SpinalFlow: An Architecture and Dataflow Tailored for Spiking Neural Networks

Surya Narayanan
School of Computing
University of Utah
Salt Lake City, USA
surya@cs.utah.edu

Karl Taht
School of Computing
University of Utah
Salt Lake City, USA
taht@cs.utah.edu

Rajeev Balasubramonian
School of Computing
University of Utah
Salt Lake City, USA
rajeev@cs.utah.edu

Edouard Giacomin
Electrical and Computer Engineering
University of Utah
Salt Lake City, USA
edouard.giacomin@utah.edu

Pierre-Emmanuel Gaillardon
Electrical and Computer Engineering
University of Utah
Salt Lake City, USA
pierre-emmanuel.gaillardon@utah.edu

***Abstract*—Spiking neural networks (SNNs) are expected to be part of the future AI portfolio, with heavy investment from industry and government, e.g., IBM TrueNorth, Intel Loihi. While Artificial Neural Network (ANN) architectures have taken large strides, few works have targeted SNN hardware efficiency. Our analysis of SNN baselines shows that at modest spike rates, SNN implementations exhibit significantly lower efficiency than accelerators for ANNs. This is primarily because SNN dataflows must consider neuron potentials for several ticks, introducing a new data structure and a new dimension to the reuse pattern. We introduce a novel SNN architecture, SpinalFlow, that processes a compressed, time-stamped, sorted sequence of input spikes. It adopts an ordering of computations such that the outputs of a network layer are also compressed, time-stamped, and sorted. All relevant computations for a neuron are performed in consecutive steps to eliminate neuron potential storage overheads. Thus, with better data reuse, we advance the energy efficiency of SNN accelerators by an order of magnitude. Even though the temporal aspect in SNNs prevents the exploitation of some reuse patterns that are more easily exploited in ANNs, at 4-bit input resolution and 90% input sparsity, SpinalFlow reduces average energy by 1.8×, compared to a 4-bit Eyeriss baseline. These improvements are seen for a range of networks and sparsity/resolution levels; SpinalFlow consumes 5× less energy and 5.4× less time than an 8-bit version of Eyeriss. We thus show that, depending on the level of observed sparsity, SNN architectures can be competitive with ANN architectures in terms of latency and energy for inference, thus lowering the barrier for practical deployment in scenarios demanding real-time learning.**

***Index Terms*—Accelerators, CNNs, SNNs**

I. Introduction

Inspired by Neuroscience, researchers have explored the potential of Spiking Neural Networks (SNNs) to achieve high prediction accuracies for various image and speech applications [2], [5], [16], [14], [55]. A spiking neuron is stateful; it maintains a potential based on previously seen inputs; as binary input spikes are received, the potential is moved up or down; a binary output spike is produced when the potential reaches a threshold. Spiking neurons mimic the operations in biological neurons, in hopes that emulating the brain will provide very high prediction accuracy at very low energy [55]. However, silicon implementations of SNNs have generally lagged behind state-of-the-art silicon implementations of ANNs. In spite of this, SNN advancements are important because of their potential benefits in specific applications. For example, a Google project shows a small-scale SNN with sparse temporal codes achieving higher accuracy than a similar-sized ANN using higher precision [10]. In the short term, SNNs are expected to be effective and useful in the following scenarios: (i) when a large/labeled training set is not available, (ii) when the inputs are expected to deviate from the training set, (iii) when continual learning [8], [43] is necessary, and (iv) to establish initial neural network weights before engaging resource-intensive training approaches [32]. In the long term, researchers need to build on the work of Comsa et al. [10] and develop new training techniques to further exploit the information content in relative spike times so that SNNs can be competitive with the best ANNs under all circumstances. The future potential of SNNs is also echoed by the many commercial projects on SNN hardware – IBM TrueNorth [2], Qualcomm Zeroth [47], Intel Loihi [12].

As Smith lays out in his FCRC'19 keynote [55], much work remains in developing SNN training methods and architectures. In theory, SNNs naturally exhibit high sparsity, i.e., they can pack the information content of an 8-bit ANN into the relative timing in a sparse spike train within a modest time window. This work attempts to realize the low energy potential of SNNs while overcoming the temporal dimension.

Unfortunately, modern SNN architectures achieve lower throughputs and higher energy per neuron, compared to ANN accelerators [13], [27], [12], [2]. This is primarily because of the temporal aspect in SNNs – inputs are received and processed across multiple ticks. Not only does this require more time, it also puts constraints on architecture dataflows and the data reuse that can be exploited. Further, the dataflow must also manage reuse for a large data structure unique to

978-1-7281-4661-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ISCA45697.2020.00038

SNNs: neuron potential.

We therefore create a baseline SNN architecture, Spiking-Eyeriss, that is modeled after a canonical ANN accelerator Eyeriss. We observe that processing SNNs (both rate-coded, and temporal-coded) in the traditional way imposes significant data movement for neuron potentials in every tick. This problem is exacerbated by the Eyeriss row-stationary dataflow, in which partial-sums (or neuron potentials) are not fully accumulated before being offloaded to its global buffer. To address this problem, we introduce a new accelerator, SpinalFlow, that processes spikes in a compressed, and chronologically sorted manner in a single time-step (like ANNs). Using row-stationary dataflow for this approach would lead to non-trivial sorting overheads; this is addressed by adapting the dataflow to use an output-stationary model. The proposed dataflow does require repeated accesses of weights from a large buffer. This large buffer reduces the compute density of SpinalFlow, relative to the ANN baseline, for some workloads. This drawback is alleviated when dealing with low resolution inputs and sparse spike trains, which is an inherent property of temporally coded SNNs. The architecture is thus designed to naturally exploit the expected high sparsity in SNN spike trains [40], [51]. Relative to the baseline ANN, SpinalFlow has simpler processing elements, but an additional hardware merge-sort unit and a large buffer to exploit weight reuse.

The main contributions of this paper are:

- An analysis of the inefficiencies in a baseline SNN design.
- A representation for spike inputs and outputs that is compressed, time-stamped, and sorted.
- An SNN architecture and dataflow that is tailored for this input/output representation and that increases reuse of neuron potential, input spikes, and weights.
- A 1.8× average energy improvement at 4-b resolution and 90% sparsity over a 4-bit version of Eyeriss, a 5× average energy improvement at 4-b resolution and 5.4× average latency improvement at a sparsity of 90% over 8-bit Eyeriss, a 1.16× average energy improvement at a sparsity of 90% over an SCNN [46] baseline, and an order of magnitude energy improvement over the baseline SNN architecture.
- The paper thus shows that SNN architectures can complete the computations required for inference in similar time and energy as an ANN architecture. This can significantly impact platforms, e.g., those requiring real-time learning, where SNNs have the potential to achieve higher accuracies than ANNs.

II. Background

Spiking Neurons

A number of projects [2], [28], [5] have attempted to implement biologically plausible neuron models in hardware. Many of these hardware projects implement neuron models that are highly simplified, but that can emulate many biologically observed neuron behaviors, e.g., the Izhikevich neurons [44]. The most popular of these simple neuron models

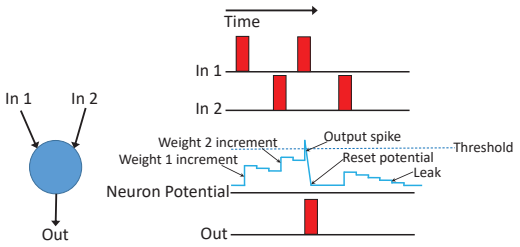

Fig. 1: *A basic 2-input LLIF spiking neuron. The figure shows how the neuron potential is incremented when input spikes are received, how a leak is subtracted when there are no input spikes, and how an output spike is produced when the potential crosses the threshold.*

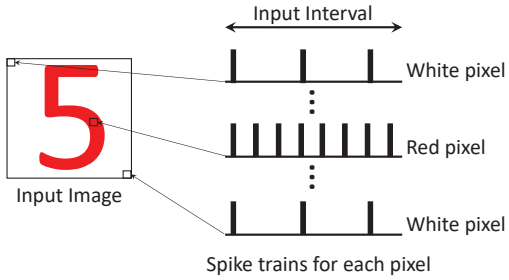

Fig. 2: *Example of an input image converted into a number of input spike trains that are fed to a rate-coded SNN (r-SNN).*

is the Integrate and Fire model (IF), shown in Figure 1. An IF neuron is stateful – it retains the value of its (membrane) potential. This *neuron potential* reflects inputs that have been received in the recent past. Inputs are received in the form of binary spikes. When a spike is received on an input, the synaptic weight for that input is added to the potential (see Figure 1). In every tick, a leak is also subtracted from the potential. When the neuron's potential eventually reaches a specified threshold, the neuron produces an output spike of its own. After the spike, the neuron potential is reset.

Spiking neurons have the potential to be hardware-efficient because inputs and outputs are binary spikes. The spiking neuron model does not require a multiplier – because the input is binary, the synaptic weight is simply added to the potential. Spikes therefore can lead to efficient communication *and* computation. This is a key feature of SNNs, but as we show later, modern SNNs have failed to exploit this advantage.

Because a neuron is designed to respond after observing spikes over time, an input (say, an image) is provided over an *input interval*, e.g., 16 ticks[1]. Figure 2 shows how each pixel of an input image is converted into a spike train that extends

[1]A *tick* is the minimum unit of time in an SNN. In one tick, a neuron evaluates its inputs, updates its potential, compares against its threshold, and produces a spike if necessary.

across an input interval. These spike trains are fed as inputs to the first layer of neurons. Prior work has primarily used *rate codes*, where for example, a red pixel value may be converted into 8 spikes in 16 ticks, while a blue pixel value may be converted into 12 spikes in the input interval. A *temporal code* converts an input pixel value into a single spike at a specific time, e.g., a red pixel value results in a single spike in the 8th tick, while a blue pixel value results in a single spike in the 12th tick. The code also includes stochasticity, e.g., a rate code may use a Poisson distribution to inject spikes [1].

We refer to rate-coded and temporally-coded SNNs as *r-SNN* and *t-SNN* respectively. Since biological neurons work at low resolution and because t-SNNs work better at low resolution, t-SNNs use short input intervals [29], [40], [55]. t-SNNs also exhibit high levels of sparsity, i.e., under 10% of all neurons produce a spike in an input interval [29], [51].

Spiking neurons are typically trained with a biologically plausible process called STDP (Spike Timing Dependent Plasticity [17]). This is an unsupervised training method where each neuron adjusts its weights based on a local process to estimate a spike's relevance [13], [52]. To increase accuracy, some recent works have also resorted to supervised backpropagation-based training for SNNs [16], [15]. The recent work of Comsa et al. [10] also employs this approach to train a t-SNN to achieve the same accuracy as an ANN.

This is a key point. *A t-SNN with its inherent lower resolution, higher sparsity, and ability to find correlations in inputs can match the accuracy of an ANN with higher-resolution operands. But the efficiency advantages of t-SNNs will not be evident until we improve its dataflow and operand reuse.*

SNN Accelerators

A variety of digital and analog SNN accelerators have been described in the literature [12], [28], [19], [42], [36], [37], [56], [2]. IBM's TrueNorth processor [2] is the most prominent example of a digital architecture for SNNs. TrueNorth is composed of many tiles, where each tile implements 256 neurons, each with 256 inputs. In every 1ms tick, a tile processes all received input spikes and sends any resulting output spikes to neurons in the next layer. Within a tick, the tile sequentially walks through every neuron in that tile and every input spike to perform several updates to each neuron potential. TrueNorth achieves relatively poor throughput and latency because of its 1 ms tick. It also does not have any parallelism within a tile. To enable an apples-to-apples comparison with state-of-the-art ANNs, we design new baseline SNN architectures that borrow some of the ANN accelerator best practices. This baseline is described in Section III-A and offers orders of magnitude better throughput and latency than TrueNorth.

ANN Accelerators

It is worth noting that in contrast to spiking neurons, artificial neurons rely on dot-product calculations, they do not retain state across consecutive inputs, and they are typically trained with supervised back-propagation based stochastic gradient descent. A number of ANN accelerator designs have been proposed in recent years [30], [53], [9], [6], [54]. In this work, we use Eyeriss [7] as a baseline because it captures many key innovations, it has been implemented in silicon, and it has many publicly available details/tools. Eyeriss uses a hierarchy of global buffers and scratchpads/registers scattered across a grid of processing elements (PEs). It uses a row-stationary dataflow for its PEs. Each PE processes a row of computation for some kernels and some input feature maps, thus exploiting reuse. The partial sums, feature maps, and kernels then move to an adjacent PE to continue the computations with high reuse. Such dataflows are a key feature in most state-of-the-art ANN accelerators, e.g., the Google TPU [25]. Many ANN accelerators also leverage sparsity in weights and/or activations [61], [46], [54]. To save energy, a multiplier/adder in Eyeriss skips its operation if either operand is zero. Architectures like SCNN [46] can also exploit ANN sparsity to reduce execution time.

III. Understanding Sources of SNN Inefficiency

A. Defining the ANN and SNN Baselines

ANN Baseline

To identify sources of SNN inefficiency, we use the Eyeriss architecture [7] as an ANN baseline. Eyeriss has the basic optimizations (dataflow, reuse, ineffectuals) that are widely adopted in both academic and commercial accelerators [25]. Much of our analysis focuses on an 8-bit version of Eyeriss, while the SNN models employ a 16-tick input interval and temporal codes with high sparsity. While the SNN design has higher sparsity and lower resolution, those advantages are inherent in the SNN design, i.e., we are not artificially introducing an accuracy/efficiency trade-off. To further understand the relative merits, we also compare SNNs (with varying resolutions) to an ANN that engages the accuracy/efficiency trade-off and operates at low resolution. Note that a 4-bit Eyeriss has the same input resolution as a 16-tick SNN.

Eyeriss has a grid of processing elements (PEs) that are fed with inputs/weights and partial sums from a global buffer. Each PE has a MAC unit and a scratchpad that stores a row of an input feature map (ifmap), a row of a filter, and partial sums (psums) for the output feature map (ofmap). Figure 3a shows the components in a single PE at 8-bit resolution. A PE can elide a computation to save energy if either input is zero. Within the 2D grid of PEs, ifmaps are shared diagonally, filters are shared horizontally, and psums are accumulated vertically. This is referred to as *Row-Stationary* dataflow [7].

SNN Baseline

Our SNN baseline, *Spiking-Eyeriss*, closely follows the Eyeriss architecture. The resulting PE, shown in Figure 3b and summarized in Table I, does not have a multiplier unit. Ifmap scratchpads only have a width of 1 bit. Weights and partial sums have the same width as the ANN baseline. After comparing the potential to the threshold, a 1-bit neuron output is produced; this 1-bit neuron output and the 8-bit neuron potential must both be saved in the global buffer or in off-chip DRAM. The 8-bit neuron potentials have to be retained for the entire input interval.

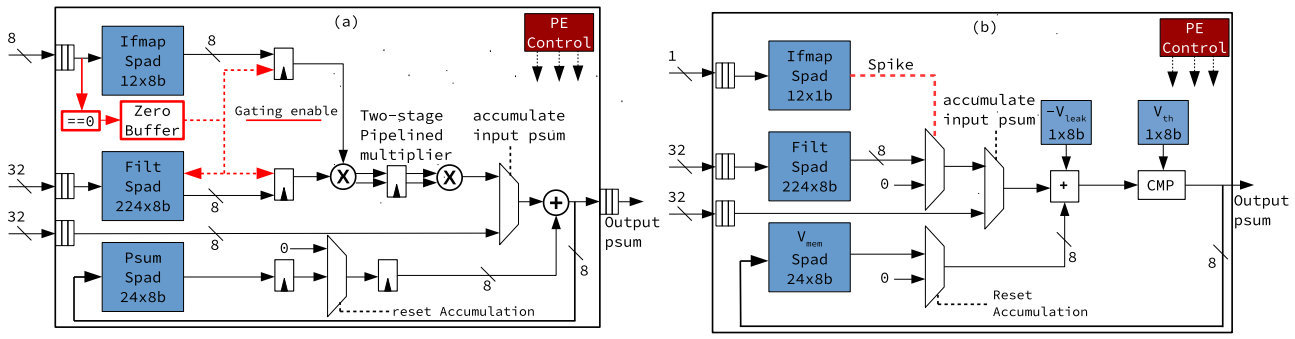

Fig. 3: *(a) PE in Eyeriss [6]. (b) PE in Spiking-Eyeriss.*

Components	Eyeriss	Spiking-Eyeriss
PE Array	12 ×14	12 ×14
ALU per PE	8-b FxP MAC	8-b FxP Add & Cmp
Filter scratchpad	224× 8-b	224× 8-b
psum scratchpad	24×8-b	24×8-b
ifmap scratchpad	12×8-b	12×1-b
Global Memory	54 KB	54 KB
Core frequency	200 MHz	200 MHz
Off-chip memory	HBM2	HBM2

TABLE I: *Parameters for our ANN (based largely on Eyeriss) and baseline SNN.*

ResNet MobileNet	33 convolutional layers, 1 FC layer 13 PWC and 13 DWC layers
STDP-Net	2 conv layered network from [29]
SC-A SC-B	3x3x64x64, 1 layer, Synthetic 3x3x512x512, 1 layer, Synthetic
DWC-A DWC-B	3x3x1x64, 1 layer, Synthetic 3x3x1x512, 1 layer, Synthetic
PWC-A PWC-B	1x1x64x64, 1 layer, Synthetic 1x1x512x512, 1 layer, Synthetic
FC-A FC-B	4096x4096, 1 layer, Synthetic 1024x1024, 1 layer, Synthetic
Sparsity	60%, 90%, 98%
Resolution	8b, 6b, 4b, 3b, 2b

TABLE II: *Workloads, degree of sparsity, and resolution. SC - Standard Conv, DWC - Depth-Wise separable Conv, PWC - Point-Wise separable Conv. The SNN network from [29] will be referred to as STDP-Net for the rest of the paper.*

The PE grid in the SNN is more efficient than in the ANN (no multipliers, ifmap scratchpads are only 1-bit wide). Even though the inputs to a layer have shrunk in size, the overall memory requirements in the SNN are higher because every neuron's potential must be retained. The architecture is agnostic to the type of data encoding used (rate or temporal). As described shortly, this SNN baseline offers significantly higher throughput/area and throughput/power than the state-of-the-art SNN architecture, TrueNorth [2].

In our analysis, we employ *Tick Batching*, where the PEs work on all ticks of the input interval for a layer before moving on to the next layer. In tick batching, the reuse distance for membrane potential is short and with appropriate tiling, the membrane potentials can be primarily accessed out of the global buffer. Meanwhile, the output spike train from a layer is likely too large to fit in the global buffer and may have to be saved in off-chip DRAM. We have analyzed other forms of batching, e.g., processing all layers before examining the next tick, and concluded that tick batching leads to overall lower data movement.

B. Evaluation Parameters

To evaluate Eyeriss and Spiking-Eyeriss, we developed an energy/performance model that captures the latencies, energy, and throughput for different networks. The model takes in the layer specifications as input, and outputs the number of accesses to different registers, scratchpads, buffers, and off-chip memory. Based on these statistics, and the average energy per operation, we calculate the energy per layer. The model uses analytical equations to capture the dataflow of Eyeriss (and Spiking-Eyeriss) based on layer dimensions and how they map to the PE array. This mapping and resource contention ultimately dictate PE utilization and performance. Note that our analysis models the zero-gating technique of Eyeriss where filter scratchpad and ALUs are gated when a zero-valued input activation is encountered.

Table II summarizes the evaluated networks and the input image sizes. We consider a range of synthetic single-layer networks to understand how the architectures impact each type of network topology. We also consider three full-fledged networks [21], [22], [29] (ResNet, MobileNet, and STDP-Net) that incorporate a number of varied layers. Previous research [18], [23] has shown that the visual cortex is organized as a cascade of simple and complex cells structured similar to a CNN.

We implement and synthesize the processing elements of both Eyeriss and Spiking-Eyeriss using 28 nm FDSOI technology node. First, we modeled the behavior of the PEs in verilog and synthesized it using Synopsys Design Compiler. We then used Innovus for the backend flow in order to get accurate post layout metrics (area, delay and power) by taking the parasitics into account. To model the global buffer, we use CACTI [41] and scale the output from 32 nm to 28 nm technology. To reduce the number of variables in our study,

we do not attempt memory compression, but simply assume a low-energy HBM-like memory interface at 4 pJ/bit [45]. Based on the above methodology, we calculate an average energy per operation for all components. This is combined with the number of operations for each component to generate the overall energy consumption for each layer. Table I has details on each accelerator component. The primary metrics, while keeping area roughly the same, are: energy per inference, and latency per inference.

C. Analysis of Spiking Eyeriss

We first try to estimate the efficiency gap between our baseline SNN and ANN. We consider the impact of rate coding (r-SNN), temporal coding (t-SNN), sparsity, and resolution on SNN energy. Figures 4 and 5 show the energy consumed by r-SNN and t-SNN respectively on Spiking-Eyeriss. All data points are normalized against an ANN operating at resolution of 8 bits and its typical activation sparsity of 60%, i.e., 60% of activations are zero.

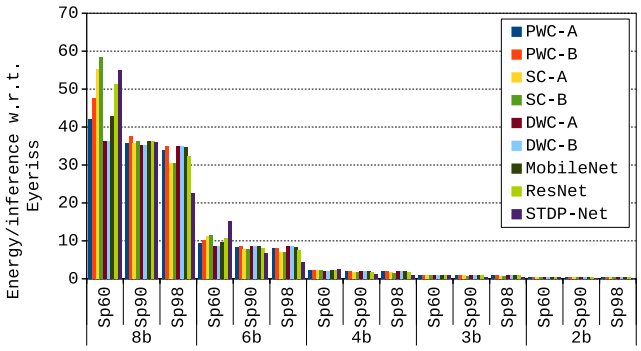

Fig. 4: *Energy consumed per inference by r-SNN on Spiking-Eyeriss normalized to Eyeriss. Sp60, Sp90, and Sp98 refers to 60%, 90%, and 98% sparsity respectively.*

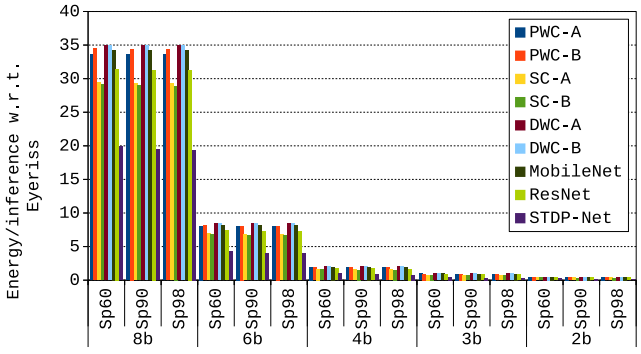

Fig. 5: *Energy consumed per inference by t-SNN on Spiking-Eyeriss normalized to Eyeriss.*

At high resolution and low sparsity, Spiking-Eyeriss consumes an order of magnitude more energy than Eyeriss. This is because of the need to repeatedly update neuron potential and fetch a weight multiple times across the input interval.

For sparse inputs, Spiking-Eyeriss and Eyeriss can avoid filter reads and partial-sum updates. Because t-SNNs encode non-zero inputs with a single spike, they consume less energy than r-SNNs. The difference is 1.63×, 1.03×, 1.08×, and 1.004× at 8bSp60, 8bSp98, 2bSp60, and 2bSp98 respectively, i.e., as one might expect, the gap shrinks when spike activity is low. More noteworthy is the "crossover point", when the SNN energy falls below that of the baseline Eyeriss. For r-SNN and t-SNN, this happens at 3bSp90 and 3bSp60 respectively. This quantifies the sparsity and resolution required for an SNN to overcome its inherent disadvantage of managing neuron potentials across many ticks and spikes.

Summary. The Spiking Eyeriss baseline has a peak throughput/area of 70 $GOps/s/mm^2$, which is 519× higher than that of TrueNorth. Truenorth is a design optimized for high levels of spike sparsity, and therefore low leakage and low energy per neuron, so its peak throughput/watt is comparable to that of Spiking Eyeriss with temporal codes and 90% sparsity. In spite of this throughput advancement relative to TrueNorth, there is a wide gap between the ANN and SNN baselines. *The performance gap (16×) is because of the need to process all (16) ticks in the input interval.* In terms of energy, we see that temporal coding is clearly superior. While a large fraction of prior SNN work has focused on rate coding, we note here that algorithmic advances in temporal coding are required so that its inherent energy efficiency can be leveraged. For the rest of the study, we will focus on the t-SNN with tick batching. However, *this SNN baseline with 90% sparsity and 16-tick intervals consumes 2× more energy per inference than the ANN because of its repeated accesses to neuron potential and filter weights*. We next devise techniques to shrink this gap.

IV. Proposed SNN Architecture: SpinalFlow

Our analysis has shown that the key drawback in our baseline SNN is the need to iteratively process (say) 16 ticks for every input interval, which in turn takes more time and requires many accesses to the memory hierarchy to update neuron potential and read filter weights. While the row-stationary dataflow of Eyeriss is highly efficient for ANNs, we must devise a new dataflow that caters to the temporal aspects and new reuse patterns exhibited by SNNs. We refer to this new architecture as *SpinalFlow*.

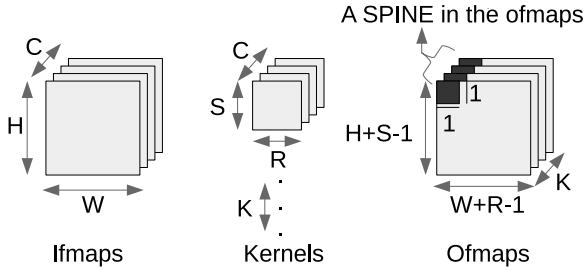

Fig. 6: *A spine in a CONV layer.*

Terminology. Before we start, Figure 6 is a quick summary of the terms we'll use in our description. We first discuss the

operations in a CONV layer. Figure 6 shows a layer with C ifmaps each of dimension H×W. Each of the K R×S×C kernels is convolved with the entire ifmap to produce K ofmaps. When the first kernel is applied to the first receptive field, i.e., the first R×S×C grid in the ifmap, the first neuron in the first ofmap is produced (the darker region in the 1st ofmap). Likewise, when K kernels are applied to this same grid of inputs, the 1st neuron in all the ofmaps are generated as shown by the dark region in Figure 6. We refer to these K neurons as a *spine*. In our discussions, we assume the value of K to be 128. The computation is ordered such that we produce the first spine of the ofmaps, followed by a shift of the input receptive field to produce the second spine, and so on.

Hardware Organization. We use an array of 128 PEs. Each PE has an accumulator (register and adder) and a comparator. Each PE is responsible for producing a neuron in an ofmap spine. Figure 7 shows the first step, where the PEs are responsible for the first spine of the ofmaps; PE-1 produces the first neuron for ofmap-1, PE-2 produces the first neuron for ofmap-2, and so on. Similar to our baselines, these PEs are fed by a global buffer that stores kernel weights. The PEs use a form of output stationary dataflow, i.e., PE-1 is dedicated to work on the first neuron for ofmap-1 till all its inputs are processed. Over the next many cycles, the PEs will receive all the spikes in their input receptive field for their input interval. The *Input Buffer* provides these input spikes. The example in Figure 7 shows that the input spikes are chronologically sorted: $< 1, 17 >, < 2, 1926 >, < 3, 75 >, < 3, 460 > \ldots$, i.e., input 17 has a spike in tick-1, input 1926 has a spike in tick-2, input 75 has a spike in tick-3. Note that in our example, the receptive field has a size of 2K, and every neuron in that receptive field can only spike at most once in its input interval (temporal code), so the input buffer can have up to 2K entries. 128 PEs also require that the global buffer feed 128 different weight values, therefore demanding a wider bus than in the baseline. We later factor this in our evaluation.

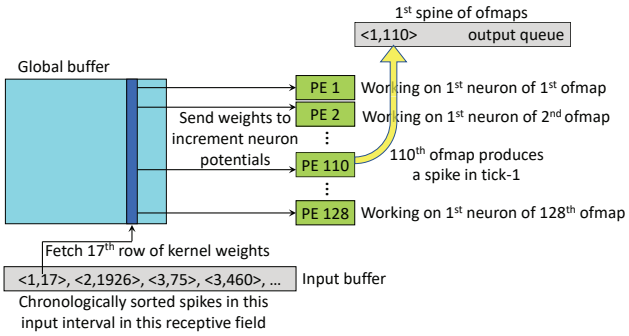

Fig. 7: *Example: Step1, Cycle 1.*

Step 1: Cycle 1. Step 1 produces the first spine in the ofmaps; producing this first spine can take up to 2K cycles. In the first cycle (shown in Figure 7), we examine the first entry in the input buffer. It represents a spike in input 17 in tick 1. Each of the PEs' neuron potential must be incremented by their corresponding kernel weight. We therefore read a row of 128 weights from the global buffer, corresponding to the 17th entry of all 128 kernels. Each PE receives one of these 128 weights and the weight is added to the neuron potential. The neuron potential is compared to its threshold. In our example, PE-110 has exceeded its threshold in tick-1, so it produces a spike $< 1, 110 >$ that is placed in its output buffer. After producing its spike, PE-110 idles for the rest of the input interval because a neuron can only produce one spike in its input interval.

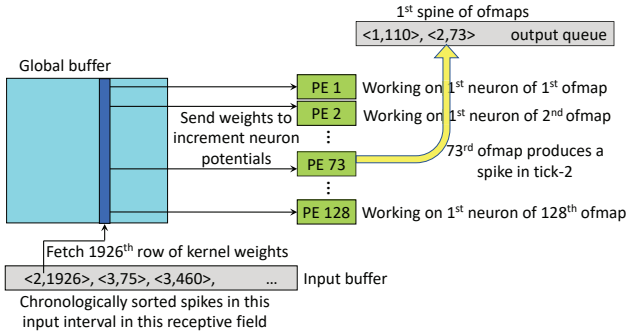

Fig. 8: *Example: Step1, Cycle 2. Step 1 continues until all receptive field entries (up to 2K) have been processed.*

Step 1: Cycle 2. In the next cycle, the next spike in the input buffer is processed (shown in Figure 8). This happens to be input 1926 spiking in tick 2. The row of 128 weights corresponding to input 1926 are read from the global buffer and fed to the PEs. Another set of neuron potential increments is performed at the PEs. PE-73 produces a spike and idles. The output buffer is appended with this new spike at tick-2: $< 2, 73 >$. We see that the spikes in the output buffer are naturally sorted, i.e., the first spine of the ofmaps is represented as a list of chronologically sorted spikes.

End of Step 1. The process repeats for up to 2K cycles until all spikes in the input buffer have been processed. Since some of the neurons in the previous layer may not have spiked in their input interval, the actual processing time can be variable and much less than the worst-case 2K cycles. In our evaluation, we assume the worst case, and leave the exploitation of activation sparsity for future work. The output buffer now contains up to 128 chronologically sorted output spikes, corresponding to a spine of the ofmaps. In practice, each spine will have less than 128 entries (since several neurons may never spike in an input interval). This spine is then written into the global buffer (see Figure 10) and will be used later as input to the next convolutional layer.

Starting Step 2. We are now ready to move to step 2, where the PEs are responsible for computing the 2nd spine of their ofmaps. The PEs reset their neuron potentials to zero. Before we start step 2, we must shift the receptive field and create a new input buffer with sorted spikes within the new receptive field. Note that the previous layer produced sorted spines of its ofmaps, which now serve as sorted ifmap spines for the current layer. To create the sorted receptive field, we must first read 16 of these ifmap spines from the global buffer into 16 ifmap spine buffers. As shown in Figure 10, these 16 pre-

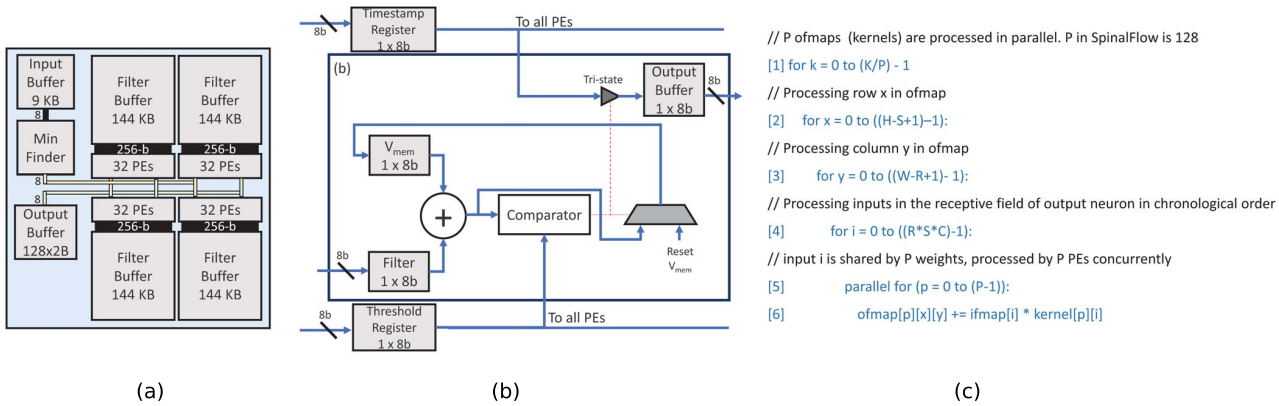

Fig. 9: *(a) SpinalFlow architecture. (b) SpinalFlow PE details. (c) Pseudocode of SpinalFlow dataflow*

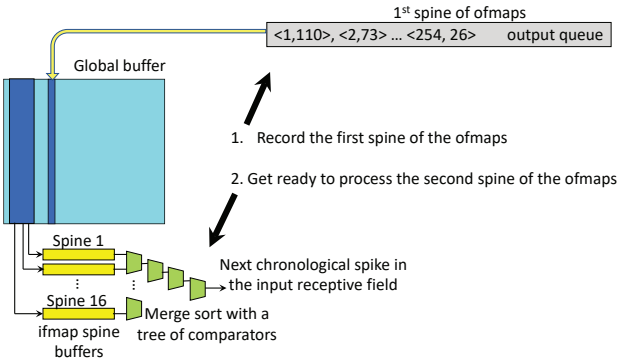

Fig. 10: *Example: End of Step 1 and set-up before Step 2.*

sorted 128-entry spines can be merge-sorted to produce the sorted 2K entries that represent the input receptive field. The 16 ifmap spine buffers and the merge-sort unit have replaced the conceptual sorted input buffer that we showed in earlier figures. The merge-sort unit is simply a tree of comparators that, in every cycle, picks the smallest entry among the heads of each ifmap spine buffer. Depending on the spine that produced that entry, an offset is added so the correct row of weights is accessed. To initiate every step, 16 cycles are required to populate the ifmap spine buffers. This is a small overhead for convolutions since the step requires hundreds of cycles of computations.

Summary. With this spine-oriented output-stationary dataflow, the proposed *SpinalFlow* architecture no longer suffers from the drawbacks in our baselines. Because we are using temporal codes and because we sequentially walk through time-stamped spikes, we need exactly as many computations as the ANN baseline, i.e., we are no longer penalized by the multi-tick input interval. Creating the compact sorted list of time-stamped spikes is trivial because of how spikes are produced by the previous layer. The architecture can yield speedups with activation sparsity with zero change, while for a similar performance effect, an ANN requires more invasive changes [46], [3]. By using an output stationary dataflow, a neuron is mapped to its PE for the entire input interval. The PE initializes its neuron potential accumulator to zero, increments it as spikes are received, produces a spike when the threshold is crossed, and discards the neuron potential before moving on to the next neuron. We are thus eliminating separate storage and data movement for neuron potential. Note that our dataflow focuses on maximizing neuron potential reuse and parallelism across a spine because of the need to sequentially process each tick; Eyeriss on the other hand optimizes a combination of reuse of inputs, kernels, partial sums.

Hardware Details. We observed that provisioning SpinalFlow with as many resources as Eyeriss led to a sub-optimal design. We therefore provide as many resources as required for the common case observed in our dataflow.

We use 128 PEs in our design because the number of feature maps per layer in large convolutional networks is often a multiple of 128. The overall architecture of SpinalFlow is shown in Figure 9a, and the details of one PE are shown in Figure 9b. The pseudo-code for our dataflow is shown in figure 9c. Each PE in our design is much simpler than the PE in Eyeriss. Since we are no longer processing an entire row at a time (the row-stationary dataflow of Eyeriss), the PE does not require large scratchpads. Such scratchpads occupy half the core area in Eyeriss, so this is a significant saving.

The SpinalFlow global buffer from earlier figures is split into a *Filter Buffer* and *Input Buffer*. While Eyeriss retains most of its weights in scratchpads, SpinalFlow retains its weights in a Filter Buffer. This buffer has to store all the weights in a receptive field for several filters because any of those weights may be required in a step. To accommodate some of the large convolutions in our workloads, we employ a 576 KB Filter Buffer organized into 32 banks, with a row width of 128 bytes (providing 128 1-byte weights at a time). In order to feed weights to 128 PEs in a cycle, the Filter Buffer needs an output bus width of 1024 bits. Depending on the type of layer being executed, bank assignment for weights and feature maps can be configured.

The inputs are received from a 9 KB Input Buffer, which stores the neuron ids and spike times for multiple spines.

Components	Eyeriss-1K 8b(4b)	SpinalFlow 8b(4b)
PEs	168	128
ALU/PE	8b (4b) MAC	8b(4b) Add, Cmp
Filt scratchpad	224× 8b (4b)	1× 8b (4b)
psum/V_{mem} spad	24×8b (4b)	1×8b (4b)
ifmap scratchpad	12×8b (4b)	1×8b (4b) (shared)
Global Buffer	54 (27) KB	585 (292.5) KB
GLB bus-width	448(224)-psum, 448(224)-filt,112(56)-ifmap	1024(512)-filt, 8(4)-spike
Core frequency	200 MHz	200 MHz
DRAM B/W	30 GB/sec	30 GB/sec
PEs A/P	0.353(0.1412)/515.5	0.024(0.012)/51.5
Min find A/P	-	0.002(0.00092)/1
Inp Buff A/P	-	0.069(0.0088)/4.3
GLB/FB A/P	0.715(0.21)/48.7	1.99(0.78)/105.6
Total A/P	1.068(0.35)/564.2	2.09(0.801)/162.4

TABLE III: *Architecture specifications of SpinalFlow and Eyeriss-1K. FB- Filter Buffer, GLB - Global Buffer, B/W - Bandwidth, A - Area in mm^2, P - Power in mW*

The spines for the input receptive field are fed to a *Min Finder* circuit (a tree of comparators) that identifies the next chronological spike and uses that neuron id to read a row of weights from the Filter Buffer. The PE array output is marshalled into an output queue that is eventually written to off-chip memory.

To evaluate the power and area of the processing elements and Min Finder, we adopt the same synthesis and SPICE methodology described in Section III-B. To model the Input and Filter SRAM buffers, we use CACTI [41]. Area and power estimates are summarized in Table III. We do not add the other exotic SNN features that can be found in TrueNorth (leak, stochasticity, various operation modes). We leave this for future work and note that the PEs account for a small fraction of chip area. Note that SpinalFlow seamlessly handles sparsity, which is an important feature in SNNs, i.e., a neuron that doesn't spike does not consume any resource bandwidth.

Other Networks. For small networks, where an input spike is seen by fewer than 128 neurons in the next layer, the PEs will be under-utilized. This is the uncommon case in our workloads. For larger networks, the computation has to be decomposed to work on 128 output feature maps at a time. The filter buffer has been sized large enough to accommodate all weights for 128 filters in our large convolutional layers. Once the filter buffer is loaded, it is reused several times to completely process the corresponding output feature maps.

The demands of a fully-connected network are different. Typically, the input receptive field is large. The spikes in this receptive field have to be chronologically sorted beforehand, with potentially multiple hierarchical passes over the Min-Finder circuit (which can only handle 16 128-entry spines at a time). The sorted list is then reused for a large set of output neurons. At a time, the PEs can process 128 output neurons. The entire input spike train for these 128 output neurons is processed before we move to the next 128 output neurons. Similar to most accelerators like Eyeriss and TPU, a fully-connected layer exhibits no weight reuse and is typically limited by the memory bandwidth required to fetch weights. Based on the input spike, a set of weights is fetched from memory, fed to the PEs, and then discarded. The only way to improve weight reuse and PE utilization is with batching, e.g., process 100 images at a time. In an SNN, such image batching is only effective if all the weights for the layer can be retained on the chip at a time (since each image in the batch has to fetch weights corresponding to its next input spike). We evaluate this in the next section.

V. Results

A. SpinalFlow vs. Eyeriss.

Energy Comparison.

Figure 11 shows the energy per inference of SpinalFlow for synthetic conv layers normalized to Eyeriss. The early analysis assumes 8-b resolution and 60% activation sparsity for Eyeriss; we later also consider lower-resolution versions of Eyeriss. Along the X-axis, we vary SNN sparsity and resolution for SNN activations and weights. Even at 8b resolution and 60% sparsity, for most synthetic workloads, SpinalFlow consumes less energy than Eyeriss. This is mainly because of the way SpinalFlow handles sparsity. Figure 12 shows the energy breakdown of different components in Eyeriss and SpinalFlow. The filter buffer and scratchpads are the dominant energy contributors in SpinalFlow and Eyeriss respectively. In SpinalFlow, no access is made to the filter buffer (which contributes 88% of total energy) whenever a zero-valued activation is encountered. Due to this, energy of SpinalFlow scales well with sparsity. Eyeriss on the other hand has to access its GLB and ifmap-spad (together contribute 44% of total energy), irrespective of activation sparsity. Therefore, the gap between SpinalFlow and Eyeriss grows as sparsity increases. Figure 11 also shows that SpinalFlow is sub-optimal when handling DWC layers. This is because ofmaps in DWC do not share inputs, so the 128-wide PEs and buffer fetches are severely under-utilized.

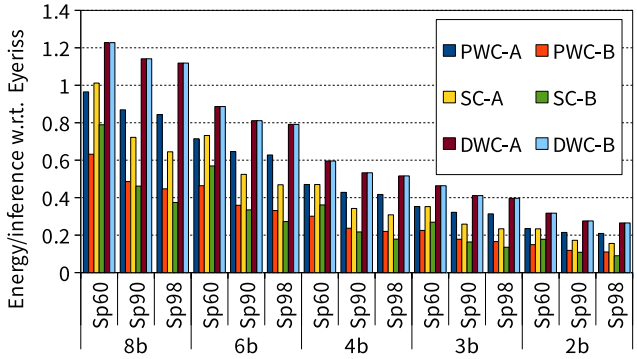

Fig. 11: *Energy/inference of SpinalFlow normalized to an 8-bit Eyeriss with 60% sparsity. Sp60, Sp90, and Sp98 refers to 60%, 90% and 98% sparsity for the SNN.*

Figure 13 shows the energy/inference of SpinalFlow relative to Eyeriss for our three full workloads. MobileNet is a combination of 13 DWC and 13 PWC layers. Even though SpinalFlow is inefficient at processing DWC layers at high

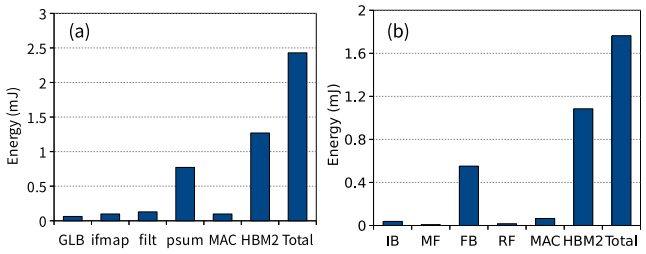

Fig. 12: *Energy/inference of ResNet at 8b resolution and 60% sparsity. (a) On Eyeriss, (b) On SpinalFlow. ifmap, filt and psum refers to corresponding scratch-pads in Eyeriss PE.*

resolution, it is overall more energy-efficient than Eyeriss for MobileNet because the DWC layers account for only 3% of execution time. The energy savings are generally higher for the other two workloads. Unlike Spiking-Eyeriss, SpinalFlow is more energy-efficient than Eyeriss at nearly all evaluated sparsity/resolution points. At 4-bit resolution and 90% sparsity, on average for the three full workloads, SpinalFlow consumes 5× less energy than the Eyeriss baseline.

Note that SNNs naturally exhibit high sparsity [51]. Prior work [29] shows that t-SNNs trained with STDP can achieve significantly higher sparsity at lower input resolutions than ANNs. While ANNs are unlikely to exhibit higher levels of sparsity than that already shown in prior work and assumed in our ANN baseline, ANNs can certainly operate at lower resolution with lower accuracy (discussed in Section VI-B). We next evaluate how SpinalFlow energy compares against Eyeriss at lower resolutions.

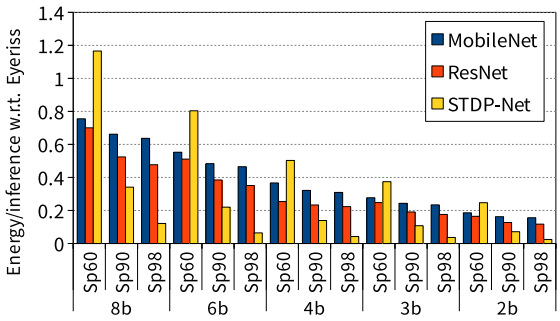

Fig. 13: *Energy per inference for SpinalFlow for full workloads, normalized to 8bSp60 Eyeriss.*

Effect of Low Resolution.

Figure 14 plots the energy per inference of our synthetic workloads on SpinalFlow – unlike earlier graphs that normalize against an 8-bit Eyeriss baseline, the data here is normalized against an Eyeriss baseline with the same resolution as SpinalFlow. The ANN sparsity is 60% throughout. In general, the SpinalFlow improvement is a little lower at lower resolutions – note how the left to right trend is slightly increasing in Figure 14, whereas it was clearly decreasing in Figure 11. This is primarily because the flip-flops in the baseline Eyeriss PE scale down better than the SRAM filter buffer in SpinalFlow. This pattern is also oberved with full workloads shown in Figure 15. At 4-bit resolution and 90% sparsity, on average, SpinalFlow consumes 1.8× less energy than a 4-bit Eyeriss baseline.

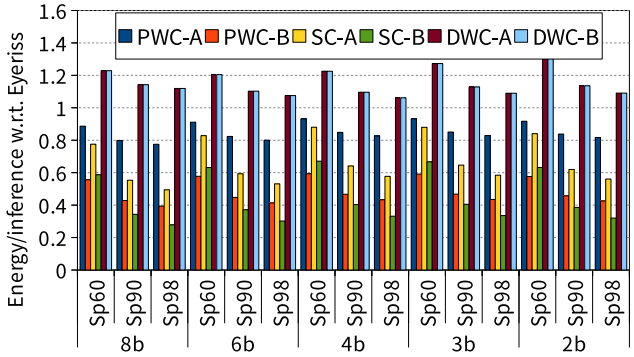

Fig. 14: *Energy/inference of SpinalFlow, normalized to an Eyeriss baseline with the same resolution as SpinalFlow. Note that sparsity of Eyeriss is fixed at 60%.*

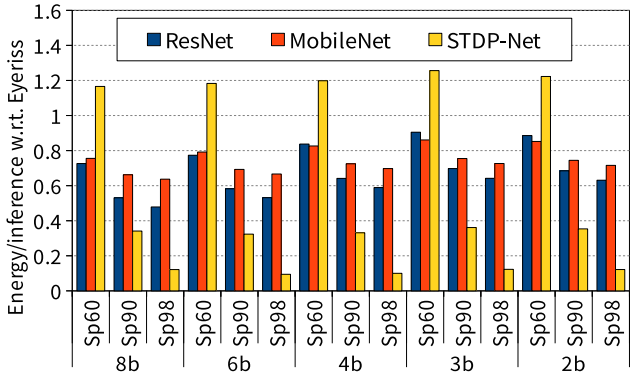

Fig. 15: *Energy/inference of SpinalFlow for full workloads, normalized to an Eyeriss baseline with the same resolution as SpinalFlow.*

Latency Comparison. Latency/inference of SpinalFlow normalized to Eyeriss (8-b resolution and 60% activation sparsity) is shown in Figure 16. We model two versions of Eyeriss, one with 1K global buffer wires (similar to SpinalFlow) and another with 72 (similar to original Eyeriss). Note that in SpinalFlow, latency changes only with the degree of sparsity, and not with resolution. While DWC is an exception because of its low utilization, the other workloads in SpinalFlow are competitive with Eyeriss at 0% sparsity because they have comparable compute and utilization. At high sparsity levels, SpinalFlow is orders of magnitude faster than Eyeriss because the execution time is a function of the spike train size; at 90% sparsity, the speedup is 5.4× on average for our three full workloads. When dealing with sparse inputs, our baseline Eyeriss already saves energy by gating the ALU, but it does

not save time by jumping to the next computation. Accelerators like SCNN [46] are able to save time when dealing with sparse inputs. SCNN adds index generation logic and a crossbar network to achieve this and offers a 2.7× performance improvement for the typical sparsity observed in ANNs. Thus, even with a better baseline like SCNN, SpinalFlow offers a significant speedup [51], [29].

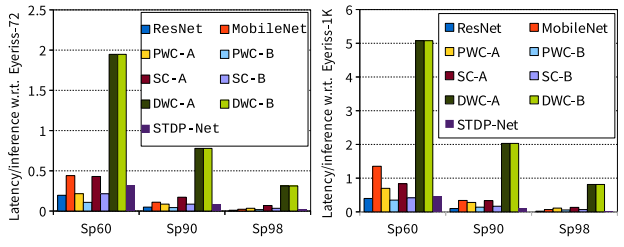

Fig. 16: *Latency per inference of SpinalFlow with respect to Eyeriss (a) with 72 GLB links and (b) with 1024 GLB links.*

B. SpinalFlow vs. Spiking-Eyeriss

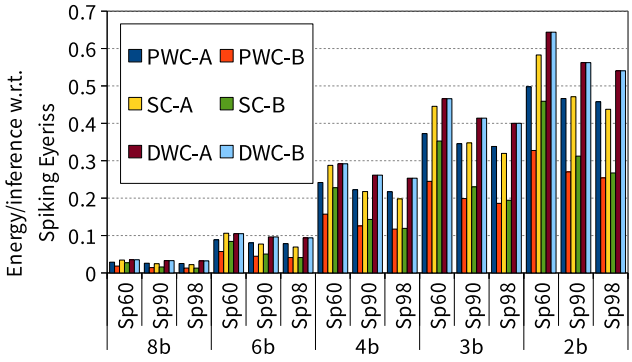

Fig. 17: *SpinalFlow energy per inference normalized to Spiking-Eyeriss for synthetic conv workloads.*

Figure 17 shows the energy per inference of SpinalFlow normalized to that of Spiking-Eyeriss at corresponding resolution and degree of sparsity. As both architectures are executing t-SNNs, the computation overhead will be similar for all design points. Recall that unlike SpinalFlow, Spiking-Eyeriss processes spikes tick-by-tick, and incurs significant off-chip and GLB overhead. 70% of the on-chip energy and 64% of the total energy of t-SNNs at 8b resolution is due to GLB accesses and off-chip accesses respectively. This results in Spiking-Eyeriss consuming an average of 35× more energy than SpinalFlow at 8b resolution. Once input resolution is decreased, the overhead of off-chip and GLB accesses reduce significantly and hence the improvement of SpinalFlow over Spiking-Eyeriss reduces as well. For similar reasons, the relative efficiency of SpinalFlow improves at higher degrees of sparsity. Figure 18 shows a similar trend on our three workloads. At 4-bit resolution and 90% sparsity, all three workloads on SpinalFlow consume roughly 5× lower energy than Spiking-Eyeriss. Spiking-Eyeriss processes inputs tick-by-tick, and hence is $2^{resolution}$ times slower than Eyeriss. It is therefore multiple orders of magnitude slower than SpinalFlow.

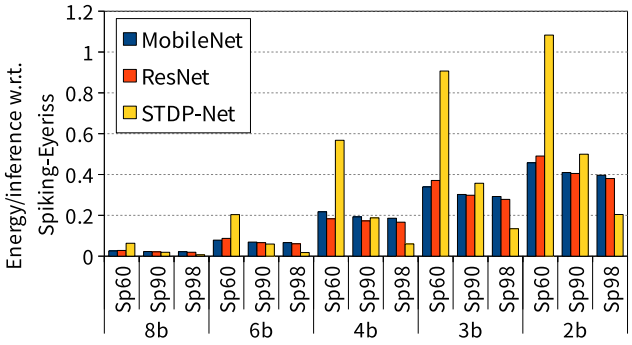

Fig. 18: *SpinalFlow energy per inference normalized to Spiking-Eyeriss for full network workloads*

C. Fully-Connected Layers

For fully-connected networks with a batch size of 1, the execution is entirely dominated by the bottleneck in fetching weights from DRAM, which accounts for 90% of the total system energy in both SpinalFlow and Eyeriss. If we assume that 200 inputs are batched, then SpinalFlow is an order of magnitude more efficient than baseline Eyeriss. This is because baseline Eyeriss has a relatively small on-chip storage capacity, requiring multiple DRAM accesses for either activations or weights (depending on the chosen dataflow). However, if we were to augment Eyeriss with substantial on-chip buffer capacity (similar to that of SpinalFlow) and a dataflow to maximize weight reuse, the energy bottleneck again shifts to the other microarchitectural components in Eyeriss and SpinalFlow.

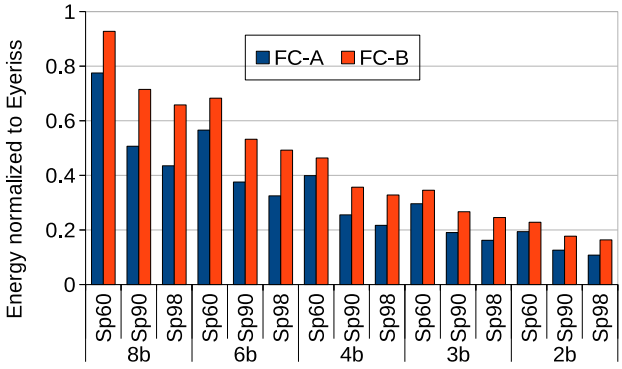

Fig. 19: *Energy per Inference of the synthetic fully connected workloads for SpinalFlow normalized to Eyeriss.*

We observe similar trends as for convolutional layers. Figure 19 shows the energy of SpinalFlow with respect to Eyeriss for executing workloads FC-A and FC-B at a batch

size of 200. At 0% sparsity, SpinalFlow consumes 20% more energy than Eyeriss, whereas at a higher sparsity level of 90% and at 4b resolution, SpinalFlow consumes 0.3× of the energy consumed by Eyeriss.

D. Scalability Study

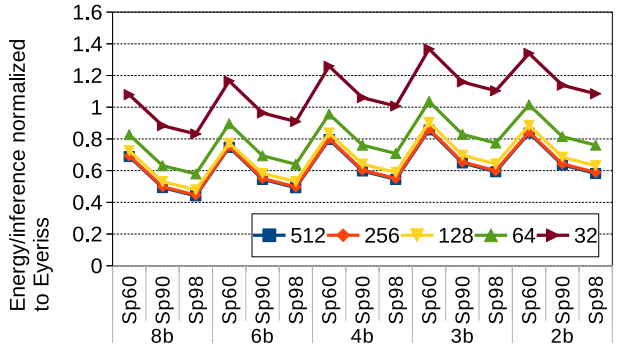

Fig. 20: *Energy per inference for ResNet on SpinalFlow, normalized to Eyeriss, as a function of the number of PEs in SpinalFlow.*

Next, we analyze the scalability of SpinalFlow as the number of PEs and, correspondingly, the size of the weight buffer are varied. Figure 20 shows the change in energy per inference as the PEs (and the weight buffer) are increased from 32 (144 KB) to 512 (2.25 MB) for ResNet. Increasing the compute and storage resources increases the efficiency, but with diminishing returns beyond 128 PEs. This is because few layers use more than 256 feature maps, and weight buffer energy increases significantly. Similar results were observed for other workloads as well.

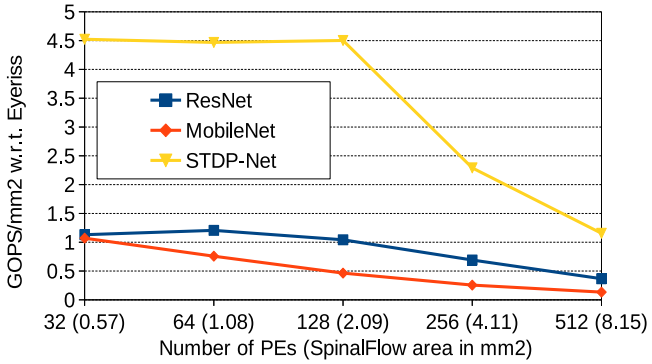

Fig. 21: *Compute density ($GOPS/mm^2$) of SpinalFlow normalized to Eyeriss and its area as the number of PEs is varied.*

Figure 21 shows the throughput-per-area for our three workloads and SpinalFlow area as the PE count (and buffer size) are varied. Because of the large weight buffer in SpinalFlow, it does not fare as well as Eyeriss in throughput-per-area in many cases. This effect can be alleviated by using fewer PEs and smaller buffers.

E. SpinalFlow vs. SCNN

As a sensitivity analysis, we also compare SpinalFlow to SCNN [46], an ANN accelerator that exploits sparsity. We model SCNN at 8-bit resolution and with 8 PEs (128 MAC units) for an iso-ALU comparison. We make favorable assumptions for SCNN – we do not model the computation and storage overheads of meta-data (indices), and the crossbar that connects MACs with the accumulator buffer. We model the accumulator buffer with 2 banks instead of 32 due to limitations with Cacti. For ResNet, SpinalFlow at 60% activation sparsity consumes 1.02× less energy than SCNN, whereas at 90% activation sparsity, it consumes 1.16× less energy than SCNN. While we assume a similar buffer organization as in the original SCNN work, we expect that a sweep of different buffer hierarchies may reveal more energy-efficient SCNN design points.

VI. Re-Visiting the SNN vs. ANN Debate

There remains a healthy debate within the community about the merits of SNNs and ANNs. These issues have been discussed in keynotes at ASPLOS 2014 (Gehlhaar [19]), HPCA 2015 (Modha [39]), ISCA 2015 (Temam [57]), ASPLOS 2016 (Williams [59]), and FCRC 2019 (Smith [55]). In this section, we summarize the current state of this debate, given our findings. In particular, our analysis is among the first to demarcate when an SNN is a better or worse choice than an ANN.

A. Comparing SNN vs. ANN Efficiency

A couple of papers have analyzed SNN vs. ANN efficiency. A MICRO 2015 paper by Du et al. [13] attempted a head-to-head comparison of ANN and SNN hardware. They compare a two-layer ANN (100 neurons in the first layer and 10 neurons in the second layer) against a one-layer SNN (300 neurons) on the MNIST workload for digit recognition. The architecture models assume some dedicated hardware per neuron, an approach that does not scale up to large networks. For this limited comparison, the authors conclude that ANNs and SNNs have similar per-neuron area and power overheads. A more recent paper by Khacef et al. [27] improves upon this prior work with more efficient and more accurate neuron models, but draws similar conclusions for a limited set of networks and dataflows. Our analysis of larger/diverse networks and architectures shows that data reuse is a key factor; with baseline dataflows, we show (contrary to prior work) that SNNs are an order of magnitude worse than ANNs in most key metrics. The execution time is $2^{resolution}$ higher and energy is 35× higher at high resolution and 2× higher at low resolution. With our new dataflow, for smaller input intervals, where t-SNNs are expected to perform best [55], SNNs consume 5× lower energy. SNN and ANN energy efficiency are almost on par even for large input intervals. When networks exhibit high sparsity, also expected for t-SNNs [51], [55], SNN execution time and energy improve significantly. The comparison is more nuanced if ANNs are also allowed to lower resolution; this is an approach that is known to significantly lower accuracy [11],

[26], [48], [62]. With this approach, as shown in Figures 14 and 15, ANNs and SNNs are comparable in terms of energy; the nature of the network and the degree of sparsity determines the winning architecture.

B. Discussion of Prediction Accuracy

The improved dataflow in this paper only improves efficiency (time and energy), and we quantify the relationship between efficiency and sparsity/resolution. The new dataflow has no impact on accuracy. But since accuracy is a primary consideration in the SNN vs. ANN debate, for completeness, we articulate the conditions under which SNNs or ANNs may be superior.

First, consider a use-case where labeled datasets are available for supervised training on GPU/TPU clusters. This is the scenario where ANNs with back-propagation based SGD represent the state-of-the-art. A number of studies have shown that r-SNNs can borrow such weights and achieve similar accuracies as ANNs [27], [13], [15], [16], [50]. This is primarily because an r-SNN neuron can emulate the behavior of an ANN. On the other hand, t-SNN training has received less attention and t-SNNs generally have lower accuracies. We summarize some of these key results in Table IV. Given that t-SNNs are more efficient in terms of time and energy on SpinalFlow, t-SNN training is an area that demands future investment, a point also made by Smith [55]. The work of Comsa et al. [10] shows an example t-SNN operating at low resolution and high sparsity that matches the accuracy of an ANN. With further advances along these lines, t-SNNs may be able to achieve higher accuracies and lower energy than high- and low-res ANNs. Note that low resolution has typically been a significant handicap for ANNs in terms of accuracy, but this is not the case for SNNs [29]. For a more complete summary of the trade-off space, we also show the impact of low-resolution ANNs on accuracy in Table IV, e.g., note that a 4-bit ANN can reduce accuracy by 2.9% (for AlexNet on ImageNet [62]).

A second use case is one where continual learning is required. The ability of STDP to efficiently perform online training allows SNNs to react faster when new inputs are encountered, e.g., new landscapes during disaster recovery or new accents during speech processing. Note that in such use cases, curated, pre-processed, and labeled datasets are often not available. The training may also have to be performed at low energy on an edge device, e.g., a rover handling disaster recovery. The area of continual learning [8] is an emerging one with a limited amount of literature. ANNs trained with SGD suffer from the concept of catastrophic forgetting [38], [49] when they are sequentially trained on two datasets, i.e., SGD's global error minimization tends to perturb all network parameters to react to the new dataset [20], [35], [31], [4]. On the other hand, STDP does not require labeled datasets and its localized training can naturally earmark a subset of neurons for the new dataset, while not perturbing the rest of the model [4]. For such use cases, SNN/STDP is a clear winner and can exploit the new dataflows to significantly reduce execution time and energy.

Workload	ANN Accuracy	SNN Accuracy
MNIST	99.8% [58] 1-bit res: 99.04% [11]	r-SNN(SGD): 99.59% [33] r-SNN(STDP+SGD): 99.28% [32] t-SNN (SGD): 97.96% [10] t-SNN (STDP): 98.4% [29]
CIFAR10	92.38% [50] 1-bit res: 88.6% [11]	r-SNN: 90.53% [60]
AlexNet on ImageNet	55.9% [62], [24] 1-bit res: 44.2% [48] 2-bit res: 49.8% [62] 4-bit res: 53.0% [62]	r-SNN: 51.8% [24]
VGG on ImageNet	70.52% [51]	r-SNN: 69.96% [51]
ResNet on ImageNet	70.69% [51]	r-SNN: 65.47% [51]

TABLE IV: *Accuracy comparison with supervised training on labeled datasets.*

VII. Conclusions

Our work first shows that the baseline SNN architecture, Spiking Eyeriss, is severely penalized by repeated accesses to neuron potential and filter weights as ticks are sequentially processed in the input interval. The Spiking-Eyeriss design consumes $2\times$ more energy than baseline Eyeriss, even at high sparsity and low resolution. It is also $2^{resolution}$ times slower than Eyeriss. We then devised a new architecture and dataflow that increases data reuse and is tailored for the high sparsity that is expected in future SNNs.

The resulting SpinalFlow design improves energy efficiency by $5\times$ over Eyeriss and by $1.8\times$ over a 4-bit version of Eyeriss. It consumes less energy than Eyeriss at most evaluated sparsity/resolution points. The new designs are effective for a range of convolutional layers, and even more effective for memory-constrained fully-connected layers. In terms of performance, SpinalFlow is faster than Eyeriss by $5.4\times$, when assuming a sparsity level of 90%. Because SpinalFlow's weight accesses are less regular, it needs a larger buffer for weights, and yields lower $throughput/mm^2$ than Eyeriss for some workloads. The new architecture also greatly improves the energy, latency, and throughput for accelerators, like TrueNorth, that will be used to simulate brain models [34], [2]. We thus show that for large neural networks, reuse management and sparsity exploitation are key in determining SNN vs. ANN relative efficiency.

Our results also serve as a useful guideline for researchers developing SNNs for various use cases. Our analysis quantifies the scenarios (resolution, sparsity, network topology) under which SNNs can surpass the energy efficiency of ANNs. We highlight the need to develop accurate training models for t-SNNs because it results in higher sparsity levels and lower energy per inference.

VIII. Acknowledgments

We thank the anonymous reviewers for many helpful suggestions. This work was supported in parts by NSF grant CNS-1718834, Google, and NSF CAREER award 1751064.

REFERENCES

[1] K. Ahmed, A. Shrestha, Q. Qiu, and Q. Wu, "Probabilistic Inference Using Stochastic Spiking Neural Networks on a Neurosynaptic Processor," in *Proceedings of IJCNN*, 2015.

[2] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. Kuang, R. Manohar, W. Risk, B. Jackson, and D. Modha, "TrueNorth: Design and Tool Flow of a 65mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34(10), 2015.

[3] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. Jerger, and A. Moshovos, "Cnvlutin: Zero-Neuron-Free Deep Convolutional Neural Network Computing," in *Proceedings of ISCA-43*, 2016.

[4] J. M. Allred and K. Roy, "Stimulating STDP to Exploit Locality for Lifelong Learning without Catastrophic Forgetting," in *arXiv preprint arXiv:1902.03187*, 2019.

[5] B. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. Arthur, P. Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102(5), 2014.

[6] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, no. 52(1), 2016.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of ISCA-43*, 2016.

[8] Z. Chen and B. Liu, *Lifelong Machine Learning*. Morgan & Claypool, 2018.

[9] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proceedings of ISCA-43*, 2016.

[10] I. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function," *arXiv preprint arXiv:1907.13223v2*, 2019.

[11] M. Courbariaux and Y. Bengio, "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 2016, arXiv preprint 1602.02830.

[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-38*, 2018.

[13] Z. Du, D. Rubin, Y. Chen, L. He, T. Chen, L. Zhang, C. Wu, and O. Temam, "Neuromorphic Accelerators: A Comparison Between Neuroscience and Machine-Learning Approaches," in *Proceedings of MICRO-48*, 2015.

[14] J. E.Smith, "Space-Time Algebra: A Model for Neocortical Computation," in *Proceedings of ISCA*, 2018.

[15] S. Esser, R. Appuswamy, P. Merolla, J. Arthur, and D. Modha, "Backpropagation for Energy-Efficient Neuromorphic Computing," in *Proceedings of NIPS*, 2015.

[16] S. Esser, P. Meroll, J.V.Arthur, A.S.Cassidy, R.Appuswamy, A. Andreopoulos, D. Berg, J. McKinstry, T.Melano, D. Barch, C. Nolfo, P. Datta, A. Amir, B. Taba, M. Flickner, and D. Modha, "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," in *arXiv*, 2016.

[17] D. Feldman, "The Spike Timing Dependence of Plasticity," *Neuron*, no. 75(4), 2012.

[18] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," pp. 267–285, 1982.

[19] J. Gehlhaar, "Neuromorphic Processing: A New Frontier in Scaling Computer Architecture," 2014, keynote at ASPLOS.

[20] I. J. Goodfellow, M. Mirza, and e. a. Da Xiao, "An empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks," in *arXiv preprint arXiv:1312.6211*, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[23] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[24] E. Hunsberger and C. Eliasmith, "Training Spiking Deep Networks for Neuromorphic Hardware," in *arXiv preprint arXiv:1611.05141*, 2016.

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of ISCA-44*, 2017.

[26] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. Jerger, R. Urtasun, and A. Moshovos, "Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets," 2016, arXiv preprint 1511.05236v4.

[27] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018.

[28] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor," in *Proceedings of IJCNN*, 2008.

[29] S. Kheradpisheh, M. Ganjtabesh, S. Thrope, and T. Masquelier, "STDP-based spiking deep neural networks for object recognition," *arXiv preprint arXiv:1611.01421v1*, 2016.

[30] D. Kim, J. H. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *Proceedings of ISCA-43*, 2016.

[31] J. Kirkpatrick, R. Pascanu, and e. a. Neil Rabinowitz, "Overcoming Catastrophic Forgetting in Neural Netowkrs," in *Proceedings of national academy of sciences 114.13*, 2017.

[32] C. Lee, P. Panda, G. Srinivasan, and K. Roy, "Training Deep Spiking Convolutional Neural Networks with STDP-Based Unsupervised Pre-Training Followed by Supervised Fine-Tuning," *Frontiers in Neuroscience*, vol. 12, 2018.

[33] C. Lee, S. S. Sarwar, and K. Roy, "Enabling Spike-based Backpropagation in State-of-the-art Deep Neural Network Architectures," *arXiv preprint arXiv:1903.06379*, 2019.

[34] D. Lee, G. Lee, D. Kwon, S. Lee, Y. Kim, and J. Kim, "Flexon: A Flexible Digital Neuron for Efficient Spiking Neural Network Simulations," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 275–288.

[35] S.-W. Lee, J.-H. Kim, and e. a. Jaehyun Jun, "Overcoming Catastrophic Forgetting by Incremental Moment Matching," in *Advances in neural information processing systems*, 2017.

[36] B. Liu, Y. Chen, B. Wysocki, and T. Huang, "Reconfigurable Neuromorphic Computing System with Memristor-Based Synapse Design," *Neural Processing Letters*, no. 41(2), 2015.

[37] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, and B. Liu, "A Spiking Neuromorphic Design with Resistive Crossbar," in *Proceedings of DAC*, 2015.

[38] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," in *Psychology of learning and motivation, Vol. 24. Academic Press*, 1989.

[39] D. Modha, "A New Architecture for Brain-Inspired Computing," 2015, keynote at HPCA.

[40] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *CoRR*, vol. abs/1606.08165, 2016. [Online]. Available: http://arxiv.org/abs/1606.08165

[41] N. Muralimanohar *et al.*, "CACTI 6.0: A Tool to Understand Large Caches," University of Utah, Tech. Rep., 2007.

[42] S. Narayanan, A. Shafiee, and R. Balasubramonian, "INXS: Bridging the Throughput and Energy Gap for Spiking Neural Networks," in *Proceedings of IJCNN*, 2017.

[43] National Science Foundation, "NSF Real-Time Machine Learning Solicitation," 2019, https://www.nsf.gov/pubs/2019/nsf19566/nsf19566.htm.

[44] A. Nere, A. Hashmi, M. Lipasti, and G. Tononi, "Bridging the Semantic Gap: Emulating Biological Neuronal Behaviors with Simple Digital Neurons," in *Proceedings of HPCA-19*, 2013.

[45] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. Keckler, and W. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *Proceedings of MICRO*, 2017.

[46] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. Keckler, and W. Dally, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," 2017.

[47] Qualcomm, "Introducing Qualcomm Zeroth Processors: Brain-Inspired Computing," 2013, https://www.qualcomm.com/news/onq/2013/10/10/introducing-qualcomm-zeroth-processors-brain-inspired-computing.

[48] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *CoRR*, vol. abs/1603.05279, 2016.
[49] R. Ratcliff, "Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions," in *Psychological review 97.2*, 1990.
[50] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks," *arXiv preprint arXiv:1612.04052*, 2016.
[51] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures," *Frontiers in Neuroscience*, vol. 13, 2019.
[52] J. Seo, B. Brezzo, Y. Liu, B. Parker, S. Esser, R. Montoye, B. Rajendran, J. Tierno, L. Chang, D. Modha, and D. Friedman, "A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons," in *Proceedings of CICC*, 2011.
[53] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *Proceedings of ISCA*, 2016.
[54] S.Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of ISCA*, 2016.
[55] J. Smith, "A Roadmap for Reverse-Architecting the Brain's Neocortex," 2019, keynote at FCRC, https://iscaconf.org/isca2019/slides/JE_Smith_keynote.pdf.
[56] T. Tang, L. Xia, B. Li, R. Luo, Y. Chen, Y. Wang, and H. Yang, "Spiking Neural Network with RRAM: Can We Use It for Real-World Application?" in *Proceedings of DATE*, 2015.
[57] O. Temam, "Hardware Neural Networks: From Inflated Expectations to Plateau of Productivity," 2015, keynote at FCRC.
[58] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of Neural Networks using DropConnect," *Proceedings of International Conference on Machine Learning - 30*, 2013.
[59] S. Williams, "Brain Inspired Computing," 2016, keynote at ASPLOS.
[60] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct Training for Spiking Neural Networks: Faster, Larger, Better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.
[61] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.
[62] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv preprint arXiv:1606.06160v2*, 2016.