ADVANCED INTELLIGENT SYSTEMS Open Access
www.advintellsyst.com

Hardware for Deep Learning Acceleration

*Choongseok Song, ChangMin Ye, Yonguk Sim, and Doo Seok Jeong**

Deep learning (DL) has proven to be one of the most pivotal components of machine learning given its notable performance in a variety of application domains. Neural networks (NNs) for DL are tailored to specific application domains by varying in their topology and activation nodes. Nevertheless, the major operation type (with the largest computational complexity) is commonly multiply-accumulate operation irrespective of their topology. Recent trends in DL highlight the evolution of NNs such that they become deeper and larger, and thus their prohibitive computational complexity. To cope with the consequent prohibitive latency for computation, 1) general-purpose hardware, e.g., central processing units and graphics processing units, has been redesigned, and 2) various DL accelerators have been newly introduced, e.g., neural processing units, and computing-in-memory units for deep NN-based DL, and neuromorphic processors for spiking NN-based DL. In this review, these accelerators and their pros and cons are overviewed with particular focus on their performance and memory bandwidth.

1. Introduction

Machine learning (ML)—coined by Arthur Samuel in 1959[1]—has had a significant impact on human life by being applied to various applications, including natural language processing,[2–5] speech recognition,[6–8] computer vision,[9–12] and machine translation.[13,14] Recently, ML has also been applied to genome analysis,[15] autonomous driving,[16] finance,[17] and game industry.[18,19] Particularly, deep learning (DL)—a subset of ML—is the key substrate of this remarkable progress in ML. DL uses deep neural networks (DNNs) as learning models, which are significantly versatile at various tasks with their excellent learning capability.

C. Song, C. M. Ye, D. S. Jeong
Division of Materials Science and Engineering
Hanyang University
222 Wangsimni-ro, Seongdong-gu, Seoul 04763, Republic of Korea
E-mail: dooseokj@hanyang.ac.kr

Y. Sim, D. S. Jeong
Department of Semiconductor Engineering
Hanyang University
222 Wangsimni-ro, Seongdong-gu, Seoul 04763, Republic of Korea

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202300762.

DOI: 10.1002/aisy.202300762

DNNs are graph models that consist of nodes (neurons) and edges (synapses), which were initially inspired by the biological behavior of the brain.[20] The node is a local processor that encodes its input (weighted sum of the outputs from its fan-in nodes) as an output value with various precision ranging from 32b floating-point (FP32) to 1b ('0' or '1') depending on the neuron model employed. Diverse neuron models of different computational complexities are available, e.g., sigmoid, hyperbolic tangent, rectified linear unit (ReLU),[21] leaky ReLU,[22] Gaussian error linear unit,[23] swish,[24] and spiking unit.[25] The last model should be distinguished from the others given that spiking units are time-dependent models unlike the others, rendering DNNs with spiking units time-dependent. Additionally, spiking units output binary numbers ('0' and '1' indicating nonspiking and spiking, respectively) unlike the others. Given these unique properties, spiking unit-based DNNs are particularly referred to as deep spiking NNs (SNNs). The output from a given neuron serves as inputs to its fan-out neurons with particular weights that are trainable parameters. Subsequently, the fan-out neurons encode the sum of weighted inputs as outputs. This neural processing continues through the layers in a DNN until the output neurons respond.

A common trend is that DNNs evolve such that they become deeper and larger, so that the numbers of weight (parameters) and computations required become significantly larger. For instance, the state-of-the-art language models incorporate a tremendous number of parameters insomuch as GPT-4 includes 1.7 trillion parameters[26] and Llma-2 70 billion parameters,[27] which causes severe space complexity. DNNs are particularly tailored to their application domains, e.g., convolutional NNs (CNNs)[28–30] for image feature extraction in computer vision and transformer-based networks to capture the degree of significance (attention) between tokens in natural language processing. Irrespective of DNN topology, the major workload arises from the multiplication of a feature and kernel weight and accumulation of the products, i.e., multiply-accumulate (MAC) operations (OPs). MAC OPs apply to a vast amount of data brought from the memory through data buses, so that the overall operation latency is dictated by the arithmetic operations (MAC OPs) and/or memory bandwidth.

The megatrend in DNN evolution is barely supported by central processing units (CPUs) because of their limited capability of parallel computing (due to the limited number of arithmetic logic units (ALUs) in a CPU) and limited memory bandwidth. Given that the major workload in DNN operation results from simple

MAC OPs (based on a single instruction) applied to massive data, graphics processing units (GPUs) for single-instruction multiple-data and single-instruction multiple-threads (SIMD/SIMT) with high-bandwidth memory are the very suitable type of general-purpose hardware for DL acceleration.[31–41] SIMD/SIMT GPUs can significantly accelerate DNN operations (i.e., MAC OPs) irrespective of DNN topology, and thus serving as the mainstream hardware for DL acceleration. However, their high power consumption, insomuch as they consume a few hundreds watts,[42] is a challenge to their applications to AI at the edge (also known as edge AI and on-device AI).

As alternatives, various types of application-specific integrated circuit (ASIC)-based accelerators have been introduced, including neural processing units (NPUs),[43–46] computing-in-memory (CIM) units,[47,48] and neuromorphic processors.[49–52] Note that neuromorphic processors (also known as event processors) accelerate SNN while the others DNN. These alternatives to GPUs aim to boost the operational efficiency at the cost of loss in versatility and flexibility to some degree. That is, a given alternative leverages its operational efficiency for a particular class of DNN. Generally, DNNs are classified as compute- and memory-bound models with regard to the overall operation-dictating factor: 1) arithmetic operation rate and 2) memory bandwidth, respectively. This classification will be elaborated in Section 3. Note that the aforementioned accelerators for DNN-based DL are responsible for MAC OPs only, so that they need host CPUs to complete the whole computations in DNNs.

Neuromorphic processors (event processors) are standalone devices without host CPU and main memory unlike the aforementioned accelerators. In this review, we refer to neuromorphic processors as event-based inference accelerators without learning engine unless otherwise stated. They can be standalone given several crucial features of SNNs such that 1) a feature map (event map) is remarkably sparse, i.e., the feature map at a given timestep includes only few '1's and mostly '0's and 2) the data required for inference are local to each spiking neuron. These allow the spiking neurons to be distributed over multiple cores in a neuromorphic processor and to send the events from them to their fan-out neurons in an ad hoc manner, significantly boosting power efficiency.

Despite proposals of various DL acceleration platforms to date, there exist no perfect acceleration platforms with high performance in all key performance metrics, e.g., operational throughput, power efficiency, versatility, flexibility, and so forth. Each of these platforms has relative pros and cons in terms of these performance metrics, so that appropriate platforms need to be chosen to accelerate key workloads for a given model. To begin, one should understand such key workloads for various models. In this review, we aims to provide comprehensive reviews on workloads for DNNs and SNNs of different topologies and various hardware platforms to accelerate their major operations. The primary contributions of this review are as follows: 1) We review generic properties of DNNs and SNNs and classify them with regard to the bottleneck in their computations. 2) We overview various acceleration platforms for DNNs, including, CPUs, GPUs, NPUs, and CIM units, and compare them with regard to the key performance metrics. 3) We overview neuromorphic processors for SNNs and comprehensively analyze their distinguishable features from DNN accelerators.

The rest of the article is organized as follows. Section 2 overviews key computations (major workload) in DNNs classified as compute- and memory-bound models, software frameworks for autodifferentiation, and the suitable topology of DNNs to use such autodifferentiation frameworks. Additionally, this section overviews SNNs and their major workload in comparison with DNNs. Section 3 addresses various processors for DL acceleration, including GPUs, NPUs, and CIM units, and overviews several recent designs for each type of accelerator. Section 4 is dedicated to neuromorphic processors that accelerate computations in SNNs. In this section, we overview various neuromorphic processor designs and explain distinguishable concerns in neuromorphic processor design from that in DNN-based DL accelerator design. Section 5 concludes this review with concluding remarks and outlook.

2. Computation in NNs

We address generic features of NNs with regard to computation in Section 2.1, which hold for DNNs and SNNs. Computational features of DNNs and SNNs are addressed in Section 2.2 and 2.3, respectively.

2.1. Generic Properties of Operations in NNs

2.1.1. Elementary Layers and Their Computation

Although NNs differ in topology for different tasks, they are commonly of layer-wise topology, and neighboring layers are joined by unidirectional connections as illustrated in **Figure 1**a. Despite the diversity in NN topology, the major types of elementary layers include a dense layer and convolutional layer (conv layer for short)—illustrated in Figure 1b,c, respectively. Note that we assume a mini-batch size of one and FP32 data format, i.e., real-value data, hereafter unless otherwise stated. For a dense layer of N neurons with a fan-in layer of M neurons, its weight matrix w is of $N \times M$ in dimension. The total number of weights

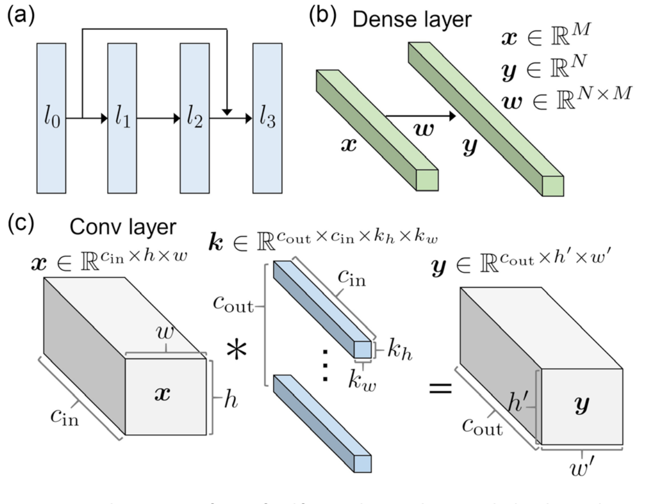

(a)
l_0
l_1
l_2
l_3
(b)
Dense layer
$\boldsymbol{x} \in \mathbb{R}^M$
$\boldsymbol{y} \in \mathbb{R}^N$
$\boldsymbol{w} \in \mathbb{R}^{N\times M}$
$\boldsymbol{x}$
$\boldsymbol{w}$
$\boldsymbol{y}$
(c)
Conv layer
$\boldsymbol{x} \in \mathbb{R}^{c_{\text{in}}\times h\times w}$
$\boldsymbol{k} \in \mathbb{R}^{c_{\text{out}}\times c_{\text{in}}\times k_h\times k_w}$
$\boldsymbol{y} \in \mathbb{R}^{c_{\text{out}}\times h'\times w'}$
c_{in}
c_{out}
k_h
k_w
w
h
h'
w'
$*$
$=$

Figure 1. Schematics of a a) feedforward neural network, b) dense layer, and c) conv layer. The dense and conv layers calculate $y = wx$ and $y = k \times x$, respectively.

Algorithm 1. Naive Matrix–Vector Multiplication. Multiplication of an input vector $x \in \mathbb{R}^M$ and weight matrix $w \in \mathbb{R}^{N \times M}$, yielding an output vector $y \in \mathbb{R}^N$.

```
Initialize $y$
for $i = 1$ to $N$ do
    $s \leftarrow 0$;
    for $j = 1$ to $M$ do
        /* Single MAC OP */
        $s \leftarrow s + w[i,j] \times x[j]$;
    end
    $y[i] \leftarrow s$;
end
```

and memory dedicated to them are NM and $32NM$ bits, respectively. The feature map z for this dense layer is calculated by a linear equation $y = wx$ (x denotes the feature map of the fan-in layer) and subsequent nonlinear equation $z = f(y)$, where f denotes a nonlinear activation function. The linear equation (matrix–vector multiplication) involves two nested for-loops, and thus NM MAC floating-point operations (FLOPs) in total, as shown in **Algorithm 1**. Therefore, the time complexity for the linear equation is $\mathcal{O}(n^2)$. The nonlinear equation $z = f(y)$ involves N FLOPs, i.e., $\mathcal{O}(n)$. Therefore, the major workload arises from the MAC FLOPs of $\mathcal{O}(n^2)$ in complexity. Given that the major workload involves NM weights and NM FLOPs, the ratio of the number of FLOPs to the number of weights (FWR) is one.

$$\text{FWR} = 1 \quad \text{for dense layers} \tag{1}$$

That is, one weight loaded is used for one FLOP, and thus, for each FLOP, one weight value needs to be loaded. In this case, memory-access latency and memory bandwidth (rather than the throughput of the arithmetic operations) likely dictate the overall operational throughput. Consequently, employing a high-bandwidth memory is an appropriate strategy to accelerate the dense layer computation.

A feature map for a conv layer is of $c_{\text{out}} \times h' \times w'$, where c_{out}, h', and w' denote the channel, height, and width of a rank-3 tensor (see Figure 1c). The feature map is calculated by convolution (linear operation) of a fan-in feature map x of $c_{\text{in}} \times h \times w$ using a rank-4 kernel k of $c_{\text{out}} \times c_{\text{in}} \times k_h \times k_w$, i.e., $y = k * x$ and subsequent nonlinear equation $z = f(y)$. The convolution ($y = k * x$) involves six nested for-loops for a unit MAC OP as shown in **Algorithm 2**, so that its time complexity is $\mathcal{O}(n^6)$. The number of FLOPs involved in the convolution is therefore $c_{\text{out}}h'w'c_{\text{in}}k_hk_w$. The nonlinear equation $z = f(y)$ involves $c_{\text{out}}h'w'$ FLOPs, i.e., the time complexity is $\mathcal{O}(n^3)$, and thus the major workload for convolution layers arises from the convolution operation of $\mathcal{O}(n^6)$ in complexity. FWR for this conv layer is therefore given by

$$\text{FWR} = \frac{c_{\text{out}}h'w'c_{\text{in}}k_hk_w}{c_{\text{out}}c_{\text{in}}k_hk_w} = h'w' \quad \text{for conv layers.} \tag{2}$$

That is, one weight loaded is $h'w'$ times reused for FLOPs, which is a distinguishable feature of conv layers from dense

Algorithm 2. Convolution of input tensor $x \in \mathbb{R}^{c_{\text{in}} \times h \times w}$ using kernel $k \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times k_h \times k_w}$, yielding output tensor $y \in \mathbb{R}^{c_{\text{out}} \times h' \times w'}$.

```
Initialize $y$;
for $i = 1$ to $c_{out}$ do
    for $j = 1$ to $h'$ do
        for $k = 1$ to $w'$ do
            $s \leftarrow 0$;
            for $l = 1$ to $c_{in}$ do
                for $m = 1$ to $k_h$ do
                    for $o = 1$ to $k_w$ do
                        /* Single MAC OP */
                        $s \leftarrow s + k[i, l, m, o] \times$
                        $x[l, j + m - 1, k + o - 1]$;
                    end
                end
            end
            $y[i, j, k] \leftarrow s$;
        end
    end
end
```

layers with FWR = 1. In this case, the arithmetic operational throughput (rather than memory bandwidth) likely dictates the overall operational throughput, so that an appropriate strategy to accelerate the conv layer computation is to increase the arithmetic operational throughput by employing multiple ALUs that work in parallel.

2.1.2. Data Formats

A commonly used data format is FP32 (single-precision floating-point): 1b sign, 8b exponent, and 23b mantissa (**Figure 2**a).

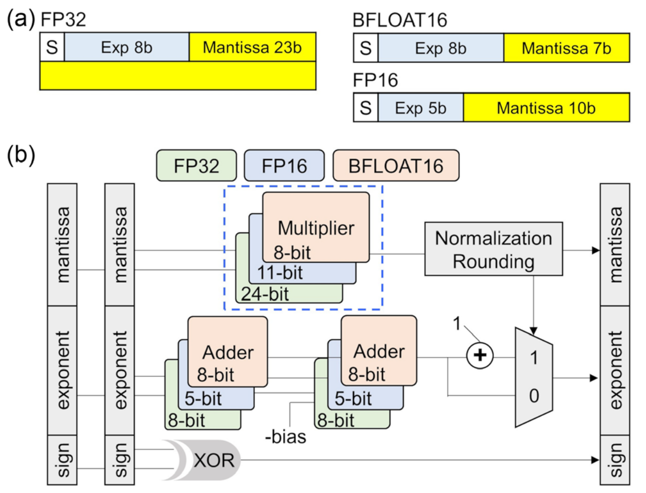


Figure 2. a) Schematics of FP32, BFLOAT16, and FP16 formats. b) Architecture of a FP multiplier.

Numbers in FP32 are considered as real values. Given the aforementioned trend in NN evolution, the space complexity (memory usage) prohibitively increases, leading to the desire for the use of lower precision FP formats such as FP16 (sign/exponent/mantissa: 1b/5b/10b) and BFLOAT16 (sign/exponent/mantissa: 1b/8b/7b). These formats are illustrated in Figure 2a. These low-precision FP formats allow the reduction in memory usage by 50% and memory-loading latency for a given memory bandwidth. Particularly, BFLOAT16 allows a similar number range to FP32 because of eight exponent bits but largely reduces power and area overheads for FP multipliers. The representative architecture of a FP multiplier is shown in Figure 2b. The multiplication of two numbers in FP involves the addition of two integer exponents and multiplication of two integer mantissa parts; integer adder and multiplier are included in Figure 2b. Given that the integer multiplier mainly dictates the power and area overheads for the FP multiplier, BFLOAT16 with ten mantissa bits can significantly reduce the power and area overheads compared with FP32 and even with FP16. Mixed precision is often used as for Tensor Processing Units (TPUs) (BFLOAT16 for multiplication and FP32 for accumulation).[44] In this case, FLOAT16 numbers are easily converted to FP32 by simply attaching 16 null bits to the right-hand side of the LSB in FLOAT16.

There exist efforts to use low-precision integer formats (e.g., INT8 and INT4) in place of floating-point formats to further reduce the memory usage, memory-loading latency, and power and area overheads for ALUs. To this end, additional algorithms for weight and activation quantization should apply to NNs, which cause inevitable loss in NN performance.[53] As such, an integer multiplier is much lighter than a FP multiplier as it is a component of a FP multiplier as shown in Figure 2b, which reduces the overheads for the multiplier logic. The extreme cases include binary weight and activation and power-of-two weight, which replace multipliers by simple XNOR logics and shift registers, respectively.

GPUs have excellent flexibility as they support various data formats, e.g., FP64/32/16, BFLOAT16, INT8/4, and even binary. ASIC-based accelerators are frequently designed for particular data formats to boost their performance at the cost of loss in flexibility. An extreme case is the CIM units based on analog MAC OPs, which limits the data format to integer only. Generally, the performance is significantly dictated by the data format used such that the lower the data precision, the higher the performance insomuch as the performance for INT4 is $\approx 64\times$ that for FP32 for NVIDIA A100.

Note that the term operation (OP) indicates format-unspecific operation, so that it is a rather general term that includes FLOP. Hereafter, we use the general terms OP and OPs to refer to operations in various data formats.

2.1.3. Directed Acyclic Graphs

Directed acyclic graphs (DAGs) define the sequence of successive computations and the consequent data flow. They are acyclic such that the data processed by a given node (function) are disallowed to be directed to the node through any paths, i.e., no self-association is allowed. NNs are mapped onto DAGs to train them by using GPUs with autodifferentiation frameworks like PyTorch[54] and TensorFlow.[55] The backpropagation of error algorithm (backprop for short) uses the backward pass of the error (evaluated at the output nodes), which is based on the chain rule on the gradient tensor computed for each node. Autodifferentiation frameworks compute the gradient tensors in a user friendly manner. The chain rule should apply to DAGs only because it yields wrong results for graphs with self-association like feedback connections.[56] Feedforward NNs can directly be mapped onto DAGs given no inherent self-association included. Recurrent NNs (RNNs) include in-layer connections, which inevitably include closed data paths within a layer. To use autodifferentiation frameworks for such RNNs, they are unrolled (duplicated) over time on DAGs in that the same node at different timesteps is considered as different nodes, which removes self-association. There exist NNs with feedback paths at the same timestep, e.g., SNN with spiking neurons whose state variables (membrane potential) are reset upon their spiking. In this case, the reset signals are delayed for one timestep to avoid self-association within the same timestep. Otherwise, particular mathematical techniques based on the implicit function theorem are required as for EXODUS.[56]

2.2. Computation in DNNs

CNNs are broadly used for computer vision, which include AlexNet,[28] VGG,[29] ResNet,[57] DenseNet,[58] GoogLeNet,[30] and so on. Most layers in CNNs are conv layers for feature extraction but with a few dense layer for classification. Therefore, the major operation type is convolution elaborated in the pseudocode in Algorithm 2. A pooling layer generally follows a given conv layer, which reduces the dimension of the conv layer. There exist various types of available pooling layers such as max pooling, average pooling, and adaptive max pooling. Max pooling for 2D kernels ($k_h \times k_w$ in size) involves a simple max function that outputs the largest feature among the features in the 2D kernel. Average pooling outputs the feature value averaged over the feature map within the kernel, requiring addition and division operations. Batch normalization (BN)[59] is commonly deployed after pooling. BN L2 normalizes the feature map after pooling using the mean and standard deviation of the features for a given channel over the samples in a mini-batch. The normalized feature map undergoes scale and shift using two trainable parameters, which require multiply and addition operations such that the normalized feature map is multiplied by the scale parameter, and the shift parameter is subsequently added. The nonlinear function finally applies to the result to compute the activation. Although various activation functions are available, ReLU and its variants are often chosen as activation functions for feedforward CNNs.

There are several lightweight DNNs that use depth-wise separable convolution in place of the aforementioned normal convolution to reduce their time and space complexities, e.g., MobileNets,[60,61] ShuffleNets,[62] and EfficientNets.[63] For a $c_{\text{out}} \times h' \times w'$ feature map with a $c_{\text{in}} \times h \times w$ fan-in feature map, depth-wise separable convolution requires the total number of OPs (time complexity) and parameters (space complexity) as follows

$$\begin{aligned}\#\ \text{OPs} &= \underbrace{c_{\text{in}}h'w'k_hk_w}_{\text{Depthwis econv}} + \underbrace{c_{\text{out}}c_{\text{in}}h'w'}_{\text{Separable conv}},\\ \#\ \text{parameters} &= \underbrace{c_{\text{in}}k_hk_w}_{\text{Depthwise conv}} + \underbrace{c_{\text{out}}c_{\text{in}}}_{\text{Separable conv}}.\end{aligned} \tag{3}$$

These highlight significant reductions in time and space complexities compared with the normal convolution whose time and space complexities are $c_{\text{out}}h'w'c_{\text{in}}k_hk_w$ and $c_{\text{out}}c_{\text{in}}k_hk_w$, respectively. However, FWR for a depth-wise separable conv layer is the same as the normal convolution.

$$\text{FWR} = \frac{c_{\text{in}}h'w'k_hk_w + c_{\text{out}}c_{\text{in}}h'w'}{c_{\text{in}}k_hk_w + c_{\text{out}}c_{\text{in}}} = h'w'. \tag{4}$$

Natural language processing depends on dense layer-based DNNs such as 1) RNNs, e.g., bidirectional RNN,[64] long short-term memory,[65] and gated recurrent units,[66] and 2) transformer[67] and its variants, e.g., bidirectional encoder representations from transformers[68] and generative pretrained transformer.[26,69] These models use either dense layers and/or dense layer-like operations, i.e., matrix–vector or matrix–matrix multiplications, so that FWR for the major operation is unity. Additionally, these models include exponential function-based nonlinear activation functions, e.g., sigmoid, hyperbolic tangent, and Gaussian error liner unit (GELU).[23] Particularly, GELUs are frequently employed in transformer-based models.

2.3. Computation in SNNs

SNNs are dynamic (time-dependent) models that extract spatio-temporal features of input data.[25] This is mainly because of spiking neurons (corresponding to activation functions in DNNs) and/or synapse models, which use spatio-temporal kernels to compute their state variables. There are diverse neuron models with different computational complexities, e.g., integrate-and-fire (IF) model, leaky IF (LIF) model,[70] and spike response model (SRM),[71] Hodgkin–Huxley model,[70] Izhikevich's model,[72] and so forth. Particularly, the first three models are often employed in deep SNNs given their low complexities (such that IF and LIF models use a single-state variable and SRM two-state variables) and simplicity in implementation. The last two models concern their biological plausibility rather than computational complexity. Hereafter, we address IF and LIF models for deep SNNs with the minimal complexity. IF and LIF models use separable spatio-temporal kernels k to compute their state variables, i.e., subthreshold membrane potential u_i for a neuron i.

$$\begin{aligned}u_i(t) &= \sum_j k_{ij} * \rho_j(t)\\ &= \sum_j \underbrace{w_{ij}}_{\text{spatial}} \underbrace{\varepsilon}_{\text{temporal}} * s_j(t)\\ s_j &= \sum_k \delta(t - \hat{t}_k)\end{aligned} \tag{5}$$

where s_j indicates a train of spikes at $\hat{t}_k$ from a fan-in (presynaptic) neuron j, and w_{ij} is the weight between neurons i and j. IF and LIF models differ for the temporal contribution ε in Equation (5) such that

$$\varepsilon = \begin{cases} e^{-t/\tau_m}H(t) & \text{for } LIF\\ H(t) & \text{for IF}\end{cases} \tag{6}$$

where H denotes the Heaviside step function. Note that the temporal kernel for the IF model is considered as a special case of the LIF model with $\tau_m \to \infty$.

Equation (5) is expressed as a form of convolution integral as follows

$$u_i(t) = \sum_j w_{ij} \int_0^t \varepsilon(\tau)s_j(t-\tau)\mathrm{d}\tau \tag{7}$$

With the temporal kernel in Equation (6), Equation (7) is equivalent to the following differential equation

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{u_i}{\tau_m} + \sum_j w_{ij}s_j \tag{8}$$

This equation can be expressed as a discrete form (with $\Delta t = 1$) using the Euler method (explicit method), and we have the following recursive form

$$u_i[t] = e^{-1/\tau_m}u_i[t-1] + \sum_j w_{ij}s_j[t] \tag{9}$$

When the membrane potential u_i exceeds a given threshold θ at t, an LIF or IF neuron fires an event (spike), which can be implemented using the Heaviside step function such that

$$s_i(t) = H(u_i - \theta) \tag{10}$$

This spike function corresponds to the nonlinear activation function. Upon the spike generation, the potential u_i is reset to a preset potential value (often to zero) and ready for the integration of subsequent spikes. The generated spike s_i is subsequently routed to the fan-out (postsynaptic) neurons with the corresponding kernel values.

The aforementioned sequence of spiking-neuron computations (Equation (9) and (10)) is mapped onto a graph in **Figure 3**. Note that the superscript of each variable indicates the layer index. The signal from $s^{(l)}$ to reset the potential $u^{(l)}$ is delayed for one timestep, i.e., $s^{(l)}[t-1]$ resets $u^{(l)}[t]$ rather than $u^{(l)}[t-1]$. This is to avoid self-association as explained in Section 2.1.3. The spike function in Eq. (10) for a layer l results a spike (event) map $s^{(l)}$ which is equivalent to a feature map for a DNN. Its dimension is determined by the layer type (conv or dense layer) as elaborated in Section 2.1. Notably, the spike map includes binary data only given the elements $s_i^{(l)} \in \{0,1\}$. A crucial feature of such spike maps is that the number of '1's in the map is remarkably sparse, i.e., the sparsity f_{sp} is close to unity, which is defined by

$$f_{\text{sp}} = \begin{cases} \dfrac{\text{Number of 0's}}{c \times h \times w} & \text{for} \quad s \in \{0,1\}^{c\times h\times w}\\ \dfrac{\text{Number of 0's}}{m} & \text{for} \quad s \in \{0,1\}^{m}\end{cases} \tag{11}$$

As a generic property of NNs (explained in Section 2.1), the major operations in SNNs are the multiplication of a weight

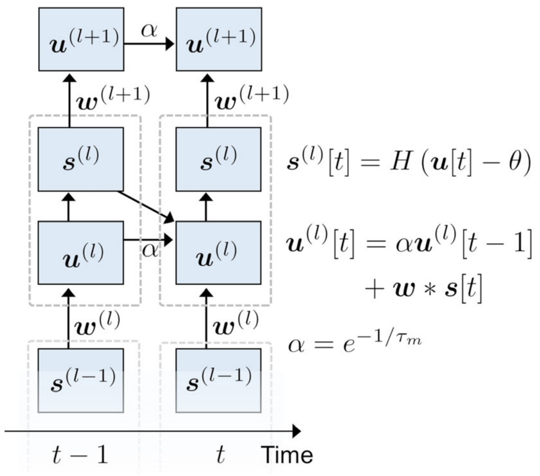


Figure 3. SNN mapped onto a DAG.

w_{ij} and activation s_j and the subsequent accumulation of the products to compute the membrane potential, which are expressed in Equation (9). This MAC OP is referred to as synaptic operation (SynOP)

$$u_i \leftarrow u_i + w_{ij}s_j \tag{12}$$

which is equivalent to MAC OP in DNNs. Given the high sparsity of spike maps, s_j in Equation (12) is mostly zero, so that SynOPs for zero s_j may be skipped using dedicated processors, particularly, event processors. A unit SynOP is equivalent to routing an event from a presynaptic (fan-in) neuron to a neuron i and subsequently updating its membrane potential u_i by w_{ij}, which is carried out only upon event, i.e., no SynOP when $s_j = 0$. This ad hoc update of membrane potential is the key to event processors that may largely reduce the computational overhead, which will be detailed in Section 4.1.

3. Accelerators for DNN-Based DL

For the moment, the megatrend in DNN evolution (deeper and larger) is supported by the hardware that enables massively parallel computing with large memory bandwidth, e.g., GPUs and NPUs. These processors support massive parallelism in arithmetic operation, significantly accelerating the major arithmetic operations. Yet, their limited memory bandwidth can limit the acceleration within the von Neumann architecture in which the main memory and processing unit are separate and communicate through buses with limited bandwidth. There barely exist ideal processors with regard to operational throughput, power efficiency, versatility, flexibility, and price. In the following sections, we address various types of DL accelerators and their pros and cons in terms of the aforementioned key metrics. To begin, Section 3.1 explains the roofline model that frequently applies to DL accelerators for performance estimation for given DNNs. Section 3.2 provides a brief overview on such accelerators mainly for comparisons among them. We address various accelerators classified as 1) general-purpose hardware-based accelerators, e.g., CPUs and GPUs, and 2) ASIC-based accelerators, e.g., NPUs and CIM units. The former is addressed in Section 3.3, and the latter in Section 3.4 and 3.5, respectively.

3.1. Roofline Model

The roofline model proposed in ref. [73] provides insights into computers with particular peak performance in OPS (OPs/s) and memory bandwidth in an easy-to-understand manner. For a given computer, the roofline model designates the attainable performance with respect to arithmetic intensity, i.e., the ratio of the number of OPs to the number of data for a given task. Particularly, this model highlights the bottleneck for given tasks. The peak performance cannot fully be utilized when the data transfer is slow due to the limited memory bandwidth, so that the attainable performance is dictated by the memory bandwidth (memory-bound case). When the data transfer is fast (high memory bandwidth), the attainable performance is dictated by the peak performance (compute-bound case). Hence, the attainable performance for a given computer is expressed as

$$\text{Attainable performance} = \min\begin{cases}\text{Peak performance} \\ \text{Arithmetic intensity} \times \text{memory bandwidth.}\end{cases} \tag{13}$$

Figure 4 illustrates a schematic of the roofline model. The arithmetic intensity value at the cross point between the two lines is referred to as a ridge point. Tasks of arithmetic intensity below the ridge point are termed memory-bound tasks while the opposite tasks compute-bound tasks.

As detailed in Section 2.1, FWR for conv and depth-wise separable conv layers (Equation (2) and (4)) is $h'w'\times$ FWR for dense layers (Equation (1)). Notably, the arithmetic intensity in the roofline model likely scales with FWR. Thus, the computation of conv layers is likely located on the right-hand side of the ridge point (i.e., compute-bound) while that of dense layers on the left-hand side of the ridge point (i.e., memory-bound). Consequently, CNNs are likely classified as compute-bound models while RNNs and transformer-based models for natural language processing as shown in ref. [44] for TPUs.

The y-intercept in Figure 4 (highlighted using a red-filled circle) corresponds to the memory bandwidth on a logarithmic scale. For a fixed peak performance, the increase of memory

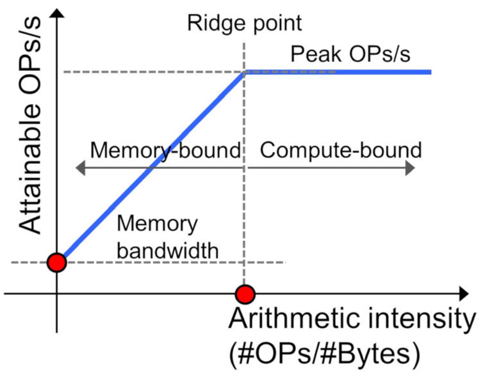


Figure 4. Schematic of the roofline model on a log–log plot.

bandwidth reduces the ridge point, widening the compute-bound range in which the peak performance is utilized. In this way, dense layer-based models may belong to the compute-bound regime. Therefore, the DL accelerator considered should establish an optimization strategy for DL workloads with limited memory bandwidth. In the following section, we overview common features for each accelerator type and present acceleration strategies with regard to the roofline model.

3.2. Overview on Accelerators

For memory- and compute-bound tasks (depicted in Figure 4), DL accelerators employ two distinct accelerating strategies: 1) memory optimization, e.g., data-transfer optimization and increase of off-chip memory bandwidth, and 2) boost in peak performance, e.g., parallel computation and analog computation. This review also discusses the power efficiency of various platforms, which is not addressed in the roofline model. DL accelerators have been developed on various hardware platforms that include CPUs and GPUs in general-purpose architecture, and NPU and CIM in application-specific architecture, but within the von Neumann architecture in which a main memory and processing units are separate. A schematic of the roofline model for CPUs, GPUs, NPUs, and CIM units is illustrated in **Figure 5**.

A CPU consists of a large number of branch control logic units and cache hierarchy of high memory bandwidth, and a small number of ALUs as illustrated in **Figure 6**. CPUs are widely used in general-purpose computers because of their high flexibility, large-size cache memories of high-bandwidth, and large number of branches supported. However, the limited number of cores (particularly, ALUs) limits their capability to process parallel workloads.[74] Hence, CPU-based accelerators need to employ additional computation engines within CPUs and/or new type of cache (detailed in Section 3.3). Additionally, an off-chip double data rate (DDR) memory of limited bandwidth is commonly used as a main memory.

Unlike CPUs, GPUs are equipped with numerous ALUs that work in parallel and most widely used as DL accelerators. A common architecture of GPUs is illustrated in **Figure 7**. GPUs exploit parallelism by SIMT in that massive multiple threads allocated by a GPU kernel are designated for computation and simultaneously processed. To boost the off-chip memory bandwidth, GPUs use graphics DDR (GDDR) or high-bandwidth memory (HBM) of a higher bandwidth than DDR. The commercialized GDDR6 realizes $\approx 4-10\times$ the bandwidth of DDR5.[75] The schematic of the roofline model for GPUs in Figure 5 highlights the higher peak performance and memory bandwidth than CPUs. However, their inherent high thermal design power (TDP) is a challenge to devices for on-device AI.[76,77] As general-purpose platforms, the key advantage of GPUs lies in their flexibility and versatility, accelerating various DL models in

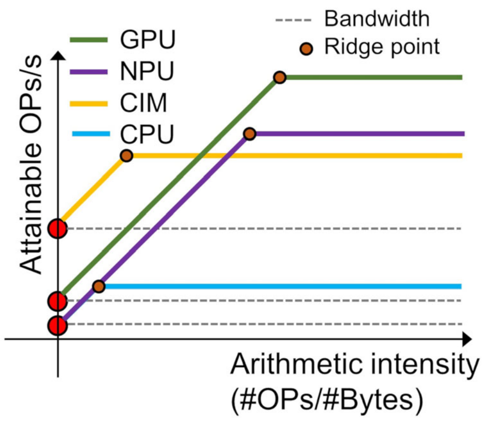


Figure 5. Schematic of the roofline model for various accelerators.

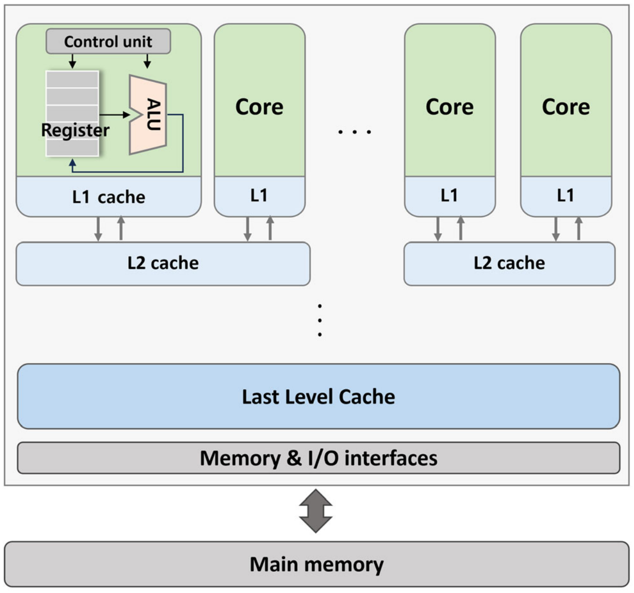


Figure 6. Architecture of a multicore CPU.

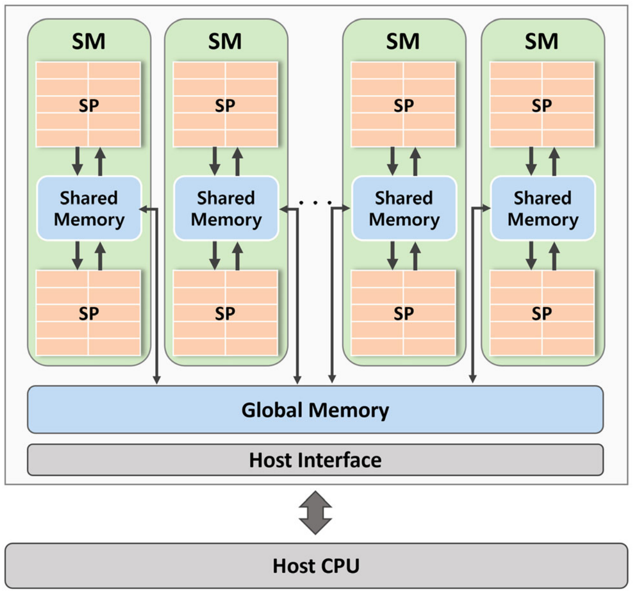


Figure 7. Architecture of a GPU. SM and SP stand for streaming multiprocessor and streaming processor, respectively.

various data formats (e.g., FP64/32/16, BFLOAT16, INT8/4, even binary). Also, GPUs can remarkably accelerate both memory- and compute-bound models given their high memory bandwidth and high peak performance based on massive parallelism. While the maximum parallelism is attainable through the flexible configuration of threads via software, data transfer between threads is realized only through the shared memory because GPUs support SIMT. This leads to a significant amount of data movement, and thus latency and power consumption .

In contrast to GPUs, NPUs (ASIC-based accelerators including TPU) can enhance computational efficiency through data-movement optimization by 1) an array of processing elements (PEs) that are connected using registers and 2) on-chip memory hierarchy (e.g., buffer, register, and scratchpad) (**Figure 8**). A unit PE carries out a unit MAC OP per cycle. Systolic arrays[78] are frequently employed in NPUs, which realize a sequence of MAC OPs based on partial sums buffered in the registers (each of which is embedded in a PE). A schematic of a generic systolic array is sketched in **Figure 9**. This systolic array-based architecture is suitable for convolution operations. Given the distributed registers over PEs in use, the off-chip memory access frequency can significantly be reduced, boosting the performance for a given memory bandwidth and reducing power consumption. Figure 5 shows a schematic of the roofline model for NPUs, indicating the higher peak performance than CPUs but the similar memory bandwidth to CPUs, assuming DDR in use. Unlike general-purpose platforms, NPUs hardly support various data formats given the PEs are designed for specific data format as for TPUv4 (BFLOAT16 and FP32 for multiplication and accumulation, respectively).[45] NPUs are suitable for compute-bound models only given their high peak performance but lower memory bandwidth than GPUs. For memory-bound models, the lower memory bandwidth hinders NPUs from fully utilizing the peak performance.

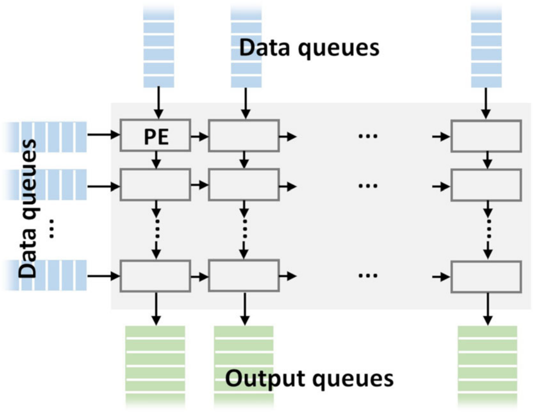


Figure 9. Schematic of a systolic array and dataflow.

CIM aims to cope with the massive data movement from the off-chip memory and the consequent latency and power consumption by placing PEs near or in the memory domain. This can significantly reduce the frequency of the off-chip memory access and data-movement latency for a given memory bandwidth. This is equivalent to the increase of memory bandwidth in effect as highlighted in Figure 5. However, CIM is endowed with a lower peak performance because of the limited parallelism in arithmetic operation compared to NPUs. A common architecture of CIM units is illustrated in **Figure 10**. Hereafter, CIM with digital PEs near or in the memory domain is referred to as digital CIM while CIM based on fully or partially analog computing as analog CIM. To date, digital CIM has been demonstrated using 1) various types of on-chip memories such as volatile memories, e.g., dynamic RAM (DRAM)[47,79,80] and static RAM (SRAM),[81–86] and nonvolatile memories, e.g., resistive RAM (RRAM),[87] and 2) various architectures, e.g., near-data processing (PEs in the vicinity of the memory domain) and

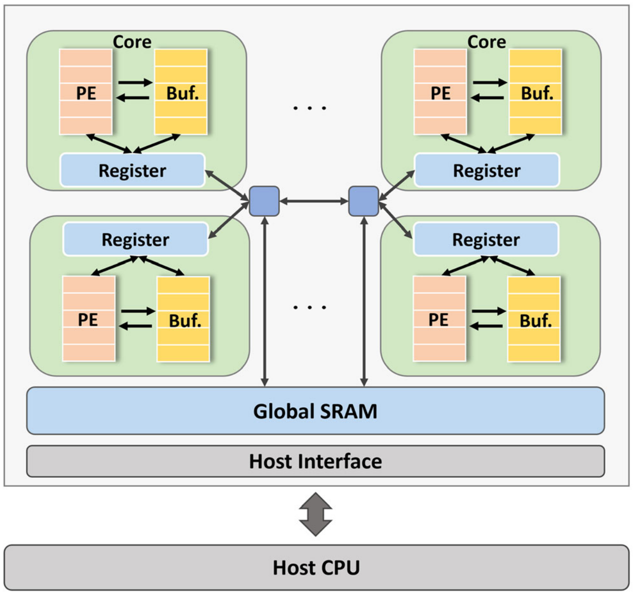


Figure 8. Architecture of NPU. Each core indicates a PE for a unit operation.

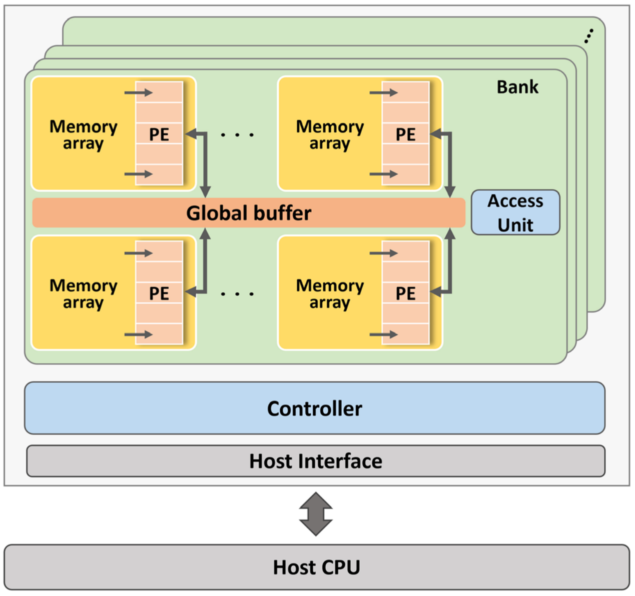


Figure 10. Architecture of a CIM unit.

PEs merged into the memory domain. High-bandwidth DRAM-based CIM units have been prototyped, for instance, a HBM-based digital CIM unit (FIMDRAM)[47,79] and GDDR6-based CIM units (AiM).[80] These examples use high-bandwidth DRAMs as embedded memories in conjunction with PEs in the vicinity of the memories. There exist diverse SRAM-based digital CIM units that fully utilize the advantages of SRAM such as fast operation, high bandwidth, and perfect compatibility with complementary metal–oxide–semiconductor (CMOS) logic circuits.[81–86]

Analog CIM is based on PEs merged into the memory domain, where full or partial MAC OPs are performed in an analog manner. Consequently, memory access and full or partial MAC OP are performed at a single cycle, so that the latency for load and store in digital CIM can be avoided. A front-runner is SRAM-based CIM in which bitwise multiplication is simply implemented by using an AND gate (digital) and accumulation by using a bitline capacitor (analog). Resistance-based nonvolatile memory (e.g., RRAM and MRAM)-based analog CIM has been successfully demonstrated.[88–96] Particularly, RRAM-based analog CIM is based on Kirschhoff's current law in a nonvolatile resistor array, inherently realizing bit-wise multiplication and accumulation of currents through RRAM bitcells that share the same bitline. Both digital and analog CIM units will be addressed in detail in Section 3.5.

3.3. CPU-Based Accelerators

CPU-based accelerators proposed to date mostly aim to perform operations in caches. As shown in Figure 6, cache hierarchy is used to efficiently retrieve data in a CPU for various applications.[97–99] CPU-based DL accelerators aim to perform operations in the cache to minimize data movement, and thus latency and power consumption. However, the limited number of cores (and thus cache memories) in a CPU limits the operational parallelism, so that its performance is hardly comparable to the other accelerators when using deep and large DNNs. Nevertheless, lightweight DNNs with high operational sparsity can solely be handled by CPU-based accelerators without the cost of data movement between CPU and other accelerators.

Compute near last level cache (CNC) realizes cache-level DL acceleration by integrating an auxiliary MAC unit in the last level cache (LLC) of 512 KB in an eight-core 64b RISC-V CPU.[100] Instead of loading data into the core for MAC OPs, performing the operations within the LLC can boost the performance and energy efficiency. The CNC MAC unit multiplies an 8×8 INT8 matrix by an 8-long INT8 vector and accumulates the product vector in INT32. Custom instructions are added to the RV64GC instruction set architecture (ISA) for the MAC OP in the LLC. The processor (fabricated using the Intel 4 CMOS process) consumes 510 mW (at 0.85 V and 1.15 GHz) and 73 mW (at 0.55 V and 350 MHz). The CNC achieved 46× and 27× performance of the scalar ISA for dense and conv layers, respectively.

Compute cache allows logical operators, e.g., AND, NOR, and XOR, in the cache without a large area overhead by designing an additional decoder for activating multiple word lines in parallel and a single-ended sense amplifier for each bitline.[101] Additionally, compound operations, e.g., compare, search, copy, and carryless multiplication, are supported. Compute cache enhances operational performance by 1.9× and reduces energy consumption by 2.4× compared to an Intel's eight-core Sandy Bridge processor with three-level cache hierarchy.

Neural cache[102] is the extension of compute cache, which supports in-cache integer arithmetic operations in addition to the logical and supplementary operations supported in compute cache. For the in-cache arithmetic operations, data are transposed and mapped onto the SRAM array, i.e., each n-bit element is stored over n word lines. Two operands sharing a single bitline are iteratively computed for each bit by simultaneously activating the two word lines. Neural cache realizes integer addition, multiplication, and division for $n+1$, n^2+5n-2, $1.5n^2+5.5n$ cycles, respectively. Compared to baseline CPU (Xeon E5) and GPU (Titan Xp) for Inception-v3,[103] neural cache reduces inference latency by 18.3× and 7.7× compared with a Xeon E5 CPU and Titan Xp GPU, respectively. Further, its energy efficiency increases by 37.1× and 16.6× compared to the CPU and GPU, respectively.

Duality cache is an in-cache computation architecture to support various in-cache arithmetic operations in integer, fixed-point, and floating-point formats.[104] For addition in FP (which is of large complexity as explained in Section 2.1.2), duality cache is equipped with a new FP addition algorithm (referred to as bit-serial) of low time complexity, which applies to multiple data in parallel. For INT multiplication and division, zero skipping algorithms are adopted to skip redundant arithmetic operations. The CORDIC algorithm[105] is used for transcendental functions in FP. Duality cache implemented in a two-socket Xeon server by using the entire cache can support 150× the number of threads supported by a NVIDIA Titan Xp GPU, achieving an average speedup of ≈3.6× and ≈4× compared with the GPU for Rodinia[106] and OpenACC benchmarks, respectively, at the cost of an increase in area overhead by merely 3.5% and TDP by 3 W.

Intel has recently announced DL acceleration using scalable Xeon CPUs with a built-in AI accelerator referred to as Intel advanced matrix extensions (Intel AMX).[107] The instruction set for Intel AMX is an extended version of the x86 ISA to optimize DL training and inference tasks. To minimize data movement, maximize parallelism, and reduce needs for additional hardware design, the Intel AMX architecture is based on 1) eight 1KB 2D registers (tiles) for data storage and 2) a tile matrix multiplication unit attached to the tiles. Intel AMX supports BFLOAT16 and INT8 formats, offering a respective speed boost of 16× and 8× compared to the previous generation Xeon CPUs without Intel AMX.

3.4. NPU

NPU was coined by Esmaeilzadeh et al. to refer to an ASIC unit that supports parallel arithmetic operations for NNs.[43] The NPU proposed by Esmaeilzadeh et al. is equipped with eight PEs to accelerate neural programs and interface with a CPU. Each PE comprises a multiply-add unit, accumulator registers, and sigmoid unit to compute multilayer perceptron (MLP). The NPU includes three first-in first-out (FIFO) modules (configuration FIFO and input and output FIFOs) to interface with the CPU in use. The configuration FIFO is used to send and retrieve the MLP parameters, and the input and output FIFOs to send and retrieve the input and output data in the execute phase of CPU cores, respectively. The programmer chooses the program segments subject to acceleration in the NPU with regard to the following conditions: execution frequency, approximability, and well-defined input and output. Subsequently, the program segments are converted into NPU instructions through a compiler and processed in the NPU. The performance of the CPU–NPU was identified using the MARSSx86 cycle-accurate x86-64 simulator.[108] With the Intel's Penryn microarchitecture, the NPU achieves up to 2.3× speedup and 3.0× energy saving.

Frequently, NPUs are designed to minimize data movement from their off-chip memory (and thus power consumption and latency) by employing systolic arrays of PEs,[78] which realize a sequence of MAC OPs based on the partial sums buffered in the registers, each of which is embedded in a PE (see Figure 9). Systolic array-based NPUs are classified as weight-stationary (storing a weight in each PE), e.g., Origami[109] and TPU[44] and row-stationary (storing more data of an input activation and weight in each PE), e.g., Eyeriss.[110] The latter is to improve data reuse and to reduce energy consumption by minimizing data movement from the off-chip memory but at the cost of a larger local memory in each PE. A schematic of a systolic array for row-stationary and weight-stationary is illustrated in **Figure 11**a,b, respectively.

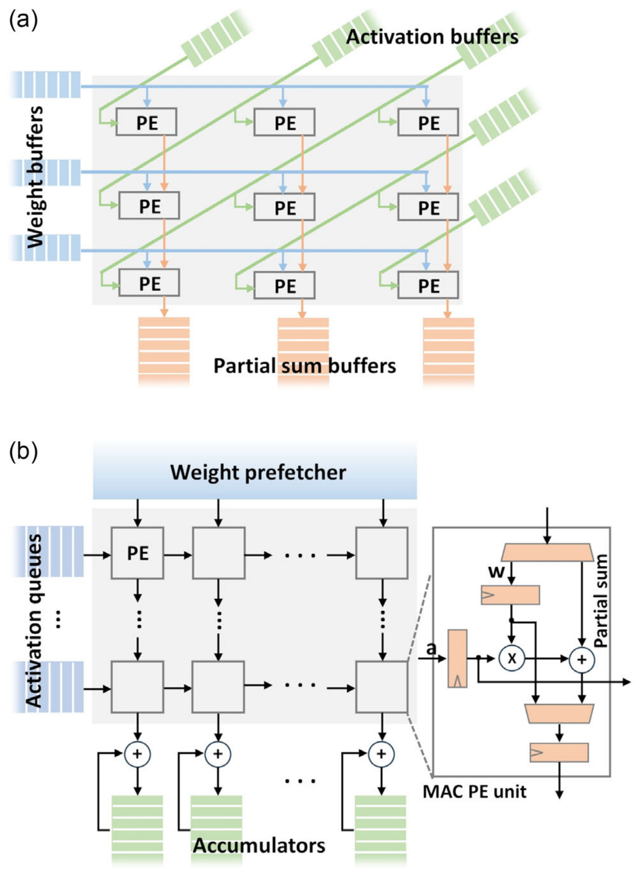


Figure 11. Schematic of a systolic array for a) row stationary and b) weight stationary.

Eyeriss is an NPU based on systolic arrays for row-stationary dataflow, which was fabricated using a 65 nm CMOS process in a 16 mm^2 die.[110,111] 2D kernel and feature map data in 16b fixed-point are distributed over the 12 × 14 PEs, which are stationary. Each PE computes dot-product of a given row of the kernel and a feature map row of the same size and buffers the result in the scratchpad. Each kernel row and activation row is horizontally and diagonally distributed, respectively; partial sums are vertically accumulated (Figure 11a). Eyeriss attains 33.6 GOPS at 200 MHz core clock and 1 V supply voltage.

Origami is an NPU based on weight-stationary systolic arrays (Figure 9b), designed using 65 nm CMOS process in a 3.09 mm^2 die.[109] This NPU comprises four processing channels, each of which is given $k_h k_w$ multipliers for parallel multiplications of 12b data and adder-tree for accumulation. The results in the processing channel are truncated to 12-b data. In each processing channel, $c_{\text{in}} \times k_h \times k_w$ kernel data are registered for weight reuse. Origami attains a maximum performance and power efficiency of 274 GOPS and 369 GOPS/W, respectively.

Another NPU based on weight-stationary systolic arrays is TPU.[44,45] TPUv1 consists of a matrix multiply unit, unified buffer for local activation, accumulators, control unit, and interface unit. The matrix multiply unit contains 256 × 256 MAC PEs for INT8 data. 64KB weights are loaded in the matrix multiply unit from a 8 GB off-chip DRAM. The 16b multiplication results are accumulated in a 4 MB accumulator for 32b data. The large on-chip memory of 28 MB in total supports the memory-hungry weight-stationary scheme, yielding a peak performance of 92 TOPS at 700 MHz. TPUv1 is 15 – 30× faster than a K80 GPU and Haswell E5-2699 v3 CPU, and its power efficiency is 30 – 80× higher. TPUv4 supports BFLOAT16 for multiplication and FP32 for accumulation.[45] Its peak performance attains 275 TOPS at a TDP of 170 W at 1 GHz.

The aforementioned stationary-based NPUs focus on minimizing data movement, but require additional registers (scratchpad memory for Eyeriss) for inter-PE data movement. This is because the full operation is split into a number of unit operations (MACs) and the results of unit operations need to be buffered in each PE. An alternative strategy is to perform the full operation using a multiply and adder-tree structure through which the data flow without data buffer. An example is DianNao developed to accelerate the computation of large scale NNs with a tiling method to alleviate memory bandwidth requirements.[112] The architecture comprises three stages of neural functional units (NFUs), three split buffers, and a control processor. In the NFU, input activations and weights (all in 16b fixed-point) are multiplied in the first stage, and the products are added through a adder-tree in the second stage. The final stage computes the activation function for the results from the adder-tree. The three buffers store input activations, weights, and output activations. DianNao was designed using a 65 nm CMOS process and simulated to identify its performance, yielding 452 GOPS at 0.98 GHz. Subsequently, Luo et al. introduced an extended version of DianNao, referred to as DaDianNao.[113] DaDianNao (designed using a 22 nm CMOS process) consists of 16 computing tiles in total and can perform 16 operations per tile (i.e., total 256 operations). Moreover, a four-bank embedded DRAM (eDRAM) in each tile accommodates a number of weights on chip.

3.5. CIM Units

3.5.1. Digital CIM

Function-In-Memory (FIM) DRAM is a digital CIM unit with programmable computing units (PCUs) supporting FP16 data and HBM2 as an embedded memory.[47] FIMDRAM includes a 16-wide SIMD engine in the memory banks to achieve bank-level parallelism. The physical dimension of HBM2 was maintained by replacing half of the memory array by PCUs. Multibank operations are supported using the FIM mode while the general memory operations using the normal mode. The PCU comprises a register group, execution unit, and interface unit, and is controlled using the conventional memory commands (CMDs) from the host without modifying the conventional memory controller. FIMDRAM was fabricated using a 20 nm DRAM process and achieves 1.2 TFLOPS in FP16 at 300 MHz.

Another example of eDRAM-based digital CIM units is AiM which is based on 4 Gb GDDR6.[80] In AiM, one processing unit (PU) is dedicated to each of 16 banks and placed in the vicinity of the bank. Notably, a set of new CMDs is introduced for bank activation, compute, and data movement unlike FIMDRAM. This new CMD set allows swift switches between the memory mode and DL operation mode without commands from the host. Each PU is equipped with 16 multipliers and a four-stage adder-tree for BFLOAT16 data. For parallel MAC OPs, the PU receives 1) 256b weights from its bank and also 256b activations from the global buffer or 2) 256b weights from its bank and activations of the same size from the paired bank. Additionally, AiM supports various activation functions, e.g., sigmoid, hyperbolic tangent, GELU, ReLU, and leaky ReLU, based on lookup tables (LUTs), each of which stores uniform inputs and their function values for each activation function. AiM attains 1 TFLOPS for BFLOAT16 at 1 GHz.

SRAM is frequently used as an embedded memory in digital CIM units.[81–86] Mostly, weights are stored in the SRAM other than a few cases in which activations are stored in the SRAM, e.g., Z-PIM.[81] Most of designs separate a processing domain from a memory domain[82–85,114] other than a few cases[81,86] as follows. Chih et al. proposed a 6 T-SRAM-based CIM macro design with integrated bit-wise multipliers (4 T NOR gate) in the SRAM domain.[86] For massive computing parallelism, the architecture employs bit-serial multipliers (in the memory domain) and parallel adder-trees (in the processing domain). This design supports the programmable precision of input activations (binary to INT8) and weights (INT4/8/12/16). The memory domain includes a 256×4 4b SRAM array in which a 4 T NOR gate for bit-wise multiplication is dedicated to each bit cell, supporting 256 bit-wise multiplications of 256 activations and 256 weights in parallel. The 256 products are summed through the add-tree. A single macro was fabricated using a 22 nm CMOS process in a 0.202 mm^2 die, yielding a performance of 3.3 TOPS (for 4b activation and 4b weight) at 0.72 V supply voltage.

Z-PIM also uses bit-wise multipliers integrated in the SRAM domain.[81] Notably, the SRAM array stores input activations instead of weights unlike the above explained CIM designs. The key feature of this CIM macro is a zero-skipping scheme that avoids MAC OPs for zero weights.

3.5.2. Analog CIM Units

Analog CIM demonstrates its high operational efficiency by reducing the area and power overheads for multipliers and adders, and eliminating on-chip data loading. However, they support integer MAC OPs only, and operational reliability is unlikely comparable to digital CIM. There exist a number of analog CIM designs using various memories such as mainstream memories, e.g., SRAM[115–119] and DRAM,[120,121] and emerging nonvolatile memories, e.g., resistive RAM (RRAM)[87–95] and magnetic RAM (MRAM).[96] We introduce a few examples in each class of analog CIM as follows.

Dong et al. introduced an 8 T SRAM-based analog CIM macro fabricated using a 7 nm CMOS process.[118] A sufficient noise margin allows the 8 T SRAM array of 64×64 in size to remain stable even when multiple words are activated for parallel MAC OPs. The macro computes multiply-average (MAV) operations for 4b activations and also 4b weights. Multibit inputs are realized in a bit-serial manner by a 4b digital counter. Multibit weights are realized by using multiple capacitors of power-of-two relative capacitance (1:2:4:8) at the end of each bitline. MAV operations for 64 4b inputs and 16 4b weights are computed using a flash analog-to-digital converter (ADC). The macro attains 455.1 GOPS and 321 TOPS/W for INT4 MAV operations at 1 V.

Fully analog CIM is prone to operational errors due to the limited signal-to-noise ratio (SNR) of ADCs. Mixed-signal CIM may be a noise-robust alternative. In this regard, Su et al. introduced 384 Kb 6 T SRAM-based CIM.[115] The proposed macro consists of SRAM subarrays, ADCs, and digital shifter and adder (DSaA). In each subarray, 32 6 T-SRAMs are connected to a local bitline pair (LBL/LBLB), and 16 sub-arrays are connected to a global bitline pair (GBL/GBLB). In a subarray, voltage-scaled 2b activations and 1b weights (stored in SRAM) are multiplied using LBL/LBLB, and the products are averaged using GBL/GBLB. The ADC converts the averaged product into a 5b digital value. These values from multiple ADCs are combined in DSaA to obtain a 20b output. The macro achieves a peak performance of 22.75 TOPS/W (peak) for INT8 data at 0.85 V supply voltage.

DynaPlasia is a system-level CIM unit with reconfigurable 3T2C eDRAM of 9.6 Mb.[121] Each bitcell is reconfigurable such that one of memory, in-memory computing, and ADC modes can be chosen. Particularly, a bitcell capacitor serves as a unit capacitor of a successive-approximation ADC in the ADC mode, significantly reducing the area overhead for ADCs. DynaPlasia was fabricated using a 28 nm CMOS technology in a 20.25 mm^2 die. It attains 56 TOPS/W peak performance for INT4 activations, INT5 weights at 250 MHz, and 1 V supply voltage.

RRAM is resistance-based nonvolatile memory[122] which was considered as storage-class memory. Following Kirschhoff's current law, the 1T1R bitcells sharing the same bitline inherently realize bit-wise multiplications and accumulation of the currents through the bitcells, equivalent to parallel MAC OPs. ISAAC is an analog dot-product machine that leverages this inherent property of RRAM crossbar arrays.[92] ISAAC comprises many tiles, each of which includes eDRAM to store input activations, output registers, in situ multiply-accumulate (IMA) units, shift-and-add, sigmoid, and max pooling units. Each IMA includes RRAM crossbar arrays, digital-to-analog converters (DACs) for input activations, and ADCs for outputs. The authors proposed three performance metrics: computational efficiency (CE in GOPS $\times$ mm^2), power efficiency (PE in GOPS/W), and storage efficiency (SE in MB/mm^2). These metrics were maximized by searching for the optimal size of each RRAM crossbar array, the numbers of crossbar arrays and ADCs in each IMA, and the number of IMAs in each tile. The system-level simulation results highlight improvements of throughput, power efficiency, and computational density by 14.8$\times$, 5.5$\times$, and 7.5$\times$, respectively, compared to DaDianNao.

TIMELY is an analog dot product machine based on RRAM crossbar arrays with three key innovations: analog local buffers (ALBs) to enhance data locality, time-domain interfaces (TDIs) to reduce the number of data conversions, and only-once input read (O^{2}IR) to reuse input activations.[94] When transferring data from an analog to a digital domain, the register in the digital domain consumes considerable energy and time. In this regard, ALBs eliminate the necessity for the data conversion. Additionally, energy cost is reduced by replacing ADCs and DACs (used in the conventional crossbar array) by time-to-digital converters (TDCs) and digital-to-time converters. O^{2}IR also reduces the energy cost by reducing memory access frequency. TIMELY was designed using a 65 nm CMOS process, working at 40 MHz and 1.2 V supply voltage. TIMELY improves energy efficiency by 18.2$\times$ and computational density by 20$\times$ compared to ISAAC.

MRAM is also resistance-based nonvolatile memory based on current-controlled magnetic tunnel junctions (MTJs) that exhibit nonvolatile high- and low-resistance states depending on their spin configuration. Jung et al. introduced a spin-transfer-torque MRAM (STT-MRAM)-based CIM unit for binary NNs (binary activation and binary weight).[96] CIM in this unit is realized by using strings of MTJs instead of MTJ crossbar arrays to reduce the power consumption in the memory domain. Given the considerable current in even high-resistance state MTJs, MTJ crossbar arrays consume high power during the current summation for parallel MAC OPs. To cope with this power issue, the proposed design uses strings of MTJs like NAND flash, where the total resistance of each string is determined by the number of high-resistance state MTJs. Each bitcell is of 2T2M, i.e., two MTJs of complementary resistance states and two transistors with complementary inputs, to realize the XNOR logic for binary NNs. Thus, the resistance sum of a given string is equivalent to the dot product of binary weight and activation vectors. The string resistance is subsequently converted to a digital value using a TDC.

4. Accelerators for SNN-Based DL

SNNs are time-dependent models for DL with unique features distinct from DNNs as addressed in Section 2.3. Particularly, their binary activation and highly sparse feature maps allow accelerator architectures of extreme power efficiency, which are essentially distinguishable from DNN accelerators. Nevertheless, SNNs for DL are of the same topology as DNNs, and SynOPs (major operations) are equivalent to MAC OPs. In this regard, GPUs are frequently used to accelerate SNN computation, which can significantly accelerate SynOPs given the high memory bandwidth and peak performance. However, given their prohibitive power consumption, GPUs hardly utilize the inherent operational efficiency of SNNs. To leverage the inherent operational efficiency due to operational sparsity and binary activation, event processors (based on ad hoc event routing) need to be used to implement SNNs, which are referred to as neuromorphic processors. In this review, we mainly address event-based neuromorphic processors, which are the mainstream neuromorphic hardware. Note that hereafter neuromorphic processors indicate SNN inference accelerators without on-chip learning engine unless otherwise specified. Additionally, we refer to event-based neuromorphic processors as neuromorphic processors unless otherwise specified. Section 4.1 introduces the generic architecture of neuromorphic processors and several key working principles. Section 4.2 is dedicated to various event-routing architectures that are the key to event processors. Section 4.3 overviews various neuromorphic processors introduced to date, which are classified as 1) mix-signal and 2) digital neuromorphic processors. Section 4.4 addresses nonevent-based neuromorphic processors and compares them with event-based neuromorphic processors.

4.1. Generic Architecture of Neuromorphic Processors

Unlike DNN accelerators based on the von Neumann architecture, neuromorphic processors are standalone hardware in need of neither host CPU nor main memory. Generally, a neuromorphic processor consists of multiple neuromorphic cores and network-on-chips (NoCs) that are responsible for communication between cores as illustrated in **Figure 12**a. Each core consists of a neuron block, weight memory, event router, and event queue. The neuron block calculates the membrane potential values. The weight memory stores synaptic weights used for SynOPs. The event router sends input spikes (events) to the postsynaptic (destination) neurons. The event queue buffers input spikes temporarily before the event router sends them to the destination neurons. Note that some designs merge the weight memory with the event router as for crossbar-based event-routing architectures.[49,123,124] Spiking neurons are distributed over multiple cores that compute the membrane potentials (state variables) of their spiking neurons in parallel. The events from a given neuron are routed to its postsynaptic (fan-out) neurons through the NoCs. Despite the limited bandwidth of the NoCs, this event routing through NoCs barely causes heavy traffic because of 1) the binary activation ('0' and '1', nonspike and spike, respectively) instead of real-value activation as for DNNs and 2) the high sparsity of the feature maps.

The forward pass of an SNN for inference depends on local data only. That is, the membrane potential update in Equation (9) (based on SynOPs in Equation (12)) uses several data like the potential time-constant τ_m, spiking threshold θ, and fan-in synaptic weights w_{ij}, which are all local. Therefore, when the fan-in synaptic weights w_{ij} are placed in the same core as the postsynaptic neurons, each core depends on its own local memory without access to a main memory. This allows the multiple cores to operate independently in parallel. Thus, event data

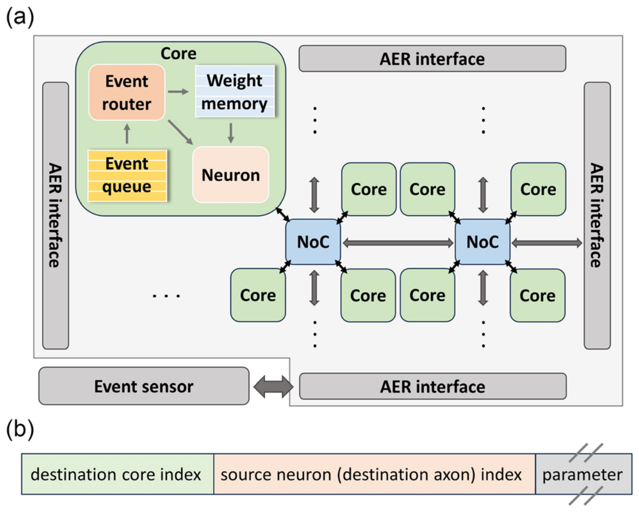


Figure 12. a) Architecture of a multicore neuromorphic processor. b) Schematic of an event-data packet for AER.

(event packets) are the only data that move between the cores through NoCs. This multicore architecture with local memory only is of excellent scalability in that the processor capacity simply scales with the number of cores without imposing additional overheads on their common hardware unlike DNN accelerators that impose additional overheads on their main memory when deploying more cores. This becomes obvious for multichip neuromorphic systems which can accommodate large SNNs by deploying many neuromorphic chips. On the other hand, the capacity of distributed on-chip local memory limits the capability of neuromorphic processors without main memory, e.g., sizes of SNNs implemented. Therefore, efficiency in memory usage is one of the key metrics.

Address event representation (AER) is a widely used protocol for event routing.[125] In AER, an event-data packet from a given presynaptic (source) neuron consists of the addresses of the source and postsynaptic (destination) neurons as illustrated in Figure 12b. This event-data packet is routed to its destination neurons through NoCs using the addresses. Upon the arrival of the event-data packet at the cores including the destination neurons, their membrane potential values are updated in parallel, i.e., SynOPs in Equation (12) are performed. That is, ad hoc SynOPs are performed by routing events in ad hoc manner instead of constructing and buffering feature (event) maps to perform synchronous SynOPs. These ad hoc SynOPs inherently skip SynOPs for zero activation, i.e., nonspike cases, and thus unnecessary computational latency and power consumption are avoided. The performance metric of a given neuromorphic processor is SynOPS (SynOPs/s), which corresponds to OPS for DNN accelerators. We define an effective performance ($\text{SynOPS}^{\text{eff}}$) that counts SynOPs for zero activation (SynOPs(0)), which is equivalent to the performance of accelerators without zero skipping like GPUs.

$$\frac{\text{SynOPs}(0)+\text{SynOPs}(1)}{\text{SynOPS}_{\text{eff}}}=\frac{\text{SynOPs}(1)}{\text{SynOPS}} \tag{14}$$

where SynOPs(1) denotes SynOPs for nonzero activation ('1'), which are actually performed. Therefore, we have

$$\text{SynOPS}_{\text{eff}}=\frac{\text{SynOPS}}{1-f_{\text{sp}}} \tag{15}$$

where the sparsity f_{sp} is given by

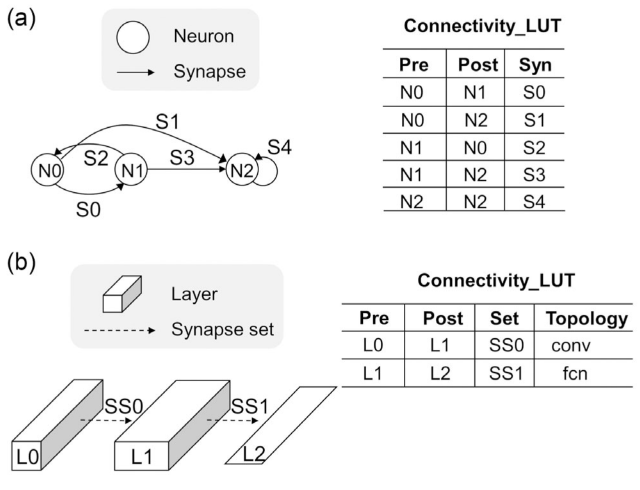
$$f_{\text{sp}}=\frac{\text{SynOPs}(0)}{(\text{SynOPs}(0)+\text{SynOPs}(1))}, \tag{16}$$

which is equivalent to Equation (11). Equation (13) highlights significant boost in effective performance by skipping SynOPs(0) for a given sparsity f_{sp}. For instance, for $f_{\text{sp}}=0.99$, $\text{SynOPS}_{\text{eff}}=100\times\text{SynOPS}$, highlighting 100× improvement in performance by skipping SynOPS(0).

4.2. Event-Routing Architectures

4.2.1. Neuron-Wise Event Routing

Although the event-routing architecture in a neuromorphic core differs for different prototypes, the architecture commonly concerns efficient event routing among neurons, which is referred to as neuron-wise event routing. A schematic of neuron-wise event routing is illustrated in **Figure 13**a, where the topology of a given SNN is defined in an LUT configured neuron-wise. Neuron-wise event routing supports full flexibility of SNN topology configuration. Crossbar-based event-routing architecture hardwires N neurons using an $N\times wN$ SRAM array for w-b weights. This architecture allows all-to-all connections. A pair of one word line and one column line (w bit lines) are dedicated to each of N neurons, and each word corresponds to a single synaptic weight. Upon event generation from a neuron, the word line dedicated to the neuron is pulled up, and all words (fan-out weights) are read simultaneously. Subsequently, each of these fan-out weights is accumulated in a dedicated PE for SynOPs. This architecture has been employed in several neuromorphic processors[49,50,124]



Connectivity_LUT

Pre	Post	Syn
N0	N1	S0
N0	N2	S1
N1	N0	S2
N1	N2	S3
N2	N2	S4

Connectivity_LUT

Pre	Post	Set	Topology
L0	L1	SS0	conv
L1	L2	SS1	fcn

Figure 13. Schematic of a) neuron-wise event-routing and b) layer-wise event-routing.

because of its simplicity. However, the quadratic increase of memory usage ($\approx N^2$) with the number of neurons N hinders it from applying to large-scale neuromorphic processors.

Hierarchical AER is a hierarchical tree-based event-routing architecture, which supports exponential expandability for a given number of event hops over the hierarchy.[126] The tree hierarchy is configured such that the leaves are dedicated to the neurons that send and receive events while the nodes on each hierarchical level relay the events from the neurons throughout the hierarchy. Given the exponential expandability of the hierarchical tree, this event-routing architecture supports the minimal number of event hops (which corresponds to the minimal latency) for event routing in large-scale SNNs.

The K-tag routing scheme[50] is a two-stage event-routing method in that the first stage routes an event to the clusters including the destination neurons and the second stage broadcasts them to their destination neurons. The key advantage of K-tag routing lies in its optimal use of event-routing memory by optimizing the number of fan-out connections for the second stage. The K-tag scheme is employed in DYNAPs[50] and Loihi.[51]

The pointer-based event-routing scheme proposed by Kornijcuk et al.[127] is an LUT-based event-routing method that aims to reduce latency for event routing and inverse lookup for spike timing-dependent plasticity -based on-chip learning. This method uses three LUTs (PTR_LUT, FOUT_LUT, and FIN_LUT). FOUT_LUT is sorted according to source neuron address such that the destination neuron addresses for a given source neuron are adjacently allocated in FOUT_LUT. FIN_LUT is sorted according to destination neuron address such that the source neuron addresses for a given destination neuron are grouped in FIN_LUT. PTR_LUT stores the ranges of the addresses in FOUT_LUT and FIN_LUT for a given neuron, so that, upon an event from a neuron, the addresses of its destination neurons in FOUT_LUT and its source neurons in FIN_LUT can be found at one cycle without sequential search of FOUT_LUT and in FIN_LUT.

4.2.2. Layer-Wise Event Routing

The neuron-wise event routing supports high flexibility in topology configuration. Yet, SNNs for DL frequently consist of basic elementary layers (see Section 2.1) instead of arbitrary connections, so that they barely need such high flexibility. When a layer type (conv or dense) and hyperparameters are fixed for a given layer, all neuron-to-neuron connections are determined, allowing us to avoid using neuron-wise configured LUTs that cause significant memory usage. That is, layer-wise rather than neuron-wise event routing can remarkably boost the efficiency in on-chip memory usage. The layer-centric event-routing architecture (LaCERA)[52] realizes this layer-wise event routing by using a tiny LUT that defines the type of each layer in a given SNN (Figure 13b**)**. Consequently, LaCERA reduces the memory usage for event routing by more than two orders of magnitude compared with the K-tag scheme as compared in **Table 1**. Further, LaCERA supports ideal weight-reuse rate for conv layers, which is barely realized in neuron-wise event-routing architectures, so that the efficiency in on-chip memory usage can be further enhanced.

Table 1. Comparison of event-routing memory usage.

Network	Memory usage		
–	Crossbar[49]	K-tag[50,51]	LaCERA[52]
cNet[a)]	42.2 Mib	4.58 Mib	15.31 Kib
LeNet[b)]	22.0 Mib	3.77 Mib	16.54 Kib

[a)]$(3 \times 32 \times 32) - 16C4@2 - 32C3 - 64C3@2 - 10C4 - 10$.
[b)]$(3 \times 32 \times 32) - 6C5 - AP2 - 16C5 - AP2 - 120C5 - 84 - 10$.

4.3. Overview on Neuromorphic Processors

Mixed-signal neuromorphic processors are implemented using both analog and digital circuits; frequently, neurons and synapses are realized using analog circuits, while network configuration and event routing are achieved using digital circuits. The advantages of analog building blocks (neurons and synapses) include their ability to mimic biological dynamics at low power consumption. However, given that analog circuits are sensitive to noise, mismatch, and power, voltage, and temperature variations, the scalability of analog neurons and synapses is somewhat limited. As such, digital neuromorphic processors are fully implemented using digital circuits. Neurons and synapses are implemented using digital logic cells. The advantages of digital circuits include excellent scalability, reliability, reconfigurability, and operation speed, allowing digital neuromorphic processors to be a solution to large-scale SNN-based DL accelerators. Section 4.3.1 and 4.3.2 introduce several neuromorphic processor designed using mixed-signal and digital circuits, respectively.

4.3.1. Mixed-Signal Neuromorphic Processors

NeuroGrid is a multichip system that works with million neurons and billions of synapses in real time.[128] This system consists of 16 mixed-signal neuromorphic chips, each of which is with a single neuromorphic core. Each chip (core) was fabricated using a 180 nm CMOS process, within a 168 mm^2 die. Each core is of the shared synapse and dendrite architecture for its area efficiency and local connectivity. The 16 cores are connected using the tree-based event-routing architecture for its higher throughput in multicasting packets than the mesh architecture. Each core consists of a 256×256 neuron array, transmitter, receiver, router, and two RAMs. The analog neuron array realizes a soma, dendrite, synapse-population, and ion-channel-population circuits. The analog neurons in the core operate in a fully parallel manner. The transmitter, receiver, and router are used to route AER packets. The transmitter encodes the coordinates of spiking neurons and sends them to the output port. The receiver decodes the coordinates of the AER packets and delivers them to the target neurons. The router multicast AER packets using the information retrieved from the memory in the core. The RAMs store the target synapse locations and configuration parameters that are shared among all neurons in the core. In NeuroGrid, a million neurons and eight billion synapses real-time operate at 2.7 W power consumption.

ROLLS[123] is a single core neuromorphic processor fabricated using a 180 nm CMOS process within a 51.4 mm^2 die.

It embodies an event-based online learning engine that allows ad hoc update of synaptic weights, which is based on analog circuit design. The learning algorithm implemented is spike-driven synaptic plasticity (SDSP).[129] The processor consists of an analog neuron circuit realizing 256 adaptive exponential IF (AdExp-IF) neurons,[130] two 256×256 arrays of trainable synapses, and synapse demultiplexer. The 256 AdExp-IF neurons work in parallel. Each of the two synapse arrays realizes short- and long-term plasticity, respectively. In the synapse arrays, weights are update using an analog circuit, while the digital circuit controls the update protocol and manages handshaking signals with AER packets. The synapse demultiplexer is to allocate one of the 256 rows in the 256×256 array to each neuron. ROLLS successfully emulate attractor networks of cortical neurons for image classification at a power consumption of 4 mW.

Moradi et al. introduced mixed-signal neuromorphic processors (DYNAPs), which support excellent reconfigurability of SNN topology.[50] DYNAP is a quad-core neuromorphic processor fabricated using a 180 nm CMOS process within a 43.79 mm^2 die. Each core consists of 256 AdExp-IF neurons and 16 k synapses that operate in parallel. The mixed-signal computing node in a core comprises an analog neuron and four synaptic dynamic circuits. The excellent reconfigurability arises from the novel *K*-tag event-routing architecture (see Section 4.2) that first routes an event to the clusters including the destination neurons and subsequently broadcasts them to the destination neurons with optimal memory usage. The *K*-tag scheme is realized using three-level routers.

4.3.2. Digital Implementation

TrueNorth is a digital multicore neuromorphic processor fabricated using a 28 nm CMOS process.[49] Globally asynchronous locally synchronous design was adopted to minimize the power consumption. Each of 4096 cores in total realizes maximal 256 LIF neurons and 64 k programmable synapses with several digital modules, including neuron, memory, scheduler, router, and controller modules. The router, scheduler, and controller modules operate asynchronously in a handshaking manner based on AER packets. The neuron module is implemented in a synchronous design that receives the generated clock from the asynchronous controller module. The 256×410-b memory stores the synaptic connections, neurons' state variables, and parameters. The scheduler and controller manage the sequence of the core operation. Event routing in TrueNorth is configured such that 1) intracore event routing uses the crossbar architecture explained in Section 4.2, and 2) mesh architecture-based event routing for intercore event routing. TrueNorth consumes 63 mW power.

Loihi is another digital multicore (128-core) neuromorphic processor designed using a 14 nm CMOS process in a 60 mm^2 die.[51] Notably, Loihi is equipped with an on-chip learning engine that supports several event-based learning algorithms. Each core realizes 1 k time-multiplexed LIF neurons and synapses whose number ranges from 114 k to 1M, depending on the user-programmable weight precision. The core implements synapse, dendrite, axon, and a learning engine modules using 2Mb distributed SRAMs in total. Upon event generation, the axon module generates AER packets containing the destination core and axon indices. The AER packets are routed to the destination cores through the NoCs. Subsequently, the synaptic fan-in states, e.g., fan-in weight and delay, are retrieved from the memory in the destination core, and event routing is completed. A barrier synchronization mechanism is employed to let the whole cores operate timestep-wise without a global clock. The authors demonstrate the performance of Loihi on various tasks.

Frenkel et al. introduced ODIN which is a digital single-core neuromorphic processor with an on-chip learning engine supporting the SDSP algorithm.[124] ODIN was fabricated using 28 nm CMOS process in a 0.086 mm^2 die. This processor supports 256 neurons conforming to the LIF model or custom phenomenological model to realize biologically plausible neural dynamics. The neuronal state variables are stored in a 4 KB SRAM memory. Similar to TrueNorth, an SRAM crossbar of 256×256 in size was adopted for event routing and weight storage, but it supports weights of 4-b precision unlike TrueNorth. The time-multiplexed learning engine for SDSP modifies the synaptic weights subject to update. ODIN successfully reproduced the dynamics of Izhikevich model[72] and demonstrated its on-chip learning capability on MNIST.

There exist several digital neuromorphic processors prototyped on field-programmable gate array (FPGA) boards. We introduce some of them as follows. Ye et al. prototyped a 32-core neuromorphic processor in a Virtex-7 FPGA, which features its novel layer-wise event-routing architecture (LaCERA introduced in Section 4.2) for convolutional SNNs (conv-SNNs). This architecture supports significant reductions in 1) event-routing memory usage compared with conventional neuron-wise routing methods, e.g., *K*-tag and crossbar architecture, and 2) weight memory usage given the ideal weight reuse supported by this architecture for conv-SNNs. Additionally, the hyperparameters for each layer is fully reconfigurable in this architecture. This processor supports the LIF and IF models, and SRM. Each core includes up to 2 k neurons whose state variables (membrane potential for LIF and membrane potential and synaptic current for SRM) are approximated using the template-scaling exponential function approximation[131,132] which allows high-precision approximations of exponential functions with minimal use of LUT memory. The 32 cores are distributed conforming to 2D mesh architecture with eight NoCs.

Yang et al. proposed a multicore neuromorphic processor of $6 \times 6 \times 6$ 3D mesh architecture, which was prototyped in an Altera Stratix III FPGA.[133] The keys to the 3D mesh architecture are a novel 3D NoCs and the router in each core, which multicasts and receives AER packets through six ports (up, down, north, east, west, and south). This processor is to real-time realize large-scale SNNs of conductance-based neuron models with high fidelity to their biological counterparts. They authors successfully reproduced the behavior of cortico-basal ganglia-thalamocortical networks real-time.

4.4. Nonevent-Based Neuromorphic Processors

We addressed event-based neuromorphic processors that allow ad hoc event routing when events are generated. This event-routing method fully leverages the generic property of SNNs, i.e., high sparsity of feature (event) maps. However, the lower the sparsity, the larger the latency for event routing because of the limited parallelism in event routing through NoCs of limited bandwidth. There exist several neuromorphic processors that build full event maps which are used for subsequent SynOPs unlike event-based neuromorphic processors, which we term nonevent-based neuromorphic processors. These processors support parallel SynOPs with multiple PEs operating in parallel as for DNN accelerators. Thus, these nonevent-based processors are advantageous over event-based processors for SNNs of low sparsity. Note that, in this case, SynOPs are identical to MAC OPs, so that the same PEs can be used for both operations.

Tianjic[134] is an example which realizes a method to leverage highly sparse feature maps by skipping SynOPs for zero events (i.e., no-spike cases). In the processor, the multiple cores communicate using AER packets through the NoCs. However, the AER packets are buffered into a memory in the core to construct the event vectors (corresponding to binary feature maps) rather than ad hoc routing of the AER packets. An input event vector to a given core undergoes parallel SynOPs using multiple PEs with the weights in the same core. SynOPs for zero activations are skipped using a zero-filtering mask before the PEs.

5. Concluding Remark and Outlook

Given that the progress in DL is regarded to continue onward, the computational complexity keeps growing. For the moment, GPUs support this DL progress as mainstream DL accelerators. This is not only because of excellent performance and memory bandwidth, flexibility, and versatility of GPUs but also because of the solidly built ecosystem of DL research based on CUDA-based DL libraries, e.g., PyTorch and TensorFlow. To bring the alternative accelerators (NPUs and CIM units) into play in the market, they should outperform GPUs on the hardware level insomuch as the users desire the change of the current solid ecosystem. However, for the moment, NPUs and CIM units are endowed with obvious disadvantages with regard to their versatility such that NPUs and CIM units are advantageous only for compute- and memory-bound models, respectively. Notably, the computational bottleneck mostly arises from the limited memory bandwidth, which is unavoidable for the conventional computer architecture. In this regard, further developments of high-bandwidth memory are the key premise of boosting the performance of DL accelerators.

Given that the DL accelerators addressed have clear pros and cons, it may be feasible to build a system equipped with various accelerators to harness their pros only for various DL tasks. An example is the heterogeneous computing using a system combining CPUs, GPUs, field-programmable gate arrays, and so forth, proposed by Intel recently. To support this computing environment, there remain several challenges including 1) minimization of data movements among different accelerators and 2) optimal task assignment to different accelerators of different run-times to maximize device utilization rate.

DL acceleration at the edge is strictly constrained by power consumption, ruling out GPUs as on-device AI accelerators. For the moment, on-device AI is an emerging market that is predicted to grow. Inference-only NPUs and CIM units at low power may be brought into play in the on-device AI market, but these low-power accelerators are endowed with strictly limited capacity of DNNs under computation. In this regard, SNN-based DL using neuromorphic processors at extremely low power may offer the largest model capacity at a given power constraint among these on-device AI accelerators.

Yet, the neuromorphic processor technology is hardly as matured as DNN-based DL accelerators, and its ecosystem has barely been solidified although there emerge several SNN libraries, e.g., snnTorch,[135] SpikingJelly,[136] and Spyx.[137] Additionally, to bring the SNN-based DL to the edge, there exist several challenges to be overcome, mainly with regard to the training efficiency and scalability of SNNs. As such, SNNs are time-dependent models like RNNs. When training, the model is commonly unrolled over time, and its parameters are optimized using backpropagation through time (BPTT).[138] Thus, the space complexity scales with the number of timesteps in use, so that the length of each training sample is strictly limited by the memory capacity of a given training platform. To cope with this issue, the online training through time (OTTT) algorithm has recently been proposed, which learns weights using the data of temporal locality unlike BPTT.[139] Yet, the scalability of OTTT to deeper and larger SNNs remains to be identified. Another challenge is the scalability of SNNs to a similar degree to DNNs. The difficulty lies in a number of hyperparameters included in SNNs, e.g., various time-constants and spiking thresholds, which should be optimized. Nonetheless, these challenges unnecessarily assure the inherent disadvantages of SNNs compared with DNNs given that they may be overcome in the near future. Certainly, the fascinating advantage (ultralow power consumption) of neuromorphic processors for on-device AI likely fuels the activities to overcome the challenges.

Acknowledgements

C.S. and C.Y. contributed equally to this work. This research was supported by the National R&D Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (grant no. NRF-2021M3F3A2A01037632). This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (grant no. RS-2023-00229689). This work was partly supported by the IITP under the artificial intelligence semiconductor support program to nurture the best talents (IITP-(2004)-RS-2023-00253914) grant funded by the Korea government (MSIT).

Conflict of Interest

The authors declare no conflict of interest.

Keywords

compute-in-memory, deep learning, deep learning accelerators, graphics processing units, neural processing units, neuromorphic processors

Received: November 13, 2023
Revised: February 6, 2024
Published online: March 21, 2024

[1] A. L. Samuel, *IBM J. Res. Dev.* **1959**, *3*, 210.
[2] M.-T. Luong, H. Pham, C. D. Manning (Preprint), arXiv:1508.04025, v1, August Submitted: **2015**.
[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 1.
[4] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur, in *Interspeech*, Vol. *2*, International Speech Communication Association (ISCA), Makuhari **2010** pp. 1045–1048.
[5] J. Hirschberg, C. D. Manning, *Science* **2015**, *349*, 261.
[6] D. Yu, L. Deng, *Automatic Speech Recognition*, Vol. *1*, Springer, New York **2016**.
[7] Y. Zhang, W. Chan, N. Jaitly, in *2017 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Piscataway, NJ **2017**, pp. 4845–4849.
[8] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al., in *Int. Conf. on Machine Learning*, JMLR, New York, NY **2016**, pp. 173–182.
[9] A. Vedaldi, B. Fulkerson, in *Proc. of the 18th ACM Int. Conf. on Multimedia*, ACM, New York, NY **2010**, pp. 1469–1472.
[10] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. Kruthiventi, R. V. Babu, *Front. Robot. AI* **2016**, *2*, 36.
[11] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Commun. ACM* **2017**, *60*, 84.
[12] A. Kendall, Y. Gal, *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1.
[13] D. Bahdanau, K. Cho, Y. Bengio (Preprint), arXiv:1409.0473, v1, Submitted: September **2014**.
[14] D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T.-Y. Liu, W.-Y. Ma, *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 1.
[15] Y. Liu, C. Niu, Z. Wang, Y. Gan, Y. Zhu, S. Sun, T. Shen, *J. Mater. Sci. Technol.* **2020**, *57* 113.
[16] Q. Yang, S. Fu, H. Wang, H. Fang, *IEEE Network* **2021**, *35*, 96.
[17] M. F. Dixon, I. Halperin, P. Bilokon, *Machine Learning in Finance*, Vol. *1170*, Springer, New York **2020**.
[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, *Nature* **2016**, *529*, 484.
[19] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, et al., *Nature* **2019**, *575*, 350.
[20] W. S. McCulloch, W. Pitts, *Bull. Math. Biophys.* **1943**, *5* 115.
[21] V. Nair, G. E. Hinton, in *Proc. of the 27th Int. Conf. on Int. Conf. on Machine Learning, ICML'10,* Omnipress, Madison, WI **2010**, pp. 807–814.
[22] A. L. Maas, A. Y. Hannun, A. Y. Ng, in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, Atlanta, June **2013**.
[23] D. Hendrycks, K. Gimpel, *Gaussian Error Linear Units (Gelus)*, Arxiv **2016**.
[24] P. Ramachandran, B. Zoph, Q. V. Le, *Searching for Activation Functions*, Arxiv **2017**.
[25] D. S. Jeong, *J. Appl. Phys.* **2018**, *124*, 152002.
[26] OpenAI (Preprint), *arXiv:2303.08774*, v1, Submitted: March **2023**.
[27] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, et al., *Llama 2: Open Foundation and Fine-Tuned Chat Models*, Arxiv **2023**.
[28] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1.
[29] K. Simonyan, A. Zisserman (Preprint), arXiv:1409.1556, v1, Submitted: September **2014**.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Piscataway, NJ **2015**.
[31] T. P. Morgan, Nvidia Rounds Out "Ampere" Lineup with Two New Accelerators **2021**, https://www.nextplatform.com/2021/04/15/nvidia-rounds-out-ampere-lineup-with-two-new-accelerators/ (accessd: April, 2021).
[32] R. Krashinsky, O. Giroux, S. Jones, N. Stam, S. Ramaswamy, Nvidia Ampere Architecture In-Depth **2020**, https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/ (accessed: May 2020).
[33] P. Alcorn, Nvidia Infuses dgx-1 with Volta, Eight v100s in a Single Chassis **2017**, https://www.tomshardware.com/news/nvidia-volta-v100-dgx-1-hgx-1,34380.html (accessed: May 2017).
[34] I. Cutress, Nvidia's DGX-2: Sixteen Tesla v100s, 30tb of NVME, Only $400k, **2018**, https://www.anandtech.com/show/12587/nvidias-dgx2-sixteen-v100-gpus-30-tb-of-nvme-only-400k (accessed: March 2018).
[35] C. Campa, C. Kawalek, H. Vo, J. Bessoudo, Defining AI Innovation with Nvidia DGX A100, **2020**, https://developer.nvidia.com/blog/defining-ai-innovation-with-dgx-a100 (accessed: May 2020).
[36] R. Smith, *Nvidia Hopper GPU Architecture and H100 Accelerator Announced: Working Smarter and Harder*, **2022**, https://www.anandtech.com/show/17327/nvidia-hopper-gpu-architecture-and-h100-accelerator-announced (accessed: March 2022).
[37] R. Smith, Nvidia Gives Jetson AGX Xaview A Trim, Announces Nano-Sized Jetson Xavier Nx **2019**, https://www.anandtech.com/show/15070/nvidia-gives-jetson-xavier-a-trim-announces-nanosized-jetson-xavier-nx (accessed: November 2019).
[38] B. Funk, Nvidia Jetson AGX Orin: The Next-Gen Platform that will Power Our AI Robot Overloads Unveiled, **2022**, https://hothardware.com/news/nvidia-jetson-agx-orin (accessed: March 2022).
[39] D. Franklin, Nvidia Jetson TX2 Delivers Twice the Intelligence to the Edge, **2017**, https://developer.nvidia.com/blog/jetson-tx2-delivers-twice-intelligence-edge/ (accessed: March 2017).
[40] B. Hill, Nvidia Unveils Ampere-Infused Drive AGX for Autonomous Cars, Isaac Robotics Platform with BMW Partnership, **2022**, https://hothardware.com/news/nvidia-drive-agx-pegasus-orin-ampere-next-gen-autonomous-cars (accessed: May 2020).
[41] R. Smith, 16gb Nvidia Tesla v100 Gets Reprieve; Remains in Production **2018**, https://www.anandtech.com/show/12809/16gb-nvidia-tesla-v100-gets-reprieve-remains-in-production (accessed: May 2018).
[42] N. C. Thompson, K. Greenewald, K. Lee, G. F. Manso (Preprint), arXiv:2007.05558, v1, Submitted: July **2020**.
[43] H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, in *2012 45th Annual IEEE/ACM Int. Symp. on Microarchitecture*, IEEE, Piscataway, NJ **2012**, pp. 449–460.
[44] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, et al., in *Proc. of the 44th Annual Int. Symp. on Computer Architecture*, **2017**, pp. 1–12.
[45] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, D. Patterson, in *2021 ACM/IEEE 48th Annual Int. Symp. on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2021**, pp. 1–14.

[46] K. J. Lee, in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning* (Eds:S. Kim, G. C. Deka)), Vol. *122,* Advances in Computers, Elsevier, Amsterdam **2021**, pp. 217–245.

[47] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, et al., in *2021 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *64*, IEEE, Piscataway, NJ **2021** pp. 350–352.

[48] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, et al., in *2022 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *65*, IEEE, Piscataway, NJ **2022**, pp. 1–3.

[49] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, D. S. Modha, *Science* **2014**, *345*, 668.

[50] S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, *IEEE Trans. Biomed. Circuits Syst.* **2017**, *12*, 106.

[51] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, *IEEE Micro* **2018**, *38*, 82.

[52] C. Ye, V. Kornijcuk, D. Yoo, J. Kim, D. S. Jeong, *Neurocomputing* **2023**, *520*, 46.

[53] G. Kim, D. S. Jeong, *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 28274.

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1.

[55] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, *Tensorflow: A System for Large-Scale Machine Learning*, Arxiv **2016**.

[56] F. C. Bauer, G. Lenz, S. Haghighatshoar, S. Sheik, *Front. Neurosci.* **2023**, *17*, 1.

[57] K. He, X. Zhang, S. Ren, J. Sun, in *The IEEE Conf. on Computer Vision and Pattern Recognition*, IEEE, Piscataway, NJ **2016**.

[58] G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberger, *Densely Connected Convolutional Networks*, Arxiv **2018**.

[59] S. Ioffe, C. Szegedy, in *Int. Conf. on Machine Learning*, PMLR, New York, NY **2015**, pp. 448–456.

[60] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, *Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, Arxiv **2017**.

[61] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Piscataway, NJ **2018**.

[62] X. Zhang, X. Zhou, M. Lin, J. Sun, *Shufflenet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*, IEEE (CVPR), New York **2017**.

[63] M. Tan, Q. V. Le, *Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks*, JMLR, New York, NY **2020**.

[64] M. Schuster, K. Paliwal, *IEEE Trans. Signal Process.* **1997**, *45*, 2673.

[65] S. Hochreiter, J. Schmidhuber, *Neural Comput.* **1997**, *9*, 1735.

[66] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*, Arxiv **2014**.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. U. Kaiser, I. Polosukhin, in *Advances in Neural Information Processing Systems* (Eds: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett) Vol. *30*. Curran Associates, Inc., Red Hook, NY **2017**.

[68] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova (Preprint), arXiv:1810.04805, v1, Submitted: October **2018**.

[69] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, et al., *Adv. Neural Inf. Process. Syst.* **2020**, *33* 1877.

[70] P. Dayan, L. F. Abbott, *Theoretical Neuroscience*, MIT Press, London **2001**.

[71] W. Gerstner, W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, Cambridge, England **2002**.

[72] E. M. Izhikevich, *IEEE Trans. Neural Netw.* **2003**, *14*, 1569.

[73] S. Williams, A. Waterman, D. Patterson, *Commun. ACM* **2009**, *52*, 65.

[74] P. Dhilleswararao, S. Boppu, M. S. Manikandan, L. R. Cenkeramaddi, *IEEE Access* **2022**.

[75] JEDEC Standards, https://www.jedec.org/standards-documents (accessed: August 2022).

[76] Nvidia a100 Tensor Core GPU, **2022**, https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet.

[77] Nvidia h100 Tensor Core GPU, **2023**, https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet.

[78] H. T. Kung, *Computer* **1982**, *15*, 37.

[79] S. Lee, S.-H. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, et al., in *2021 ACM/IEEE 48th Annual Int. Symp. on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2021**, pp. 43–56.

[80] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, et al., in *2022 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *65*, IEEE, Piscataway ,NJ **2022**, pp. 1–3.

[81] J.-H. Kim, J. Lee, J. Lee, J. Heo, J.-Y. Kim, *IEEE J. Solid-State Circuits* **2021**, *56*, 1093.

[82] H. Fujiwara, H. Mori, W.-C. Zhao, M.-C. Chuang, R. Naous, C.-K. Chuang, T. Hashizume, D. Sun, C.-F. Lee, K. Akarvardar, S. Adham, T.-L. Chou, M. E. Sinangil, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao, T.-Y. J. Chang, in *2022 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *65*, IEEE, Piscataway, NJ **2022**, pp. 1–3.

[83] C.-F. Lee, C.-H. Lu, C.-E. Lee, H. Mori, H. Fujiwara, Y.-C. Shih, T.-L. Chou, Y.-D. Chih, T.-Y. J. Chang, in *2022 IEEE Symp. on VLSI Technology and Circuits (VLSI Technology and Circuits)*, IEEE, Piscataway, NJ **2022**, pp. 24–25.

[84] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, S. Yin, in *2022 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *65*, IEEE, Piscataway, NJ **2022**, pp. 1–3.

[85] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, M. Liu, in *2023 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, IEEE, Piscataway, NJ **2023** pp. 250–252.

[86] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori, et al., in *2021 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *64*, IEEE, Piscataway, NJ **2021**, pp. 252–254.

[87] Z. Li, Z. Wang, L. Xu, Q. Dong, B. Liu, C.-I. Su, W.-T. Chu, G. Tsou, Y.-D. Chih, T.-Y. J. Chang, et al., *IEEE J. Solid-State Circuits* **2020**, *56*, 1105.

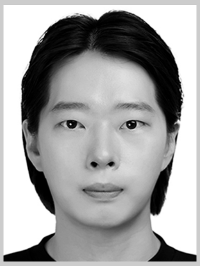
[88] A. Nag, R. Balasubramonian, V. Srikumar, R. Walker, A. Shafiee, J. P. Strachan, N. Muralimanohar, *IEEE Micro* **2018**, *38*, 41.

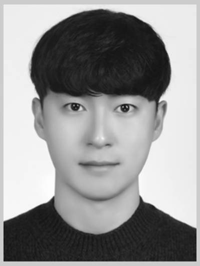
[89] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, H. Qian, *Nature* **2020**, *577*, 641.

[90] S. Yin, X. Sun, S. Yu, J. S. Seo, *IEEE Trans. Electron Devices* **2020**, *67*, 4185.

[91] J.-H. Yoon, M. Chang, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, A. Raychowdhury, in *2021 IEEE Int. Solid- State Circuits Conf. (ISSCC)*, Vol. *64*, IEEE, Piscataway, NJ **2021** pp. 404–406.

[92] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 14.
[93] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, Y. Xie, *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 27.
[94] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, Y. Lin, in *2020 ACM/IEEE 47th Annual Int. Symp. on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2020**, pp. 832–845.
[95] C. Song, J. Kim, D. S. Jeong, *Adv. Intell. Syst.* **2023**, *5*, 2200289.
[96] S. Jung, H. Lee, S. Myung, H. Kim, S. K. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, et al., *Nature* **2022**, *601*, 211.
[97] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, in *Proc. of the 42nd Annual Int. Symp. on Computer Architecture*, ACM, New York, NY **2015**, pp. 105–117.
[98] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. Pileggi, F. Franchetti, in *Proc. of the 52nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, IEEE, Piscataway, NJ **2019**, pp. 347–358.
[99] M. Zhu, T. Zhang, Z. Gu, Y. Xie, in *Proc. of the 52nd Annual IEEE/ ACM Int. Symp. on Microarchitecture*, IEEE, Piscataway, NJ **2019**, pp. 359–371.
[100] G. K. Chen, P. C. Knag, C. Tokunaga, R. K. Krishnamurthy, *IEEE J. Solid-State Circuits* **2022**, *58*, 1117.
[101] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, R. Das, in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Piscataway, NJ **2017**, pp. 481–492.
[102] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, R. Das, in *2018 ACM/IEEE 45Th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2018**, pp. 383–396.
[103] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, IEEE, Piscataway, NJ **2016**, pp. 2818–2826.
[104] D. Fujiki, S. Mahlke, R. Das, in *Proc. of the 46th Int. Symp. on Computer Architecture*, **2019**, pp. 397–410.
[105] J. E. Volder, *IRE Trans. Electron. Comput.* **1959**, *EC-8*, 330.
[106] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, K. Skadron, in *2009 IEEE Int. Symp. on Workload Characterization (IISWC)*, IEEE, Piscataway, NJ **2009**, pp. 44–54.
[107] Intel AMX, https://www.intel.com/content/www/us/en/content-details/785250/accelerate-artificial-intelligence-ai-workloads-with-intel-advanced-matrix-extensions-intel-amx.html (accessed: January 2024).
[108] A. Patel, F. Afram, S. Chen, K. Ghose, in *Proc. of the 48th Design Automation Conf.*, ACM, New York, NY **2011**, pp. 1050–1055.
[109] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, L. Benini, in *Proc. of the 25th Edition on Great Lakes Symp. on VLSI*, ACM, New York, NY **2015**, pp. 199–204.
[110] Y.-H. Chen, J. Emer, V. Sze, *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 367.
[111] Y.-H. Chen, T. Krishna, J. S. Emer, V. Sze, *IEEE J. Solid-State Circuits* **2016**, *52*, 127.
[112] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, O. Temam, *ACM SIGARCH Comput. Archit. News* **2014**, *42*, 269.
[113] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, Y. Chen, *IEEE Trans. Comput.* **2016**, *66*, 73.
[114] J. Yue, C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang, et al., in *2023 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, IEEE, Piscataway, NJ **2023**, pp. 1–3.
[115] J.-W. Su, Y.-C. Chou, R. Liu, T.-W. Liu, P.-J. Lu, P.-C. Wu, Y.-L. Chung, L.-Y. Hung, J.-S. Ren, T. Pan, et al., in *2021 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *64*, IEEE, Piscataway, NJ **2021**, pp. 250–252.
[116] K. Ueyoshi, I. A. Papistas, P. Houshmand, G. M. Sarda, V. Jain, M. Shi, Q. Zheng, S. Giraldo, P. Vrancx, J. Doevenspeck, et al., in *2022 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Vol. *65*, IEEE, Piscataway, NJ **2022**, pp. 1–3.
[117] I. A. Papistas, S. Cosemans, B. Rooseleer, J. Doevenspeck, M.-H. Na, A. Mallik, P. Debacker, D. Verkest, in *2021 IEEE Custom Integrated Circuits Conf. (CICC)*, IEEE, Piscataway, NJ **2021**, pp. 1–2.
[118] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, J. Chang, in *2020 IEEE Int. Solid-State Circuits Conf.-(ISSCC)*, IEEE, Piscataway, NJ **2020**, pp. 242–244.
[119] B. Wang, C. Xue, Z. Feng, Z. Zhang, H. Liu, L. Ren, X. Li, A. Yin, T. Xiong, Y. Xue, et al., in *2023 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, IEEE, Piscataway, NJ **2023**, pp. 134–136.
[120] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, J. P. Kulkarni, in *2021 IEEE Int. Solid- State Circuits Conf. (ISSCC)*, Vol. *64*, **2021**, pp. 248–250.
[121] S. Kim, Z. Li, S. Um, W. Jo, S. Ha, J. Lee, S. Kim, D. Han, H.-J. Yoo, in *2023 IEEE Int. Solid-State Circuits Conf. (ISSCC)*, IEEE, Piscataway, NJ **2023**, pp. 256–258.
[122] D. S. Jeong, R. Thomas, R. S. Katiyar, J. F. Scott, H. Kohlstedt, A. Petraru, C. S. Hwang, *Rep. Prog. Phys.* **2012**, *75*, 076502.
[123] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, G. Indiveri, *Front. Neurosci.* **2015**, *9* 141.
[124] C. Frenkel, M. Lefebvre, J.-D. Legat, D. Bol, *IEEE Trans. Biomed. Circuits Syst.* **2018**, *13*, 145.
[125] K. A. Boahen, *IEEE Trans. Circuits Syst. II* **2000**, *47*, 416.
[126] J. Park, T. Yu, S. Joshi, C. Maier, G. Cauwenberghs, *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *28*, 2408.
[127] V. Kornijcuk, J. Park, G. Kim, D. Kim, I. Kim, J. Kim, J. Y. Kwak, D. S. Jeong, *Adv. Mater. Technol.* **2019**, *4*, 1800345.
[128] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, K. Boahen, *Proc. IEEE* **2014**, *102*, 699.
[129] J. M. Brader, W. Senn, S. Fusi, *Neural Comput.* **2007**, *19,* 2881.
[130] R. Brette, W. Gerstner, *J. Neurophysiol.* **2005**, *94*, 3637.
[131] J. Kim, V. Kornijcuk, D. S. Jeong, in *2020 21st Int. Symp. on Quality Electronic Design (ISQED)*, IEEE, New York, Ny **2020**, pp. 358–363.
[132] J. Kim, V. Kornijcuk, C. Ye, D. S. Jeong, *IEEE Trans. Circuits Syst. I* **2021**, *68*, 350.
[133] S. Yang, J. Wang, B. Deng, C. Liu, H. Li, C. Fietkiewicz, K. A. Loparo, *IEEE Trans. Cybern.* **2018**, *49*, 2490.
[134] L. Deng, G. Wang, G. Li, S. Li, L. Liang, M. Zhu, Y. Wu, Z. Yang, Z. Zou, J. Pei, Z. Wu, X. Hu, Y. Ding, W. He, Y. Xie, L. Shi, *IEEE J. Solid-State Circuits* **2020**, *55*, 2228.
[135] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, W. D. Lu, *Proc. IEEE* **2023**, *111*.
[136] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, Y. Tian, *Sci. Adv.* **2023**, *9*, eadi1480.
[137] K. Heckel, kmheckel/spyx: v0.1.0-beta **2023**, https://doi.org/10.5281/zenodo.8241588.
[138] P. Werbos, *Proc. IEEE* **1990**, *78*, 1550.
[139] M. Xiao, Q. Meng, Z. Zhang, D. He, Z. Lin, in *Advances in Neural Information Processing Systems* (Eds:S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh), Vol. *35*, Curran Associates, Inc., Red Hook NJ **2022**, pp. 20717–20730.


Choongseok Song received his B.S. degree in electronics and information engineering from Sejong University, Seoul, South Korea, in 2020. He is currently pursuing his Ph.D. degree in materials science and engineering from Hanyang University, Seoul, South Korea. His research interests include computer architecture for deep learning acceleration based on neural processing unit and processing in memory.


ChangMin Ye received his B.S. degree in materials science and engineering from Hanyang University, Seoul, South Korea, in 2020, where he is currently pursuing his Ph.D. degree in materials science and engineering. Since 2020, he has been focusing on learning digital neuromorphic processor design.

Yonguk Sim received his B.S. degree in electronic engineering from Hanyang University, Seoul, South Korea, in 2023, where he is currently pursuing the integrated Ph.D. degree in semiconductor engineering. Since 2023, he has been focusing on NVM-based deep learning accelerator design.

Doo Seok Jeong is a professor at Hanyang University, Republic of Korea. He received his B.E. and M.E. in materials science from Seoul National University in 2002 and 2005, respectively. He received his Ph.D. degree in materials science from RWTH Aachen, Germany in 2008. He was with the Korea Institute of Science and Technology from 2008 to 2018. His research interest includes spiking neural networks for sequence learning and future prediction. Learning algorithms, spiking neural network design, and digital neuromorphic processor design are his current research focus.