Customization of a Deep Learning Accelerator

Shien-Chun Luo
Industrial Technology Research Institute, Taiwan
scluo@itri.org.tw

ABSTRACT

Deep-learning algorithms require large and parallel multiplication and accumulation (MAC), which fits hardware accelerators consist of parallel processing elements (PEs) to speed up. As the PE number increases, how to distribute data in time becomes the key problem. Improving the accelerator performance needs to balance the computation power and the data communication bandwidth, which forms a roofline model of the throughput. In addition to hardware setup, the combination of neural network operators, layers, also causes the roofline curve to shift. Optimizing the performance, power, cost of the accelerators needs to link the neural network models to the physical hardware setup, which indicates custom and model-specific are essential. This presentation introduces an example of custom deep-learning accelerating system developed from open-source NVIDIA deep-learning accelerator (DLA). We supplement an environment for developing the system, from model quantization, model compilation, test generation, to driving tools. FPGA prototypes and test chips are designed, running an application of object detection, offering 70% MAC network utilization.

INTRODUCTION

Deep-learning algorithms comprise parallel multiplication and accumulation (MAC), suitable for application-specific IC (ASIC) to accelerate. These MACs are operated by relatively simple processing elements (PEs) that are usually constructed in an array structure. The performance of an accelerator is proportional to its PE number, and the data distribution bandwidth constrains the upper bond. The computation to communication power (C2C) ratio forms a roofline shape of the throughput curve [1]. Besides this C2C ratio, practical inference performance depends on the neural network (NN) structure, the operator layer, and the combination and dimension of NN layers. For example, a fully-connected operation requires higher memory bandwidth than computing power. Similarly, some NN operations like depth-wise convolution [2], element-wise arithmetic, residual addition [3], and data transpose, also prefer higher communication than computing power. In contrast, a convolution operation requires more computation than the memory bandwidth. Besides single-layer operation, multiple-layer operations along with model compression skills, like folded batch normalization into convolution [4], hetero-layer fusion, and data dimension reshape or concatenation also affect. Designing an efficient accelerator, especially having candidate inference models, needs careful profile and proper setup the hardware.

ARCHITECTURE AND DESIGN ENVIRONMENT

System Architecture

Figure 1 shows our customizable system architecture, which is developed and revised based on the open source of NVIDIA deep-learning accelerator (DLA) [5]. The accelerator core has four processors: convolutional processor and element-wise processors are essential, but the configurable part is their MAC number. The pool processor and nonlinear processors are optional. The other custom parts include the interface, the flow controller, and the host system.

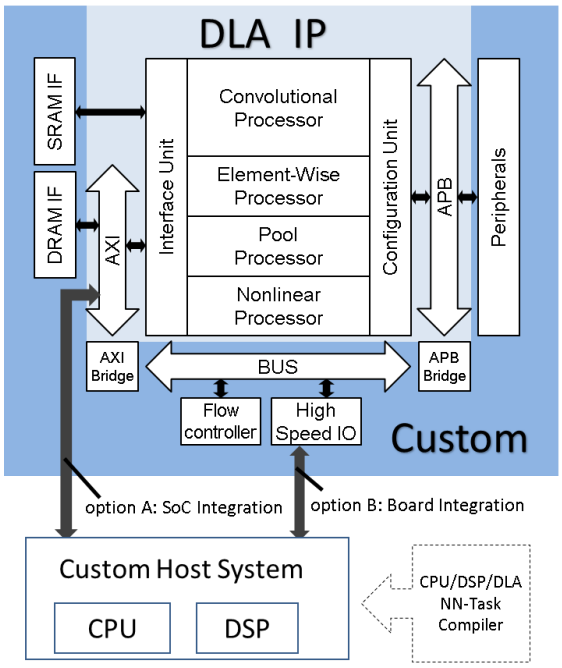

Fig. 1 Customization Architecture

Simulation and Customization

Four classical classification NN models, Alexnet [6], Inception v1 [7], Resnet50 [3], Mobilenet v1 [2], show interesting trade-off in C2C ratio. A DLA profiler calculates these simulation data based on register transfer level (RTL) simulation results. Here, the weights and activations are 8-bit, and the DLA comprises 256 MACs and 128-KB convolutional buffers. We scale core frequency to represent the computation power and scale memory bandwidth to represent communication bandwidth. The variable ranges are set as core frequency from 100MHz to 1GHz, and memory bandwidth from 1 GBps to 12 GBps. For a quick review, Table 1 shows the sensitivity, using the ratio of frame per second (FPS), at the corner cases. Alexnet and Mobilenet have higher sensitivity (> 5X) to memory bandwidth, and Inception has higher sensitivity (> 5X) to core frequency. These results indicate to build higher memory bandwidth when running Alexnet and Mobilenet than running Inception, otherwise wasting the computation power.

Table. 1 Performance sensitivity shown by the ratio of FPS

Network	Fixed Core Frequency FPS Ratio = [@12GBps] / [@1GBps]		Fixed Memory Bandwidth FPS Ratio=[@1GHz] / [@100MHz]	
	CLK=100MHz	CLK= 1GHz	BW= 1GBps	BW= 12GBps
Alexnet	1.6 X	**5.2 X**	**2.1 X**	6.6 X
Inception v1	1.1 X	**2.0 X**	**5.1 X**	9.6 X
Resnet50	1.0 X	**3.1 X**	**3.2 X**	9.9 X
Mobilenet v1	1.1 X	**5.0 X**	**2.1 X**	9.5 X

Compilation and Testbench

Compiling and profiling an NN model can be simultaneously. We construct a bare-metal compiling tool for NN model simulation and test-bench generation. The tool parses NN models in Caffe prototxt format, fuses the operating layers based on the DLA architecture, and then partitions the fused layer to fit the convolutional buffer size. The memory of activations and weights are allocated arbitrarily, where the further optimization should call for professional compilers. However, an NN network whose whole operators are supported by DLA will be executable from end to end. Testbenches including accelerator configurations, inputs, weights, golden patterns are generated after the NN model is compiled. The testbenches are called by C functions, which join the hardware RTL verification using a Verilog program interface (VPI).

Model Quantization and Bare-metal API

Accelerators using integer inference need weight quantization and retraining, which must follow the precision propagation inside, for example, to align the internal bit-width of the accumulators and to set truncation location of the output activations. Our accelerator use both 8-bit inputs and weights, but 16-bit parameters in the element-wise and non-linear processors. Executing by the bare-metal runtime, all the weights, parameters, and driving configurations are stored in a preserved and private location of system memory, called by an application program interface (API) in the inference program.

PROTOTYPE AND TEST CHIP

FPGA Prototype

Two FPGA prototypes are implemented: standalone (Fig. 2(a)) and USB-stick (Fig. 2(b)). Both prototypes equip 64 MACs and 128-KB convolutional buffers. The standalone version uses two ARM Cortex A53 cores on Xilinx ZCU102 as a flow controller and a host processor, respectively. The USB-stick version uses an USB bridge to connect to a Linux computer, where we built specific USB driving libraries. Tiny YOLO v1 [8], an object detection algorithm, was selected to demonstration. The FPGA runs at 150MHz and offers 19.2 GOP/s peak performance. The MAC utilization of Tiny YOLO v1 inference is close to 70%, reaching 4 FPS at runtime. In addition to the 64-MAC configuration, a 256-MAC setup is also verified and can boost up the performance to 8 FPS, which is not proportional to MAC number constrained by DRAM speed.

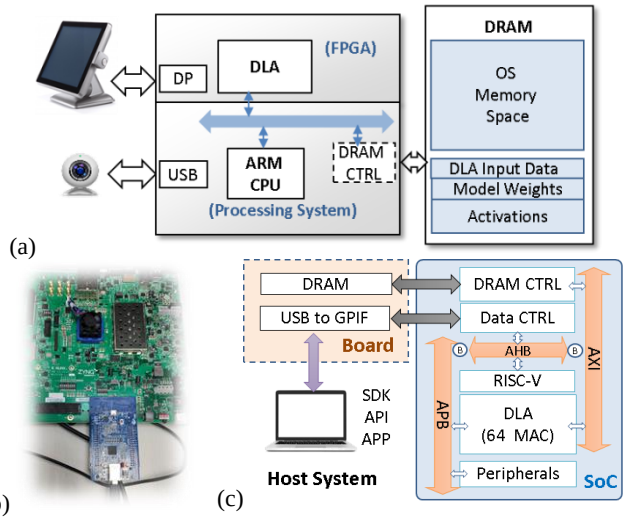

Fig. 2 (a) Standalone FPGA prototype (b) USB-bridge and FPGA (c) ASIC SoC design and system overview.

ASIC Test Chip

ASIC test chip is implemented using the DLA of 64 MACs and 128-KB convolutional buffers. A 5-stage RISC-V processor (RV-32IM) is implemented as the main flow controller. This chip uses TSMC 65-nm process, operates at 400 MHz and offers 50 GOPs peak performance. The test chip links to a host computer by the same USB bridge referred to in the USB-version FPGA prototype. The silicon area including pads is 3200 μm by 3200 μm. Registers and clock network has been optimized for reducing power consumption. The average power consumption of convolution operation (DLA only) is 60mW, achieving 0.8 TOPs/W peak energy efficiency.

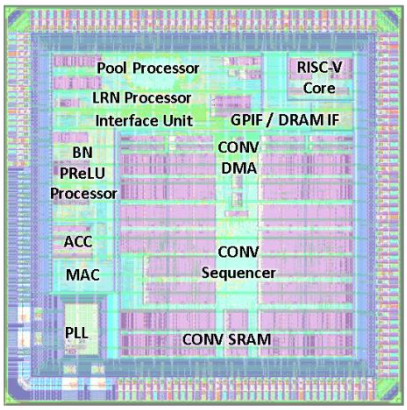

Fig. 3 Test chip layout

ACKNOWLEDGEMENT

The presenter sincerely appreciates all the team members, especially KC, Weber, Jane, Bryan, CY, Jeff, and Jerry who contribute very much to the success of this system.

REFERENCES

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[2] Andrew G. Howard, Menglong Zhu, Bo Chen,Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR," *arXiv: 1704.04861[cs.CV]*, Apr 2017.

[3] Kaiming He, Xiangyu Zhang, et al. "Deep residual learning for image recognition." *arXiv:1512.03385[cs.CV]*, Dec 2015.

[4] D. Jung, W. Jung, , B. Kim, S. Lee, W. Rhee, and J. H. Ahn, "Restructuring batch normalization to accelerate CNN training," *arXiv:1807.01702[cs.CV]*, July 2018.

[5] NVDLA open source project: http://nvdla.org/

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, "ImageNet classification with deep convolutional neural networks" *Communications of the ACM*. 60 (6): 84–90, 2012.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan,V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, June 2015.

[8] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.