Approximate Computing for Spiking Neural Networks

Sanchari Sen, Swagath Venkataramani, Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{sen9,venkata0,raghunathan}@purdue.edu

***Abstract*—Spiking Neural Networks (SNNs) are widely regarded as the third generation of artificial neural networks, and are expected to drive new classes of recognition, data analytics and computer vision applications. However, large-scale SNNs (*e.g.*, of the scale of the human visual cortex) are highly compute and data intensive, requiring new approaches to improve their efficiency. Complementary to prior efforts that focus on parallel software and the design of specialized hardware, we propose AxSNN, the first effort to apply approximate computing to improve the computational efficiency of evaluating SNNs.**

In SNNs, the inputs and outputs of neurons are encoded as a time series of spikes. A spike at a neuron's output triggers updates to the potentials (internal states) of neurons to which it is connected. AxSNN determines spike-triggered neuron updates that can be skipped with little or no impact on output quality and selectively skips them to improve both compute and memory energy. Neurons that can be approximated are identified by utilizing various static and dynamic parameters such as the average spiking rates and current potentials of neurons, and the weights of synaptic connections. Such a neuron is placed into one of many approximation modes, wherein the neuron is sensitive only to a subset of its inputs and sends spikes only to a subset of its outputs. A controller periodically updates the approximation modes of neurons in the network to achieve energy savings with minimal loss in quality. We apply AxSNN to both hardware and software implementations of SNNs. For hardware evaluation, we designed SNNAP, a Spiking Neural Network Approximate Processor that embodies the proposed approximation strategy, and synthesized it to 45nm technology. The software implementation of AxSNN was evaluated on a 2.7 GHz Intel Xeon server with 128 GB memory. Across a suite of 6 image recognition benchmarks, AxSNN achieves 1.4-5.5× reduction in scalar operations for network evaluation, which translates to 1.2-3.62× and 1.26-3.9× improvement in hardware and software energies respectively, for no loss in application quality. Progressively higher energy savings are achieved with modest reductions in output quality.

***Index Terms*—Approximate Computing, Spiking Neural Networks, Approximate Neural Networks**

I. Introduction

The explosion in digital data, ushered in by the growth in the number of connected and intelligent devices, has led to the widespread use of machine learning algorithms to analyze, organize and draw inferences from data. Among the different algorithms developed over the years, Neural Networks (NNs) have demonstrated superior performance on a variety of image, video, audio and text analytics tasks [1], and are deployed in many real-world products [2], [3]. We focus on an emerging class of NNs, called Spiking Neural Networks (SNNs), which are often referred to as the 3^{rd} generation NNs. Compared to prior generations of NNs, SNNs exhibit higher biological fidelity *i.e.,* they mimic the spiking behavior of biological neurons. Therefore, SNNs have the potential to achieve better algorithmic performance with lower network complexity, especially in applications where temporal streams of data are processed [4]. SNNs are an active area of research, and in the recent past, SNNs have demonstrated state-of-the-art recognition performance on popular datasets such as MNIST [5] and CIFAR [6].

Computational Challenges. SNNs are compute and data intensive workloads. For example, spiking networks emulating the functionality of the visual cortex may contain over a million neurons and a billion synapses [7]. When used to process an image of size 256×256, this translates to ∼2 giga scalar operations per frame, and over 4 GB memory. With growth in data and the need for higher accuracy, these requirements are only expected to increase further in the future. Hence, exploring avenues to improve the energy efficiency of SNNs is fundamental to their adoption.

Realizing this need, prior approaches have explored three key directions for efficient realization of SNNs. The first is software parallelization on commercial platforms such as multi-cores and GPUs [8]–[11]. The event driven nature of SNNs makes software parallelization quite challenging. In SNNs, work is generated dynamically as neurons spike, which renders the control flow and memory access patterns irregular. In contrast, commercial platforms, such as GPUs, are optimized for regular memory access patterns and fine-grained SIMD parallelism. The second direction is to build hardware architectures specialized for SNNs. A range of architectures, from low-power IP cores [12] to large-scale systems [13], [14], have been proposed. The final set of efforts investigate alternate device technologies, such as memristor ans spintronics to realize SNNs [15]–[17].

Approximate SNNs. In this work, we explore a new direction - *approximate computing* - to improve the efficiency of SNNs. Due to their large-scale structure and the application context in which they are deployed, NNs are highly resilient to approximations in a significant fraction of their computations. Approximate computing has been applied to prior generations of (non-spiking) NNs [18]–[20]. However, due to the unique characteristics of SNNs (described below), such methodologies are not directly applicable. We believe ours to be the first effort to explore approximate computing for SNNs.

In SNNs, information is encoded and processed using trains of spikes. Each neuron is associated with a *membrane potential*, and a spike is dynamically generated when the potential goes above a specified threshold. When a neuron spikes, the potentials of all its fanout neurons are incremented by the weights of the respective connections. Thus *spike-triggered neuron updates* are the fundamental compute kernel in SNNs. To approximate SNNs, we develop a methodology, called AxSNN, to identify the criticality of spike-triggered updates and skip a subset of them to lower computational requirements thereby energy. AxSNN associates an approximation level with each neuron. The approximation level determines which fraction of a neuron's successors will be updated when it spikes, as well as which fraction of its inputs it is sensitive

Swagath Venkataramani is currently a research staff member at IBM T.J. Watson Research Center, Yorktown Heights, NY

This work was supported in part by the National Science Foundation under grant nos. 1423290 and 1320808 and in part by the Center for Computational Brain Research at IIT Madras.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

to. All update operations are carried out when a neuron at the most accurate (or least approximate) level spikes. Progressively fewer update operations are performed as the approximation level of the neuron is increased. Spikes are entirely skipped when the neuron is in its most approximate state. To determine the approximation level of the neuron at runtime, AxSNN estimates the probability of the neuron spiking, and the significance of its spikes. For this purpose, it utilizes static network-level characteristics such as the number of fanout paths from the neuron to SNN outputs and their average path weights, as well as dynamic local characteristics such as spike rate and current membrane potential of the neuron.

SNNs of any desired output quality can be realized using the above approach. We develop a framework that automatically tunes how aggressively neurons transition between approximation levels, thereby yielding an efficiency *vs.* quality trade-off. It is worth noting that the proposed approach is intrinsically input-adaptive *i.e.,* the approximate SNN modulates its computational effort across input samples, based on how often the neurons spike and the significance of the spikes to the eventual output.

In summary, the key contributions of this work are:

- We propose approximate computing as a new approach to improve the efficiency of SNNs.
- We develop a systematic approach to identify the criticality of spikes generated by each neuron at runtime. We correspondingly skip some or all updates due to the spike, thereby improving both compute and memory energy for a minimal loss in quality.
- We evaluate our approach in both software and hardware. For software, we utilize a C++ implementation of SNNs on a commodity server. In the case of hardware, we develop SNNAP, a Spiking Neural Network Approximate Processor. We achieve 1.2×-3.9× improvement in energy across a suite of 6 image recognition SNNs that contain ∼3K-14K neurons and ∼1.3M-48.8M connections.

The rest of the paper is organized as follows. Section II presents related research efforts. Section III provides the necessary background on SNNs. Sections IV describes the design approach and methodology. Section V details the SNNAP architecture. The experimental methodology is presented in Section VI followed by the results in Section VII. Section VIII concludes the paper.

II. Related Work

Our work lies at the intersection of two active fields of research, namely, efficient implementations of SNNs and approximate computing. In this section, we summarize previous efforts in both the domains and highlight the unique aspects of our work.

Efficient implementation of SNNs. Prior efforts that improve the efficiency of SNNs can be categorized into three classes. The first set of efforts explore methods to parallelize SNNs in software on commercial multi-cores [10], [11] and GPUs [7]–[9]. However the dynamic and event-driven nature of SNNs makes parallelization challenging, as it leads to irregular and unpredictable memory access patterns, and incurs significant communication and synchronization overheads. The second class of efforts design specialized hardware for SNNs to derive efficiency. A spectrum of architectures ranging from low power IP cores [12], reconfigurable fabrics [21], and mixed-signal implementations [22] to large-scale systems [13], [14] have been proposed. The third approach aims to realize SNNs using post-CMOS technologies such as spintronics and memristor crossbar arrays [15]–[17].

Our work is complementary to the above efforts, as we explore a new approach, approximate computing, to address the computational challenges imposed by SNNs.

Approximate computing. Applications from several important domains, including recognition, data mining, analytics, vision, search *etc.,* have the intrinsic ability to produce outputs of acceptable quality even when some of their computations are performed in an approximate or imprecise manner. Approximate computing leverages this forgiving nature to improve the efficiency of computing systems. A range of approximate design techniques, spanning circuits, architecture and software [23], have been proposed in recent years.

In the context of NNs, approximate computing has been previously applied to non-spiking networks in [18]–[20]. These approaches utilize backpropagation, one of the key steps involved in training non-spiking NNs, to characterize the criticality of neurons in the network, and correspondingly subject them to varying levels of approximation. Such approaches cannot be directly applied to SNNs, as their computational characteristics are quite different. For example, work is dynamically generated in SNNs as neurons spike. Therefore, it is difficult to statically determine an approximation level for a neuron and evaluate its impact on energy and output quality. Also, training mechanisms such as backpropagation are not applicable in the context of SNNs. Our work applies approximate computing to SNNs, and demonstrates significant energy benefits in both software and hardware.

III. Spiking Neural Networks: Preliminaries

SNNs are interconnected networks of primitive compute units, called neurons, that are organized in layers, with neurons in each layer connected to those in the layer succeeding it. The junction between connected neurons is called a synapse, which is associated with a parameter, the weight, that signifies the strength of the connection. The synaptic weights are learnt during the training process. In SNNs, as shown in Figure 1, information is represented and processed using *spikes*, which take a binary value 0 or 1. Inputs are presented to the neurons in the first layer as a time series of spikes. The spike trains propagate through the network until the output layer is reached. Each neuron in the output layer is associated with a class label, and the input is assigned the class corresponding to the output neuron that spiked the largest number of times. The number of time steps for which the SNN is evaluated is a key network parameter, and is determined during training.

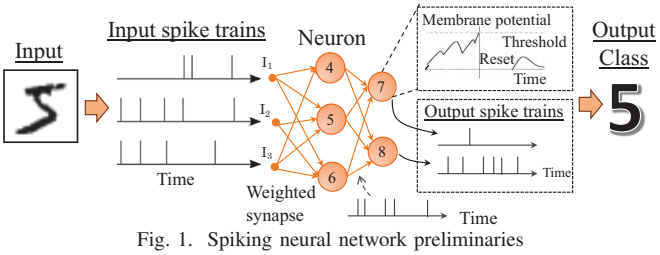

Fig. 1. Spiking neural network preliminaries

Several spiking neuron models, with varying levels of biological fidelity, have been proposed [24]. In this work, we consider the most commonly used model, called Leaky-Integrate-and-Fire (LIF); however, our approximation method is independent of the neuron model used in the SNN. The LIF

neuron has 3 key parameters *viz.,* the membrane, threshold and reset potentials. At the start of evaluation, the membrane potential is initialized to the reset value. Whenever a spike is observed at an input, the membrane potential is updated by the weight of the connection. In addition, in each time step, the membrane potential leaks (is decremented) by a fixed value. Mathematically, the LIF neuron is represented by Equation 1.

$$V_i(t) = V_i(t-1) - V_{leak} + \sum_j w_{ji} A_j(t) \quad (1)$$

where $V_i(t)$ is the membrane potential of neuron i at time t, V_{leak} is the leakage potential, w_{ji} is the weight of the synapse connecting neuron i and its input j, and A_j is a binary variable indicating whether input j spiked at time t. The LIF neuron produces an output spike whenever its membrane potential exceeds the threshold, following which the potential is re-initialized to the reset value.

The key compute primitive in SNNs is the set of updates triggered by a neuron spike. In this case, the potentials of all of its fanouts are updated by the respective synaptic weights. These updates may in turn cause new spikes to be generated and subsequently processed.

In summary, SNNs are event-driven workloads, wherein work is dynamically generated as neurons spike in the network. The set of updates triggered by each spike is the basic unit of work.

IV. AxSNN: Design Approach and Methodology

To address the computational challenges imposed by SNNs, we propose AxSNN, a new design approach that leverages approximate computing to improve their efficiency. In this section, we present the key concepts behind AxSNN and describe the design methodology in detail.

A. Approximating Spike-triggered Updates

As described in Section III,spike triggered neuron updates form the key compute primitive in SNNs. In our benchmark suite comprising of 6 image recognition SNNs, spike-triggered updates accounted for ∼97% of the overall operations, and consumed ∼93% of the total software execution time on a commodity Intel Xeon server. Therefore, we target these operations for approximation.

We associate an approximation level (α) with each neuron in the SNN. The approximation level (α) takes a value between 0 and 1, and is modulated dynamically during network evaluation. Based on α, we approximate the update operations associated with the neuron, as shown in Figure 2. An α of 1 indicates that the neuron is at its highest accuracy level, in which case all its fan-in and fan-out connections are active. As α is reduced, the neuron is progressively made more approximate by only enabling a fraction α of its fan-in and fan-out connections to be active. In this case, when the neuron spikes, only its active fan-out connections are updated, while the rest are skipped. Similarly, its membrane potential is updated only when one of its active fan-in connections spike. By dynamically deactivating input and output connections of a neuron, we reduce computation and save energy.

An important aspect of our approach is that we eliminate connections in a significance-aware manner, *i.e.,* based on synaptic weights. Once the SNN is trained, we pre-sort input and output connections to each neuron in increasing order of weight magnitude and deactivate the ones with lower magnitude first. For ease of implementation, we restrict the number of number of approximation levels to 5 *viz.* 1, 0.5, 0.25, 0.125 and 0. Note that α of 0 indicates that the neuron is completely removed from the network as none of its fan-in and fan-out connections are active.

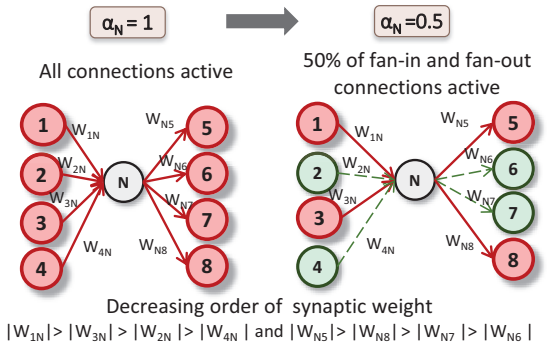

Fig. 2. Neuron approximation mechanism

B. AxSNN: Overview

With the aforementioned approximation mechanism in place, we now present the overall approximation strategy adopted in AxSNN, as illustrated in Figure 3. The proposed strategy is dynamic, *i.e.,* the approximation levels of different neurons are determined at runtime in the course of evaluating an input. As shown in Figure 3, at the start of evaluation (t = 0), all neurons are set to their most accurate level (α = 1). We augment the SNN with an *AxSNN controller*, which is invoked periodically after every λ time steps. The AxSNN controller loops through each neuron in the network and determines the approximation level with which it should be executed for the next λ time steps. To make this decision, the AxSNN controller considers several key factors as described below.

1) Determining Approximation Levels: In order to determine the approximation level of a neuron, the AxSNN controller estimates the potential impact of approximations on the overall output quality using two key factors: (i) Spike probability, which captures the probability of the neuron spiking in the next λ time steps, and (ii) Spike significance, which denotes the relative importance of the neuron's spike to the overall network. Since the AxSNN controller is invoked periodically during execution, the above factors need to be estimated with very low overhead, as they directly offsets the benefits derived from approximate computing.

The AxSNN controller utilizes a mix of static and dynamic parameters to determine the spike probability ($SpikeProb$) and significance ($SpikeSig$) of each neuron. For a neuron to spike, its membrane potential should exceed its threshold value. Therefore, to estimate the $SpikeProb$ at runtime, as shown in Equation 2, we first identify how far the neuron's current potential is from its threshold, normalized to the reset value. We then divide the rate at which the neuron spiked in the past by the normalized potential difference to compute $SpikeProb$.

$$SpikeProb = \frac{SpikeRate}{(Thresh. - Potential)/(Thresh. - Reset)} \quad (2)$$

Intuitively, from Equation 2, the spike probability of a neuron is higher if it has spiked frequently in the past, or if its potential is close to the threshold value. $SpikeSig$, as shown in Equation 3, is computed as the product of the number of

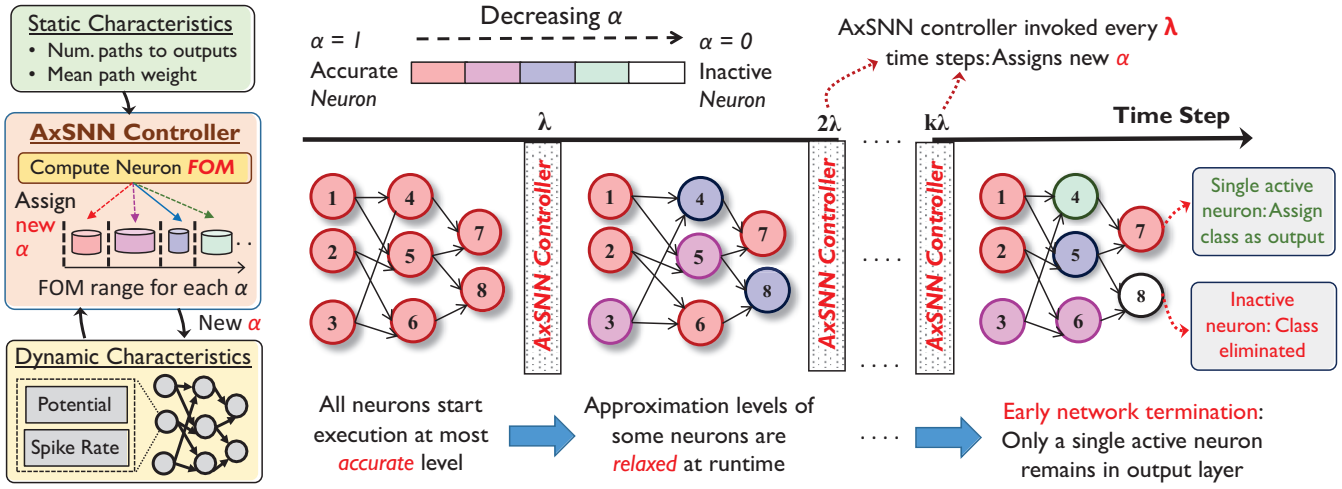

Fig. 3. Overview of approximation strategy in AxSNN

paths connecting the neuron to the network outputs and the mean of all path weights.

$$SpikeSig = NumPathsToOutputs * MeanPathWeight \quad (3)$$

We note that, since the number of paths and the mean path weight remain constant across all time steps, we can pre-calculate $SpikeSig$ for each neuron once the SNN is trained. $SpikeProb$ and $SpikeSig$ are combined into a single Figure-Of-Merit (FOM), as shown in Equation 4.

$$FOM = SpikeProb * SpikeSig \quad (4)$$

The AxSNN controller contains a set of pre-defined FOM ranges for each approximation level. Based on the range in which the FOM of a neuron falls, the AxSNN controller assigns it the corresponding approximation level. Given an output quality requirement, the methodology used to obtain the FOM ranges is described in Section IV-C.

2) Early Network Termination: Another key aspect of the proposed approximation strategy is that it enables the SNN to classify an input even before all the time steps are complete. We note that, in the most approximate level, the neuron is completely disconnected from the network and no further spike activity can occur at its output. Therefore, when all but one neuron in the output layer reach an α of 0, the execution is terminated and the input is assigned the class of the neuron that is active.

C. AxSNN: Design Methodology

We now describe how the FOM range for each approximation level is obtained. The FOM ranges determine how aggressively neurons transition across approximation levels, leading to different points in the efficiency *vs.* quality space. At the finest granularity, the FOM ranges can be defined individually for each neuron in the network. However, this leads to a prohibitively large design space, and further incurs significant overhead to store the FOM ranges in the AxSNN controller. We address this challenge by leveraging the fact that neurons in a layer are computationally similar, and constrain them to utilize the same FOM ranges. In other words, the FOM range for each approximation level is defined layer-wise.

We constrain the search space further by imposing a constraint that the FOM range endpoints for the different approximation levels are spaced in the proportion to the value of α. For example, the threshold to transition from $\alpha : 0.5 \rightarrow 0.25$ is constrained to be half of the threshold to transition from $\alpha : 1 \rightarrow 0.5$ and so on. This simplifies the search to finding one parameter per layer, which represents the threshold to transition from the most accurate level to the first approximate level ($FOM_{\alpha:1\rightarrow a1}$).

Algorithm 1 presents the pseudocode to find $FOM_{\alpha:1\rightarrow a1}$ for each layer. A trained SNN, the training dataset and the output quality constraint are provided as inputs. We first identify the maximum FOM for each layer, by setting its $SpikeProb$ to 1 (Line 2), and initialize its $FOM_{\alpha:1\rightarrow a1}$ to this value (Line 3). We then iteratively search the space of $FOM_{\alpha:1\rightarrow a1}$ as follows (Lines 4-9). For each layer, the corresponding $FOM^{Li}_{\alpha:1\rightarrow a1}$ is decreased by a small constant Δ, and the energy (E_{Li}) and quality (Q_{Li}) of the resultant AxSNN is computed by evaluating it on the training dataset (Lines 5-6). Amongst these, we commit to the change in $FOM^{Li}_{\alpha:1\rightarrow a1}$ for the layer with the minimum E_{Li}/Q_{Li} ratio (Line 8). This process is repeated until Q_{Li} for none of the layers meet the specified quality constraint (Line 7).

Algorithm 1 Identifying FOM transition thresholds

Input: SNN: Trained spiking network, $TrData$: Training dataset, Q: Quality constraint
Output: $FOM^{Li}_{\alpha:1\rightarrow a1}$: Threshold to transition from most accurate to first approximate level for each layer

1: **For** each Layer Li:
2: Compute Max. FOM FOM^{Li}_{max}
3: Initialize $FOM^{Li}_{\alpha:1\rightarrow a1}$ = FOM^{Li}_{max}
4: **while** (1) **do**
5: **For** each Layer Li:
6: Set $FOM^{Li}_{\alpha:1\rightarrow a1}$ -= Δ and compute E_{Li}, Q_{Li}
7: **if** ($Q_{Li} < Q \ \forall \ Li$) **break**
8: Commit $FOM^{Li}_{\alpha:1\rightarrow a1}$ in layer with min. (E_{Li}/Q_{Li}) and $Q_{Li} > Q$
9: **end while**

In summary, by selectively skipping updates triggered by spikes, AxSNN achieves significant improvements in the implementation efficiency of SNNs.

V. SNNAP: Architecture

In this work, we evaluate AxSNN, using both hardware and software SNN implementations. To demonstrate the benefits in hardware, we propose Spiking Neural Network Approximate Processor (SNNAP), a new hardware architecture for SNNs,

enhanced to support the proposed approximation mechanism. Figure 4 shows the block diagram of SNNAP. It consists of two types of processing units: (i) a scalar Leak-and-Spike (LnS) unit, and (ii) a 1D array of Spiking Neuron Processing Elements (SNPEs). The architecture also contains two memory banks, the State Memory (SM) and the Weight Memory (WM), which store the neuron potentials and the weights respectively. A global controller orchestrates the overall execution.

We now describe how SNNs are realized in SNNAP. The LnS unit loops over all neurons in all time steps, leaks its membrane potential and checks if the potential is above its threshold. If not, the execution moves on to the next neuron. In the case the neuron spiked, the SNPE array is activated, which reads the SM and WM banks and updates the potentials of the fan-out neurons. The network state and weights are statically partitioned across the SM and WM banks, such that the compute load to each SNPE is roughly balanced.

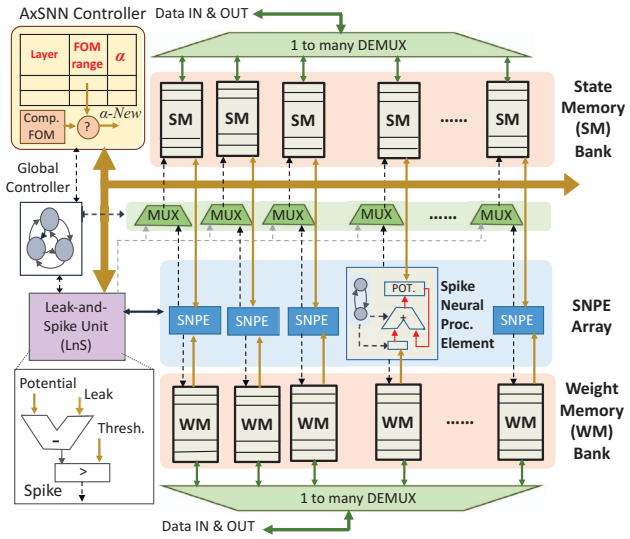

Fig. 4. Block diagram of SNNAP

To support approximate operation, SNNAP is enhanced with an AxSNN controller that is periodically invoked by the global controller. We also add 3 bits to the neuron states in the SM banks to store their approximation levels. Further, the weights in WM are sorted in accordance to their magnitude, such that updates can be skipped with no overhead when neurons transition approximation levels. Overall, SNNAP incurs ∼3.5% area overhead to support approximate operation.

VI. Experimental Methodology

In this section, we describe the methodology used in our experiments to evaluate AxSNN.

Runtime and Energy Evaluation. We evaluate AxSNN in both hardware and software. The software was implemented in C++ and run on a 2.7GHz Intel Xeon server with 128GB memory. In the case of hardware, the SNNAP architecture was implemented at the Register Transfer Level (RTL) using Verilog HDL and synthesized to IBM 45nm technology using Synopsys Design Compiler. The micro-architectural and circuit level parameters of the implementation are shown in Figure 5a. We measured performance through cycle-accurate simulations using ModelSim, and the switching activity traces were fed to Synopsys Power Compiler to estimate (static and dynamic) logic power at the gate-level. Energy was computed as the product of power and execution time. CACTI was used to model the memory structures. The memory energy was computed by profiling the number of memory reads and writes and multiplying them with the corresponding energy values obtained from CACTI.

Metric	Value
Feature Size	45nm
Area	0.34 mm^2
Power	175.74 mW
Gate Count	193723
Frequency	1 GHz
No. of SNPE lanes	64

(a)

Dataset	Type	Layers	Neurons	Connections
MNIST	F.C	4	3194	2392800
	Conv.	6	13594	6527300
NORB	F.C	4	3053	1276500
	Conv.	6	89806	4012960
SVHN	F.C.	5	7582	11149000
CIFAR-10	F.C.	4	14082	48854400

(b)

Fig. 5. SNNAP parameters and application benchmarks

Application Benchmarks. Our benchmark suite, listed in Figure 5b, comprises of 6 image recognition SNNs, of which two are convolutional networks and the rest are fully-connected networks. Figure 5b also lists the number of layers, neurons and connections in each benchmark. The networks were trained using the methods described in [6] and [5]. We utilized classification accuracy, *i.e.,* fraction of inputs classified correctly, as our metric to evaluate application quality.

VII. Results

This section presents the results of our experiments that demonstrate the benefits of AxSNN in both HW and SW.

A. Energy Benefits at Iso-Accuracy

Figure 6 shows the normalized benefits using AxSNN with no loss in accuracy across all benchmarks. The improvements are quantified using three metrics: (i) number of spike update operations, (ii) hardware energy, and (iii) software energy. We achieve 1.4×-5.5× reduction in spike update operations across the benchmarks, which translates to 1.2×-3.62× and 1.26×-3.9× improvement in hardware and software energies respectively, at iso-accuracy. The benefits largely depend on the difficulty of classifying the inputs in each dataset. For example, in the case of MNIST where we observe the most improvements, almost 99% of the inputs can be terminated early. In contrast, only 63% and 52% of inputs are amenable to early termination in CIFAR and SVHN respectively. The hardware and software energies include the energy spent in performing other control and compute operations involved in SNN evaluation in addition to the spike update operations. Further, they also reflect the overheads associated with realizing the approximation methodology. Due to the above reasons, the reduction in the spike update operations don't translate entirely into the hardware and software benefits.

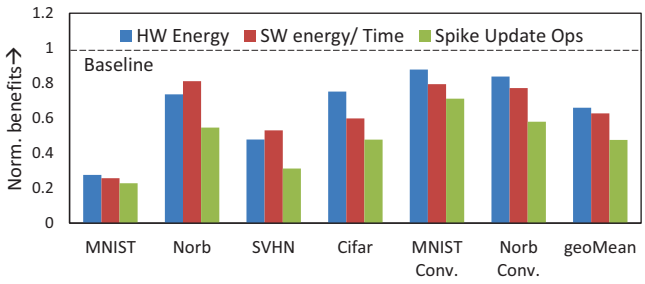

Fig. 6. Normalized OPS and energy benefits for different applications

B. Energy vs. Accuracy Tradeoff

By modulating how aggressively the neurons transition to higher approximation levels, different application-level qualities can be achieved in AxSNN with corresponding benefits in energy. Figure 7 shows the energy *vs.* quality trade-off

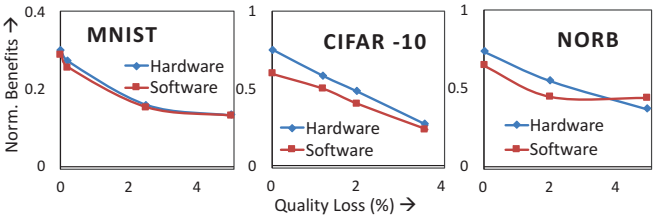

Fig. 7. Normalized energy *vs.* accuracy trade-off for 3 SNN benchmarks

curves thus achieved for 3 benchmarks. On an average, we obtain 1.86×, 2.51× and 3.37× energy improvement for a quality loss of 1%, 2% and 5% respectively. Since modulating application quality only requires a change in the FOM ranges set in the AxSNN controller, the same implementation can easily support multiple application quality levels and can switch between them at runtime.

C. Input Adaptive Approximations: Easy vs. Hard Inputs

An important aspect of AxSNN is that the approximations are intrinsically input adaptive. Inputs that are easy-to-classify are approximated more than the harder inputs, thereby scaling computational effort in proportion to input difficulty. We illustrate this in Figure 8, using 2 examples from the MNIST handwritten digit recognition dataset. Figure 8 plots the approximation level of each neuron in the SNN at each time step. Red denotes the most accurate level, while white indicates the neuron is inactive. The AxSNN controller is invoked after every 5 time steps, and the neurons may switch approximation levels after this interval. For ease of understanding, we plot the approximation level for the output neurons separately (graph on the right for both inputs).

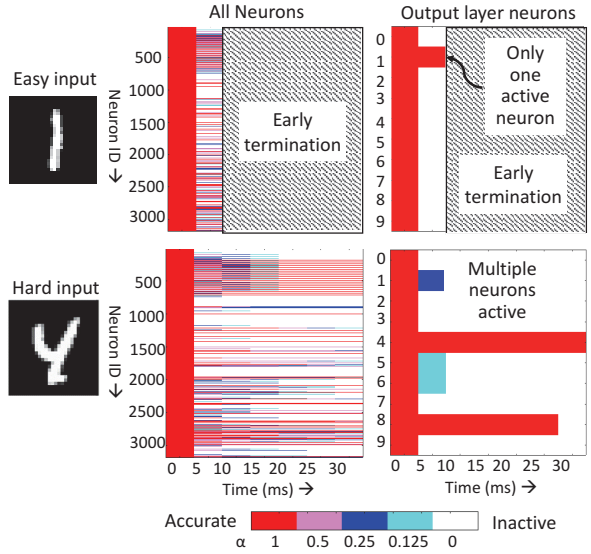

Fig. 8. Approximation levels of neurons at each time step

We observe that all neurons start execution (t=0) in their most accurate state. However, after the AxSNN controller is first invoked (t=5), a substantial fraction of the neurons change approximation levels. Specifically, we observe that several output neurons have been rendered inactive, effectively eliminating the respective class labels from consideration. For example, in the case of the easy input, except for the class '1', neurons corresponding to all other classes are inactive. Therefore, the execution is terminated and the input is classified. For the hard input, 5 classes are in contention, albeit with their neurons at different approximation levels. This reduces to 2 class labels ('4' and '8') at t=10, and the input is classified after 35 time steps. We still achieve significant benefits in the case of the hard input, as neurons in other layers are progressively approximated.

VIII. Conclusion

Spiking neural networks (SNNs) are an emerging class of neural networks that have demonstrated significant promise in realizing several recognition, data analytics and computer vision applications. To improve the implementation efficiency of SNNs, we utilize approximate computing, a design paradigm in which selected computations are performed in an approximate manner, saving energy with minimal loss in quality. We identified updates triggered by a neuron spike as the key compute primitive in SNNs. We target our approximations at this primitive by skipping some or all updates caused by a spike. We develop a methodology, AxSNN, to identify the spike triggered neuron updates that can be skipped while meeting the specified output quality requirement. Across a suite of 6 image recognition SNN benchmarks, we demonstrate significant benefits in energy for both hardware and software implementations.

References

[1] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61, 2015.
[2] Y. Taigman *et al.* Deepface: Closing the gap to human-level performance in face verification. In *Proc. CVPR*, June 2014.
[3] How google translate squeezes deep learning onto a phone research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html.
[4] J. Vreeken. Spiking neural networks, an introduction. Technical report, 2003.
[5] P. Diehl *et al.* Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proc. IJCNN*, 2015.
[6] Y. Cao *et al.* Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vision*, 113(1), May 2015.
[7] J. L. Krichmar *et al.* Large-scale spiking neural networks using neuromorphic hardware compatible models. *JETC*, 11(4), April 2015.
[8] J. M. Nageswaran *et al.* Efficient simulation of large-scale spiking neural networks using cuda graphics processors. In *Proc. IJCNN*, 2009.
[9] A. Fidjeland *et al.* Accelerated simulation of spiking neural networks using gpus. In *IJCNN*, 2010.
[10] A. Morrison *et al.* Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural Comput.*, 17(8), 2005.
[11] R. Ananthanarayanan *et al.* Anatomy of a cortical simulator. In *Proc. Supercomputing*, SC '07, 2007.
[12] T. Schoenauer *et al.* Neuropipe-chip: A digital neuro-processor for spiking neural networks. *Trans. Neur. Netw.*, 13(1).
[13] S. B. Furber *et al.* Overview of the spinnaker system architecture. *IEEE Trans. Comput.*, 62(12), December 2013.
[14] P. *et al.* Merolla. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197), 2014.
[15] D. Kuzum *et al.* Synaptic electronics: materials, devices and applications. *Nanotechnology*, 24(38), 2013.
[16] X. Liu *et al.* Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators. *IEEE Trans. on CAS*, 63(5):617–628, May 2016.
[17] A. Sengupta *et al.* Magnetic tunnel junction mimics stochastic cortical spiking neurons. *Scientific Reports*, 6, July 2016.
[18] S. Venkataramani *et al.* Axnn: Energy-efficient neuromorphic systems using approximate computing. In *Proc. ISLPED*, 2014.
[19] Q. Zhang *et al.* Approxann: An approximate computing framework for artificial neural network. In *Proc. DATE*, 2015.
[20] Z. Du *et al.* Leveraging the error resilience of neural networks for designing highly energy efficient accelerators. *IEEE TCAD*, Aug 2015.
[21] A. S. Cassidy *et al.* Design of silicon brains in the nano-cmos era: Spiking neurons, learning synapses and neural architecture optimization. *Neural Netw.*, 45, September 2013.
[22] B. V. Benjamin *et al.* Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE*, 102(5), May 2014.
[23] S. Venkataramani *et al.* Approximate computing and the quest for computing efficiency. In *Proc. DAC*, 2015.
[24] E. M. Izhikevich. Which model to use for cortical spiking neurons? *Neural Networks, IEEE Trans. on*, 15(5), September 2004.