INXS: Bridging the Throughput and Energy Gap for Spiking Neural Networks

Surya Narayanan
School of Computing
University of Utah

Ali Shafiee
School of Computing
University of Utah

Rajeev Balasubramonian
School of Computing
University of Utah

***Abstract*— In recent years, multiple neuromorphic architectures have been designed to execute cognitive applications that deal with image and speech analysis. These architectures have followed one of two approaches. One class of architectures is based on machine learning with artificial neural networks. A second class is focused on emulating biology with spiking neuron models, in an attempt to eventually approach the brain's accuracy and energy efficiency. A prominent example of the second class is IBM's TrueNorth processor that can execute large spiking networks on a low-power tiled architecture, and achieve high accuracy on a variety of tasks. However, as we show in this work, there are many inefficiencies in the TrueNorth design. We propose a new architecture, INXS, for spiking neural networks that improves upon the computational efficiency and energy efficiency of the TrueNorth design by 3,129× and 10× respectively. The architecture uses memristor crossbars to compute the effects of input spikes on several neurons in parallel. Digital units are then used to update neuron state. We show that the parallelism offered by crossbars is critical in achieving high throughput and energy efficiency.**

I. Introduction

The stagnation of Moore's Law scaling has shifted industry and academia's focus away from general-purpose systems and towards specialized systems. Neuromorphic architectures are an important class of specialized systems because:

1) They are efficient at a variety of machine learning tasks that are growing in prominence – image analysis in self driving cars, information discovery from massive datasets, etc.
2) They target the grand challenge of emulating brain mechanics in hopes of matching the brain's cognitive power and energy efficiency.

These two separate needs – machine learning efficiency and brain emulation – have also led to a bifurcation in neuromorphic architectures.

A number of architectures, DaDianNao [8], ISAAC [37], EIE [38], Cnvlutin [3], and Eyeriss [7] to name a few, are based on the artificial neurons (perceptrons) that have formed the basis for decades of research in machine learning. We refer to these architectures as artificial neural network accelerators, or *ANN accelerators*.

The second class of architectures, TrueNorth [2], SpiNNaker [25], and Neurogrid [4] to name a few, are based on biologically plausible models of spiking neurons. We refer to these architectures as spiking neural network accelerators, or *SNN accelerators*.

The goal of this paper is not to compare ANN vs. SNN accelerators. To date, only one study, by Du et al. [13], has performed a head-to-head comparison of ANN and SNN accelerators. While that study is an excellent start to an important debate, it draws limited conclusions for small-scale chips executing small-scale networks. For example, they conclude that SNNs achieve significantly lower accuracy than ANNs on MNIST, but concurrent work [14, 15] developed better training algorithms for SNNs and achieved 99.42% accuracy on MNIST. Du et al. investigate chips with no more than 110 artificial neurons and 300 spiking neurons. We mention these examples to highlight that the comparison between ANNs and SNNs is far from being resolved, and will likely play out over the coming decade.

In the meantime, advances are required for both ANNs and SNNs. At the moment, much of the architecture research has focused on ANNs. As a result, ANNs are ahead of SNNs on a variety of metrics (see Table I). This paper attempts to bridge that gap by designing better SNN architectures that can keep up with the high throughput and energy efficiency being achieved on state-of-the-art ANNs.

The most high-profile and most efficient SNN architecture to date is IBM's TrueNorth. It is a 5.4 billion transistor chip that can model 1 million neurons and 256 million synapses while consuming less than 100 mW. TrueNorth achieves high tile-level parallelism, and makes a number of design choices that impose constraints on the application, while reducing power and storage requirements. However, we see in Table I that TrueNorth lags behind state-of-the-art ANN accelerators on all metrics, notably throughput and energy. Therefore, drawing inspiration from recent ANN architectures, we undertake an overhaul of the TrueNorth design.

We describe an SNN accelerator that leverages memristor crossbars to aggregate the effects of input spikes in the analog domain. By effectively using in-situ computing, memristor crossbars have been shown to achieve high parallelism and storage density in the ISAAC [37] and PRIME [9] ANN accelerators. We describe the many changes required to adapt a crossbar-based architecture for an SNN. In particular, the management of neuron potentials represents the biggest challenge, and the sparse spike rate represents the biggest opportunity. The former requires additional storage overheads, and the

Network Type	Accuracy			Throughput (TOPs/s)		Energy per operation (pJ/op)	
	MNIST	CIFAR-10	AlexNet	Digital	Analog	Digital	Analog
ANN	99.77% [10]	96.53% [17]	89% [26]	9 [3, 8]	45 [37]	3.2 [3, 8]	1.5 [37]
SNN	99.42% [14]	89.32% [15]	82.5% [21]	0.058 [2]	0.07 [29]	41 [2]	0.35 [29]

TABLE I

COMPARISON OF ACCURACY, THROUGHPUT, AND ENERGY EFFICIENCY FOR STATE-OF-THE-ART ANNS AND SNNS. THE DIGITAL SNN NUMBERS CORRESPOND TO TRUENORTH [2]. THE ENERGY NUMBER FOR THE ANALOG SNN ACCELERATOR IS FOR A SMALL-SCALE 32-NEURON IMPLEMENTATION [29].

latter enables low overheads for analog-to-digital conversion (ADC). We carry out a design space exploration to identify the best provisioning of resources for this mixed-signal architecture.

II. BACKGROUND

Artificial neurons were developed more than 70 years ago [30]. Artificial neurons receive synchronous real-valued inputs, perform a dot-product of these inputs with weights, apply an activation function (often ReLU), and pass real-valued outputs to the next layer of artificial neurons. In addition to many decades of progress, the past decade has seen significant advances with artificial neurons, primarily because of our ability to train deep networks with a combination of new techniques.

A. Spiking Neurons

While neuroscientists have delved into the mechanics of the biological neuron for decades [20], it has only recently received attention from the architecture community. A number of high-profile projects [2, 4, 25] 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 [34].

The most popular of these simple neuron models is the Linear Leaky Integrate and Fire model (LLIF), shown in Figure 1. An LLIF neuron is stateful – in addition to synaptic weights, it retains the value of its (membrane) potential. This 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 cycle, 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, i.e., a communication link between neurons requires a single bit. Further, the spiking neuron model does not require a multiplier – because the input is binary, the synaptic weight is simply added to the potential. Spikes can therefore lead to efficient communication and computation.

Because a neuron is designed to respond after observing spikes over time, the input is provided over an *input interval*, say 500 cycles. Figure 2 shows how each pixel of an input

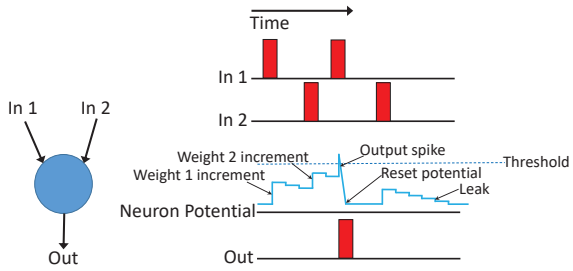

Fig. 1. A basic 2-input LLIF spiking neuron. The figure shows how the neuron potential is incremented when input spikes are received, how a leak is subtracted when there are no input spikes, and how an output spike is produced when the potential crosses the threshold.

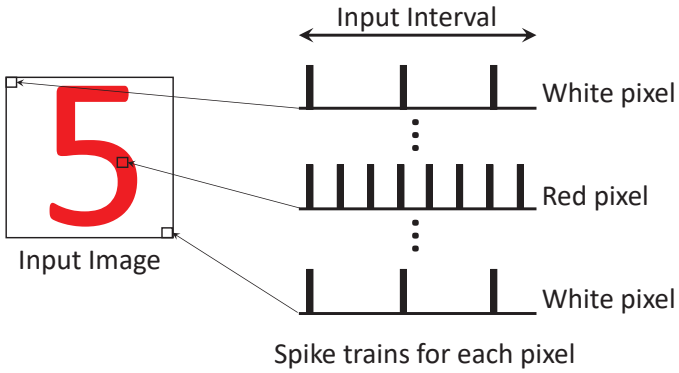

Fig. 2. Example of an input image being converted into a number of input spike trains that are fed to a spiking neural network.

image is converted into a spike train that extends across an input interval. These spike trains are fed as inputs to the first layer of neurons. Prior work has primarily used *rate codes* that convert an input pixel value into a certain number of spikes. For example, a red pixel value may be converted into 50 evenly spaced spikes in 500 cycles, while a blue pixel value may be converted into 125 evenly spaced spikes in the input interval. The same encoding is typically used throughout the network, i.e., information is carried in terms of spike intensity. The code also includes an element of stochasticity, e.g., a rate code typically uses a Poisson distribution to inject spikes [1].

Spiking neurons are typically trained with a biologically plausible process called STDP (Spike Timing Dependent Plasticity [16]). This is an unsupervised training method where each neuron adjusts its weights based on a local process. Recent studies have been unable to achieve high accuracies with STDP-based training [13, 36]. Therefore, more recent works

have resorted to supervised backpropagation-based training for spiking networks [14, 15].

B. SNN Accelerators

IBM's TrueNorth processor [32] is the most prominent example of a digital architecture for large SNNs. We will use TrueNorth as the SNN baseline in this work because it achieves best-in-class throughput and energy efficiency.

TrueNorth is composed of many tiles, where each tile implements 256 neurons, each with 256 inputs. The tiles communicate through an on-chip and potentially an off-chip network. The tiles use a mix of asynchronous and synchronous circuits to boost energy efficiency. In every 1ms *"tick"*, a tile processes all received input spikes; any resulting output spikes are sent through the network to neurons in the next layer so they can be processed in a subsequent tick. TrueNorth implements an LLIF neuron model with a number of configurable parameters, including some that allow stochastic behavior. 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. For each neuron, it reads a 410-bit SRAM row that contains all parameters for that neuron, including a 256-bit vector indicating which tile inputs connect to that neuron. This bit vector is reconciled with the list of input spikes in that tick to identify spiking connections for that neuron. The synaptic weight for each of these connections is then sent to a synchronous neuron unit that performs the necessary arithmetic operations. This unit adds the synaptic weights to the neuron's potential. Finally, the leak is subtracted and the potential is compared against the threshold. In case of an output spike, the neuron potential is reset. The final neuron potential is then written back to the SRAM bank. The 12.8 KB SRAM bank occupies nearly half the tile area and one-third the tile power. A tile processes a single synapse at a time. The tick is long enough (1ms) to process all possible input spikes and neurons sequentially.

To further reduce storage requirements and energy, TrueNorth imposes several constraints on the neural network. It only uses 4 quantized 9-bit weights per neuron. It also forces an input spike to share the same weight type with all neurons in that tile. A neuron's output can only be seen by the 256 neurons in one tile. A neuron can only receive at most 256 inputs.

SpiNNaker [25] is another prominent SNN architecture that uses many low-power general-purpose ARM cores to perform several parallel neuron updates. It is well known that custom ASICs will out-perform general-purpose cores by at least two orders of magnitude [18], so we will not explore SpiNNaker-style architectures in this paper.

A few projects have attempted to implement neurons and synapses with analog devices, typically using capacitors or memristors to emulate neuronal behavior [2, 28, 29, 42]. These projects have focused more on device innovations to reproduce neuron behavior, and have not focused on architectural innovations to boost throughput. For example, Liu et al. [29] implement a single 32×64 memristive crossbar to execute feedforward and Hopfield networks. The crossbar performs the synaptic operations, and an analog integrate-and-fire circuit models the neuron. But, maintaining the neuron potential in an analog circuit can incur a very high area overhead, especially in large-scale convolutional networks where the number of neurons far exceeds the number of (shared) synapses. However, we do believe that analog circuits have a lot to offer [23] and we will use the analog domain in a limited manner to accelerate the neuron update.

C. ANN Accelerators

Our proposed architecture is inspired by the best practices in state-of-the-art ANN accelerators. We first discuss the analog approach, followed by the digital approach.

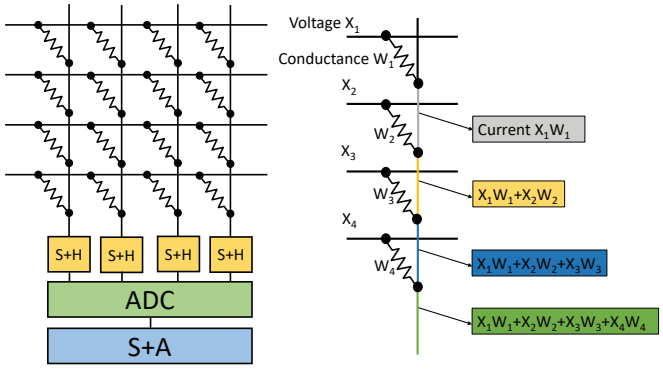

Fig. 3. (a) An example 4x4 memristor xbar connected to peripheral circuits. (b) The conductance of the memristor corresponds to the synaptic weight and the voltage corresponds to the spike input. The current at the end of the bitline is the dot-product of input spikes and synaptic weights.

Two architectures introduced in the past year, ISAAC [37] and PRIME [9], have leveraged memristor crossbars to perform dot product operations in the analog domain and accelerate deep convolutional neural networks. We will focus on ISAAC here because it out-performs PRIME in terms of throughput, accuracy, and ability to handle signed values. We note that a few other papers have also analyzed the circuits required in crossbar-based accelerators [11, 40, 43].

A memristor crossbar uses Kirchoff's Law to produce a sum of products, as shown in Figure 3. Inputs are provided as a vector of voltages; the memristor conductances in the crossbar represent synaptic weights of neurons; the emerging bitline currents are neuron outputs (before the activation function) because they represent the dot products of input voltages and synaptic weights. This is an example of in-situ computing because the crossbars are not only used to store weights, but also perform computations on them. ISAAC uses a number of crossbars in a tiled architecture to process all layers of a deep network in parallel. It distributes computations across time and space to manage the high costs of analog-to-digital conversion (ADC). Even with such techniques, the ADCs account for a large fraction of chip power and area. To support sufficient precision, ISAAC employs 8-bit ADCs to capture the largest possible dot-product emerging from a crossbar bitline. The dot

products, after analog to digital conversion, are aggregated with digital ALUs. eDRAM banks are used to store neuron outputs until they are consumed by the next layer. By setting up a pipeline from layer to layer, a relatively small set of outputs has to be buffered, which can be accommodated in a 64 KB eDRAM unit per tile. Since a dense crossbar is used to store the weights and perform computation, ISAAC is able to dramatically reduce data movement, and increase computation/storage density.

We next describe recent digital ANN architectures. The DianNao [6] and DaDianNao [8] accelerators were among the first to target deep convolutional networks. DianNao designs the digital circuits for a basic NFU (Neural Functional Unit) that can process 16 inputs to 16 neurons in parallel. DaDianNao is a tiled architecture where each tile has an NFU and eDRAM banks that feed synaptic weights to that NFU. DaDianNao uses many tiles on many chips to parallelize the processing of a single network layer. Once that layer is processed, all the tiles then move on to processing the next layer in parallel. Thus, the keys to DaDianNao's efficiency are: (i) localized data movement (from local eDRAM bank to nearby NFU), and (ii) time-multiplexed execution of several neurons and several network layers on a small set of SIMD execution units (the NFUs). Other recent papers have proposed innovations to digital ANN accelerators that primarily exploit sparsity [3, 35, 38]. Since digital ANN accelerators are nearly an order of magnitude slower than ISAAC [37], we will not consider them further in this paper.

III. The INXS Architecture

A. Overview

As described in the previous section, the best SNN accelerator to date, TrueNorth, suffers from a few weaknesses:

1) There is no intra-tile parallelism while performing neuron updates.
2) Each input spike to a neuron is handled sequentially.
3) To reduce the storage and energy overheads, significant approximations have to be made for the synaptic weight values.
4) A neuron can only have at most 256 inputs and its output can be seen by at most 256 other neurons connected to one axon.

We design a mixed-signal architecture, INXS[1], that addresses all of the above problems, and can efficiently handle state-of-the-art deep networks. A large number of memristor crossbars are used to process the many incoming spikes in a tick, and compute the resulting potential increments in parallel. The potential increments are immediately converted to digital signals. The neuron potentials are retrieved from SRAM buffers with wide reads, added to the increments, thresholded, and written back to SRAM. The resulting spikes are routed to the next layers so they can be processed in the next tick. Many crossbars work in unison on different layers of the neural network to set up an efficient pipeline.

[1]INXS, pronounced "in excess" is short for IN-situ Xbar Spiking

The key contributions of this design are:

1) It offers very high pipelined parallelism with many crossbars, not only working on many SNN layers in parallel (as in TrueNorth), but also working on many neuron update values and many input spikes in parallel (unlike TrueNorth). In most cases, the pipeline operates as an odd-even pipeline, working on analog crossbar operations in odd ticks, and digital neuron updates in even ticks.
2) While some prior works [29] have implemented a crossbar in tandem with analog neurons, we observe here that in a convolutional network, a set of shared weights are used to compute several neurons. The use of analog neurons would require a single crossbar bitline to be multiplexed across many analog circuits, resulting in significant overheads. Therefore, we immediately convert the analog crossbar output into a digital signal and perform the even phase in the digital domain.
3) We lay out several design details and carefully consider the overheads of each module. We follow with a design space exploration to identify how best to provision the resources per tile.
4) The resulting architecture differs from the state-of-the-art ANN accelerator, ISAAC, in the following ways: (i) ISAAC requires a 22-stage pipeline while INXS only requires a 2-stage pipeline to process a single neuron in one layer, (ii) INXS uses a low-resolution ADC because of observed sparsity, thus achieving higher throughput per area, and (iii) it allocates more area for central storage and neuron update.
5) The resulting architecture differs from the state-of-the-art SNN accelerator, TrueNorth, in the following ways: (i) INXS does not constrain neuron input/outputs and weights in any way, (ii) it offers orders of magnitude higher parallelism and throughput, and (iii) it achieves lower energy per operation by boosting throughput and lowering the contribution of leakage.

B. Implementation Details

Overall Chip Organization

INXS is designed to be modular and hierarchical. Figure 4 shows that a chip is composed of several tiles connected with a mesh network. The many layers of an SNN are scattered across these tiles. Figure 4 also zooms into one of these tiles. A tile has *central SRAM buffers*, *Neuron Units*, *Synaptic Units*, and a router. The Synaptic Units, Neuron Units, and the router in a tile are connected by a unidirectional ring network. The ring network has North, South, East, West, and Hub stations – the Hub is used to switch to the inter-tile mesh network. Figure 5 shows the details of a Synaptic and Neuron Unit in a tile. Each Synaptic Unit is composed of multiple memristor crossbars and ADCs. The Neuron Unit has the adders and thresholding logic to implement the neuron model. Next, we'll walk through the operations required to execute a single convolutional layer.

The Odd Phase

In every odd tick, all the crossbars on the chip receive inputs from their input buffers. A tick is assumed to be at least

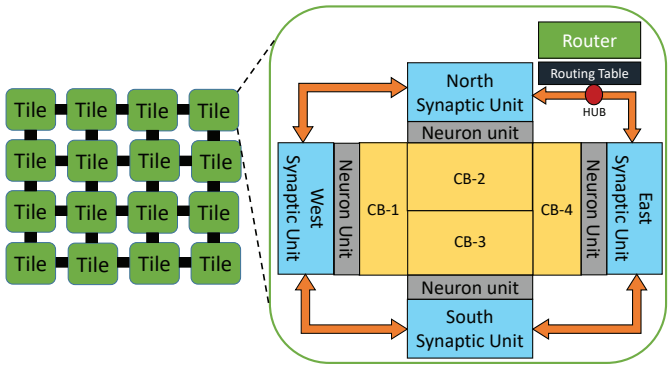

Fig. 4. INXS tiled architecture and details of one tile.

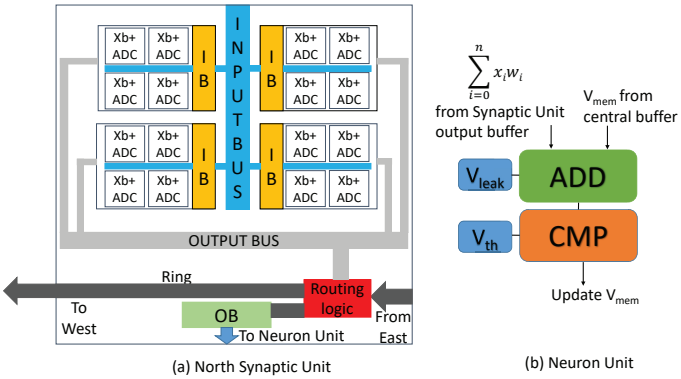

Fig. 5. (a) Synaptic Unit. (b) Neuron Unit. (The exact number of xbars, ADCs, adders, etc. vary in our optimal design points.)

100 ns [37] to allow sufficient time to perform a crossbar read and capture all the bitline outputs in sample and hold circuits. This phase exploits very high parallelism in the analog domain to estimate the effect of every incoming spike on the potential of several neurons.

We'll assume that a crossbar has R rows and C columns of w-bit cells. We'll assume that weights and neuron potentials are stored with p-bit fixed-point precision. Depending on the values of p and w, a single synaptic weight may be spread across multiple cells in a row. If a neuron has more than R inputs, its calculation will be spread across multiple crossbars, and potentially multiple Synaptic units.

The Even Phase

The even phase is itself composed of several small *"cycles"*. In the first cycle, the ADC processes the first bitline output. The results of multiple bitlines (after ADC) have to be aggregated with shift-and-add circuits in the Synaptic Unit because they represent contributions from different bits of the synaptic weights. If a neuron has more than R inputs and is spread across multiple crossbars, those partial results have to be aggregated as well. Once the partial result within a Synaptic Unit has been aggregated, this potential increment is placed on the output bus. The potential increment is routed to that neuron's home, possibly navigating 0 or more hops on the ring network and 0 or more hops on the mesh network. Once the partial sums are generated, the routing logic in each synaptic unit routes them to their respective neuron home through the ring bus if it resides in the same tile, or through the mesh network if the neuron home is in another tile. The partial sums are stored in the output buffer, which acts as input to the Neuron Unit.

Once the increment reaches the neuron home, it is added to the neuron's potential and leak in the Neuron Unit. To enable this addition, the neuron potential has to be read from the central SRAM buffer in the previous cycle. Once the new neuron potential is calculated, it is thresholded, and the final neuron potential is written back to the SRAM buffer. The generated spike is then sent over the ring and mesh networks to a destination input buffer, where it will be accessed in the next Odd Phase.

All of these operations are performed deterministically and controlled by finite state machines in Synaptic and Neuron Units. The control signals for the finite state machines would be generated at compile time and loaded into the chip along with the weights for each network layer. We assume that the leak and threshold are the same for all neurons in a layer [5, 12]. We provision the network, SRAM buffer, and adders with sufficient bandwidth so they can handle the worst-case network layers in our evaluated workloads without introducing any structural hazards and contention. If such a chip had to evaluate an even larger network (say, more neurons or more inputs per neuron), it can do so, but would require multiple ticks to process the Even Phase.

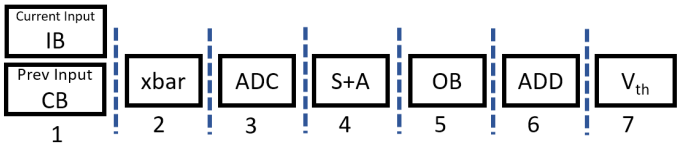

Fig. 6. INXS pipeline. IB - Input Buffer, CB - Central Buffer, OB - Output Buffer

Figure 6 shows the many pipeline stages that must be navigated for one neuron computation. At compile time, we would estimate the number of cycles required for one neuron increment, starting from the ADC, all the way until the resulting spike is placed in the next layer's input buffer. This number S will vary across layers and workloads depending on the required number of network hops. As each ADC sequentially walks through the C bitlines in its corresponding crossbar, new values begin navigating this S-stage pipeline. Therefore, the Even Phase will complete after $C + S$ cycles. The length of a tick is therefore variable across workloads, and is a function of the worst-case network layer in each workload.

Because consecutive network layers will typically map to adjacent tile quadrants or tiles, most communication in the network tends to be nearest-neighbor communication.

An Example Design Point

We carry out a design space exploration, where we vary a number of INXS parameters:

- Number of memristor xbars in a Synaptic Unit
- Number of Synaptic Units in a tile
- Number of Neuron Units in a tile
- Central buffer size

For each of these design parameters, we sweep through our example workloads and provision the bandwidths of each module so they can handle the worst-case layers. We then estimate the throughput, area, and power for each design point for our workloads. In Section V, we show the design points that optimize throughput/area and throughput/power.

To make our architecture more concrete, we walk through the parameters selected for the optimal throughput/area design point, while assuming 16-bit fixed-point computations. This design uses 64 crossbars in a Synaptic Unit, and 8 Synaptic Units in each tile. The crossbar has 256 rows and 128 columns, and stores 2 bits per cell. The bus within a Synaptic Unit, the ring network, and the mesh network all have a width of 128 bits. Each of the 8 SRAM central buffers in a tile has a capacity of 128 KB, a row width of 128 bytes, and a read latency of 0.57 ns. The Neuron Unit contains 8 3-input adders and 8 comparators to perform the neuron activation.

Balancing the Pipeline

In one Odd/Even Phase, a convolutional kernel in a layer is applied to one set of inputs to produce one output neuron. This process has to be repeated over several Odd/Even Phases until the kernel has been applied to an entire set of input feature maps. Some layers have less work to do than others. Those layers can either idle in some cycles or we can replicate the weights and boost the throughputs of the work-intensive layers so every crossbar is busy in every cycle. Such replication leads to a balanced pipeline, similar to the one employed in ISAAC.

In our workload evaluations, we also observe that the spike rate is relatively sparse and that the maximum observed output from a bitline is significantly smaller than the worst-case output. While a 256x128 memristor xbar would need a 10-bit ADC to capture its worst-case output, we observe enough sparsity in our applications that an 8-bit ADC is actually sufficient. This significantly boosts the throughput/area and throughput/power metrics, while having zero impact on accuracy. We envision that a developer would have to run simulations to confirm that the ADC precision is rarely exceeded at run-time; if it is, such overflows can be avoided by mapping fewer inputs to every crossbar column.

Neuron Model

Note that we are implementing a simple LLIF neuron model. We are not modeling the many modes and stochastic features implemented by TrueNorth. The adder/thresholding unit can be augmented to handle these additional modes and we leave these as future work. It is worth noting that the adder/thresholding unit occupies 0.6% of tile area, so even if its size is increased by 10×, its overhead would be small. For this study, we assume that all pooling layers use average pooling, because average pooling is more amenable to crossbar acceleration than max pooling.

Routing Table

The outputs produced by bitlines of a crossbar are routed to the same neuron unit, and eventually to the same set of destination crossbars in the next layer. Each crossbar therefore has a single register that is used to route the result to its neuron home. The neuron home has a routing table that has one entry for each neuron. That entry keeps track of all the crossbars in the next layer that must receive the spike resulting from that neuron. We calculate the number of entries by analyzing the state-of-the-art neural networks like MSRA and VGG-NET. Based on our analysis, we fix the number of crossbars that each neuron output can connect to as 512 (128K neurons, 512× better than TrueNorth). We size these structures so they can handle the largest deep network to date; the resulting size of the routing table is 25.5 KB.

IV. Methodology

We use the following metrics to evaluate the various design points:

- Computational Efficiency(**CE**): Peak number of 16-bit operations performed per second per mm^2.
- Energy Efficiency (**EE**): Peak number of 16-bit operations performed per second per Watt.
- Storage Efficiency (**SE**): Mega bytes of storage per mm^2. This includes synaptic storage in crossbars and neuron potential storage in SRAM central buffer.
- Energy consumed for entire state-of-the-art deep networks (VGG-NET and MSRA).

For our power and area analyses at 32nm technology, we use CACTI [33] for SRAM buffers, ORION 2.0 [24] for router and interconnect evaluation, the models of Shafiee et al. [37] for CMOS-compatible TaOx memristor crossbars, and recent adder/comparator models [31, 41]. We use [27] for ADC evaluation.

We use a manual process (emulating a future compiler) to map the different network layers of our workloads to crossbars/tiles while not exceeding any of the available resources, and while replicating layers to maintain a balanced pipeline. As described in Section III-B, we estimate the length of a tick (107 ns) based on the worst-case $C + S$ value for our workloads. The length of each cycle is 0.78 ns and is determined by the latency to process one ADC sample. The overall performance is determined with an analytical model that considers the sizes of each network layer. For our evaluvation we calculate the $C + S$ value for each layer. The delay of one access to the central buffer dictates the frequency in configurations with really large central buffer. Note that cycle-accurate simulations are not required because the workloads do not encounter any conditional structural hazards.

Hunsberger et al. [22] show that convolutional neural networks can be mapped to SNNs while achieving very similar accuracy. In a similar vein, we use two state-of-the-art deep convolutional networks for image classification, VGG-NET [39] and MSRA [19], to evaluate INXS. Since we need to find a design point that performs well on both fully connected

SNNs and convolutional SNNs, we pick VGG-NET (it has the most neurons/layer in Conv1) and MSRA (it has the most number of inputs/neuron in FC1).

ISAAC uses a 128×128 crossbar and an 8-bit ADC so bits are never dropped. The use of a modest crossbar size keeps noise in check and allows use of a low-resolution ADC. Given the inherent sparsity of spikes in an SNN, we allow use of a 256×128 crossbar while still using an 8-bit ADC. We also explore the use of a 6-bit ADC that assumes sufficient sparsity in spikes. Note that applications with high spike rates would be forced to use a subset of crossbar rows so the ADC precision is rarely exceeded. As a sensitivity study, we also explore use of a 6-bit ADC in tandem with 1-bit memristor cells that is guaranteed to not drop bits – while this design has a lower overhead for ADCs, it uses more crossbars to represent synaptic weights. Our results show that this design point does not match a design with an 8-bit ADC in tandem with 2-bit memristor cells, so we will not discuss it further.

V. Results

A. INXS design space exploration

For all our results, we evaluate metrics across a number of design points. The X-axis in most figures describes these design points as $a \times b \times c \times d$, where $a \times b$ describes the number of crossbars in a Synaptic Unit, c represents the number of Synaptic Units per tile, and d represents the capacity of central buffer in a tile (in KB). Note that in a convolutional layer, a single crossbar can produce results for several neurons across several ticks. The size of the central buffer puts a cap on the number of neurons that can be produced locally by the corresponding synaptic units.

We first evaluate peak computational efficiency, shown in Figure 7, for the INXS design as we vary our design parameters. Similarly, Figures 8 and 9 quantify the EE and SE metrics respectively. The main observation from these figures is that all these metrics improve when the central buffer size is reduced. This is because peak metrics are primarily impacted by the number of crossbars, which offer high storage and computation. Note that SE is a sum of neuron and synaptic density. Providing a large SRAM buffer increases neuron density (and is helpful to convolutional layers), but decreases synaptic density (not helpful to fully connected layers). Clearly, the latter effect is more dominant in this analysis of peak performance, so we see a drop in SE when CB size is increased. Figure 10 further breaks the SE metric into neuron and synaptic density.

While peak CE, EE, and SE are useful metrics and favor crossbar computation over neuron potential storage, deep networks with large convolutional layers benefit more from neuron potential storage. Therefore, ultimately, we need to evaluate INXS designs on state-of-the-art deep networks, e.g., VGG and MSRA. Figures 11 and 12 show the energy for different design points for these two workloads. These real workloads exhibit the best metrics when using larger central buffer sizes. Based on this analysis, we pick an ideal design point that does reasonably well for both workloads:

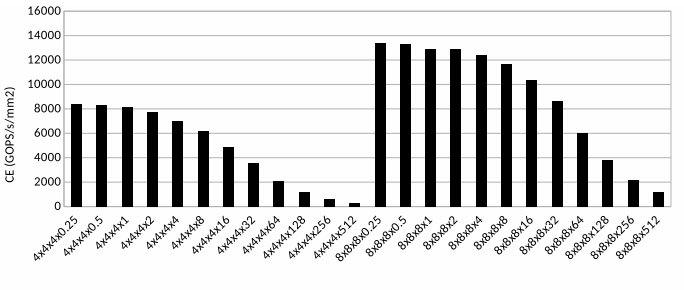

Fig. 7. Computational efficiency of INXS for various configurations.

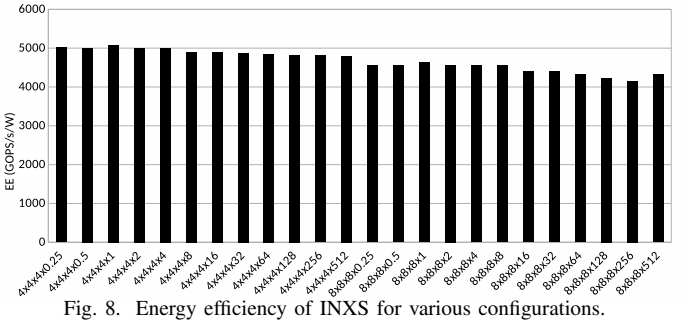

Fig. 8. Energy efficiency of INXS for various configurations.

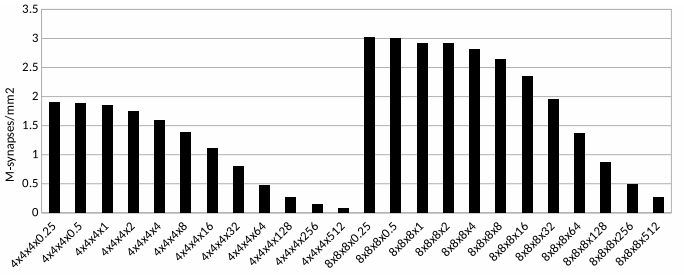

Fig. 9. Storage efficiency of INXS for various configurations.

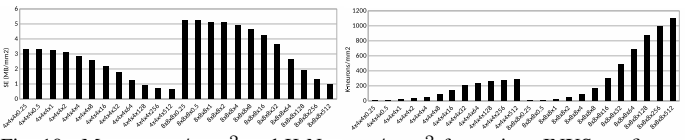

Fig. 10. M-synapses/mm^2 and K-Neurons/mm^2 for various INXS configurations.

8x8x8x128. For this ideal design point, Figures 13 and 14 show a breakdown of the area and energy required by the different layers of the deep networks. Table II summarizes the area and power of each component in an INXS tile.

Contrary to what we saw earlier for peak CE, EE, and SE metrics, the configurations with large central buffer perform well on MSRA and VGG-NET. The reason for this is the overhead of inter-tile and intra-tile interconnect energy. Configurations with small central buffers engage in more intra- and inter-tile communication which increases energy significantly. This is especially true for convolutional layers. Fully-connected classifier layers, on the other hand, benefit more from crossbars than from large buffers. But in these large workloads, the convolutional layers dominate (see per layer breakdown in Figures 13 and 14).

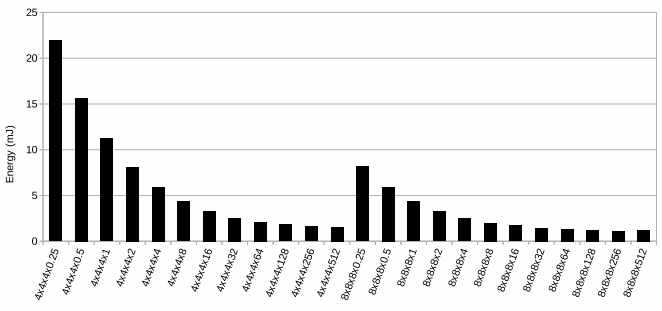

Fig. 11. Energy (for 1 image) estimates of different INXS configurations for VGG-NET.

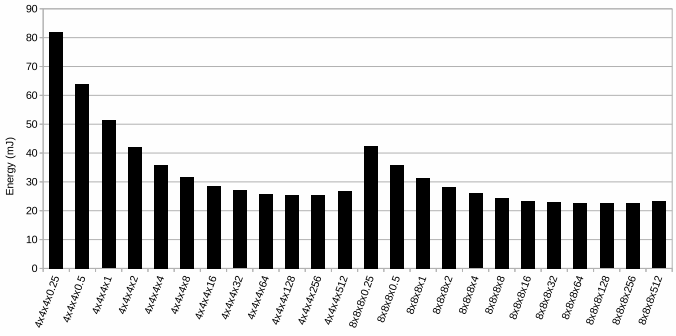

Fig. 12. Energy (for 1 image) estimates of different INXS configurations for MSRA.

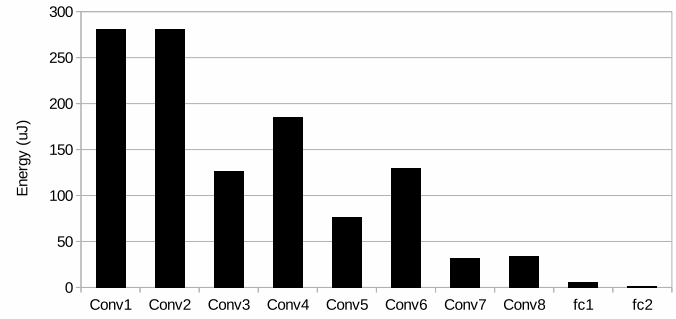

Fig. 13. Energy estimate of INXS (8x8x8x128) for different layers of VGG-NET.

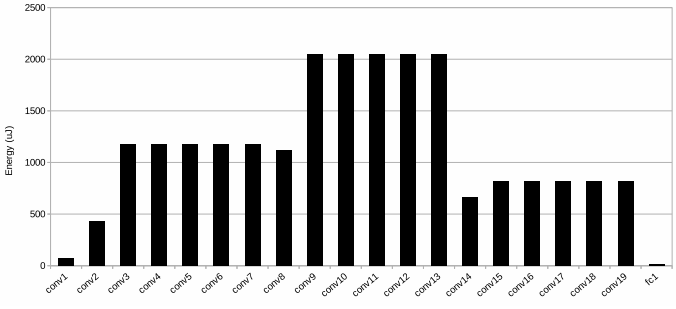

Fig. 14. Energy estimate of INXS (8x8x8x128) for different layers of MSRA.

B. Comparison to TrueNorth

Next, we compare INXS metrics to those of TrueNorth. Table III summarizes the key metrics. Because of the high parallelism in INXS, it achieves three orders of magnitude higher performance/area than TrueNorth. Because of the high ADC overhead, INXS has only a 10× improvement over TrueNorth in terms of EE. Using memristor xbars gives us significant density for synaptic storage compared to the SRAM storage used in TrueNorth. This also helps us eliminate the need for quantization of synaptic weights. INXS has a much larger advantage in terms of neuron density – it balances resources appropriately by recognizing that convolutional layers need large central buffers and low synaptic storage. The routing table employed in each neuron unit removes the severe constraints imposed by TrueNorth on the number of inputs/outputs to a neuron. As explained earlier, we achieve a more flexible architecture with the maximum number of inputs a neuron can receive being 512× higher than that TrueNorth.

Component	Description	Area (μm^2)	Power (mW)
Memristor xbar	512	8K	307.2
ADC	512	136K	1024
Shift and Add	512	30.7K	25.6
Router	128b Flit, 5-ports	253K	107
Input Buffer	4KB	489K	102.4
Output Buffer	2.5KB	36.6K	7.2
Central Buffer	128KB	1,738K	304
Interconnects	128b output bus, 128b ring bus	61K	135.7
Functional Units	8 adders and comparators	15K	4
Routing table	12.75KB	21K	12.1
Tile	64 xbars/SU, 8 SU, 8 NU	**2.8** mm^2	**2030 mW**

TABLE II
INXS AREA AND POWER BREAKDOWN (FOR ONE 8X8X8X128 TILE CONFIGURATION). SU: SYNAPTIC UNIT, NU: NEURON UNIT.

Metric	TrueNorth	INXS	Improvement
EE (GOPS/s/W)	400	4.1K	10.4x
CE (GOPS/mm^2)	0.703	2.2K	3129x
SE (MB)/mm^2	0.138	2.06	15x
ND (Neurons/mm^2)	2.73K	866K	363.5x
SD (Synapses/mm^2)	0.65M	1.497M	2.2x
Neuron connectivity	256	128K	512x

TABLE III
INXS COMPARISON WITH TRUENORTH. ND-NEURON DENSITY, SD-SYNAPTIC DENSITY

VI. CONCLUSIONS

In this work, we show that the use of a memristor crossbar can significantly accelerate the computations required by SNNs. The resulting architecture not only removes the constraints posed by the state-of-the-art TrueNorth architecture, it also surpasses TrueNorth on all metrics by one to three orders of magnitude. The INXS architecture takes a significant step in bridging the current large gap between ANN and SNN accelerators.

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] 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.

[5] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113(1), 2015.

[6] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *Proceedings of ASPLOS*, 2014.

[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] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of MICRO-47*, 2014.

[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] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-Column Deep Neural Networks for Image Classification," in *Proceedings of CVPR*, 2012.

[11] C.Yakopcic, M.Z.Alom, and T.M.Taha, "Memristor Crossbar Deep Network Implementation Based on a Convolutional Neural Network," in *Proceedings of IJCNN*, 2016.

[12] P. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing," in *Proceedings of IJCNN*, 2015.

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

[15] 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.

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

[17] B. Graham, "Fractional Max-Pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[18] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding Sources of Inefficiency in General-Purpose Chips," in *Proceedings of ISCA*, 2010.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv preprint arXiv:1502.01852*, 2015.

[20] A. Hodgkin and A. Huxley, "A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve," *Journal of Physiology*, no. 117(4), 1952.

[21] E. Hunsberger and C. Eliasmith, "Spiking Deep Networks with LIF Neurons," 2015, arXiv preprint 1510.08829.

[22] ——, "Training Spiking Deep Networks for Neuromorphic Hardware," in *arXiv preprint arXiv:1611.05141*, 2016.

[23] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware Spiking Neurons Design: Analog or Digital?" in *Proceedings of IJCNN*, 2012.

[24] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *Proceedings of DATE*, 2009.

[25] 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.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of NIPS*, 2012.

[27] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Brandli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s Single-Channel Asynchronous SAR ADC with Alternate Comparators for Enhanced Speed in 32 nm Digital SOI CMOS," *Journal of Solid-State Circuits*, 2013.

[28] 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.

[29] 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.

[30] W. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, no. 5, 1943.

[31] E. Menendez, D. Maduike, R. Garg, and S. Khatri, "CMOS Comparators for high-Speed and Low-Power Applications," in *Proceedings of ICCD*, 2006.

[32] P. Merolla, J. Arthur, R. Alvarez-Icaza, A. Cassidy, J. Sawada, F. Akopyan, B. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. Risk, R. Manohar, and D. Modha, "A Million Spiking-Neuron Integrated Circuit with a Scalable Communication Network and Interface," *Science, vol 345, no. 6197*, 2014.

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

[34] 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.

[35] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. Lee, J. M. Hernandez, Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, High-Accuracy Deep Neural Network Accelerators," in *Proceedings of ISCA-43*, 2016.

[36] 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.

[37] 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.

[38] 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.

[39] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[40] T. Taha, R. Hasan, C. Yakopcic, and M. McLean, "Exploring the Design Space of Specialized Multicore Neural Processors," in *Proceedings of IJCNN*, 2013.

[41] M. Talsania and E. John, "A Comparative Analysis of Parallel Prefix Adders," in *Proceedings of the International Conference on Computer Design (CDES)*, 2013.

[42] 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.

[43] C. Yakopcic and T. M. Taha, "Energy Efficient Perceptron Pattern Recognition using Segmented Memristor Crossbar Arrays," in *Proceedings of IJCNN*, 2013.