Average} = E_T / t_T$

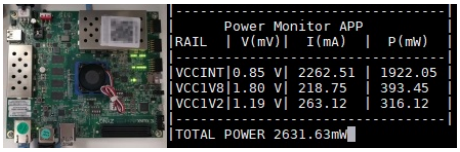

Fig. 5. ZCU104 board and power monitor.

IV. Experiment Results and Analysis

In this section, we first introduce our experimental setup and then give the performance and power consumption results of our implemented CNN-VFS system. After that, we show the energy optimization results of the CNN accelerators after applying our policies. At last we compare our work with state-of-the-art approaches.

A. Experimental Setup

We first build an SDSoC [10] hardware platform with VFS support targeting ZCU104 in Vivado 2018.2, and then synthesize the CNN accelerator using SDSoC 2018.2. To better describe the relationship between the supply voltage and the power consumption, we monitor the power of the PL rather than the whole chip. Thanks to the PMIC, we can monitor the power consumption of each component on FPGA without any extra equipment, as shown in Fig.5. VGG16 [11] is chosen as the case study and we cut off half of the kernels of the CONV layers (except CONV5) and replace the FC layer with average pooling to fit the embedded FPGA. In order to demonstrate the effect of our data reuse strategy, we implement two accelerators, the one with our data reuse strategy is called Accelerator A, and the one without is called Accelerator B. Both of the accelerators' parallelism are set to 32×32. We quantize the VGG16 to Fix8 according to the guidance of [12] and achieve 66.28% Top-1 accuracy on imagenet dataet.

B. Performance and Power Consumption

Fig.6 shows the performance and power efficiency change with frequency respectively. The peak performance of Accelerator A is 300GOPS while that of Accelerator B is 205GOPS, about 50% performance improvement is achieved with our data reuse strategy. The power efficiency is given by Giga-Operations-Per-Second-Per-Watt (GOPS/W) and all the following data is measured with Accelerator A. Fig.7 shows the CNN accelerator's optimal voltage and power consumption change with frequency respectively, where O stands for optimal, I stands for idle, P stands for power, and N stands for normal. We sweep the voltage from 850mV to 650mV at each frequency and record the optimal voltage. Our experiment shows that the minimum supply voltage for ZCU104 is 680mV, under which the system may be unstable. The optimal idle power (OIP) is measured when the accelerator is not running but the accelerator's

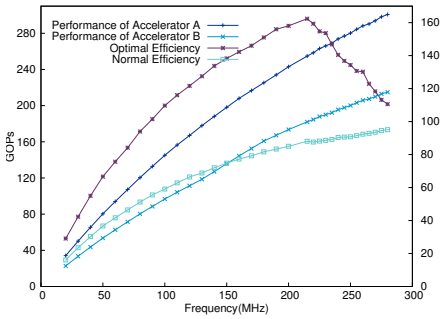

Fig. 6. Performance and power efficiency change with frequency respectively.

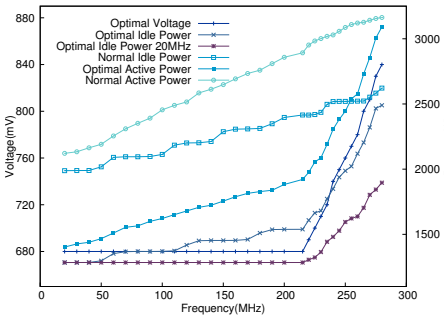

Fig. 7. CNN accelerator's optimal voltage and power consumption change with frequency respectively.

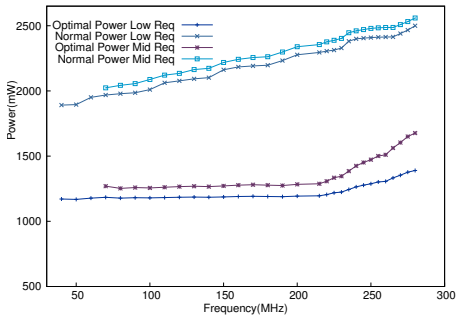

Fig. 8. Average power under different performance requirements changes with frequency respectively.

TABLE II
ENERGY EFFICIENCY COMPARISON WITH OTHER WORKS

Reference	[6]	[13]	our work
FPGA chip	XC7Z045	XC7Z045	XCZU7
Frequency	150MHz	100MHz	220MHz
Network	VGG16	VGG16	VGG16
Precision	Fix16	Fix16	Fix8
DSPs(used/total)	780/900	824/900	524/1702
Performance(GOPs)	137	230	258
Power(W)	3.15	2.92	2.12
Effiency(GOPs/W)	43.5	78.7	121.7

frequency and voltage remains. The optimal idle power at 20MHz (OIP20) is measured when the accelerator's clock is set to 20MHz and the voltage remains at the original voltage. The optimal active power (OAP) is measured when the accelerator is running at the optimal voltage while the normal active power is measured when the accelerator is running at 850mV. We can see that the OAP is about 800mW lower than normal active power (NAP) when the frequency is under 200MHz. As the frequency goes up, the optimal voltage goes up correspondingly thus the gap narrows. Fig.6 shows the power efficiency under normal voltage goes up with the frequency while that under optimal voltage first goes up and then goes down as the optimal voltage gets closer to normal voltage. The peak optimal power efficiency is 160GOPS/W, achieved at 210MHz.

C. Power Optimization

We define two performance requirements, namely low performance and middle performance. The low performance scenario requires 5FPS, middle performance scenario requires 10FPS and high performance scenario requires 30FPS. For low and middle performance scenario, there remains design space to optimize the power consumption with our VFS policy. Fig.8 shows the average power of the accelerator using our VFS policy under different performance requirements. In the low performance scenario, the minimum average power with VFS is 1167mW while that without VFS is 1891mW, in the middle performance scenario, the minimum average power with VFS is 1252mW while that without VFS is 2176mW. In summary, about 40% energy can be saved with VFS under both low and middle performance scenarios.

D. Comparison with other works

Most of the previous works targeting edge-computing choose ZC706. To give a fair comparison, we measure the static power of the CPU of both ZC706 and ZCU104 and remove it from the total power consumption. The static power of the CPU part of ZC706 is 6.5W while that of ZCU104 is 12W. We choose the performance and power at 220MHz with the optimal voltage as a comparison. As shown in Table II, our work achieve the best performance and power efficiency.

V. CONCLUSION

In this paper, we propose a novel CNN accelerator that optimizes the power and performance utilizing data reuse and loop unrolling. We then present a novel FPGA-based VFS platform which is efficient to measure and alter the CNN accelerator's voltage and frequency settings. Based on the platform, we devise a model-free VFS policy that achieves power optimization. We apply the VFS policy to our proposed CNN accelerator platform, and save about 40% energy under both low performance and middle performance requirements.

REFERENCES

[1] Eriko Nurvitadhi, Ganesh Venkatesh, et al. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proc. FPGA*, pages 5–14. ACM, 2017.

[2] Xiaofan Zhang, Junsong Wang, et al. Dnnbuilder: an automated tool for building high-performance dnn hardware accelerators for fpgas. In *Proc. ICCAD*, page 56. ACM, 2018.

[3] Lin Bai, Yiming Zhao, et al. A cnn accelerator on fpga using depthwise separable convolution. *IEEE TCAS-II*, 65(10):1415–1419, 2018.

[4] Kaiyuan Guo, Lingzhi Sui, et al. Angel-eye: A complete design flow for mapping cnn onto embedded fpga. *IEEE TCAD*, 37(1):35–47, 2018.

[5] Yufei Ma, Yu Cao, et al. Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. In *Proc. FPGA*, pages 45–54. ACM, 2017.

[6] Jiantao Qiu, Jie Wang, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proc. FPGA*, pages 26–35. ACM, 2016.

[7] Jose Luis Nunez-Yanez, Mohammad Hosseinabady, and Arash Beldachi. Energy optimization in commercial fpgas with voltage, frequency and logic scaling. *IEEE Trans. Computers*, 65(5):1484–1493, 2016.

[8] Heng Yu, Yajun Ha, and Jing Wang. Quality optimization of resilient applications under temperature constraints. In *Proc. Computing Frontiers*, pages 9–16. ACM, 2017.

[9] Heng Yu, Bharadwaj Veeravalli, et al. Dynamic scheduling of imprecise-computation tasks on real-time embedded multiprocessors. In *Intl. Conf. Computational Science and Engineering*, pages 770–777. IEEE, 2013.

[10] https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[12] https://github.com/Xilinx/CHaiDNN.

[13] Qingcheng Xiao, Yun Liang, et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas. In *Proc. DAC*, pages 1–6. IEEE, 2017.