Chapter 1

Hardware Accelerator Design for Machine Learning

Li Du and Yuan Du

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72845

Abstract

Machine learning is widely used in many modern artificial intelligence applications. Various hardware platforms are implemented to support such applications. Among them, graphics processing unit (GPU) is the most widely used one due to its fast computation speed and compatibility with various algorithms. Field programmable gate arrays (FPGA) show better energy efficiency compared with GPU when computing machine learning algorithm at the cost of low speed. Finally, various application specific integrated circuit (ASIC) architecture is proposed to achieve the best energy efficiency at the cost of less reconfigurability which makes it suitable for special kinds of machine learning algorithms such as a deep convolutional neural network. Finally, analog computing shows a promising methodology to compute large-sized machine learning algorithm due to its low design cost and fast computing speed; however, due to the requirement of the analog-to-digital converter (ADC) in the analog computing, this kind of technique is only applicable to low computation resolution, making it unsuitable for most artificial intelligence (AI) applications.

Keywords: machine learning, hardware accelerator, model compression, analog computing, GPU, FPGA, ASIC

1. Introduction

Machine learning (ML) is currently widely used in many modern artificial intelligence (AI) applications [1]. The breakthrough of the computation ability has enabled the system to compute complicated different ML algorithm in a relatively short time, providing real-time human-machine interaction such as face detection for video surveillance, advanced driver-assistance systems (ADAS), and image recognition early cancer detection [2, 3]. Among all those applications, a high detection accuracy requires complicated ML computation, which comes at the cost of high computational complexity. This results in a high requirement on the hardware platform. Currently, most applications are implemented on general-purpose compute engines, especially graphics processing units (GPUs). However, work recently reported

from both industry and academy shows a trend on the design of application specific integrated circuit (ASIC) for ML, especially in the field of deep neural network (DNN). This chapter gives an overview of the hardware accelerator design, the various types of the ML acceleration, and the technique used in improving the hardware computation efficiency of ML computation.

2. Recent development on deep learning hardware accelerator

2.1. GPU/FPGA-based accelerator in datacenter

Over the past decades, graphics processing units (GPUs) have become popular and standard in training deep-learning algorithms or convolutional neural networks for face, object detection/recognition, data mining, and other artificial intelligence (AI) applications. GPUs offer a wide range of hardware selections, a high-performance throughput/computing power, and a stable but ever-expanding ecosystem. The GPU architecture is usually implemented with several mini graphics processors. Each graphics processor has its own computation unit and local cache which fits for the matrix multiplication. A shared high-speed bus is included in multiple mini processors to enable fast data exchange among mini processors. In addition, it also acts as a bridge to connect the main CPU and multiple mini graphics processors.

Taking NVIDIA's DGX-1 as an example [4], DGX-1 has eight Tesla P100-SXM2 GPUs conforming to Pascal architecture. Each GPU has 56 multiprocessors with 64 CUDA cores per multiprocessor. This makes each GPU equipped with 3584 CUDA cores. The GPU and memory clock frequencies are 1.3 GHz and 700 MHz, respectively. The GPU has 4096-bit memory bus width, 16 GB global memory, and 4 MB L2 cache. **Figure 1** shows the system-level topology of DGX-1. The network of NVLink interconnect is wired so that any two GPUs can hop away from less than one another GPU. The GPU cluster is connected to a switch (PLX) with a PCIe × 16

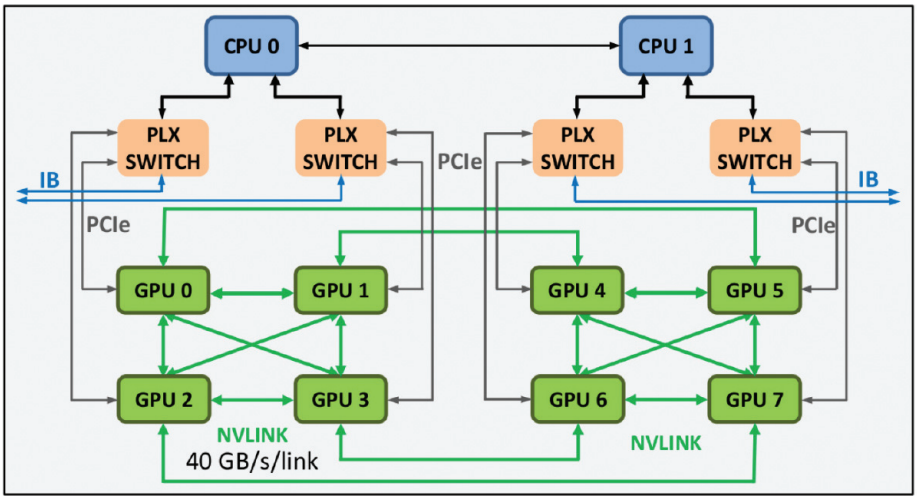

Figure 1. Diagram of NVIDIA DGX-1 system-level topology.

interconnect. The maximum bandwidth of NVLink interconnect with Tesla P100 is reported at 160 GB/s. In a clustering or multicore parallel computation scenario, the communication interconnect performance becomes the bottleneck to achieving high throughput, low latency, and high energy efficiency. **Figure 2(a)** and **(b)** shows that DGX-1 GPU outperforms comparable Intel CPU (KNL) in power efficiency and computing throughput for two different batch sizes when running CLfarNet.

The GPU offers significant computation speed due to a lot of parallel processing cores. However, a relatively large power consumption is also requested for the computation and data movement. In addition, a high-speed interconnect interface is required to support the fast data exchange. Thus, compared with other techniques, GPU offers power computation ability at the expense of high design cost (unit price) and power consumption.

As the industry matures, field programmable gate arrays (FPGAs) are now starting to emerge as credible competition to GPUs for implementing CNN-based deep learning algorithms. Microsoft

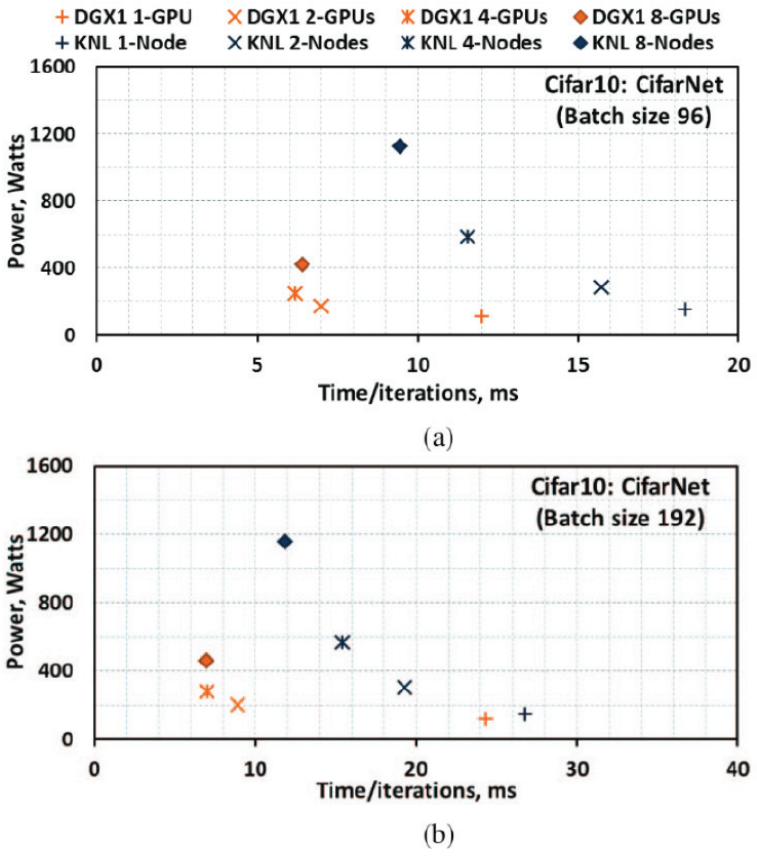

Figure 2. Power and performance of CifarNet/Cifar 10 with batch sizes (a) 96 and (b) 192.

Research's Catapult Project garnered quite a bit of attention in the industry when it contended that using FPGAs could be as much as 10 times more power efficient compared to GPUs [5]. Although the performance of single FPGA was much lower than comparable-price GPUs, the fact that power consumption was much lower could have significant implications for many applications where high performance may not be the top priority. **Figure 3(a)** shows a logical view of FPGAs in cloud-scale application and **Figure 3(b)** shows how the FPGA-based accelerator fits into a host server.

As **Figure 3(b)** shows, the FPGA-based machine learning accelerator typically involves hardware blocks such as DRAM, CPUs, network interface controller (NIC), and FPGAs. The DRAMs act as a large buffer to store the temporary data while the CPU is in charge of managing the computation, including sending instructions to FPGAs. The FPGA is programmed to fit the ML algorithm. Since the ML algorithm is optimized at a hardware level through FPGA programming, a high data access efficiency is obtained compared with regular GPU computation which does not have any hardware optimization on the corresponding ML algorithms.

Although the FPGA reduces the power consumption in computing through optimizing the ML algorithms on the hardware design, the overall efficiency is still much lower compared with the ASIC for single kind of algorithms. Compared with the ASIC, the programmability introduced by the FPGA also brings complicated logic which increases the hardware design cost. In addition, the speed of the FPGA is usually limited to 300 MHz, which is 4–5× times lower than a typical ASIC [6].

2.2. ASIC-based CNN accelerator at edge

2.2.1. Introduction

In the HPC or datacenter, hardware accelerator solutions are dominated by GPU and FPGA solution. State-of-the-art machine-learning computation mostly relies on the cloud servers.

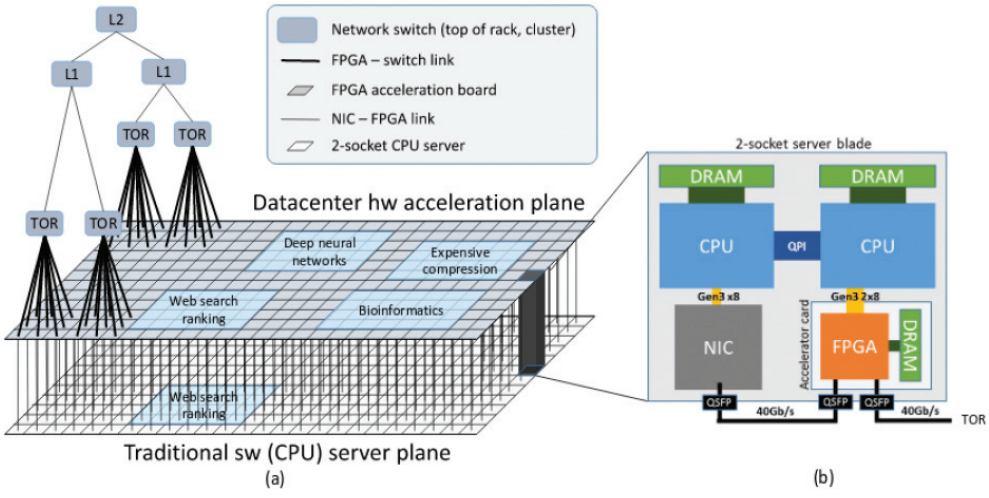

Figure 3. (a) De-couples programmable hardware plane, (b) server plus FPGA schematic.

However, high-power consumption makes this approach limited in many real application scenarios. Since cloud-based AI applications on portable devices require network connection capability, the quality of network connection affects user experience. Furthermore, the network and communication latency is not acceptable for real-time AI applications. In addition, most of IoT AI applications have a strict power and cost constrain, which could support neither high-power GPU nor transmitting a large amount of data to cloud servers.

To address the abovementioned issues, several edge-based AI processing schemes were introduced in [7–9]. The edge-based AI processing scheme targets utilizing the localized data at the edge side and avoids network communication overhead. Currently, most localized AI processors focus on processing convolutional neural network (CNN) which is widely used for computation vision algorithms and requests a lot of computing resources.

2.2.2. CNN accelerator layer function definition

The state-of-art convolutional neural networks commonly include three different computational layers: convolution layer, pooling layer, and fully connected layer. Convolution layer is the most computation intensive part of the neural network, with pooling layer inserted between two convolution layers with the function of reducing intermediate data size and remapping feature maps. Fully connected layer is usually the last layer of the CNN to predict labels of input data, which is memory bandwidth limited, rather than computation resource limited.

The primary role of a convolution layer is to apply convolution function to map the input (previous) layer's images to the next layer. Data from each input layer are composed of multiple channels as a three-dimensional tensor. One set of regional filter windows is defined as one filter or weight. The results run through inner product computation by the filter weight and input data. Output feature is defined by using the filter or weight to scan and accumulate different input channels. After interproduct computation, a separated bias vector (the same dimension as output feature number) will be added in each final result. The analytical representation of convolution layer is shown in Eq. (1) and **Figure 4**.

$$\boldsymbol{O}[o][m][x][y] = \boldsymbol{B}[o] + \sum_{k=1}^{M}\sum_{i=1}^{K}\sum_{j=1}^{K}\boldsymbol{I}[o][k][\alpha x + i][\alpha y + j] \times \boldsymbol{W}[m][k][i][j]$$

$$1 \le o \le N, 1 \le m \le M, 1 \le x, y \le S_o \tag{1}$$

O, B, I, and **W** are the output features, biases, input features, and filters, respectively.

In addition to the convolution layer, pooling layer is to compress important information through a group of local image pixel data in each input channel. There are two types of pooling operations: max pooling and average pooling. For max pooling operation, the output of pooling layer collects the maximum of pixel data in the local group window, while for average pooling operation, the output of pooling layers calculates the mean of pixel data in the local group window. The representations of these two pooling operations are defined as Eqs. (2) and (3). **Figure 5** is an example of the max pooling function.

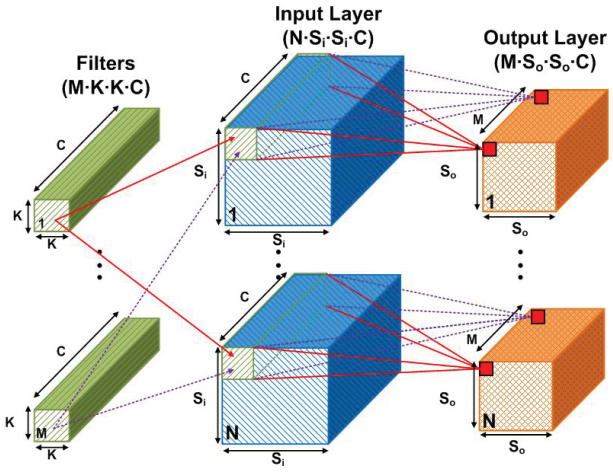

Figure 4. Concept of computation of CONV layer.

$$O_{avg}[r][c] = avg\begin{bmatrix} I[r][c] & \cdots & I[r][c+K-1] \\ \vdots & \ddots & \vdots \\ I[r+K-1][c] & \cdots & I[r+k-1][c+K-1] \end{bmatrix} \tag{2}$$

$$O_{max}[r][c] = max\begin{bmatrix} I[r][c] & \cdots & I[r][c+K-1] \\ \vdots & \ddots & \vdots \\ I[r+K-1][c] & \cdots & I[r+k-1][c+K-1] \end{bmatrix} \tag{3}$$

Here I[r][c] represents the input channel's data at the position (r,c) and the kernel size of the pooling window is K.

2.2.3. *CNN accelerator architecture overview*

Today's CNN accelerator architecture can mainly separate into two categories. The central computation architecture and the sparse computation architecture. **Figure 6** is a typical central

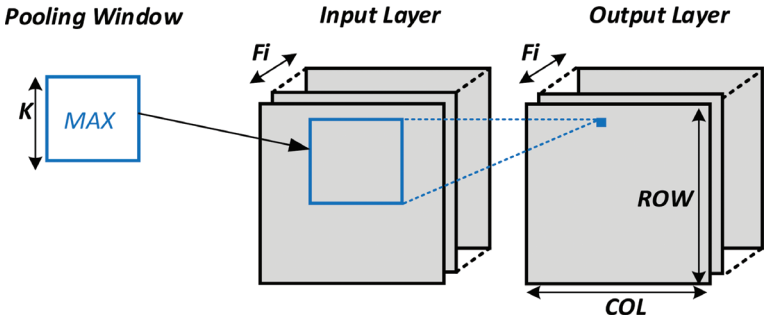

Figure 5. Example of computation of a max pooling layer.

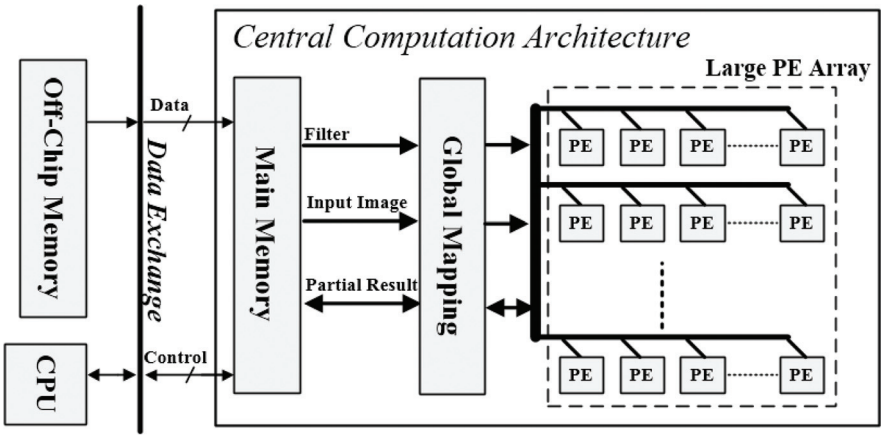

Figure 6. Central computation architecture of the CNN accelerator.

computation architecture that reports in 2015 [10]. The central computation architecture has one large PE array. Multiple filters will be sent out into the PE array to enable parallel computation. The output result of each filter will be gathered at the PE array's output to feedback to the memory for next layer computation. This large PE array in the central computation architecture provides a benefit to computing large kernel-sized CNN; however, it needs to reconstruct the array when computing the small kernel-sized CNN.

On the other hand, a sparse computation architecture is made of many parallel small convolution units that fit for small-sized kernel [11]. **Figure 7** is one of such implementations. The computing unit (CU) Engine Array is made of 16 3 × 3 kernel-sized convolution units. It provides a benefit to compute small kernel-sized convolution operations and simplify the data flow. However, the computing unit is only supported for 3 × 3 convolution. So when computing a kernel size that is larger than 3 × 3, a kernel decomposition technique is proposed in the following section.

2.2.4. Kernel decomposition technique

The filter's kernel size in a typical CNN network can range from a very small size (1 × 1) to a very large size (11 × 11). A hardware engine needs design to support various sized convolutional operation. However, for sparse architecture, the computation units are not separated into many small blocks. Each block consists of a small-sized processing engine array and can only support small-sized convolution, making each block hard to process large convolution. To minimize the hardware resource usage, a filter decomposition algorithm is proposed to compute any large kernel-sized (>3 × 3) convolution through using only 3 × 3-sized CU [11]. The algorithm is separated into three steps: (1) It first examines the kernel size of the filter. If the original filter's kernel size is not an exact multiple of three, zero padding weights will be added in the original filter's kernel boundary to extend the original filter's kernel size to be a multiple of three. The added weights are all zero to keep the extended filter convolution result to be same as the original one. (2) The extended filters will be decomposed into several 3 × 3-sized filters. Each filter will be

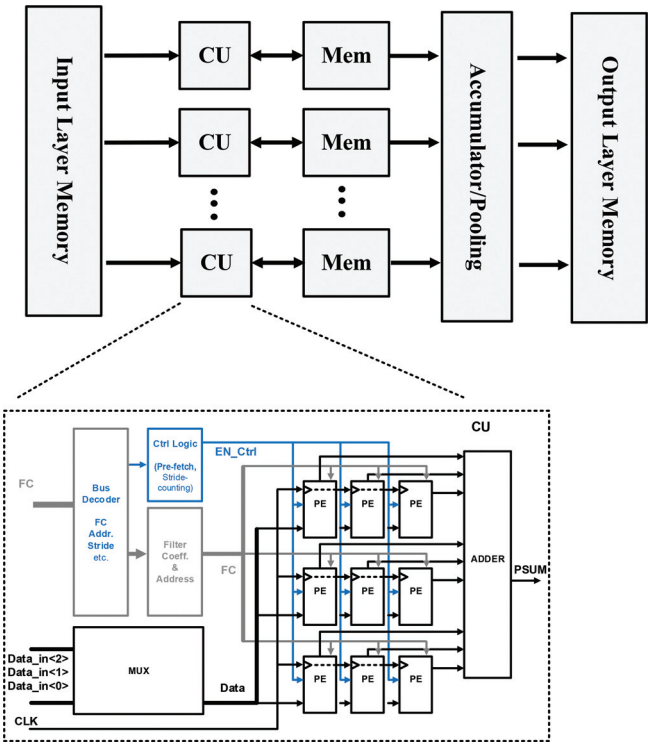

Figure 7. Sparse computation architecture of the CNN accelerator in [11].

assigned a shift address based on its top left weight's relative position in the original filter and each decomposed filter will be computed individually. (3) The output result of each decomposed filter will be summed together based on its shift address to generate the final output. The mathematical derivation of this decomposition technique is also explained in [11].

Figure 8 is an example of decomposing a 5 × 5 filter into four 3 × 3 filters using this technique. One row and column zero padding are added in the original filter. The decomposed filters F0, F1, F2, F3's shift address are (0,0), (0,3), (3,0), (3,3). **Figure 9** shows the detailed procedure.

2.3. Model compression

In addition to the hardware architecture level development, model compression is also reported as a way to improve the hardware computation efficiency of the machine learning. Ref [12] reported a methodology to prune the neural network and achieve up to 35× to 49× model parameters reduction. The procedure is shown in **Figure 10**. The original network will be pruned and retrained several times to achieve parameters reduction. After that, quantization is implemented with clustered weights to achieve additional parameter size reduction. Finally, Huffman encoding is added into the final weights to achieve further model size reduction.

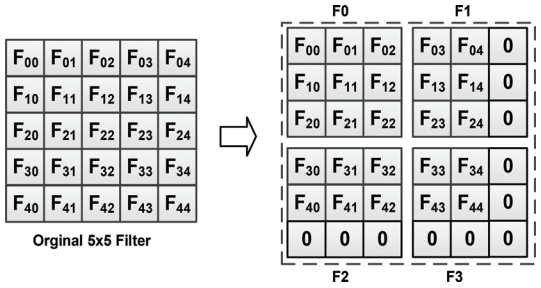

Figure 8. A 5 × 5 Filter decomposed into four 3 × 3 sub-filter.

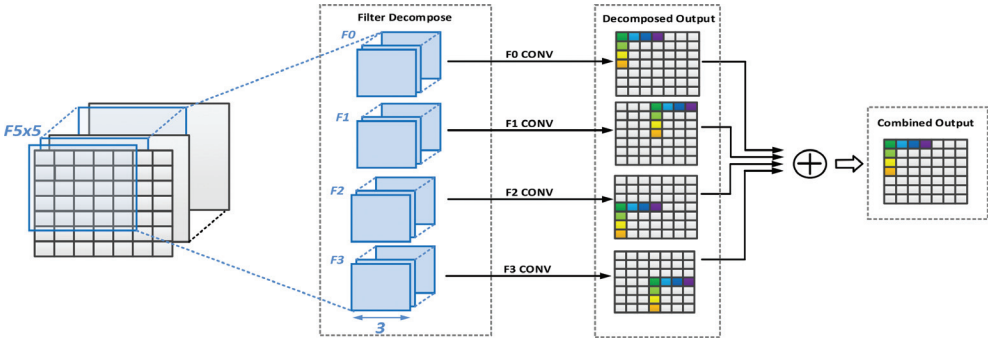

Figure 9. Filter decomposition technique to compute a 5 × 5 filter on the 7 × 7 image. The 5 × 5 filter is decomposed into four separated 3 × 3 filters F0, F1, F2, F3, and generating four sub-images. The sub-images are summed together to generate the final output. Same color's pixels in each sub-image will be added together to generate the corresponding pixels in the output image.

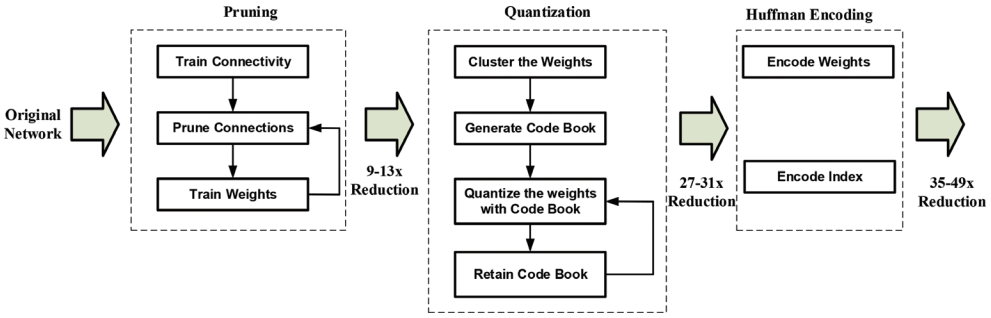

Figure 10. Neural network compression reported in Ref [12].

Due to the rapid increment of the deep learning model size, model compression becomes more and more important for machine-learning hardware acceleration, especially for the edge-side user case. In addition, the fixed-point data format is also used in many deep learning applications to reduce the computation cost [13].

2.4. Analog computing

In addition to the traditional digital accelerator design, analog computing is also becoming one of the trends to improve the processor computation ability in solving machine learning problems. Here, we use the charge-trapping transistors (CTTs) technique as an example to introduce analog computing [14]. The complementary metal oxide semiconductor (CMOS)-compatible feature of the CTTs makes them very promising devices to implement large-sized computation using analog methodology.

As the scaling of transistors is reaching its manufacturing limit, the computation throughput using current architectures will also inevitably saturate. Recent research reports the development of analog computing engines. Compared to traditional digital computation, analog computing shows tremendous advantages regarding the power, design cost, and computation speed. Among many analog computing systems, memristor-based ones have been widely reported [14]. Recently, more promising charge-trapping transistors (CTTs)

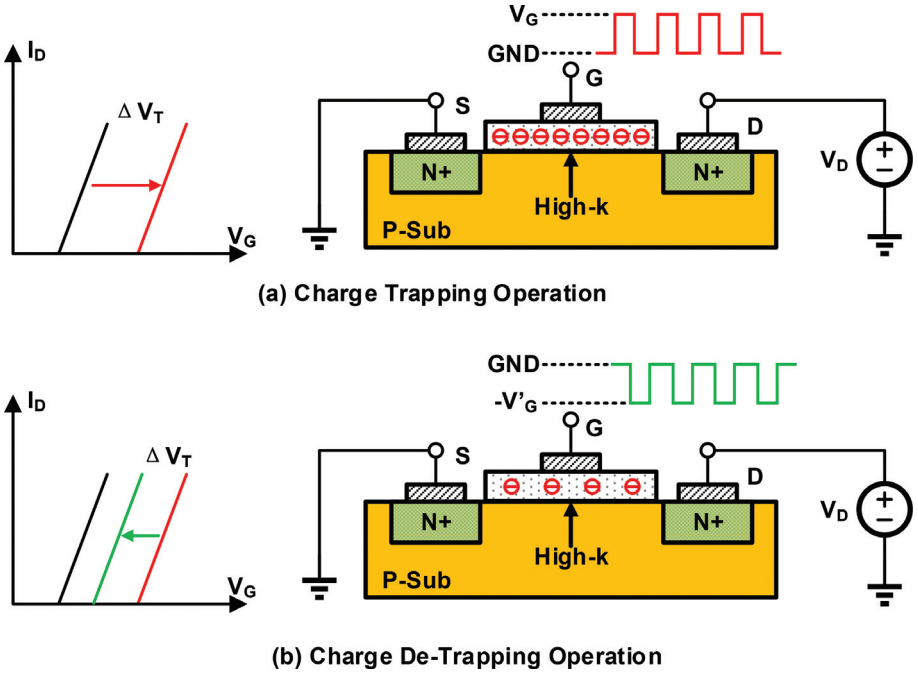

Figure 11. A schematic showing the basic operation of CTT device (equally applicable to FinFET-based CTTs): (1) charge trapping operation, (2) charge de-trapping operation.

were reported to be used as digital memory devices with reliable trapping and de-trapping behavior. Different from other charge-trapping devices such as floating-gate transistors, transistors with an organic gate dielectric, and carbon nanotube transistors, CTTs are manufacturing ready and fully CMOS compatible in terms of process and operating. IT shows that more than 90% of the trapped charge can be retained after 10 years even when the device is baked at 85°C [15].

A schematic of the basic operation of a CTT device is depicted in **Figure 11**. The device threshold voltage, VT, is modulated by the charge trapped in the gate dielectric of the transistor. VT increases when positive pulses are applied to the gate to trap electrons in the high-k layer and decreases when negative pulses are applied to the gate to de-trap electrons from the high-k layer. CTT devices can be programmed by applying logic-compatible voltages.

A memristive computing engine based on the charge-trapping transistor (CTT). The proposed memristive computing engine consists of 784 by 784 CTT analog multipliers and achieves 100× power and area reduction compared to the conventional digital approach. Through implementing a novel sequential analog fabric (SAF), the mixed-signal interfaces are simplified and it only requires an 8-bit analog-to-digital converter (ADC) in the system. The top-level system architecture is shown in **Figure 12**. A 784 by 784 CTT computing engine is implemented using TSMC 28 nm CMOS technology and occupies 0.68mm^2 as shown in **Figure 13**. It achieves 69.9 TOPS with 500 MHz clock frequency and consumes 14.8 mW.

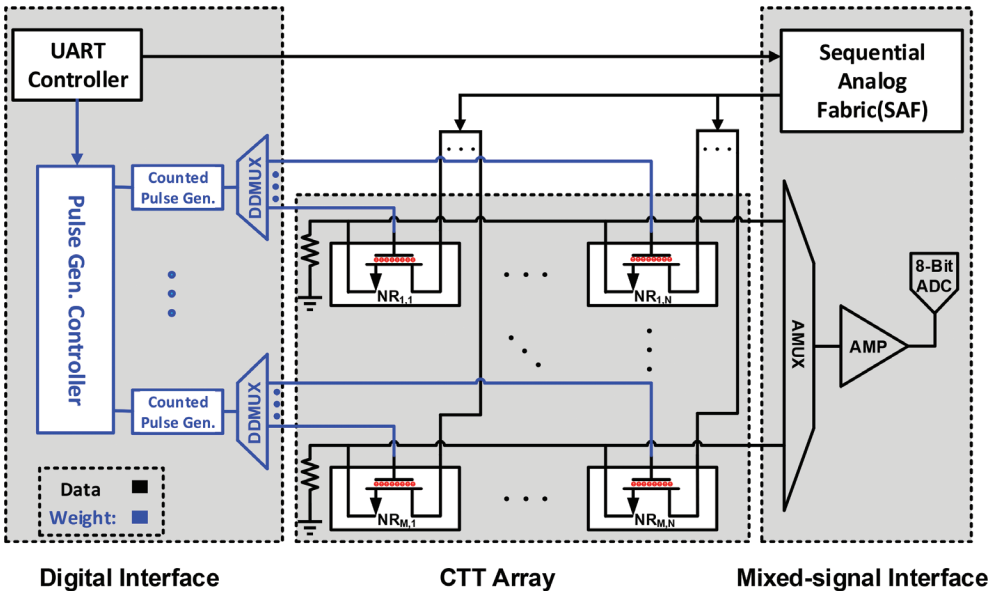

Figure 12. Top-level system architecture of the proposed memristive computing engine, including CTT array, mixed-signal interfaces including tunable low-dropout regulator (LDO), analog-to-digital converter (ADC), and novel sequential analog fabrics (SAF).

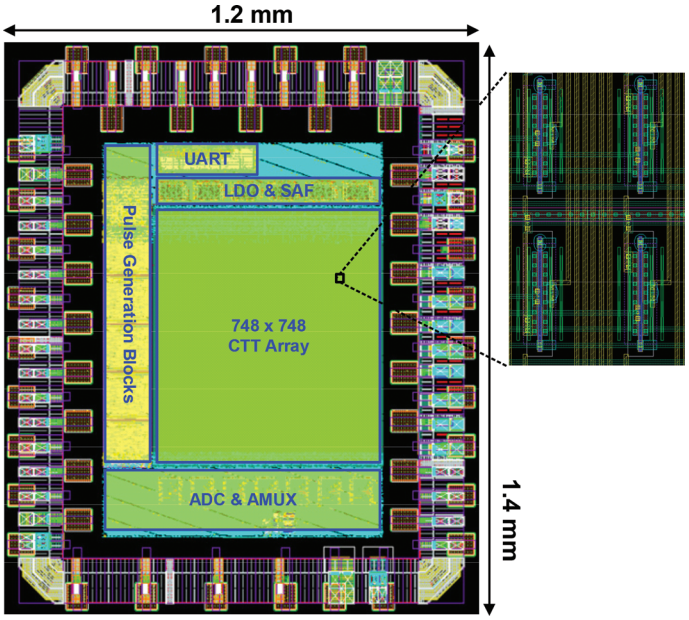

Figure 13. Layout view in TSMC 28 nm CMOS technology.

Compared with the traditional digital processor, analog-based computing processor achieves much less power as well as large area reduction in the design. **Table 1** is a comparison of the computation ability between the analog processor and digital processor. As it shows, analog processor achieved more than 100 times computing speed with 1/10 times area consumption compared to digital processor.

Even the analog computing shows advantages in the computation speed and design cost, a low computing resolution limits its application in most ML algorithms. Due to the design challenges of the ADC in the analog processor, the processor can only handle computation resolution that is less or equal to around 10 bits, making it not suitable for most AI applications.

Merits	Digital [16]	This work
Process	Standard 28 nm FD-SOI CMOS	Standard 28 nm CMOS
Core Area (mm^2)	5.8	0.68
Power (mW)	41	14.8
Clock Speed	200–1175 MHz	500 MHz
Peak MACs #	0.64 K	69.9 K
SRAM Size	128 KB	0
Non-Volatile	No	Yes

Table 1. Comparison table between analog computing and digital computing in Ref [14].

3. Conclusion

In this chapter, various computation hardware platforms for machine learning algorithms are discussed. Among them, GPU is the most widely used one due to its fast computation speed and compatibility with various algorithms. FPGA shows better energy efficiency compared with GPU when computing machine learning algorithm at the cost of low speed. Finally, different ASIC architectures are proposed to support certain kinds of the machine learning algorithms such as a deep convolutional neural network with model compression technique to improve hardware performance. Compared with the GPU and FPGA, ASIC shows the best energy efficiency and computation speed, however, at the cost of reconfigurability to various ML algorithms. Depending on the specific applications, the designers should select the most suitable computation hardware platform.

Author details

Li Du* and Yuan Du

*Address all correspondence to: dl1989113@ucla.edu

Hardware Architecture Research Engineer, Kneron Inc., Research Scientist, UCLA, Los Angeles, USA

References

[1] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. May 2015;**521**:436-444

[2] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolution neural networks. Proceeding of Advances in Neural Information Processing Systems. 2012;**25**:1097-1105

[3] Silver D et al. Mastering the game of go with deep neural networks and tree search. Nature. Jan. 2016;**529**(7587):484-489

[4] Gawande NA, Landwehr JB, Daily JA, Tallent NR, Vishnu A, Kerbyson DJ. Scaling deep learning workloads: NVIDIA DGX-1/Pascal and intel knights landing. In: IEEE International Parallel And Distributed Processing Symposium Workshops (IPDPSW); Lake Buena Vista. 2017 pp. 399-408

[5] Putnam A. The configurable cloud – accelerating hyperscale datacenter services with FPGA. In: IEEE 33rd International Conference on Data Engineering (ICDE); San Diego. 2017. p. 1587

[6] Chang AXM, Culurciello E. Hardware accelerators for recurrent neural networks on FPGA. In: IEEE International Symposium on Circuits and Systems (ISCAS); MD, Baltimore. 2017. pp. 1-4

[7] Sim J, Park J-S, Kim M, Bae D, Choi Y, Kim L-S. A 1.42TOPS/W deep convolution neural network recognition processor for intelligent IoE systems. In: Proceeding of IEEE International Solid-State Circuits Conference (ISSCC). Jan/Feb. 2016. pp. 264-265

[8] Bong K, Choi S, Kim C, Kang S, Kim Y, Yoo HJ. 14.6 A 0.62mW ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on haar-like face detector. IEEE International Solid-State Circuits Conference (ISSCC); San Francisco. 2017. pp. 248–249

[9] Desoli G et al. 4.1 A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In: IEEE International Solid-State Circuits Conference (ISSCC); San Francisco. 2017. pp. 238–239

[10] Chen YH, Krishna T, Emer J, Sze V. 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolution neural networks. In: IEEE International Solid-State Circuits Conference (ISSCC); San Francisco. 2016. pp. 262–263

[11] Du L et al. A reconfigurable streaming deep convolutional neural network accelerator for internet of things. IEEE Transactions on Circuits and Systems I: Regular Papers. 2017; **99**:1-11

[12] Han S, Mao H, Dally W. Deep compression: Compressing DNNs with pruning, trained quantization and huffman coding. 2015. arxiv:1510.00149v3

[13] Du Y et al. A streaming accelerator for deep convolutional neural networks with image and feature decomposition for resource-limited system applications. Sep. 2017. arXiv:1709.05116 [cs.AR]

[14] Du Y et al. A memristive neural network computing engine using CMOS-compatible Charge-Trap-Transistor (CTT). Sep 2017. arXiv:1709.06614 [cs.ET]

[15] Shin S, Kim K, Kang SM. Memristive computing- multiplication and correlation. In: IEEE International Symposium on Circuits and Systems; Seoul. 2012. pp. 1608-1611

[16] Desoli G et al. 14.1 A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In: IEEE International Solid-State Circuits Conference (ISSCC); San Francisco. 2017. pp. 238-239