IDLA: An Instruction-based Adaptive CNN Accelerator

Peng Gao[1], Zhize Huang[1], Hanchen Ye[2], Gengsheng Chen[1]*

[1]State Key Laboratory of ASIC and System, Fudan University, No.825 Zhangheng Road, Shanghai, 201203, China
[2]University of Illinois at Urbana-Champaign, 1308 W Main St, Urbana, Illinois 61801, US
* Email: gschen@fudan.edu.cn

Abstract
In this paper, we propose an instruction-based adaptive CNN accelerator named IDLA for fast and efficient deployments of CNN models on FPGA. The hardware engine of IDLA accelerates the computation of CNN models by adaptively using different functional modules. Following a modular design fashion, the hardware engine is attentively designed to enable all these modules to work concurrently and to improve the usage efficiency of on-chip resources. Besides, layer fusion and weight reuse strategies are applied to reduce data access to DDR. Coordinating with this hardware engine, a network parser is developed to automatically analyze different CNN models to generate an optimal scheduling scheme for each CNN model. Moreover, a customized instruction set with moderate-granularity is designed to further enhance the flexibility in joint-optimization between software and hardware. We build the IDLA on a Xilinx VU9P FPGA. The experimental results show that our proposed IDLA accelerator has reached an overwhelming performance of 168.76 (ResNet18) and 277.63 (VGG16-SVD) GOPS, and an DSP efficiency of 1.62 Ops/DSP/cycle (VGG16-SVD), much better than existing works [9] and [10].

1. Introduction

In recent years, convolutional neural networks (CNNs) have replaced traditional algorithms in many application fields such as image classification, target tracking, face detection and etc. Meanwhile, being a field programmable hardware platform, FPGA helps to shorten development cycle under the assistance of high-level synthesis tools, leading to a soaring demand of FPGA-based accelerators in both cloud and edge computations.

CNN accelerators on FPGA can be generally classified into two categories: specialized and general purposed. Specialized accelerators, with customized circuit design in calculating units, memory access and data flow, can provide a significant accelerating performance [1,9]. But they usually have a big limitation in their adaptability to different CNN algorithms, which largely prevents them from being used in various scenarios. Based on these specialized accelerators, more recent researches have placed their efforts on more flexible and general-purposed solutions. Ma Y, et al. [2] use a purely modularized design to fulfill quick deployments of different deep residual CNNs. But their work ignores the optimization of data flow and memory access, which unavoidably leads to a low model adaptability and a low computing efficiency. VTA [3] takes another approach of placing its emphasis on system stack design. By using a model compiler, it has a high generality for various kinds of CNN models. However, the full functional design in its hardware processing units and data access scheme makes it difficult to provide with a competing accelerating performance.

To tackle these problems, in this paper, we present a new CNN accelerator named IDLA (Instruction-based Deep Learning Accelerator), which automatically analyzes the network structures of different CNN models and adaptively assembles hardware processing modules with custom designed data access & reuse strategies to reach a subtle balance between the adaptability and the accelerating performance. Our major contributions can be summarized as follows:

1) A customized instruction set with moderate control granularity is designed to support a wide range of CNN models and to provide a sufficient optimization space for inference.
2) A software & hardware co-optimization strategy is raised for computing acceleration. At software side, different data reuse strategies are selected adaptively and dynamically to accommodate different network structures. At hardware side, strategies of modular design method, weight reuse and layer fusion are applied to optimize the computation and to minimize the data access to DDR.
3) We build and test IDLA on a Xilinx VU9P FPGA. Compared to existing advanced works [8,9], the new design exhibits a higher performance in both its computing acceleration and DSP efficiency, together with a wide adaptability to various CNN models.

The rest of this paper is organized as follows: Section 2 gives a brief introduction of CNN calculations. Section 3 presents the design of IDLA. Section 4 gives the experimental results and analysis. Section 5 makes the summary.

2. Background

2.1 Convolutional Neural Networks

CNNs, such as VGG [4] and ResNet [5], are generally cascaded by convolutional (CONV) layers, activation layers, residual layers, fully connected (FCN) layers and etc. Among them, the CONV layers contribute a major cost in computation time and resource. The FCN layers

conduct lots of matrix-vector multiplications leading to a mass use of network weights. The other layers contribute little to computation and storage cost, but have non-negligible influence on the data flow of CNNs' inference. They also need to be treated attentively in designing an optimal CNN accelerating scheme.

2.2 Model Compression of CNN

There are already many researches on CNNs' model compression for the purpose of reducing computation and storage cost. In [6], Gupta et al. point out that using 16-bit fixed-point numbers has little impact on classification accuracy compared to using floating-point numbers. In [7], Gysel et al. propose a quantization method of using dynamic fixed-point numbers which can greatly reduce the computation on FPGA. In [8], Girshick et al. reduce the computation of FCN layers by using singular value decomposition (SVD) method, which could be widely applicable to many other CNNs' accelerating solutions.

3. IDLA Design Overview

3.1 System Design

As shown in Figure 1(a), the IDLA accelerator consists of 2 major parts. The network parser analyzes the parameters of a CNN model, then generates an optimal scheduling scheme with least data access to DDR for computing acceleration. The general-purposed hardware engine assembles different hardware modules to construct an optimized computing flow for CNN inference and accelerate calculations. Besides, an instruction set is customized to coordinate the joint-work of software and hardware, and to provide with more adaptability and optimization space for inference.

The hardware engine is composed of 4 hardware modules of Ctrl, Load, Comp and Save, as shown in Figure 1(b). The dotted lines in Figure 1(b) indicate the data dependencies between modules, where handshake FIFOs are used to prevent hazards. Based on FPGA, the accelerating engine is built with a modular design fashion to make different modules work concurrently. Multiple hardware optimization strategies, such as weight reuse and layer fusion, are applied in its detail design for a better balance between performance and resource utilization.

3.2 Instruction Set

The instruction set is used to pilot the operations of the hardware engine. Four 128-bit instructions (Load, Comp, Save and Comp_cfg) are defined, as shown in Figure 2.

The Comp_cfg instruction is used to configure the bias data and the control registers of the Comp module. The field of CFG_OP determines the selection of configuring the registers or the bias data. CFG_ADDR gives the starting address and CFG_DATA gives the data values for registers configuration. DRAM_BASE gives the starting address and CFG_CH_SIZE gives the size of the bias data.

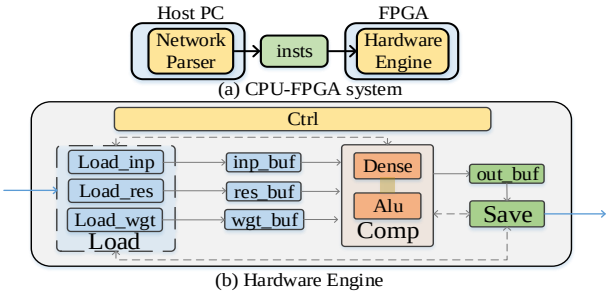

Figure 1. Architecture of IDLA

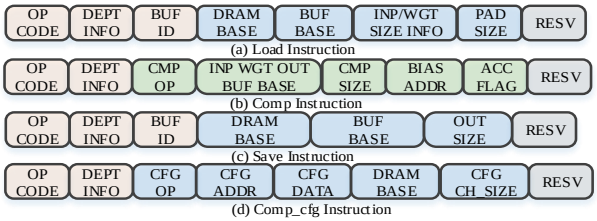

Figure 2. Instruction Set of IDLA

For all the four instructions, the field of DEPT_INFO indicates the data dependencies among modules. BUF_ID selects the buffer for load or save operations. CMP_OP selects the operation (convolution/average pooling/max pooling) for the Comp module. CMP_SIZE indicates the sizes of input/output feature map, kernel and the value of stride. ACC_FLAG is used to clear accumulation buffer.

3.3 Accelerator Design and Optimization

3.3.1 Ctrl & Load & Save Module

The Ctrl module fetches instructions from DDR, parses their OP_CODE and distributes them to corresponding modules. The Load module has 3 submodules: Load_inp, Load_res and Load_wgt. Load_inp judges its data dependencies with other modules by fetching flags from handshake FIFOs, then it loads data from DDR to inp_buf. Load_res and Load_wgt are used to load residual data and weights from DDR to on-chip buffer. The Save module writes computed results back to DDR.

3.3.2 Comp Module

Comp module contains 3 submodules: Cfg, Dense and Alu, as shown in Figure 4(a). Cfg configures the registers and the bias data. Dense executes the convolution and pooling calculations. As an example, Figure 3(a) shows a data flow of a normal convolutional computation, in which the variable loop range makes it difficult to unroll the loops and to accelerate the computation. To figure out such a problem, we propose a new data flow in our hardware design, as shown in Figure 3(b). We let the innermost two loops (co, ci) have a fixed loop boundary value of T_p. These two loops can be completed in one cycle by the way of unrolling them completely. The unrolled loops can be regarded as a matrix-vector multiplication between a weight matrix in the size of

$[T_p, T_p]$ and an input feature map in the size of T_p, which can be calculated by using a multiply-add array as shown in Figure 4(c). Thus, the data access of this convolution can get a large amount of reduction since all the weights can be reused in loop h and w.

The Alu submodule is designed for adding bias, residual function and ReLU. Batch normalization layers can be fused into CONV layers. Alu is designed by using a layer fusion strategy to further reduce data access, as shown in Figure 4(b). The operations of Alu are executed optionally according to the configuration of the instructions.

```
for co, ci in range(OC, IC):{
   sum[OC][OH][OW]=0;
   for ky, kx in range(K, K):
      for h, w in range(OH, OW){
sum[co][h][w] +=
fmap[ci*Tp+ci][hin][win]* \
weight[co][ci][ky][kx];}
   }      }
```

(a) Data Flow of Normal Convolution

```
int BOC=ceil(OC/Tp); int BIC=ceil(IC/Tp);
for coo, cii in range(BOC, BIC){
   sum[Tp][OH][OW]=0;
   for ky, kx in range(K, K):
      for h, w in range(OH, OW):
         for co in range(Tp):
            for ci in range(Tp):{
sum[co][h][w] +=
fmap[cii*Tp+ci][hin][win]* \
weight[coo*Tp+co][cii*Tp+ci][ky][kx];}}
```

(b) Data Flow of Proposed Convolution

Figure 3. Data flow of Normal/Proposed Convolution

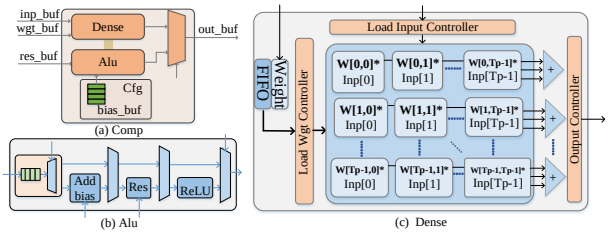

Figure 4. Hardware Architecture

3.4 Convolution Mapping Optimization

In order to minimize data access to DDR in each CONV layer's computation, optimal operation schemes are applied for the reuse of input data and weight data. The input stationary scheme loads an input feature map block into Inp_buf, and then loads weight data block to Wgt_buf. Weight stationary scheme reuses the weight data. At the system level, for each CONV layer, the network parser calculates the total data access of the two schemes and selects the smaller one.

4. Experimental Results

In the experiment, we build an IDLA instance on a Xilinx VU9P FPGA and connect it to an Intel Xeon E5-2680 CPU through a PCIE x16 interface. IDLA reaches its best performance when setting T_p=32 (T_p is the width and height of the multiply-add array). Table 1 gives the FPGA resource utilization of IDLA, running at a maximum frequency of 167MHz.

We then implement a ResNet18 model and a VGG16-SVD model respectively on this IDLA instance. Table 2 presents their computing latencies and their used instruction numbers. We compare IDLA with existing works on VGG16 in Table 3, we can see that IDLA has reached a computing performance of 277.63 GOPS on VGG16-SVD, better than the existing advanced works in [9] and [10]. The DSP utilization index Ops/DSP/cycle is close to the limit value 2, which is far above [9] and [10].

Table 1.Resource Utilization (@167MHz)

	LUTs	DSPs	18k BRAMs	URAM
Used	205667	1089	1344	54
Utilization	17.4%	15.9%	31.1%	5.63%

Table 2.Test Results of ResNet18 and VGG16-SVD

CNN model	GOPS	Latency(ms)	Inst Number
ResNet18	168.76	21.51	1636
VGG16-SVD	277.63	110.7	4621

Table 3.Comparison of IDLA against existing works

	Qiu[9]	Zhang[10]	IDLA
Platform	Zynq XC7Z045	Xilinx KU060	Xilinx VU9P
Network	VGG16-SVD	VGG16	VGG16-SVD
FRQZ(MHz)	150	200	167
DSP for Conv	780	1024	1024
Data precision	16bit	16bit	16bit
PRFM (GOPS)	136.97	266	**277.63**
Ops/DSP/cycle	1.17	1.30	**1.62**

5. Summary

This paper proposes an instruction-based adaptive CNN accelerator IDLA, including a hardware engine, a network parser and a customized instruction set. The network parser generates optimal scheduling schemes with least data access for each CNN model. The hardware engine uses a modular design fashion to make different modules work concurrently, and adopts layer fusion, weight reuse and etc. strategies to minimize data access. The instruction set is designed to coordinate the joint-work of software and hardware. We deploy and test IDLA on a Xilinx VU9P FPGA. Experimental results show that, compared with the existing advanced works [9] and [10], IDLA has achieved an overwhelming performance in both the computing (277.64 GOPS on VGG16-SVD) and the DSP efficiency (1.62 Ops/DSP/cycle on VGG16-SVD).

References

[1] Huang C, et al., International Conference on ASIC (ASICON), p.1037-1040 (2017).
[2] Ma Y, et al., International Symposium on Circuits and Systems, p.1-4 (2017).
[3] Moreau T, et al., arXiv:1807.04188v1 (2018).
[4] Simonyan K, et al., arXiv:1409.1556 (2014).
[5] Kaiming He, et al., arXiv:1512.03385 (2015)
[6] Gupta S, et al., arXiv:1502.02551 (2015).
[7] Gysel P, et al., arXiv:1604.03168 (2016)
[8] Girshick R, arXiv:1504.08083 (2015).
[9] Qiu J, et al., ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, p.26-35(2016).
[10] Zhang C, et al. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, p.2072-2085 (2018).