www.ijiarec.com

ISSN:2348-2079

Volume-7 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Optimizing the convolution operation to accelerate deep neural networks on FPGA

V. Saravanan[1], R.Susmitha[2]

Assistant Professor[1]/ VLSI Design PG Scholar[2]

Department of ECE, Knowledge Institute of Technology, Salem.

ABSTRACT

As convolution contributes most operations in convolutional neural network (CNN), the convolution acceleration scheme significantly affects the efficiency and performance of hardware CNN accelerator. Convolution involves multiply and accumulates operation with four levels of loops, which results in a large design space. Prior works either employ limited loop optimization techniques ,e.g, loop unrolling, tiling, and interchange, or only tune some of the design variables after the accelerator architecture and dataflow are already fixed .Without fully studying the convolution loop optimization before the hardware design phase ,the resulting accelerator can hardly exploit the data reuse and manage data movement efficiently. To overcome these barriers by quantitatively analyzing and optimizing the design objectives (e.g. Memory access) of the CNN accelerator based on multiple design variables. The presented CNN acceleration scheme and architecture are demonstrated by implementing end-to-end CNNs including NiN, VGG-16, and ResNet-50/ResNet-152 for interference. For VGG-16 CNN, overall throughputs348 GOPS and 715 GOPS on Intel StatrixV and Arria 10 Fpgas, respectively.

Keywords: Convolution, Acceleration, Field programmable gate arrays, Computer architecture, System-on chip, Hardware, Optimization.

INTRODUCTION

The field-programmable gate arrays (FPGA) are fast becoming the platform of choice for accelerating the inference phase of deep convolutional neural networks (CNNs). In addition to their conventional advantages of re configurability and shorter design time over application-specific integrated circuits (ASICs) to catch up with the rapid evolving of CNNs, FPGA can realize low latency inference with competitive energy efficiency (_10–50 GOP/s/W) when compared to software implementations on multicore processors with GPUs. Deep CNN algorithms have tens to hundreds of layers, with significant differences between layers in terms of sizes and configurations. The limited computational resources and storage capacity on FPGA makes the task of optimal mapping of CNNs (e.g., minimizing latency subject to energy constraints or vice versa) a complex and multidimensional optimization problem.

The high cost of off-chip communications another major impediment to achieving higher performance and lower energy. In fact, the energy cost associated with the large amount of data movements and memory accesses often exceeds the energy consumption of the computations. For these reasons, energy-efficient hardware acceleration of CNNs on a FPGA requires simultaneous maximization of resource utilization and data reuse, and minimization of data communication. More than 90% of the operations in a CNN involve convolutions. Therefore, it stands to reason that acceleration schemes should focus on the management of parallel computations and the organization of data storage and access

Author for correspondence:
Department of ECE, Knowledge Institute of Technology, Salem.

across multiple levels of memories, e.g., off-chip dynamic random access memory (DRAM), on-chip memory, and local registers. In CNNs, convolutions are performed by four levels of loops that slide along both kernel and feature maps.

MAIN CONTRIBUTIONS OF THIS PAPER INCLUDE THE FOLLOWING

- ✓ We provide an in-depth analysis of the three loop optimization techniques for convolution operations and use corresponding design variables to numerically characterize the acceleration scheme.
- ✓ The design objectives of CNN accelerators (e.g., latency, memory) are quantitatively estimated based on the configurations of the design variables.
- ✓ An efficient convolution acceleration strategy and dataflow is proposed aimed at minimizing data communication and memory access.
- ✓ A data router is designed to handle different settings for convolution sliding operations, e.g., strides and zero padding, especially for highly irregular CNNs.
- ✓ Corresponding hardware architecture is designed that fully utilizes the computing resources for high performance and efficiency, which is uniform and reusable for all the layers.
- ✓ The proposed acceleration scheme and architecture is validated by implementing large-scale deep CNN algorithms.

LITERATURE SURVEY

Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolution Neural Networks

Eyeriss is an accelerator for state of heart deep convolutional neural networks (CNNs). It optimizes for the energy efficiency of the entire system, including the accelerator chip and off-chip DRAM, for various CNN shapes by reconfiguring the architecture. CNNs are widely used in modern AI systems but also bring challenges on throughput and energy efficiency to the underlying hardware. This is because its computation requires a large amount of data, creating significant data movement from on-chip and off-chip that is more energy-consuming than computation. Minimizing data movement energy cost for any CNN shape, therefore, is the key to high throughput and energy efficiency.

Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA

Convolutional Neural Network (CNN) has become a successful algorithm in the region of artificial intelligence and a strong candidate for many computer vision (CV) algorithms. But the computation complexity of CNN is much higher than traditional algorithms. With the help of GPU acceleration, CNN based applications are widely deployed in servers. 6 However, for embedded platforms, CNN-based solutions are still too complex to be applied. Various dedicated hardware designs on FPGAs have been carried out to accelerate CNNs, while few of them explore the whole design flow for both fast deployment and high power efficiency. In this paper, we investigate state-of-the-art CNN models and CNN based applications.

Scalable and Modularized RTL Compilation of Convolutional Neural Networks onto FPGA

Despite its popularity, deploying Convolutional Neural Networks (CNNs) on a portable system is still challenging due to large data volume, intensive computation and frequent memory access. Although previous FPGA acceleration schemes generated by high-level synthesis tools (i.e., HLS, Open CL) have allowed for fast design optimization, Hardware inefficiency still exists when allocating FPGA resources to maximize parallelism and throughput. A direct hardware-level design (i.e., RTL) can improve the efficiency and achieve greater acceleration. However, this requires an in-depth understanding of both the algorithm structure and the FPGA system architecture.

EXISTING METHOD

Recently reported CNN algorithms involve a large amount of data and weights. For them, the

on-chip memory is insufficient to store all the data, requiring gigabytes of external memory. Therefore, a typical CNN accelerator consists of three levels of storage hierarchy:

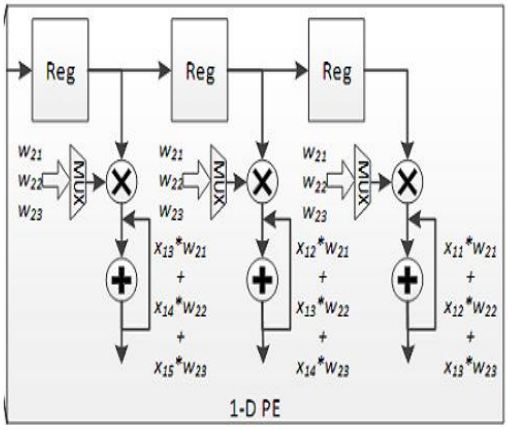
- External memory;
- on-chip buffers; and
- Registers associated with the processing

Engines (PEs)

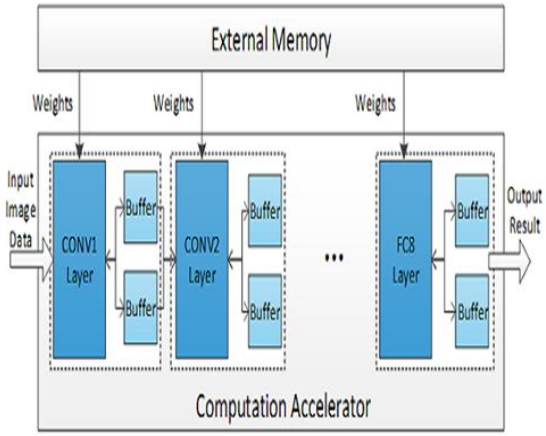
The basic flow is to fetch data from external memory to on-chip buffer, and then feed them into registers and PEs. After the PE computation completes, results are 10 transferred back to on-chip buffers and to the external memory if necessary, which will be used as input to the subsequent layer. To improve the performance and increase the throughput, a pipelined structure can be applied on the CNN model with different parallelism strategies for different layers. Consequently, numerous input data and weights are required, which leads to a high data access workload.

Figure General Architecture

To solve this problem, intermediate results between layers are stored in on-chip buffers. In this way, the data access workload can be reduced significantly. Weights are stored in the external memory because the number of weight is too large. Furthermore, on-chip buffers also facilitate data reuse.



Architecture of processing element



Architecture of accelerator

In the accelerator, layers are implemented by modules where 2 ping-pong buffers are utilized to store the intermediate data. During computing, these 8 layers request for weights to the "Read

Weight Controller". Then, the controller makes arbitration within the request signals and read weights from the external memory.

DRAWBACKS

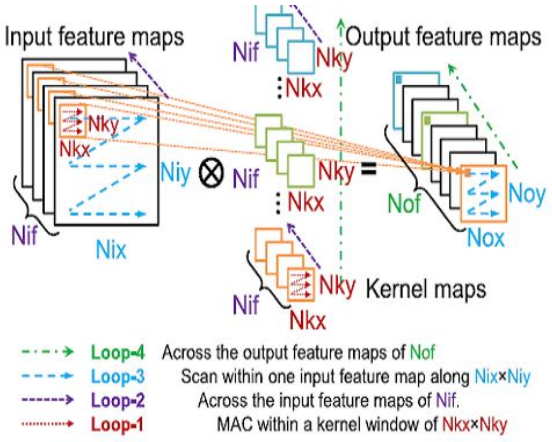
- Convolutions involve multiply and accumulate operations with four levels of loops, which results in a large design space. Prior works either employ limited loop optimization techniques, e.g., loop unrolling, tiling, and interchange, or only tune some of the design variables after the accelerator architecture and dataflow are already fixed.
- This is due to the fact that modern FPGAs allow customization of the architecture and can exploit the availability of hundreds to thousands of on-chip DSP blocks. However, significant challenges remain in mapping CNNs onto FPGAs. The state of-the-art CNNs require a large number (> 1 billion) of computationally intensive task (e.g., matrix multiplications on large numbers), involving a very large number of weights. Deep CNN algorithms have tens to hundreds of layers, with significant differences between layers in terms of sizes and configurations.
- The limited computational resources and storage capacity on FPGA make the task of optimal mapping of CNNs (e.g., minimizing latency subject to energy constraints or vice versa) a complex and multidimensional optimization problem.
- The old convolution acceleration scheme significantly affects the efficiency and performance of a hardware CNN accelerator.

PROPOSED SYSTEM

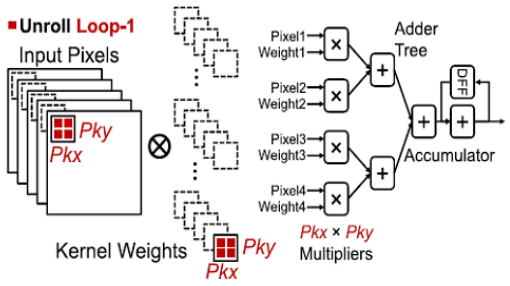
To overcome the drawback of existing method we can use loop optimization technique for either adder or multiplier algorithm.

Loop Optimization and Design Variables Multiple dimensions are used to describe the sizes of the feature and kernel maps of each convolution layer for a given CNN. The hardware design variables of loop unrolling and loop tiling will determine the acceleration factor and hardware footprint.

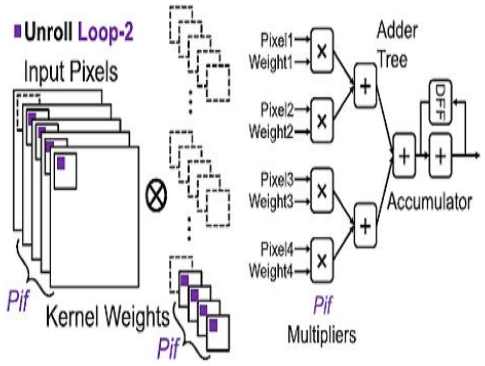
The loop unrolling design variables are (Pkx, Pky), Pif, (Pox, Poy), and Pof, which denote the number of parallel computations. The loop tiling design variables are (Tkx, Tky), Tif, (Tox, Toy), and Tof, which represent the portion of data of the four loops stored in on-chip buffers. The constraints of these dimension and variables aregiven by 1 _ *P*_ _ *T*_ _ N_, where N_, *T*_, and *P*_ denoteany dimension or variable that has a prefix of capital N, T, andP, respectively. For instance, 1_ Pkx_ *Tkx*_ *Nkx*. By default, *P*_, *T*_ and *N*_ are applied to all convolution layers.



Four levels of convolution loops and their dimensions.



Unroll loop-1 and its corresponding computing architecture.



Unroll loop-2 and its corresponding computing architecture.

The relationship of input and output variables is constraintby (1)–(3), where S is the stride of the sliding window andthe zero padding size is included in *Nix*, *Niy*, *Tix*, and

TiyNix = (Nox−1) S+ Nkx
Niy= (Noy−1) S+ Nky (1)
Tix = (Tox−1) S+ Nkx
Tiy= (Toy −1) S+ Nky (2)
Pix = Pox
Piy= Poy (3)

PROPOSED ACCELERATION SCHEME

The optimization process of our proposed acceleration scheme is presented in this section, which includes appropriate selection of the convolution loop design variables.

Minimizing Computing Latency

We set variables P_ to be the common factors of T_ for all the convolution layers to fully utilize PEs, and T_ to be the common factors of N_ to make full use of external 17 memory transactions. For CNN models with only small common factors,it is recommended to set _N_/T__ −N_/T_ and _T_/P__−T_/P_ as small as possible to minimize the inefficiencycaused by the difference in sizes of CNN models.

Minimizing Partial Sum Storage

To reduce the number and movements of partial sums, bothLoop-1 and Loop-2 should be computed as early as possible or unrolled as much as possible. To avoid the drawback of unrolling Loop-1 as discussed in Section IV and maximize the data reuse, we decide to unrollLoop-3 (*Pox* >1 or *Poy*>1) and Loop-4 (*Pof*>1). By this means, we cannot attain the minimum partial sum storage, as Constrained by 1 _ *P*_ _ *T*_ _ N_, the second least number of partial sum storage is achieved by (9.2) among (9.2)–(9.9)To satisfy the condition for (9.2), we serially compute Loop-1 and Loop-2 first and ensure the required data of Loop-1 and Loop-2 are buffered,

i.e.,
Tkx= Nkx,
*Tky=Nky*and
Tif= Nif.

Therefore, we only need to store *Pof*×*Pox* × *Poy*number of partial sums, This can be retained in local registers with minimum data movements.

Minimizing Access of On-Chip Buffer

The number of on-chip buffer accesses is minimized by unrolling Loop-3 to reuse weights andunrolling Loop-4 to reuse pixels. As our partial sums are kept on local registers, they do not add overhead to the buffer access and storage.

Minimizing Access of External Memory

As we first compute Loop-1 and Loop-2 to reduce partial sums, we cannot achieve the minimum number of DRAM access described in (10.1) and (10.3) inside, where neither the pixels nor the weights are fully buffered for one convolution layer.

Therefore, we can only attain the minimum DRAM access by assigning sufficient buffer size for all pixels or all weights of each layer as in (10.8) then, the optimization of minimizing the on-chip buffer size while having minimum DRAM access is formulated

Asmin

bits_BUF_px_wts.t.#T ile_pxL= 1or#Tile_wtL = 1 with_L [1, #CONVs] (20)
where#Tile_pxLand #Tile _wtL
denote the number of tilingblocks for input pixels and weights of layer *L*, respectively, and #CONVs is the number of convolution layers.bits_BUF*_px_wt*is the sum of pixel buffer size (bits_BUF*_px*) and weight buffer size (bits_BUF*_wt*), which are given by*bi ts_BUF_px_wt= bi ts_BUF_px+ bits_BUF_wt.* Both pixel and weight buffers need to be large enough tocover the data in one tiling block for all the convolution layers.This is expressed as

bits_BUF_px = MAX (words_pxL) ×pixel_datawidth with L [1, *#CONVs*] (22)
bits_BUF_wt = MAX (words_wtL) ×weight_datawidth with L [1, *#CONVs*] (23)
Where words*_pxL*and words*_wtL*denote the number ofpixels and weights of one tiling block in layer *L*, respectively.These are expressed in terms of loop tiling variables asfollows:

words_pxL=TixL×TiyL×TifL+ToxL×ToyL×TofL (24) *words_ wtL= TofL×TifL×TkxL×TkyL* (25)
Where words *_pxL* is comprised of both input and output pixels. The number of tiles in (20) is also determined by *T_* variables. In this paper, we propose to empirically find a satisfactory solution for a given on-chip memory capacity that takes advantage of the property of CNNs. CNNs normally have largepixel data volume and small weight sizes in the beginning few layers.

As we proceed into deeper layers, the pixel sizesbecome smaller with extracted features, and the weight sizesbecome larger with more channels. Where the bars denote data sizes in each convolutionlayer. To benefit from the data distribution property in different layers, we only need to make pixel buffers fully cover the last few layers and weight buffers fully cover the beginning few layers.

Then, the middle layers with both relatively large pixel and weight sizes become the constraints of the buffer sizes, and we only need to take care of these bounding layers, which significantly shrinks the solution space. The dashed are the minimal buffer sizes we found while guaranteeing minimum DRAM accesses and the bounding layers are pointed out by arrows. If this buffer size still cannot be fit into the FPGA on-chip memory, then we need to either change the tiling strategy or decrease the buffer sizes at the cost of more DRAM accesses.

SOFTWARE-DESCRIPTION

Xilinx ISE (**I**ntegrated **S**ynthesis **E**nvironment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.

The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the Model Sim logic simulator is used for system-level testing. Other components shipped with the Xilinx ISE

include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and Chip Scope Pro. Since 2012, Xilinx ISE has been discontinued in favor of Vivado Design Suite, that serves the same roles as ISE with additional features for system on a chip development. Xilinx released the last version of ISE in October 2013 (version 14.7), and states that "ISE has moved into the sustaining phase of its product life cycle, and there are no more planned ISE releases.

RESULT AND DISCUSSION

Simulation results

Coding Output

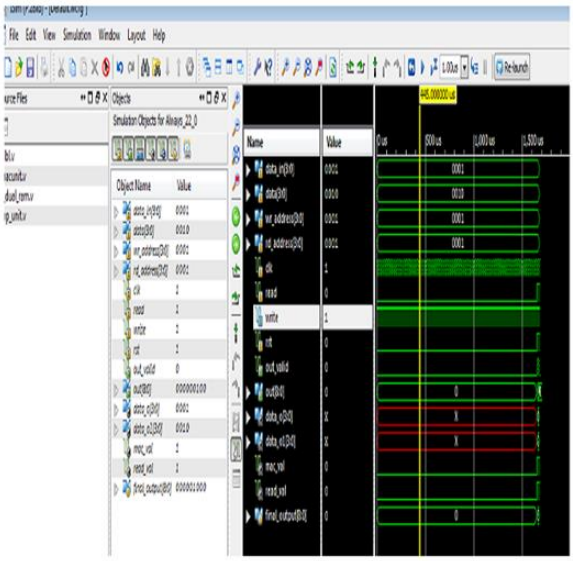
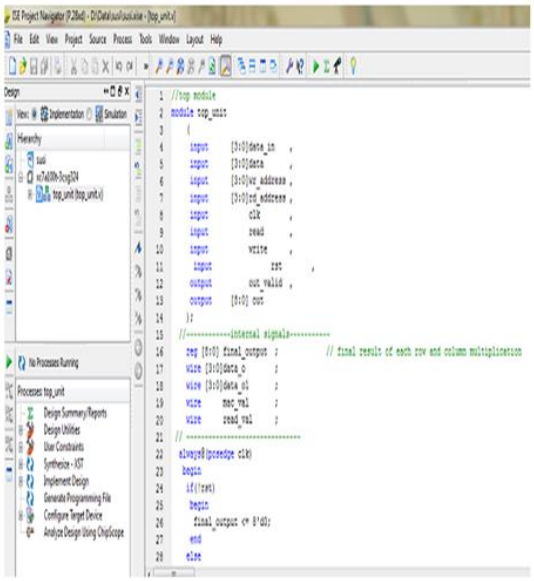


Figure 4.2 SIMULATION OUTPUT 1



CREATION OF RTL SCHEMATIC

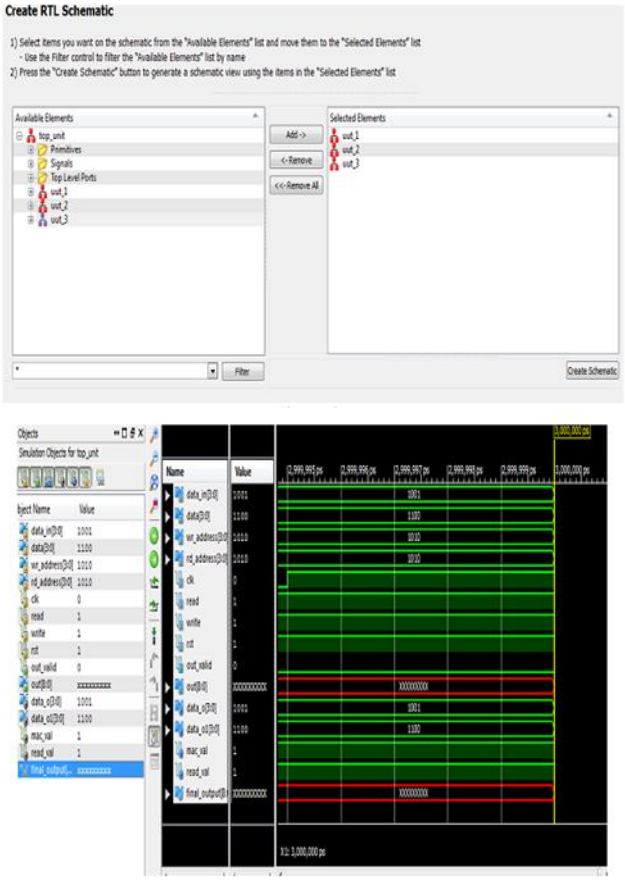


Figure 4.3 SIMULATION OUTPUT 2

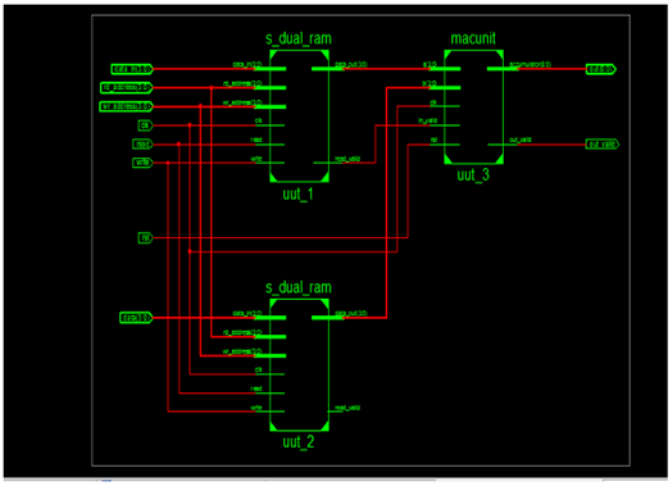
OUTPUT STEPS

data in = 0001
data = 0010
write address= 0001
read address=0001 (both write and read address must be same)
clk =clk
read = 0
write =1
rst=0

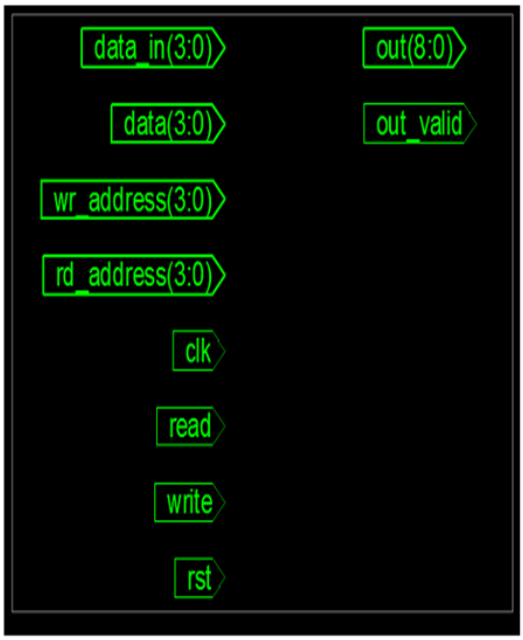
STEP 2:
rst =1

STEP 3:
read = 1

RTL SCHEMATIC



SCHEMATIC OF INPUT DATA



CONCLUSION

In this paper, we present an in-depth analysis of convolution loop acceleration strategy by numerically characterizing the loop optimization techniques. The relationship between accelerator objectives and design variables are quantitatively investigated. A corresponding new dataflow and architecture is proposed to minimize data communication and enhance throughput. Our CNN accelerator implements end-to end. The pipelining technique can increase the throughput but not necessarily the latency. The relationship between accelerator objectives and design variables are quantitatively investigated. A corresponding new dataflow and architecture is proposed to minimize data communication and enhance throughput.

REFERENCES

[1]. O. Russakovskyet al., "ImageNet large scale visual recognition challenge," Int. J. Comput. Vis., 115(3), 2015, 211–252.
[2]. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Image Net classification with deep convolutional neural networks," in Proc. Adv. Neural Inf. Process. Syst. (NIPS), 2012, 1097–1105.
[3]. M. Lin, Q. Chen, and S. Yan. "Network in network."[Online]. 2014.Available: https://arxiv.org/abs/1312.4400
[4]. K. Simonyan and A. Zisserman, "Very deep convolutional networksfor large-scale image recognition," in Proc. Int. Conf. Learn. Represent. (ICLR), 2015.
[5]. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learningfor image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, 770–778.
[6]. D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," ACM Comput. Surv. 26(4), 2014, 345–420.
[7]. Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: energy efficient reconfigurable accelerator for deep convolutional neural networks, "IEEE J. Solid-State Circuits, 51(1), 2017, 127–138.
[8]. Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture forenergy-efficient dataflow for convolutional neural networks," in Proc.ACM/IEEE Int. Symp. Comput. Archit. (ISCA), 2016, 367–379.
[9]. C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks, "in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA), 2015, 161–170.
[10]. C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in Proc. ACM Int. Symp.Low Power Electron. Design (ISLPED), 2016, 326–331.
[11]. N. Sudaet al., "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA), 2016, 16–25.
[12]. 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 Symp. Field-Program. Gate Arrays (FPGA), 2017, 55–64.