PULSE: Parametric Hardware Units for Low-power Sparsity-Aware Convolution Engine

Ilkin Aliyev and Tosiron Adegbija
Department of Electrical and Computer Engineering
The University of Arizona, Tucson, AZ, USA
Email: {ilkina, tosiron}@arizona.edu

2024 IEEE International Symposium on Circuits and Systems (ISCAS) | 979-8-3503-3099-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCAS58744.2024.10558062

***Abstract*—Spiking Neural Networks (SNNs) have become popular for their more bio-realistic behavior than Artificial Neural Networks (ANNs). However, effectively leveraging the intrinsic, unstructured sparsity of SNNs in hardware is challenging, especially due to the variability in sparsity across network layers. This variability depends on several factors, including the input dataset, encoding scheme, and neuron model. Most existing SNN accelerators fail to account for the layer-specific workloads of an application (model + dataset), leading to high energy consumption. To address this, we propose a design-time parametric hardware generator that takes layer-wise sparsity and the number of processing elements as inputs and synthesizes the corresponding hardware. The proposed design compresses sparse spike trains using a priority encoder and efficiently shifts the activations across the network's layers. We demonstrate the robustness of our proposed approach by first profiling a given application's characteristics followed by performing efficient resource allocation. Results on a Xilinx Kintex FPGA (Field Programmable Gate Arrays) using MNIST, FashionMNIST, and SVHN datasets show a $3.14\times$ improvement in accelerator efficiency (FPS/W) compared to a sparsity-oblivious systolic array-based accelerator. Compared to the most recent sparsity-aware work, our solution improves efficiency by $1.72\times$.**

***Index Terms*—Sparsity-aware SNN, FPGA, parametric hardware design, neuromorphic computing.**

I. Introduction

Spiking Neural Networks (SNN) are emerging as a competitive alternative to traditional Artificial Neural Networks (ANNs) due to the biologically plausible neuron model (i.e. binary activation and multiplication-free) which operates on sparse activations. However, most studies in SNN hardware literature ignore this characteristic despite its significant impact on hardware performance. As evidence of the potential for optimization, prior work [1] improved training energy by $5.58\times$ compared to SNNs that did not exploit sparsity. Similarly, Wang et. al. [2] improved inference delay by $2.1\times$ compared to sparsity-oblivious hardware.

To fully realize the efficiency benefits of SNNs, the hardware implementations must match the SNN's computation needs. A key aspect of this involves determining the number of processing elements (PEs) for deep learning (DL) accelerators, as this significantly influences resource utilization and energy consumption. This ongoing research challenge has prompted various approaches [3], such as SCNN's configuration of 1024 Multiply-and-Accumulate (MAC) units for HD video processing [4] and Eyerissv2's exploration of diverse models with different layer shapes [5]. Additionally, it is crucial to effectively and explicitly exploit the sparsity characteristic of SNNs. This requires rethinking traditional methods used in conventional convolutional neural networks (CNNs), like the shifting window-based approach [6], [7], due to the variable sparsity across network layers, feature maps (fmaps), and within each fmap's timesteps. Addressing this complexity is essential for meeting the latency, area, and memory requirements of complex SNNs like spiking CNNs.

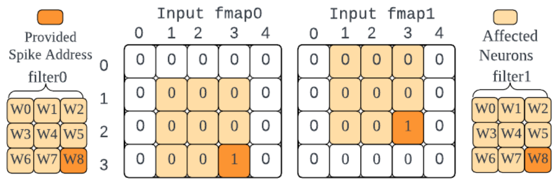

Fig. 1. Illustration of spike-based convolution operation flow. The "membrane potential" of neurons with non-zero activation values is updated along with the surrounding neurons determined by the filter weights.

Event-driven SNN hardware: Figure 1 illustrates the concept of convolution (CONV) operations on sparse activations. Unlike classical CONV operations where the filter is slid over the input activations, the spike event addresses in the input activations determine which neurons in the post-synaptic layer will be updated. We argue that by specializing hardware resource allocation to the sparsity of each layer, we can reduce excessive resource demands and optimize the efficiency of SNN accelerators. Recent studies [8]–[10] have shown reductions in both hardware resources and inference time by simply considering sparsity in the input layer in a two-layer network with only input and output layers.

Prior sparsity-aware studies have two main limitations. Firstly, works using systolic array-based hardware follow a "one size fits all" approach to sparsity exploitation across layers [11]. Consequently, while small layers under-utilize resources, leaving PEs idle and causing increased leakage power, larger layers overutilize resources causing latency overhead. Drastically unbalanced sparsity across the layers necessitates a layer-wise resource specialization. Secondly, network-on-chip (NoC)-based event-driven designs [12], [13] face substantial runtime overhead due to managing (routing and storing) large spike trains, impacting the system's energy efficiency. As a result, the non-trivial problem of mapping the model onto hardware [5] becomes even more cumbersome due to sparsity.

To alleviate these issues, we propose ***PULSE*** *(Parametric Hardware Units for Low-power Sparsity-Aware Convolution*

Algorithm 1 Even-based Convolution Loop Flow

```
 1: for each output channel do                          ▷ ▷ unrollable
 2:     for each time step do
 3:         for each input channel do
 4:             for each spike event do
 5:                 for each kernel column do           ▷ ▷ unrollable
 6:                     for each kernel row do          ▷ ▷ unrollable
 7:                         Calculate affected neuron index
 8:                         Update neuron membrane potential
                             with the corresponding filter weight
 9:         for each neuron's membrane pot. in OFM do
10:             Add filter bias to it
11:             Perform comparison & thresholding on it
12:             Apply leakage to it
```

Engine), a novel event-driven modular hardware design that accelerates sparsely active convolution operations. PULSE is a fully parametrized hardware design that can address the flexibility and efficiency needs of different spiking CNN models. We employ design time parameters to partition hardware resources across network layers such that the *layer-wise* workload (i.e., input activation sparsity, layer size) is taken into account. Implementation results on a Xilinx Kintex UltraScale FPGA (Field Programmable Gate Arrays) using three datasets—MNIST, Fashion MNIST, and Street View House Numbers (SVHN)—show that PULSE can achieve 1.73× and 3.14× better frames per second per watt (FPS/W) accelerator efficiency compared to sparsity-aware [11] and sparsity-oblivious [14] studies, respectively.

II. Related Work

To contextualize our work with the state-of-the-art, we briefly highlight two prior studies that provide sparse convolution engines. Sommer et al. [11] propose an FPGA-based SNN accelerator with at least 9 PEs, each PE handling one filter coefficient add operation, and utilized "Memory Interlacing" for parallel access to neurons' membrane potentials. Their design also supports parallelization of output channels, allowing deployment of multiple 9 PE clusters to handle various output feature maps in a given CONV layer. Although we employ output channel-wise parallelization, our approach differs by not parallelizing filter coefficients to simplify memory management (details in Section III). Mauro et al. [13] propose an ASIC implementation of a NoC-based hardware capable of handling entire CNN layers on a single chip. Each PE in their NoC handles 16 hardware neuron instances, with custom packet routing logic for output spikes. While their results show the NoC's flexibility in exploiting SNN layers' irregular dataflow, it incurs significant routing delays of the encoded spike packets and requires additional hardware logic in each PE for packet handling, increasing the hardware complexity.

III. Balancing sparse workload

The primary challenge in implementing a parallelization strategy in sparsity-aware SNN accelerators is maintaining the continuous operation of hardware units to prevent the associated power overheads of idle units. Algorithm 1 shows the convolution loop flow carried out in an event-based fashion. The algorithm also identifies the loops that can be unrolled without causing data hazards, like conflicts that arise when multiple feature maps write to the same output channel.

To identify the best parallelization strategy for event-based spiking convolution, we examine various options in Algorithm 1. Firstly, input channel-wise parallelization (lines 3 and 4) is inefficient due to fluctuating activation sparsity in each input feature map (IFM), leading to workload imbalance among the PEs, and data hazards when IFMs contribute to the same output feature map (OFM) with overlapping spike events. Secondly, while kernel weight parallelization (lines 5 and 6) avoids these issues as it involves updating independent neurons' membrane potentials (see Figure 1), it requires on-chip parallel memory access. Sommer et. al. [11] address this challenge through memory interlacing, but at the cost of hardware overheads. Lastly, output channel-wise parallelization (line 1) appears most effective, as each PE can independently process the same spike events and update its neurons.

However, output channel-wise parallelization comes at the expense of additional on-chip memory since each PE needs an OFM-sized memory to keep track of neuron membrane potentials. Furthermore, additional comparator and subtraction logic are needed for the activation phase of the leaky integrate-and-fire (LIF) neuron. The latter can be efficiently circumvented by employing low-cost OR gates instead of a full 32-bit fixed point comparison (see Section IV). We argue that the former can be addressed by further chunking the OFM and assigning a fraction of activations to each parallel PE. Once each PE completes all time steps and all spikes, it moves on to the next spatial chunk of the OFM. As a result, we can reduce the on-chip memory required for membrane potential storage, while maintaining the parallelization degree and, most notably, avoiding the hardware cost of managing interlacing.

The characteristics of the LIF neuron, used in this work, are shown in Equations 1 and 2.

$$u_j[t+1] = \beta u_j[t] + \sum_i w_{ij} s_i[t] - s_j[t]\theta \tag{1}$$

$$s_j[t] = \begin{cases} 1, & \text{if } u_j[t] > \theta \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where, $u_j[t+1]$ represents the membrane potential for previous and current time steps, β is the decay, and θ is the threshold value that the membrane potential must surpass to trigger a firing event, as indicated by a spike $s_j[t]$.

IV. PULSE Architecture

Figure 2 depicts PULSE's hardware architecture, which has two main components: *Event Control Unit (ECU)* and *Neural Core (NC)*. This design unrolls the output channels by a factor of N, defined at design time, to determine the number of *NC* instances. Within the *ECU*, the *controller* submodule implements lines 1-6 in Algorithm 1, striding through output channels by N. The *controller*'s signals are connected to all

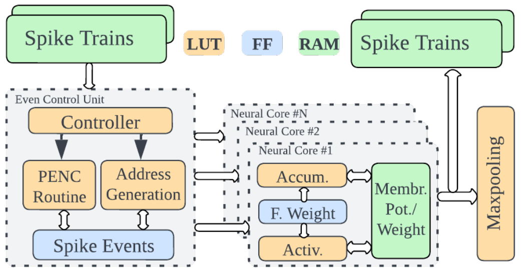

Fig. 2. The proposed layer architecture for a CONV/FC layer hardware. PENC stands for Priority Encoder routine. F. weight stands for Filter weights

parallel *NCs* and interpreted by the *NCs* as a base address. This base address is then paired with an offset parameter with which each *NC* is instantiated to calculate the OFM index. For instance, an *NC* with an offset of 2 and N=8, will process OFMs with indices 1, 9, 17, etc.

The *controller* fetches a spike train from the input Spike RAM, then initiates the Priority Encoder (PENC) routine. This routine compresses the spike train by eliminating the non-spiking (i.e., '0') bits, translating an n-bit spike train into a register array, $SpikeEvents$ (see Figure 2). PENC processes n bits in each cycle, and outputs the address of the first set bit to the $SpikeEvents$ array. The ECU's bit reset component then resets the bit value of 1 to 0 for this address in the previous cycle's version of the spike train, enabling the PENC to identify the next set bit in the spike train. After compression, the controller begins the accumulation phase, reading from the $SpikeEvents$ register set and iterating through the filter coefficients (see Figure 1). For each coefficient, the *Address Generation* routine calculates the (row, col) addresses of neurons associated with that spike event and filter sizes (i.e., for spike event at (row, col), all 9 affected neurons are from $(row - 3, col - 3)$ to (row, col)). These (row, col) signals are connected to the *NCs*. Importantly, the PENC and accumulation phases are overlapped, allowing PENC to process new spike trains simultaneously with the accumulation phase handling the previously compressed spike events.

With the (row, col) pairs provided, the *Accum* routine within each *NC* then performs the accumulation phase of the LIF neuron. It reads the membrane potential value from the Block RAM (BRAM), adds the weight coefficient to it, and writes back the result to the BRAM (line 8, Algorithm 1). Both the *Address Generation* and *Accum* routines are fully pipelined and can update one neuron per cycle. We store the filter weights of the CONV layer in flip flops (FFs). For fully-connected (FC) weight storage, we use Ultra RAMs (URAMs), which have a higher density than BRAMs, enabling a larger storage space and better energy efficiency. Unlike BRAMs, URAMs are fixed in size (e.g., each URAM tile is fixed to 72-bit width and 4K depth). Therefore, we concatenate two FC neuron weights (each weight is 32 bits) in a single row to efficiently utilize the URAM rows.

The controller, after processing all input feature maps, enables the *Activ.* unit within the *NCs* to start the LIF neuron's spiking phase. This triggers each *NC* to perform lines 9-12 in Algorithm 1 on the output features. Our design uses a simple, low-cost comparator to avoid full 32-bit fixed-point comparisons. We set the threshold value to a constant 1.0 during training and employ a Q3.29 fixed-point representation in our hardware, with the most significant bit (MSB) as the sign bit. As such, for efficiency, we only check the first 3 bits. If the MSB (bit 31) is 1, then the number is less than 1.0, while a 1 in the second or third bits indicates a value of 1.0 or greater, signaling a spiked neuron. Then, thresholding is performed by subtracting 1.0 from its membrane potential. The architecture also features max pooling, which is effective in SNNs for computer vision datasets. Its implementation on binary feature maps only requires sliding an OR gate over an $N \times N$ input area, where N is the downsampling ratio [11].

V. Implementation Results

1) Experimental Setup: We used the Xilinx Kintex® UltraScale+™ FPGA to implement the proposed hardware due to its higher logic resources than Artix and lower power than the Virtex FPGAs. We used *snntorch*[1] [15] for training our models using surrogate gradient learning [16]. We used MNIST, Fashion MNIST (FMNIST), and Street View House Numbers (SVHN) datasets as the driving applications. We set the `beta` hyperparameter to 0.15 for all datasets and training batch size to 64. We used two different networks with the following architectures: (1) 32C3-32C3-10C3-MP3-10 and (2) 32C3-P2-32C3-MP2-256-10 (where XCY stands for X filters with size $Y \times Y$ and MPZ for max pooling with size of $Z \times Z$)[2]. We evaluated our accelerator with respect to throughput and efficiency (FPS/W) in comparison to two state-of-the-art (SOTA) SNN accelerators representing sparsity-aware [11] and regular sparsity-oblivious [14] designs.

We used standard rate coding due to its ability to achieve SOTA accuracy. However, rate codes are known to require lengthy spike trains to achieve high accuracy, and population coding in the classification layer is often effective in reducing the length, at the expense of increased neuron count in the classification layer [17]. As such, each class gets mapped to 2 or more neurons (instead of a single neuron) which effectively increases the firing rate of neurons with shorter time steps. This method fits particularly well with our design scheme since the output layer is the most passive in the network. That is, although the neuron count is increased by employing a population of neurons in the classification layer, the highly sparse nature of this layer incurs negligible hardware overhead.

2) Results: Table I shows the comparison results. It also shows the sparsity range in the dataset images, indicating how sparsity varies even within the same dataset. To determine hardware configurations for each comparison, we calculate the application workload distribution using Equation 3. Let the workload of a spiking convolutional and fully-connected neural network layer (W_{CONV} and W_{FC}) be defined as:

$$W_{\text{CONV}} = F \times C_{\text{out}} \times \sum_{i=1}^{N} S_i, \quad W_{\text{FC}} = N \times S \qquad (3)$$

[1]Available online at https://github.com/jeshraghian/snntorch
[2]Pre-trained models available at https://github.com/githubofaliyev/model_zoo

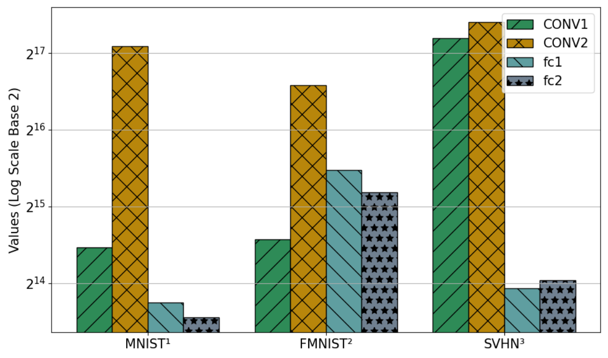

Fig. 3. Per-layer workload distribution for each application (model+dataset). The superscripts represent the network used for the dataset, shown below Table I.

where F is the number of filter coefficients (e.g., 9 for 3×3), C_out is the number of output channels, and S_i is the number of spikes for input feature map i. PULSE is equipped with network-specific performance counters and provides layer-wise execution statistics like the average number of spikes, latency cycles, etc. Therefore, we obtain S_i by first naively mapping the network onto PULSE with one neural core per layer to reveal each layer's total workload. Then, we refine the hardware to better align with the workload needs.

Figure 3 shows the layer workload distributions for each application scenario. We observe that the second CONV layer's latency dominates the overall network latency in all three applications due to 32 input features. The most notable gap can be seen in MNIST[1] because, unlike others, there is no max pooling applied to the CONV2 input. On the other hand, the weight of both CONV1 and CONV2 becomes identical in SVHN[3] because SVHN images have 3 channels. Based on this analysis, we partitioned the hardware resources to $8, 32, 4, 2$ *NCs*, $16, 32, 8, 8$, and $32, 32, 3, 3$ *NCs* for the four consecutive layers in MNIST[1], FMNIST[2], and SVHN[3], respectively. This partitioning provides the top throughput with the highest utilization.

Comparison to previous work: Compared to the sparsity-aware prior work [11], PULSE achieved $1.73\times$ efficiency improvement for MNIST. Prior work outperformed our work in throughput and latency because they utilized TTFS coding, whereas we used the Poisson-based rate coding. However, their power usage is worse due to high frequency, DSP usage, and idle PEs, leading to increased leakage power. Despite a slightly higher area footprint in our work, we achieved a power advantage due to the utilization of URAMs instead of BRAMs. Note that we report power without any power optimizations such as power/clock gating. Compared to [14], which ignored sparsity, PULSE achieved $3.14\times$ better efficiency for FMNIST[2]. For SVHN[3], which has lower sparsity (54% median sparsity), PULSE achieved $2\times$ improvement.

While PULSE outperformed prior work in efficiency in all cases, it lagged in latency. The latency degraded because each layer needed to finish processing an entire image before moving on to the next layer. Therefore, PULSE favors high-throughput and low-power scenarios. Furthermore, by employing population coding, we significantly reduced the number of time steps without degrading the model accuracy and hence balanced the intensive spiking characteristics of the rate coding for MNIST and FMNIST. For SVHN, we observed accuracy degradations below 18 time steps. Overall, PULSE's innovative hardware design improves efficiency while enabling flexibility for further performance-power tradeoffs.

TABLE I
COMPARISON TO PRIOR WORK. THE SUPERSCRIPTS REPRESENT THE NETWORKS DEPICTED BELOW THE TABLE. IN THE *platform* ROW, K. STANDS FOR KINTEX, AND Z. STANDS FOR ZYNQ.

Dataset	MNIST[1]		FMNIST[2]		SVHN[3]	
Work	PULSE	[11]	PULSE	[14]	PULSE	[14]
Latency [#cycles]	131.7K	13.3K	163K	120K	214K	121K
Through-put [FPS]	10.8K	21.4K	2907	833	2155	826
LUT	43.3K	33K	47.5K	80K	49.2	80K
FF	32.6	21	38.7K	138K	40.3	138K
BRAM	42	125	34	246	52	246
URAM	50	0	53	0	56	0
DSP	0	64	0	0	0	0
Power [W]	0.92	2.9	1.09	0.98	1.28	0.98
Efficiency [FPS/W]	12.5K	7.2K	2667	848	1684	841
Eff. Gain	$1.73\times$		$3.14\times$		$2\times$	
Input Sprs.[%]	high=95 low=83		high=76 low=68		high=78 low=51	
Input Coding.	time step=3 pop. size=500		time step=8 pop. size=600		time step=18 pop. size=400	
$\mathbf{F}_{\text{Max}}$ [MHz]	125	333	125	100	125	100
Plat-form	K. U+	Z. U+	K. U+	K. 7	K. U+	K. 7
Acc. [%]	98.5	98.2	88.32	90.04	82.68	80.24

[1]28x28-32C3-32C3-P3-10C3-10, [2]28x28-32C3-MP2-32C3-MP2-256-10, [3]32x32x3-32C3-P2-32C3-P2-256-10

VI. CONCLUSION

This paper proposes PULSE, an event-driven, *layer-wise*, and flexible hardware design for spiking neural networks. The proposed design mitigates latency overhead by employing *spike-based* processing and an output-channel-wise parallelization strategy. It also reduces power consumption by balancing hardware resources in a layer-wise manner, specializing resources to individual layers' sparsity characteristics. Implementation results on a Kintex FPGA show that PULSE achieves better efficiency (FPS/W) than both sparsity-aware and sparsity-oblivious prior works. Moreover, the paper highlights the challenges inherent in existing SNN hardware designs by analyzing the variability in the sparsity of input datasets, the length of time steps, and the impact of these factors on network latency and hardware efficiency. Although the factors affecting network latency arise primarily from training, their influence on hardware performance mirrors that of the hardware's design.

ACKNOWLEDGMENT

This work was partially supported by the Technology and Research Initiative Fund (TRIF) provided to the University of Arizona by the Arizona Board of Regents (ABOR).

References

[1] R. Yin, A. Moitra, A. Bhattacharjee, Y. Kim, and P. Panda, “Sata: Sparsity-aware training accelerator for spiking neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[2] Z. Wang, Y. Zhong, X. Cui, Y. Kuang, and Y. Wang, “A spiking neural network accelerator based on ping-pong architecture with sparse spike and weight,” in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2023.

[3] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, *et al.*, “Magnet: A modular accelerator generator for neural networks,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

[4] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[6] Z. Shen, F. Tian, J. Jiang, C. Fang, X. Xue, J. Yang, and M. Sawan, “Nbssn: A neuromorphic binary single-spike neural network for efficient edge intelligence,” in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2023.

[7] H. Liu, Y. Chen, Z. Zeng, M. Zhang, and H. Qu, “A low power and low latency fpga-based spiking neural network accelerator,” in *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2023.

[8] A. J. Leigh, M. Heidarpur, and M. Mirhassani, “Selective input sparsity in spiking neural networks for pattern classification,” in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 799–803, IEEE, 2022.

[9] A. J. Leigh, M. Heidarpur, and M. Mirhassani, “Digital hardware implementations of spiking neural networks with selective input sparsity for edge inferences in controlled image acquisition environments,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[10] I. Aliyev, K. Svoboda, and T. Adegbija, “Design space exploration of sparsity-aware application-specific spiking neural network accelerators,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[11] J. Sommer, M. A. Özkan, O. Keszocze, and J. Teich, “Efficient hardware acceleration of sparsely active convolutional spiking neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 3767–3778, 2022.

[12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[13] A. Di Mauro, A. S. Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, “Sne: an energy-proportional digital accelerator for sparse event-based convolutions,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 825–830, IEEE, 2022.

[14] W. Ye, Y. Chen, and Y. Liu, “The implementation and optimization of neuromorphic hardware for supporting spiking neural networks with mlp and cnn topologies,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 448–461, 2022.

[15] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, “Training spiking neural networks using lessons from deep learning,” *Proceedings of the IEEE*, 2023.

[16] Y. Li, Y. Guo, S. Zhang, S. Deng, Y. Hai, and S. Gu, “Differentiable spike: Rethinking gradient-descent for training spiking neural networks,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 23426–23439, 2021.

[17] Z. Pan, J. Wu, M. Zhang, H. Li, and Y. Chua, “Neural population coding for effective temporal classification,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.