PipeCNN: An OpenCL-Based Open-Source FPGA Accelerator for Convolution Neural Networks

Dong Wang, Ke Xu and Diankun Jiang
Institute of Information Science
Beijing Jiaotong University
Beijing 100044, China
Email: {wangdong, 17112071, 16125141}@bjtu.edu.cn

***Abstract*—Convolutional neural networks (CNNs) have been employed in many applications, such as image classification, video analysis and speech recognition. Being compute-intensive, CNNs are widely accelerated by GPUs with high power dissipations. Recently, studies were carried out exploiting FPGA as CNN accelerator because of its reconfigurability and advantage on energy efficiency over GPU, especially when OpenCL-based high-level synthesis tools are now available providing fast verification and implementation flows. In this paper, we demonstrate PipeCNN – an efficient FPGA accelerator that can be implemented on a variety of FPGA platforms with reconfigurable performance and cost. The PipeCNN project is openly accessible, and thus can be used either by researchers as a generic framework to explore new hardware architectures or by teachers as a off-the-self design example for any academic courses related to FPGAs.**

I. Introduction

Convolutional neural network (CNN) [1], [2], as an emerging deep learning architecture, has received huge attentions in various applications, such as video surveillance, image searching, speech recognition, and robot vision. Currently, GPUs are widely adopted as hardware accelerators for training deep neuron networks. Yet they are generally energy inefficient for embedded applications. FPGAs, which provide massive processing elements, reconfigurable interconnections and lower power dissipation, are naturally suitable to implement neural network circuits. Moreover, FPGAs are also flexible with reduced data precision at circuit level, which will reduce the memory footprint and bandwidth requirements, resulting in better energy efficiency than GPUs.

Studies, such as [4], [5], have reported efficient CNN accelerators on embedded FPGA platforms. However, traditional register-transfer-level (RTL) design flows adopted in these studies require deep background knowledge in digital circuit design and great effort in writing complex RTL codes, practicing time-consuming simulations and compilations before one can actually run accelerators on hardware. As the rapid development in deep learning areas, the unfriendly features of RTL-based design scheme hinder domain experts from utilizing FPGAs to explore new architectures for neural network accelerators.

High-Level Synthesis (HLS) tools, which enable automatic compilation from high-level programs (C/C++) to low-level RTL specifications, have became increasingly popular in both academic and industrial fields. Compared with traditional methodology, the HLS tools provide faster hardware development cycle and software-friendly program interfaces that can be easily integrated with user applications [3].

Ke Xu (17112071@bjtu.edu.cn) is corresponding author of this paper. This work was partially supported by NNSF of China Grant No. 61574013.

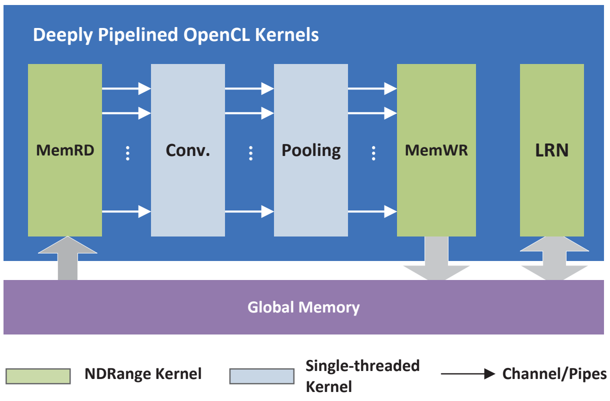


Fig. 1. The top-level architecture of PipeCNN.

In this paper, we introduce PipeCNN, an efficient OpenCL-based CNN accelerator on FPGAs. A set of configurable OpenCL kernels are designed to accelerate a wide range of neural network models. Throughput and memory bandwidth optimization schemes are also presented and discussed. All the design files are openly accessible and can be downloaded from [6].

In the final demo, PipeCNN was implemented and evaluated on three different FPGA platforms, including Cyclone-V SEA5 SoC, Stratix-V GXA7 and Arria-10 AX115. CNN-based image classification applications were accelerated by PipeCNN. The processing speed and power consumption were measured and demonstrated at runtime showing scalable performance and cost that can meet different application requirements and resource constrains.

II. Architecture Design and Optimization

A. Accelerator Architecture

As shown in Fig. 1, PipeCNN consists of a group of OpenCL kernels that are cascaded by using Altera's OpenCL extension Channels. Two data mover kernels, namely MemRD

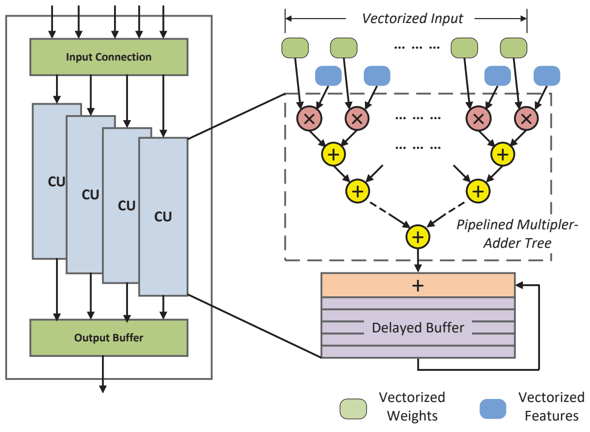


Fig. 2. The hardware architecture of the convolution kernel.

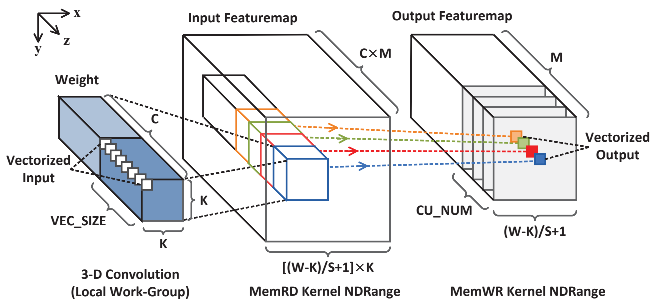


Fig. 3. Data and work-item mapping scheme of the data mover kernels.

and MemWR, transfer feature map and weight data from/to the global memory (i.e., the external DDR memory) feeding other kernels with high throughput data streams. The Convolution kernel (Conv.) is designed to accelerate the most compute-intensive computations in CNNs, i.e., convolution layer and the FC layer. The Pooling kernel performs subsampling operations directly on the output data stream of the Conv. kernel. The Local Response Normalization (LRN) kernel fetches data from global memory and performs normalization on the feature map of neighboring neurons [1]. This architecture has the following advantages: 1) the cascaded kernels form a deep pipeline, which can execute a serial of basic CNN operations without the need of storing interlayer data back to external memory. It significantly relieves the demand on memory bandwidth which is essential for embedded FPGAs. 2) we use a single hardware kernel to implement both the convolution and FC layers which further improves the efficiency of hardware resource utilizations. Detailed kernel designs and corresponding optimization schemes are as follow:

1) Convolution Kernel: The convolution operation is essentially a 3-dimensional (3-D) multiply-accumulate (MAC) operation that can be defined by

$$D_o(f_o, y, x) = \sum_{f_i=1}^{C_l} \sum_{k_y=0}^{K-1} \sum_{k_x=0}^{K-1} W_l(f_o, f_i, k_y, k_x) \cdot D_i(f_i, y + k_y, x + k_x) \tag{1}$$

where $D_i(f_i, y, x)$ and $D_o(f_o, y, x)$ denote the neurons at position (x, y) in the input feature map f_i and output feature map f_o, respectively. $W_l(f_o, f_i, y, x)$ represents the corresponding weights in the l-th layer that gets convolved with f_i. The size of the convolution filters is $K \times K$, while the total number of input feature maps is C_l. In this paper, we propose to implement (1) by using a HLS-friendly 1-D convolution structure which flattened the 3-D convolution as follow:

$$D_o(f_o) = \sum_{f_i'=1}^{C_l \times K \times K} W_l(f_o, f_i') \cdot D_i(f_i') \tag{2}$$

In this way, nested-loops can be avoided in kernel code, and an efficient convolution pipeline structure consisted of a multiplier-adder tree with a delayed buffer is generated by the compiler as Fig. 2 shows. When an appropriate buffer depth is selected, the proposed structure can be efficiently pipelined by the OpenCL compiler with an initial interval of only one clock cycle. Each convolution pipeline constitutes a compute unit (CU) and the kernel consists of multiple CUs to perform parallel convolutions.

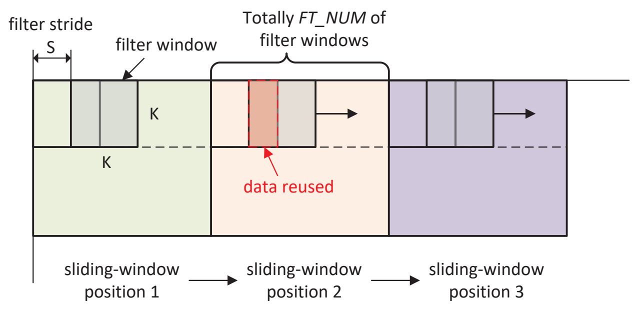
2) Data Mover Kernels: Two multi-mode 3-D NDRange kernels are designed to fetch/store data from/to the global memory for the computation pipelines. Data and work-item mapping schemes are illustrated in Fig. 3. In convolution mode, the MemRD kernel launches with a global work-item number of $([(W-K)/S+1] \times K, [(W-K)/S+1] \times K, C' \times M)$, while the MemWR kernel works in an NDRange of $((W-K)/S+1, (W-K)/S+1, M)$. Variables W and H represent the width and height of the input feature map, while S denotes the stride of each filtering operations. To enable concurrent work-group processing, the work-items are arranged into multiple concurrent work-groups, each of which has a local work-group size of *(K, K, C')*.

In FC mode, both the input feature and weight data are 1-D vectors as defined in Eq. (2). Directly launching MemRD kernel with only one classification task will reduce the opportunity of data reuse in weights. Therefore, we introduce batched processing capability in MemRD. For instance, a batch of 64 classification tasks can be processed with a single kernel launch by mapping all the input feature maps as a single 3-D data set with the NDRange size of $(8, 8, C)$.

3) Other Kernels: Besides the most compute-intensive convolution kernel, we also designed other OpenCL kernels to accelerate the widely used layer operations in CNNs, such as pooling, LRN and etc. Therefore, PipeCNN can process the complete CNN forward computation flow with very little involvement of host CPU, resulting in high throughput and low latency.

B. Performance and Bandwidth Optimizations

1) Throughput Optimization: To further improve the throughput of the convolution kernel, data vectorization and parallel CUs are introduced. As shown in Fig. 3, the input features D_i and weights W_l at same the position (x, y) from adjacent feature maps are grouped as one vectorized input. The size of the vectorized data is controlled by the



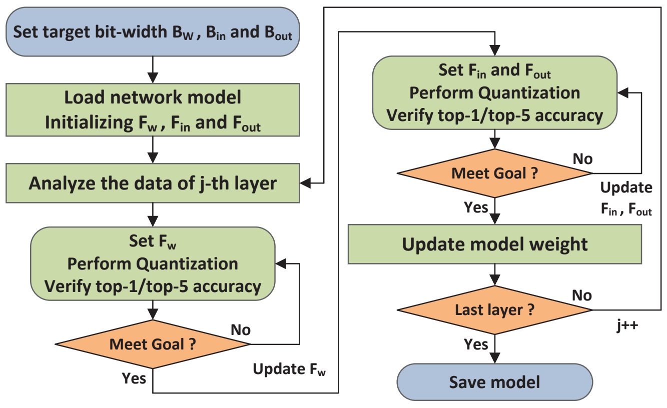
filter stride
S
filter window
Totally FT_NUM of filter windows
K
K
data reused
sliding-window position 1
sliding-window position 2
sliding-window position 3

Fig. 4. Sliding-window-based data buffering scheme.

design parameter *VEC_SIZE*. The vectorized data streams are fetched by the MemRD kernel and send to multiple CUs in the Conv. kernel by OpenCL Channels as the colored line shows. The number of parallel CUs used is controlled by another parameter *CU_NUM*. Simply changing the value of the parameters *VEC_SIZE* and *CU_NUM*, the implemented design can achieve scalable performance and hardware cost without the need of modifying the kernel code. In the final design, two 8×8 multipliers were also grouped and mapped into one DSP block by manually inserting Altera's IP blocks in the kernel code to improve the efficiency of the pipeline.

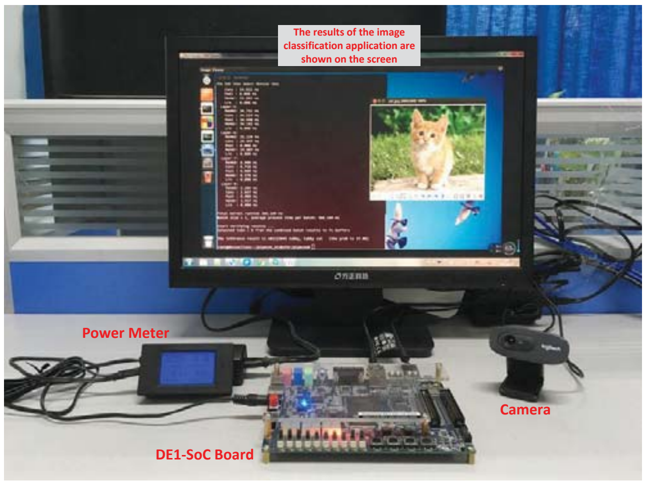
2) Bandwidth Optimization: To relieve the pressure on external memory bandwidth, we introduce a sliding-window-based data buffering scheme. As shown in Fig. 4, the filter stride S of the convolution window is usually smaller than the filter size K (in most cases, $S = 1$). Therefore, a large portion of data can be reused during the convolution computation. To exploiting data reuse, the MemRD kernel fetches a window of data that covers the area of *FT_NUM* of convolution filters each time, and caches the data in the on-chip buffers. And then, for successive convolution filtering operations, feature-map data and weight are repeatedly loaded from local memories avoiding access of external memory. To demonstrate the effectiveness of this scheme, we profiled the DDR memory bandwidth of implementations with different values of *FT_NUM* on different FPGA platforms. The average bandwidth reductions achieved reached up to 50%.

3) Fixed-point Optimization: Implementing fixed-point arithmetics instead of floating-point computations on FPGAs can significantly reduce hardware costs and memory bandwidth requirements. In this paper, we quantize both the model and intermediate computation results with a uniformed word length and variable fractional bit-width for each CNN layers. Fig. 5 illustrates how data quantization is performed in this demo. In each layer, the weight, input feature map and output feature map data are presented in fixed-point numbers as $Q_w \cdot 2^{-F_w}$, $Q_{in} \cdot 2^{-F_{in}}$ and $Q_{out} \cdot 2^{-F_{out}}$, respectively. The variable Q denotes the fixed-point binary word with B-bit length while F denotes a bias presenting the number of fractional bits of the fixed-point number. The quantization flow searches for the group of parameters B_w, F_{in} and F_{out} that minimizes the hardware cost while satisfying the accuracy



Set target bit-width B_W, B_in and B_out
Load network model Initializing F_w, F_in and F_out
Analyze the data of j-th layer
Set F_w Perform Quantization Verify top-1/top-5 accuracy
Meet Goal ?
No
Update F_w
Yes
Set F_in and F_out Perform Quantization Verify top-1/top-5 accuracy
Meet Goal ?
No
Update F_in, F_out
Yes
Update model weight
Last layer ?
No
j++
Yes
Save model

Fig. 5. Fixed-point model quantization flow used in this demo.



The results of the image classification application are shown on the screen
Power Meter
Camera
DE1-SoC Board

Fig. 7. Demo setup of the DE1-SoC board. The power dissipation was measured at runtime when performing image classification application.

bound at the same time. In this demo, the AlexNet and VGG-16 models are quantized with 8-bit word length with less than 1% loss in top1/5 accuracy.

III. Demonstration and Evaluation

A. Demonstration Setup

In the demonstration, we implemented CNN-based image classification applications on three different FPGA platforms. The detailed information are summarized in Table I. The DE5-net and DE5a-net boards were installed in a desktop computer, which was equipped with Intel i5-4690K CPU and 64GB memories, while the DE1-SoC board was connected to the computer and accessed through VNC viewer. The OpenCL kernel codes are compiled by using Altera OpenCL SDK v16.1. The host programs first load images from hard disks or web camera, and then send them to the FPGA accelerators to perform CNN forward computations. Two large-scale CNN models: AlexNet (8 layers) and VGG-16 (16 layers) were used as benchmarks to measure the performance. For each board, an external power meter was used to measure the power

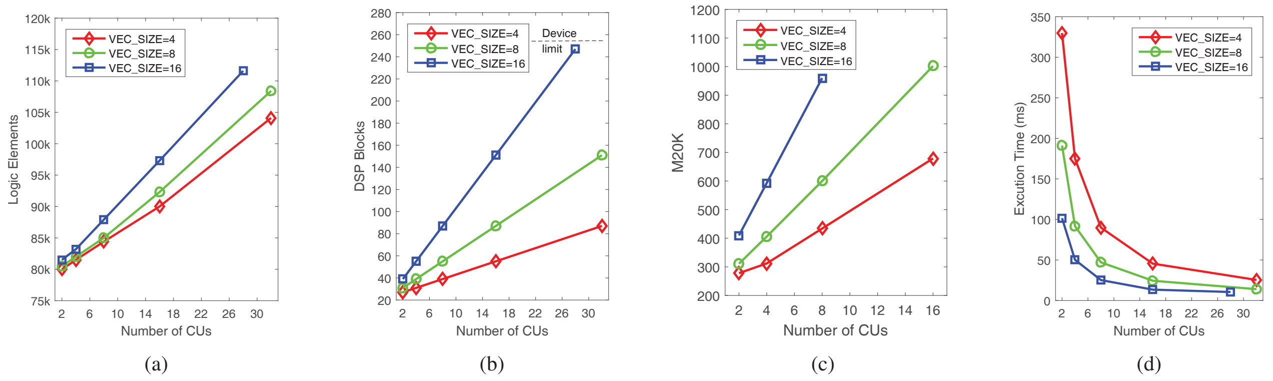


Fig. 6. Design space exploration for PipeCNN on the DE5-net FPGA platform. The execution time was measured by using AlexNet model.

TABLE I
SUMMARY OF THE MEASURED PERFORMANCE, COST AND POWER CONSUMPTION ON DIFFERENT PLATFORMS.

Platform	FPGA Type	Resource Capacity	Resource Consumed	Execution Time		Frequency	Board Power
				AlexNet	VGG-16		
DE1-soc	Cyclone-V SEA5	85K LEs 87 DSPs	45K LEs 68 DSPs	140ms	1,928ms	122Mhz	2.1W
DE5-net	Stratix-V GXA7	622K LEs 256 DSPs	112K LEs 247 DSPs	15ms	254ms	198Mhz	27W
DE5a-net	Arria-10 GX1150	1,150K LEs 1,518 DSPs	322K LEs 683 DSPs	5ms	110ms	218Mhz	26W

consumption at runtime. Fig 7 shows how the DE1-SoC board was setup for the demonstration.

B. Design Space Exploration

As discussed in Section II-B, two design parameters *VEC_SIZE* and *CU_NUM* are used to control the throughput and hardware cost of the FPGA accelerator. Therefore, design space explorations can be quantitatively performed by implementing the accelerator with different parameter configurations. Fig. 6 illustrates the exploration results on the DE5-net platform. It can be observed from Fig. 6-(b) and (d) that the accelerator with parameters *VEC_SIZE=16* and *CU_NUM=28* maximizes the DSP utilization and uses the shortest time for image classification.

C. Performance Evaluation

For each platform, we have performed the design space exploration and found out the implementation that achieved the best performance. The results are summarized in Table I. For instance, the best performance that PipeCNN achieved on the DE5-net platform is 67 fps (15 ms/img) for AlexNet model. To demonstrate how fast PipeCNN can accelerate CNN computations, we also performed image classifications on CPU by using the Caffe tool installed on our desktop computer. The execution time for AlexNet and VGG-16 is 189 ms and 1547 ms, respectively. We can see that using FPGA-based accelerator can achieve up to 37× performance improvement in implementing CNN-based image classification applications. The work of [3] also presents an OpenCL-based CNN accelerator on Arria-10 FPGA. The design adopted Winograd transformations to reduce the number of computations required by the convolution layer, and thus achieved much higher performance (i.e., 1020 fps for AlexNet) than ours. In future works, we will explore sparse convolution algorithms to further improve the performance of PipeCNN.

IV. CONCLUSION

This paper demonstrated an open-source OpenCL-based FPGA accelerator for convolutional neural networks. An efficient hardware architecture with pipelined kernels was presented. Throughput and memory bandwidth optimization schemes were also discussed. The implemented design show scalable performance and cost on multiple FPGA platforms.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton and et al., "ImageNet classification with deep convolutional neural networks," *in Proc. Neural Information Processing Systems (NIPS'12)*, 2012.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[3] U. Aydonat, S. O'Connell, D. Capalija, A.C. Ling and G.R. Chiu, "An OpenCL Deep Learning Accelerator on Arria 10" *in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*, 2017.
[4] J. Qiu, J. Wang, S. Yao and et al., "Going deeper with embedded FPGA platform for convolutional neural network," *in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*, 2016.
[5] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: a scalable deep learning accelerator unit on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.
[6] https://github.com/doonny/PipeCNN.