Journal of Signal Processing Systems
https://doi.org/10.1007/s11265-020-01546-x

CyNAPSE: A Low-power Reconfigurable Neural Inference Accelerator for Spiking Neural Networks

Saunak Saha[1] · **Henry Duwe**[1] · **Joseph Zambreno**[1]

Received: 30 November 2019 / Revised: 17 March 2020 / Accepted: 5 May 2020

Abstract

While neural network models keep scaling in depth and computational requirements, biologically accurate models are becoming more interesting for low-cost inference. Coupled with the need to bring more computation to the edge in resource-constrained embedded and IoT devices, specialized ultra-low power accelerators for spiking neural networks are being developed. Having a large variance in the models employed in these networks, these accelerators need to be flexible, user-configurable, performant and energy efficient. In this paper, we describe CyNAPSE, a fully digital accelerator designed to emulate neural dynamics of diverse spiking networks. Since the use case of our implementation is primarily concerned with energy efficiency, we take a closer look at the factors that could improve its energy consumption. We observe that while majority of its dynamic power consumption can be credited to memory traffic, its on-chip components suffer greatly from static leakage. Given that the event-driven spike processing algorithm is naturally memory-intensive and has a large number of idle processing elements, it makes sense to tackle each of these problems towards a more efficient hardware implementation. With a diverse set of network benchmarks, we incorporate a detailed study of memory patterns that ultimately informs our choice of an application-specific network-adaptive memory management strategy to reduce dynamic power consumption of the chip. Subsequently, we also propose and evaluate a leakage mitigation strategy for runtime control of idle power. Using both the RTL implementation and a software simulation of CyNAPSE, we measure the relative benefits of these undertakings. Results show that our adaptive memory management policy results in up to 22% more reduction in dynamic power consumption compared to conventional policies. The runtime leakage mitigation techniques show that up to 99.92% and at least 14% savings in leakage energy consumption is achievable in CyNAPSE hardware modules.

Keywords Neuromorphic · Spiking neural networks · Reconfigurable · Accelerator · Memory · Caching · Leakage · Energy efficiency

1 Introduction

While deep neural networks provide state-of-the-art performance in classification, regression and even generative tasks, they have to pay steep dividends when deployed on conventional architectures [8]. Recently, there has been an unprecedented increase in the depth of neural networks owing to their application in extremely complicated tasks of perception and generation [45]. As these networks grow wider and deeper, the number of processing elements (i.e., neurons) grow substantially and the number of learnable parameters can grow up to quadratically with respect to the number of processing elements. This makes them extremely demanding in terms of silicon real estate, especially memory, as well as compute performance and power. To bring this computation closer to the edge in resource-constrained devices, recently there has been considerable interest in building special-purpose hardware accelerators to support inference [12, 35, 52, 60], training [44, 63] as well as compilers to bridge the gap between software simulation and

✉ Saunak Saha
saha@iastate.edu

Henry Duwe
duwe@iastate.edu

Joseph Zambreno
zambreno@iastate.edu

[1] Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Published online: 19 June 2020

hardware acceleration [62]. However, while microarchitectural techniques have been able to improve on the efficiency of neural network processing, it is nowhere near the biological neocortex, which is not only substantially deeper and wider but is also significantly more efficient in terms of energy and data [3].

The major inefficiency of these networks result from continuous and expensive computational primitives at every discrete timestep of the simulation. *Spiking neural networks (SNNs)* attempt to marry the approaches of computational neuroscience and deep learning by using more biologically accurate processing elements, *spiking neurons*. SNNs are extremely energy efficient, fast, noise-invariant and give great insight into neuroscientific understanding. However, the processing substrate for SNNs in common use today, which can both accelerate their applications as well as exploit their advantages, is completely different from artificial neural network accelerators.

Introduced by Carver Mead in 1990 [47], *neuromorphic engineering* has concerned itself with computing fabric that is able to emulate biologically plausible dynamics so as to perform efficient processing of neural information. Neuromorphic hardware acceleration has been achieved by both mixed-signal and digital hardware [57]. While analog hardware can emulate biological realism and energy efficiency to a much greater extent [49, 54, 61], they are plagued by process, voltage and temperature (PVT) variations and is especially difficult to scale to today's technology nodes [40]. Digital implementations provide these advantages and are suitable for integration to embedded systems and software ecosystems [1, 13, 50]. Neural response latency in digital circuits is orders of magnitude lower than the diffusion time of ions across the biological membrane. Hence, neural ensembles in silicon can achieve faster-than-real-time performance. However, the usefulness of a digital neural accelerator is strongly dependent on its energy efficiency. A common approach is to trade-off some inference latency if that can lead to improved energy consumption. To that end, this paper makes the following contributions:

1. We present CyNAPSE: a digital SNN accelerator with reconfigurable network topology and flexible generalized neural dynamics.
2. We study the memory access patterns of several SNN workloads and observe that spike processing is predominantly memory-intensive with respect to power consumption.
3. We propose a memory management strategy to reduce the power consumption resulting from redundant memory accesses in the baseline. Results range from 13-44% power savings over the baseline and 8-23% over best conventional replacement policies.
4. We observe that large amounts of on-chip power consumption is accounted for by static leakage owing to the predominantly idle resources typical of an SNN inference fabric.
5. Accordingly, we implement simple runtime leakage control techniques to arrest up to 98% of leakage energy in CyNAPSE modules that could be especially relevant in multicore manifestations of the accelerator.

The rest of the paper is organized as follows. Section 2 covers a motivational account for SNNs in general, describes our generalized neuron model and lists the benchmaks used in the study. Section 3 describes the CyNAPSE system, its working in brief and other related IP developed and used in this work. Section 4 attempts to explain the problem at hand and presents the proposed efficient memory management scheme in detail. Section 5 describes our implementation of runtime leakage control techniques to arrest on-chip energy consumption and prescribes the experimental approach to arrive at the results. We evaluate our work and present the results in Section 7 and conclude in Section 8

2 Spiking Neural Networks

Classical Artificial Neural Network (ANN) models can be defined as topological variations of the Multilayer Perceptron (MLP) where a large number of *perceptrons* or artificial neurons are connected in different ways. Depending on connectivity, these networks can have *dense* fully connected layers or *convolutional* sparsely connected layers. The flow of information can be unidirectional as in *feed-forward* layers or bidirectional as in *recurrent* layers. For computer vision applications, some specialized layers of computation are commonly employed between these layers, such as *pooling*, *normalization* etc. However, the basic unit of computation in these networks are these artificial neurons employed to encode and compute data in a collective fashion. Figure 1 shows the basic computation of such a neuron. In ANNs, a neuron j in layer L receives N floating-point inputs ${x_i}^{L-1}$ and a single floating-point bias b^L. Each connection into this neuron is weighted by a floating-point weight ${w_{i,j}}^L$. The total input to this neuron is therefore a weighted sum of all inputs on which it applies an *activation* ${f_{act}}^L$ or nonliearity to compute its output and propagates the same to the subsequent layer. This output ${\phi_j}^L$ can be given by:

$$ {\phi_j}^L(\mathbf{x}) = {f_{act}}^L\left(\sum_{i=1}^{N} {x_i}^{L-1} {w_{i,j}}^L + b^L\right) \tag{1} $$

Therefore, to compute the input to all neurons of layer L, a floating-point *General Matrix Vector* (GEMV)

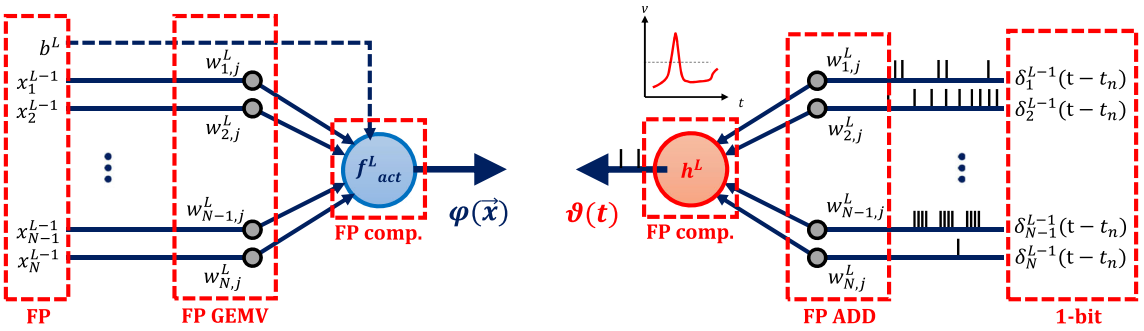


Figure 1 Schematic descriptions of a typical perceptron or artificial neuron (left) compared to the working of a spiking neuron (right) in their respective networks.

product is computed between the input vector $\mathbf{x}$ and the weight matrix W before the respective nonlinearities can be applied. This is followed by a floating-point computation to apply the activation. These inputs are communicated continuously at every discrete timestep of the inference and therefore require large synchronous floating point GEMV operations for the entirety of the simulation period. Furthermore, the inherent nature of computation in these neurons is spatial and therefore, without the added complexities of recurrent connections, fail to capture any temporal variance in the data distribution. ANNs learn using *backpropagation* of classification errors and tuning of weights using gradient descent. Not only is gradient descent difficult to implement in hardware emulation [48], but requires labeled data, the dearth of which is well acknowledged [24].

However, these ANNs are only high-level abstractions of neural inference as it is believed to occur in their biological counterparts. For example, biological neural cultures have shown to have very inexpensive, error-tolerant and sustainable computation [3]. Contrary to perceptrons, biological neurons encode and process real-world information through *spikes*. Spikes or action potentials are transient excursions of a neuron's membrane voltage above its usual range of operation [25, 26]. Information is communicated amongst neurons in the precise timing of these spikes rather than the morphology of the signal itself. Therefore, these communications can be represented in a temporal distribution of binary all-or-nothing signals. These signals flow through *synapses* or connections between neurons with a synaptic weight that determines the efficacy of signal transfer [58]. A *population* of neurons and synapses form *Spiking Neural Networks* (SNNs). These are biologically plausible networks that use low-level mathematical abstractions of real neurons to develop energy-efficient alternatives to classical network architectures. Figure 1 shows a single spiking neuron in an SNN topology. Temporal distribution of spikes $\sum_n {\delta_i}^{L-1}(t-t_n)$ from the previous layer arrive as a weighted sum via synaptic weights ${w_{i,j}}^L$ into a spiking neuron j of the layer L. At a given timestep of the simulation t_n, all incoming spike inputs will contribute to the membrane voltage of the neuron $v(t)$ according to the spiking neuron model function h^L as:

$$ {v_j}^L(\mathbf{x}, t_n) = h^L\left[\sum_{i=1}^{N}\left(\sum_n {\delta_i}^{L-1}(t-t_n)\right){w_{i,j}}^L\right] \tag{2} $$

where h^L is mathematical model of the neuron that describes the integration of inputs into the membrane voltage and conditional generation of action potential typically following a trend as shown in the figure (inset). Note that computation using binary spikes alleviate the need for large expensive GEMV operations. In a single timestep, now the processing is reduced to conditional floating-point adds. There is no need for computation at every timestep, rather only when there is a spike from the preceding layer. Since spikes are sparse events depending on the input intensity and neuron parameters, this not only relieves the computation requirements but also brings an inherent temporal nature in the computation, even without explicit recurrent connections. It should also be emphasized that SNNs are more amenable to *unsupervised* learning using variants of *Spike Timing Dependent Plasticity* or STDP where the synaptic weights are dynamically tuned according to the difference in timing of post-synaptic and pre-synaptic action potentials [7]. Although this paper is concerned primarily with inference, reducing the dependence on labeled data should be highlighted as one of the driving forces behind emerging interest in SNNs [15, 36].

2.1 Neuron Model

Spiking neurons can be described in varying levels of detail ranging from biophysically accurate and computationally

expensive models [25] to simpler phenomenological models that trade off some biological detail for tractable simulation of large SNNs [19, 30]. One popular example of the latter is the Leaky Integrate and Fire (LIF) model [19, 41]. It empirically reproduces the generation of neuron action potential via imperfect (or leaky) integration of synaptic inputs into a thresholded membrane voltage. It is a *point* neuron model in that it accounts for temporal integration of spikes without attention to spatial variation in these integrations within a single simulated neuron. This simple abstraction provides an excellent balance between biological plausibility and large scale tractability for application in SNNs for object recognition [15, 16, 51]. Further, to agree with various neuroscientific observations, the LIF model has been equipped with different application-specific parameters to produce mathematical variants [9, 20, 34]. Since our approach is to cater to a wide range of applications without losing generality, we have adopted a *generalized* and hardware reconfigurable model of the LIF [33]. This model can be mathematically described as follows:

$$R_m C_m \frac{dV_m}{dt} = -g_l(V_m(t) - V_{rest}) - g_{Na}(t)(V_m(t) - V_{rNa}) - g_k(t)(V_m(t) - V_{rK}) \quad (3)$$

where $R_m C_m$ can be collectively denoted as τ_m which signifies the time constant of exponential leak of the membrane voltage. The voltage leaks through a constant conductance of g_l with a tendency to return to its resting potential V_{rest}. Any synaptic input from connected neurons arrive either as Excitatory Post Synaptic Current (EPSC) through a time-varying sodium conductance g_{Na} or as Inhibitory Post Synaptic Current (IPSC) through a potassium conductance g_K. These terms mimic the sodium and potassium *ion channels* of a biological neuron having reversal potentials V_{rNa} and V_{rK} respectively, upon reaching which, the respective currents cease to integrate. The dynamic behavior of these ion channel conductances themselves follow the equations:

$$\tau_{ion} \frac{dg_{ion}(t)}{dt} = -g_{ion} t \quad (4)$$

that describes the decay of conductances similarly as Eq. 3. The action potential is generated by a simple thresholding and reset behavior:

$$S_i(t) = \begin{cases} 0, & V_m(t) < V_{th} \\ 1, & V_m(t) \geq V_{th} \end{cases} \quad (5)$$

$$V_m(t) = \begin{cases} V_{reset}, & t_n \leq t \leq t_n + t_{ref} \\ V_m(t), & otherwise \end{cases} \quad (6)$$

where $S_i(t)$ is the binary spike signal as a result of thresholding and t_n is the time whenever the neuron membrane voltage reaches the threshold potential V_{th} going into a *refractory period* t_{ref} during which it exhibits a state of *hyperpolarization* and sits at a reset potential V_{reset} thereby ignoring any synaptic effects during such a time. For a detailed discussion on neuron model parameters, their origin and significance, we point the readers to [19]

The generalized LIF model affords, among other factors, *conductance-based synapses* and *voltage-dependent current integration*, two widely accepted ingredients for biological plausibility in real neurons [36]. As a testimonial, we show membrane voltage characteristics of our model with test stimulus in Figure 2 using the Brian2 simulation environment [21] to concur with observations in real neural cultures [30]. The main advantage of the generalized LIF model is its flexibility. For example, it can be easily configured into a simpler and less plausible perfect Integrate and Fire (IF) model by fixing τ_m to unity and disabling g_l. This can be useful in very deep network of spiking neurons on account of a great performance boost. If the simple LIF model with constant current integration is desired, varying degrees of simulation gains can be achieved by gradually diminishing the τ_{Na} and τ_K values modelling the effect of directly integrating into the membrane voltage more and more closely. V_{th} of neurons are independent parameters that can be set as desired in order to model the *homeostatic* effect after competitive learning by incorporating variable thresholds during inference.

2.2 Benchmarks

We have selected three different SNNs as our benchmarks for this work. They bring significant diversity to our case study by using different neural dynamics and training algorithms to solve the standard task of MNIST digit recognition [42].

The first network is a winner-take-all network with lateral inhibition trained using STDP [7]. It is built from LIF neurons with conductance based synapses and voltage-dependent current integration. This network achieves a maximum accuracy of 95% on MNIST as shown in [15]. Hereafter, we call it a *Spiking Competitive Winner-take-all Network* (SCWN). The second network is a feed-forward restricted boltzmann machine that has been trained using *contrastive divergence* [23], an unsupervised learning technique, and has been converted to an event-based network post-training, achieving a maximum accuracy of 92% [51]. This network has no recurrent connections, is built from simple LIF neurons and is referred to as the

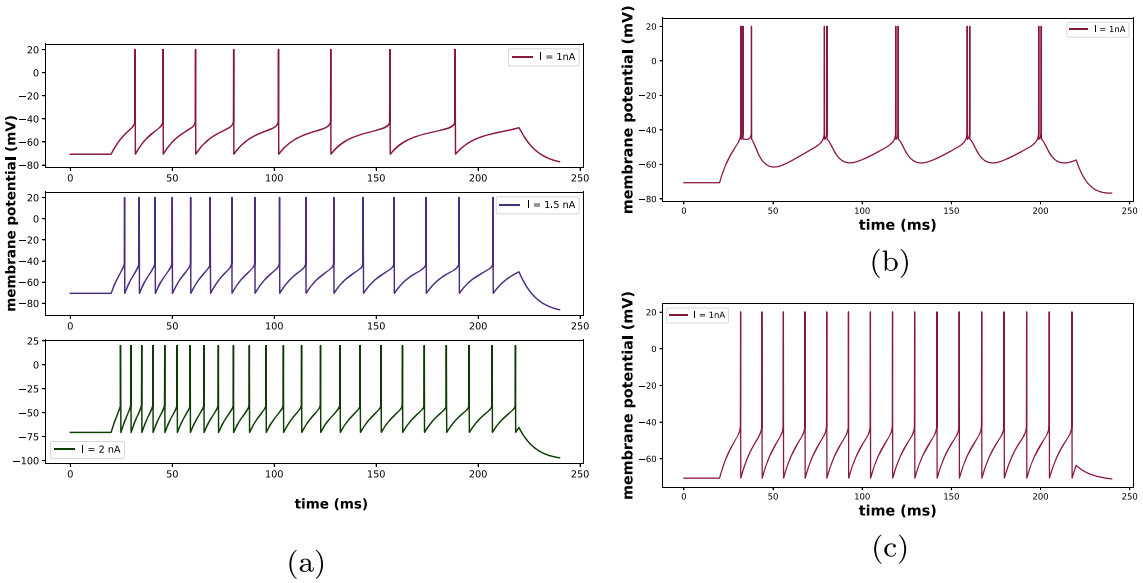


Figure 2 Simulation characteristics of the LIF neuron showing membrane potential traces in **a** regular spiking, **b** tonic bursting and **c** fast spiking behaviors.

Spiking Deep Belief Network (SDBN). The third network follows a convolutional topology and has been trained using standard *error backpropagation* [56]. Thereafter, it has been converted into the spiking domain following the methodology advised by [16]. It achieves a maximum accuracy of 97%, although the loss of accuracy from the equivalent analog network is negligible. This network is built from *integrate and fire* (IF) neurons that are perfect integrators with no leak or ion-channel conductance modelling. This network is significantly larger than the others but the connectivity is much sparser. Hereafter, we refer to it as the *Spiking Convolutional Neural Network* (SCNN).

The activity of each of these networks along with their internal connectivity will determine the energy footprint of these networks. Activity of a network depends on a large number of tunable *hyperparameters*. As discussed before, SNNs have an inherent temporal nature to them which is tunable by controlling the maximum input frequency and/or exposure time of a single stimulus. The network architect may want to select a high frequency that might lead to fast inference or a low frequency which can mean consolidation of energy by compromising on the inference latency and/or accuracy. The minimum discrete timestep for processing the exposure time is called the resolution. Other such hyperparameters include the range of weights, individual neuron thresholds, refractory periods, membrane time constants etc. As we discuss further in the following sections, the layer-wise spiking activities and spike fractions of the three networks show the variety of spike signatures that we can expect. It shows that network activity can attenuate as we go deeper, but it might also grow (for the SDBN). There may (for the SCNN) or may not (for the SCWN) be a good ratio between input-generated and internally-generated events. Table 1 lists succinct details of the benchmarks.

3 The CyNAPSE Neural Accelerator

3.1 Chip Microarchitecture

Figure 3 shows the overall design of the CyNAPSE accelerator excluding the memory hierarchy. This accelerator works in a coprocessor configuration with a host CPU or interfaces with dedicated sensory processing hardware like a spiking retina [46], cochlea [59] or a dynamic vision sensor [14]. CyNAPSE is an accelerator for SNN inference simulation and is assisted by the CPU and/or embedded spike generation/consumption circuits to complete the end to end application in question.

Each network simulated by the system consists of its own biological timeframe (different from the system clock domains). Any given simulation time is expressed in *Biological Time* (BT) which has a network-specified granularity of δt. The accelerator communicates with external interrupts through the I/O FIFO queues. All communications take place using the *Address Event Representation (AER)* protocol. Under this protocol, each spike is expressed as a packet consisting of the BT and a unique ID of the neuron that produced it. All spike inputs to the network are queued into the input FIFO. Internally generated spikes are queued into the auxiliary FIFO for rerouting and if produced by an output layer, they are

Table 1 Spiking neural network benchmarks used for this study.

Spiking Competitive Winner-Take-All Network (SCWN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
3	1584	473600	LIF	63.75 Hz	500 ms	0.5 ms	STDP - WTA	95%
Layer	Input Layer			Excitatory Layer (forward single)			Inhibitory Layer (recurrent dense)	
Spke Fraction	97.8%			1.1%			1.1%	
Spiking Deep Belief Network (SDBN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
4	1794	647000	LIF	6 Hz	1000 ms	1 ms	CD	92%
Layer	Input Layer		Layer 2 (dense)		Layer 3 (dense)		Output Layer (dense)	
Spke Fraction	15.6%		23.7%		59.0%		1.7%	
Spiking Convolutional Neural Network (SCNN)								
Layers	Neurons	Synapses	Neuron Model	Max. Input Freq.	Exposure	Resolution	Training	Max. Accuracy
6	13594	652800	IF	1000 Hz	100 ms	1 ms	Backpropagation	97%
Layer	Input Layer		Layer 2 (conv2D)	Layer 3 (subsampling)	Layer 4 (conv2D)		Layer5 (subsampling)	Output Layer (dense)
Spike Fraction	47.2%		35.7%	7.6%	7.7%		1.7%	0.1%

listed into the output FIFO waiting to get dequeued by an external interrupt for inference. It consists of two routing state machines for handling the input spikes (input router) and to reroute internally generated spikes (internal router). The neuron unit in CyNAPSE supports a maximum of N *logical* neurons (i.e. neurons in the network). To support deep network kernels within constrained chip resources, all logical neurons are not provisioned with dedicated circuits. Instead, a fewer number X of *physical* neurons (i.e. neuron circuits in the hardware) are used that support time-multiplexed access to a subset of logical neurons. This requires persistent storage of neuron states of all N neurons across X dendritic SRAM stacks that contain the current membrane, threshold, conductance and refractory states of each logical neuron. A memory controller handles all outgoing queries from the routing state machine to supply network weights for simulation to proceed. A system controller maintains the timeline of simulation, synchronizes control at the BT barriers and switches simulation phases at appropriate times to ensure correctness.

3.2 Neuron Design

Figure 4 shows the design of a single *physical* neuron circuit used in the CyNAPSE accelerator. This circuit emulates a discrete-time model of the generalized LIF neuron membrane dynamics as well as supports synaptic integration through nonlinear ion-channel dynamics as given by Eqs. 3 and 4. The neuron circuit shows three distinct channels in operation. The Na^+ and K^+ ion-channels integrate synaptic weights from spikes that arrive into this neuron depending on whether they arrive from *excitatory* or *inhibitory* pre-synaptic neurons respectively. This updates the current conductance values of the logical neuron. The leak-channel dynamics update the membrane voltage of the current logical neuron by using current conductance values. There is a comparator circuit that checks for membrane thresholding and accordingly releases a spike AER packet into the buffer. Updated status of each logical neuron are written back to the respective dendritic SRAM stacks.

3.3 Scheduling and core control

The CyNAPSE system is initialized by copying all trained weights into the device memory (similar to a memcpy) and configuring the CyNAPSE parameters to match the appropriate network architecture and neuron model desired for the particular network we want to run. Figure 6a shows a graphical depiction of how CyNAPSE schedules the various phases of inference. First, the top event of the Input FIFO is checked. If it is in the current BT, it is dequeued for routing. The Input-to-Dendrite route

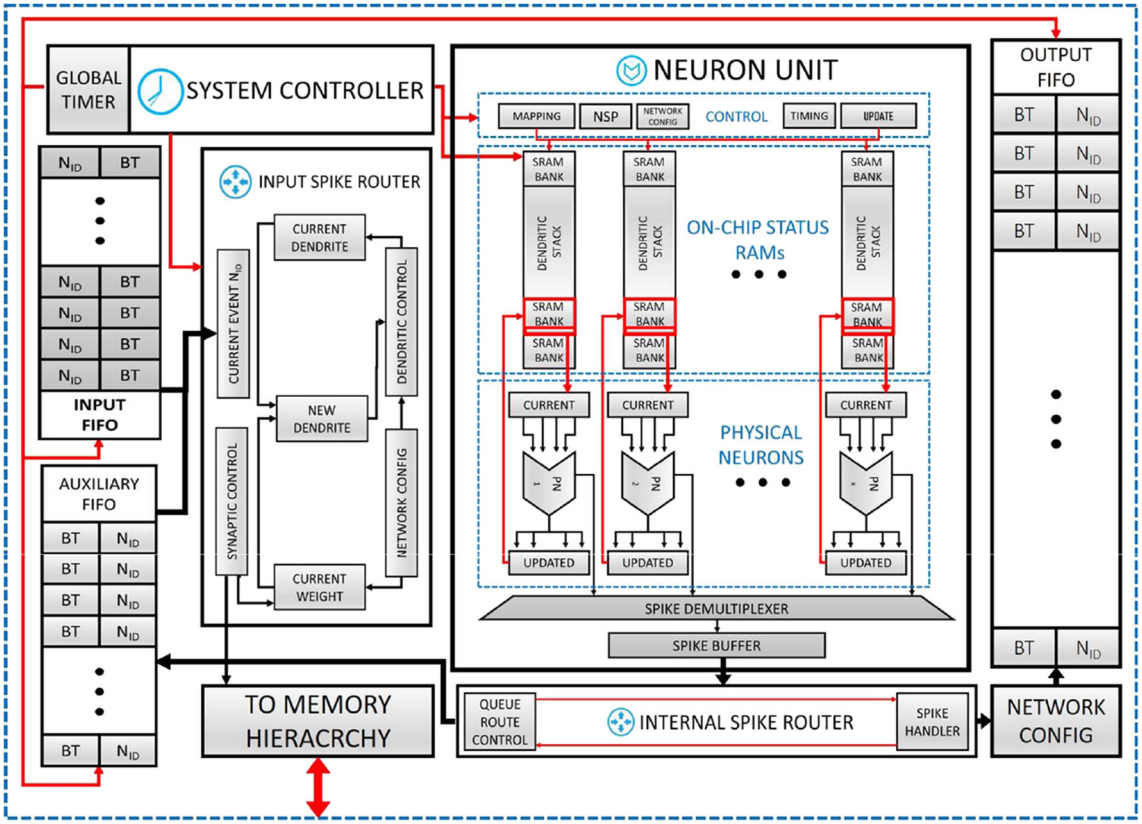


Figure 3 The overall chip microarchitecture of the CyNAPSE neural accelerator.

pipeline fetches all post-synaptic weights from the memory hierarchy and updates the appropriate post-synaptic neuron dendritic statuses on the SRAM stacks. All events of the current BT are similarly dequeued for routing from the input FIFO and then from the auxiliary FIFO as well until the top events in both queues belong to a future timestep. This shifts control to the Neuron Unit for the update pipeline. Here, all valid status words in an SRAM stack use the corresponding physical neuron ALU to update itself in a pipelined manner and any spikes produced are filtered out. When it is completed, any spike produced are rerouted to the auxiliary and, if required, to the output queues for further

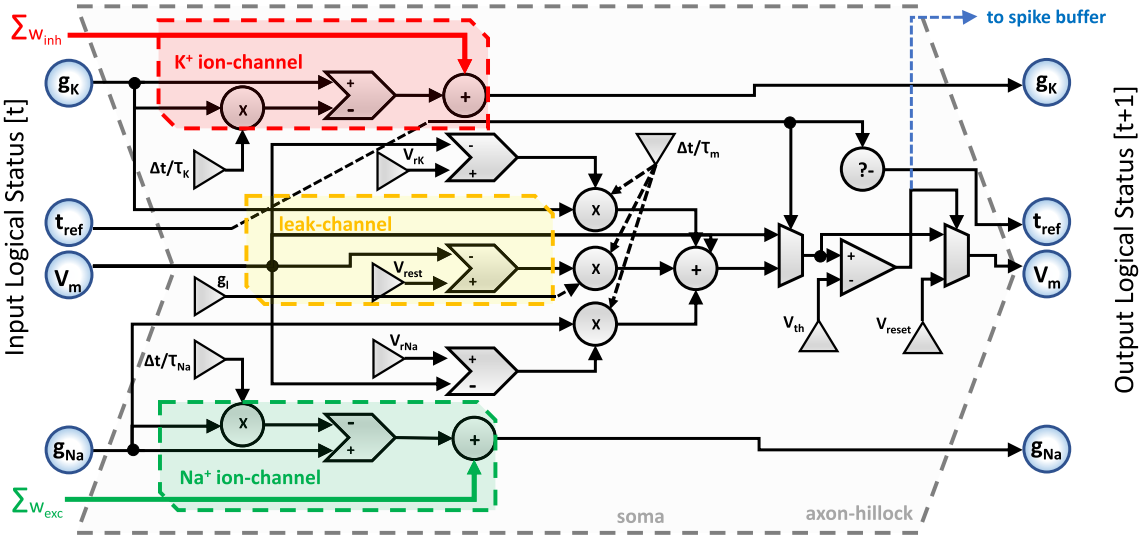


Figure 4 The full-custom digital generalized integrate and fire neuron.

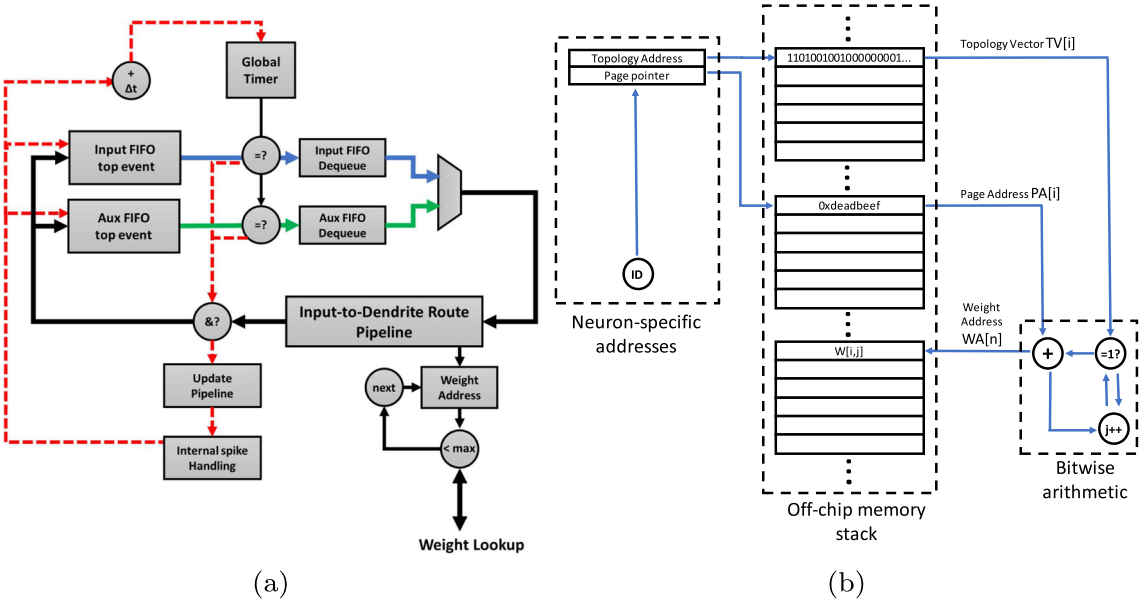


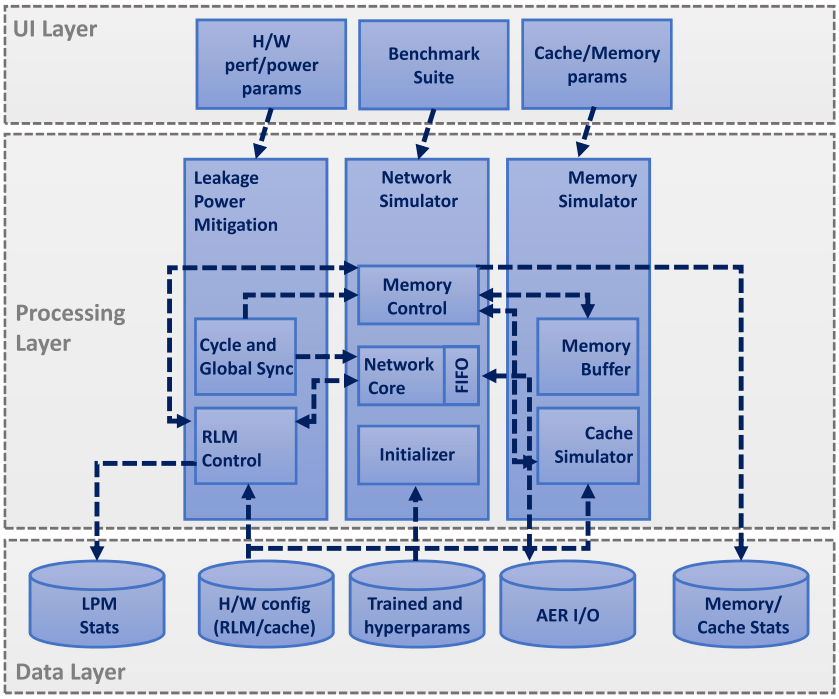
Figure 5 **a** Control flow of SNN emulation in the CyNAPSE Core and **b** Control flow of a single synaptic weight lookup by the input router.

handling. This marks the completion of one BT. The system synchronizes at this barrier, ticks the timer to the next BT and simulation resumes.

3.4 Memory Control

To support N logical neurons, maximum synaptic storage required would be $O(N^2)$. The closest physical implementation of the max. would be a fully interconnected Hopfield pool of neurons [27]. However, all modern applications have deeper and sparser topologies leading to much lower number of synapses. This makes a fixed $N \times N$ table storage very inefficient. Instead, we add a layer of indirection in the primary read path to tradeoff some performance for excellent gains in storage efficiency. This is done by storing synaptic weights in *pages* corresponding to each pre-synaptic neuron. This adds a second layer of indirection via a translation table that directs the router to the relevant page that belongs

Figure 6 The CyNAPSE Simulator architecture.



to the post-synaptic weights of the relevant neuron. This scheme is most effective when pages are of variable sizes (so as to reserve only as much memory as the actual fanout of a certain neuron) and, thus, are marked by the starting address of the page, or a *page pointer*. The number of entries in a single page is determined by one row of a *topology matrix* which defines every logical neuron's connections and instructs the router state machine to access the next D page offsets only when a connection exists, where D is the data-width of the current network. So each event induces the access of three data items (in chronological order) as shown in Figure 5b i.e. a topology vector, a page address, and all the weights in that page, all via a simple bitwise arithmetic scheme.

3.5 The CyNAPSE Simulator

While we depend on our synthesized and verified RTL implementation to collect low level statistics and diagnostics about the system's behavior (see Section 6), for large design space explorations of the system, we have developed a CyNAPSE simulator that efficiently implements the execution model of the hardware accelerator. Figure 6 describes the architecture of the simulation tool. It collects the kernel parameters (model hyperparameters, architecture, layer definitions etc.), trained parameters (weights, thresholds etc.) and input/output AER packets to get started with the desired SNN benchmark. Additionally, it takes some hardware configuration options that correspond to the design-time configurable parameters of the CyNAPSE chip. As we will see later, the simulator is also capable of running explorations on the off-chip memory system as well as mitigation of on-chip static leakage. Accordingly, some configs on the L1 Cache/Main memory system as well as leakage power management are also accepted. The processing is divided into three software routines: Caching/Memory management, Network simulation management and Leakage power management. The simulator is not inherently cycle accurate in nature but for static power mitigation, it adds a global cycle and synchronization layer for timing purposes. For consumption of the user, we provide scripts to mention benchmark details, leakage management details and caching or memory management details.

4 Adaptive Memory Management

Since the amount of storage required for weights in large networks is infeasible for realization on chip, off-chip, flexible, high-density storage is used to store synaptic weights of reconfigurable network kernels simulated by CyNAPSE. This has serious implications on both network performance as well as energy consumption because performance of SNN kernels is largely memory-limited and a large amount of energy is consumed in bringing weights back into the computation engine to fuel SNN inference. For each input or internal spike processed by the network, all post-synaptic weights are loaded by the memory controller and atomically added to the relevant dendritic SRAM bank. Depending on the average connectivity of a network, the amount of time spent in the dendritic routing phase is variable but it is consistently seen to be a majority of network simulation time. We show a roofline analysis of the dendritic routing cycle of CyNAPSE in Figure 7a. We have tallied the peak Mega Dendritic (atomic) Operations per second (MDOPS/s) possible in CyNAPSE with a peak reported bandwidth of commodity DDR3 DRAM as well as with a lower steady state bandwidth that is expected in simulation with CyNAPSE. X denotes the number of physical on-chip neurons in the chosen configuration while W denotes the precision of synaptic weights. We can see that with lower peak memory bandwidth, a larger region of

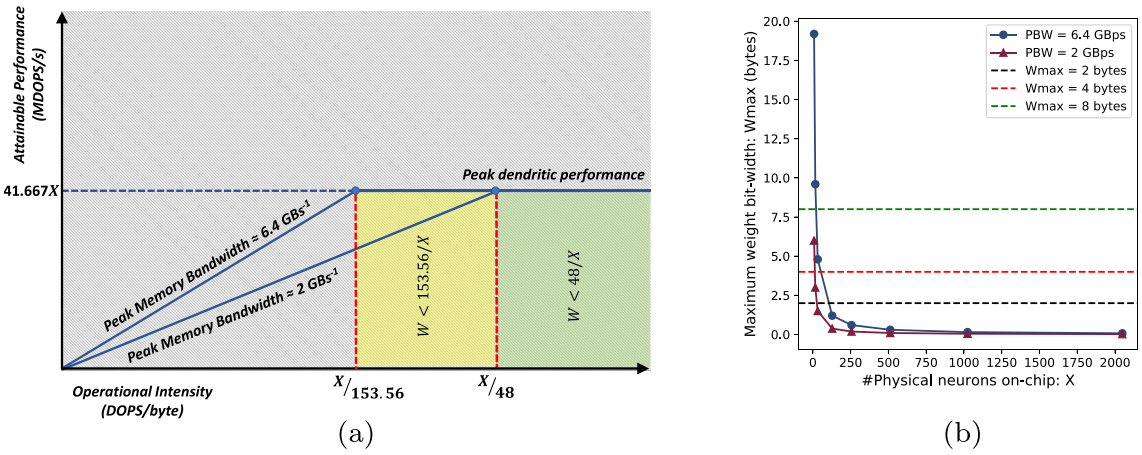


Figure 7 **a** Roofline analysis of CyNAPSE routing performance and **b** compute-bound performance requirements for CyNAPSE routing cycle.

the operational space is memory bound. Figure 7a confirms this hypothesis. Under steady-state bandwidth and 64 on-chip physical neurons, greater than half-precision synaptic weights will result in memory-limited performance.

When compared with the activity factor of a representative CPU workload, SNN inference has very little switching computation on-chip because spiking is essentially a sparse event. In Figure 8, we show the full power consumption profile of the CyNAPSE system (see Section 6 for details). It can be noted that for all the benchmarks, with negligible differences, the off-chip memory accesses makes up for a large share of the total system power consumption. More physical neurons on chip allow more performance and according to the use case, may be an energy-dominated or performance-dominated choice [10]. Regardless, the opportunity and importance of saving simulation time and power consumption in off-chip memory traffic is clear and calls for architectural investigation to explore possible solutions.

While algorithmic optimizations like pruning and quantization of weights [22, 29] provide some viable approaches to relax the memory traffic, their effectiveness is limited by the allowable degradation in accuracy. We attempt to make microarchitectural optimizations that do not affect accuracy while maintaining compatibility with all algorithmic changes. We carefully investigate domain-specific access patterns and make recommendations to cleverly mitigate large redundant losses.

4.1 Cache Management Policy

Conventional Cache policies like *Least Recently Used (LRU)* and *Random* can capture, to a great extent, the data localities in general purpose programs. However, their ability to model unique references to the same block is limited by associativity [17]. While Belady's optimal replacement policy [5] requires an infeasible view of the future, policies like DIP [55], RRIP [38] and LIRS [32] have looked at speculative techniques to predict re-reference of a cache block in general purpose processors by collecting past information. However, unlike general purpose programs, we already have some knowledge about the upcoming accesses, courtesy of the event-queue. Furthermore, for inference, we do not have to deal with writes. In other words, it is similar to an instruction cache in nature. Therefore, we attempt to design a new management policy that can efficiently capture subtle behavior particular to our applications and pattern of memory accesses that conventional policies like LRU or Random fail to account for.

In CyNAPSE, initially the events are stacked up into the input FIFO queue until it is full. Thereafter, an event is enqueued at the write pointer only when another is dequeued from the read pointer. The input spike router routes an event that is dequeued and looks into the memory hierarchy for the relevant neuron's connectivity and weights. However, since the queue already contains up to *<queue length>* events, the hardware can always look ahead of the actual execution in terms of events that are to come. As such, we define two different times for each event, namely the *'read-time'* i.e. when a certain event is read (but not dequeued) from the FIFO queue and the *'route-time'* i.e. when this event is eventually dequeued. Before simulation starts, the cache is warmed up with events up to a certain *lookahead distance*. This distance is selected carefully to maintain sufficient reuse information from the future without letting these events thrash the ones that are required sooner. After the initial warm-up, each event dequed from the top of the FIFO at its route-time means one event from the bottom is added to the cache at its read-time. We explain this policy in detail as follows using each type of cache access scenario:

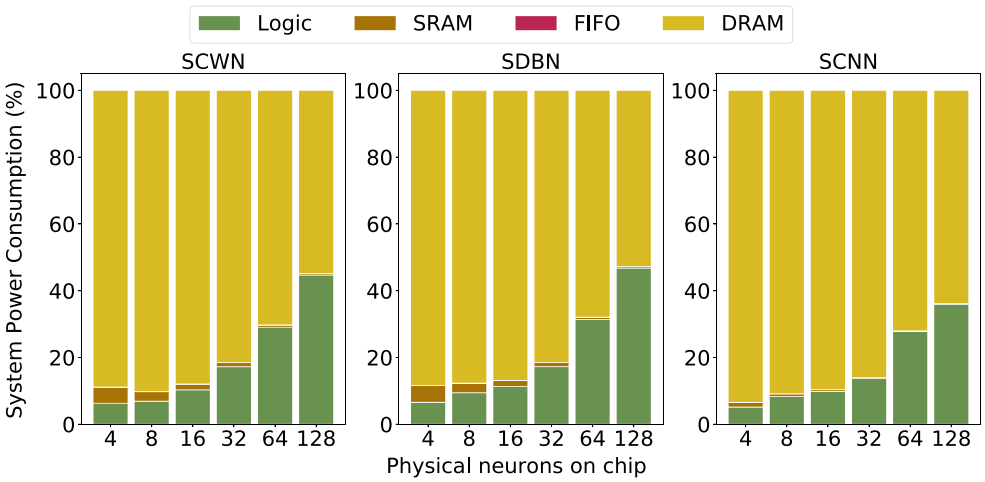


Figure 8 System power consumption of CyNAPSE broken down for each benchmark shows significant DRAM percentage.

4.1.1 Compulsory Miss at Warm-Up and Read-Time

An *"event reader"* circuit reads the neuron ID of the first event that is on the queue. The CyNAPSE simulator (see Section 3.5) provides computation to generate all addresses associated with any particular ID of a given network. So, the circuit will now reserve an unallocated way in the requested index and start bringing in the data from the main memory (i.e., DRAM). At warm-up, there is no contending process inside the cache but queue requests can also be processed while routing occurs. The cache is configured with two independent read-write ports for processing simultaneous requests. Depending on the DRAM steady-state bandwidth and the latency of a single-route cycle, a cache with exclusive read-write ports can complete multiple *read* requests within the regime of one *route* request. However, we consider one read request per route request post warm-up to keep our study simple and our solution sufficiently easy to achieve. A compulsory miss means this neuron ID is encountered by CyNAPSE for the first time since the cache was last flushed. Therefore, all blocks tagged by the corresponding addresses are marked with a *reuse score* of 1, which essentially means 1 guaranteed route accesses to this block in the future.

4.1.2 Hit at Read-Time

At some point, the event circuit will come across a neuron that it had already encountered before in the queue. If this information still resides in the cache, it is a hit. On a hit, the circuit will simply increment the reuse score value by 1 and move on to the next event in the queue issuing no further request to the next level cache or main memory.

4.1.3 Capacity or Conflict Miss at Read-Time

At a certain point down the queue, the cache is bound to fill up considerably. This leads to a possible capacity/conflict miss at read-time. In this case, the naive approach is to consider evicting the way which has the least reuse score value. However, replacement at read-time is not compulsory. There are certain potential concerns that can occur with replacements at read-time. Accordingly, we consider three different approaches to handle read-time replacements:

- *Conservative approach*: Multiple-reuse blocks can be easily thrashed by blocks that end up not being reused much and will lead to severe thrashing of blocks resulting in unnecessary memory traffic and energy expense. Hence, no replacements are allowed at read-time.
- *Aggressive approach*: Not allowing replacements at read-time will lead the cache to lose out on potential opportunities to reuse blocks that could have been loaded into the cache. Hence, this approach always replaces the minimum reuse score block at read-time.
- *Intelligent approach*: Replace at read-time only when the minimum reuse score in a set is below a specific, reconfigurable *reuse threshold*.

If a read-time replacement occurs, the new import will be marked with a reuse score of 1.

4.1.4 Compulsory Miss at Route-Time

There are no compulsory misses at route-time in this policy. Misses happen only when the policy opts out of read-time replacements for all read-time references of a particular block before its route-time arrives.

4.1.5 Hit at route-time

Hit at route-time means one promised reuse has been realized. This is, therefore, accompanied by a decrement of the reuse score of the corresponding cache block by 1.

4.1.6 Policy Miss at Route-Time

As mentioned above, there is a finite probability of encountering misses at route-time if the particular read-time replacement policy that is employed fails to warrant the (pre)fetching of a certain block. This requires the router to request an import from the next level cache or main memory. The route-time policy simply asks the router to replace the block with the lowest reuse score in the cache set by the new block since it is compulsory to bring the new cache block into the cache at route-time (unless there is a bypass mechanism). This time we put a zero into the reuse score field of the block on a fresh import since there are no guaranteed reuses after this point for this particular block. Figure 9 summarizes the baseline memory control scheme.

4.2 Network-Adaptive Enhancements

Our domain-specific cache policy only accounts for events in the event queue that are generated at a much higher throughput than the expected compute-latency of processing a single event and are visible to the event reader prior to their individual route-time. While input events necessarily satisfy this criterion, all internal events generated within the network are routed in the biological timestep immediately following the one in which they are generated, making them unsuitable for our scheme. As our benchmarks show, input

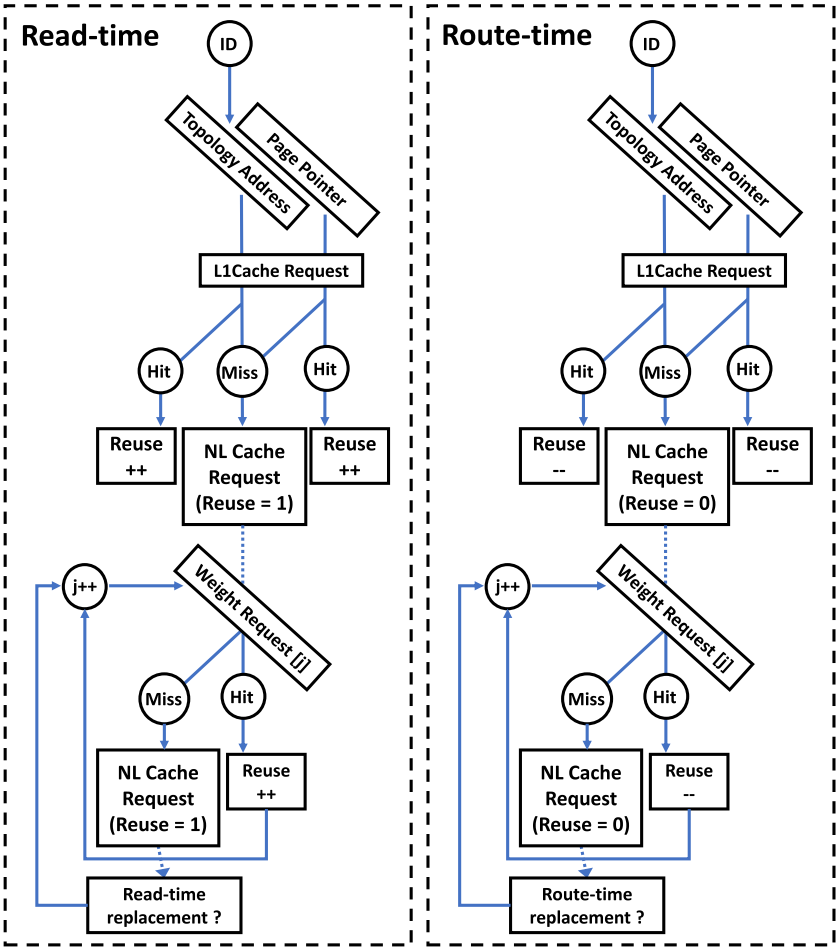


Figure 9 Baseline read-time and route-time memory control schemes.

activity can have different relative importance to internal activity and, hence, could affect our scheme to varying degrees. It is clear that we need flexible and dynamic network-specific enhancements that help make our scheme robust to these variances.

The CyNAPSE core requires compile-time information about network layer types (dense, conv2d, subsampling, etc.) as well as neuron ID ranges corresponding to these layers. Some of this static information can help us adaptively extend our scheme to perform better. We attempt to extend this static kernel information to include dynamic behavioral data. Simulation statistics can dynamically provide information like heightened areas of activity as well as dormant regions in the network. This can be collected from the auxiliary queue over time which contains all internal activity of the network. We dump queue statistics during simulation of a *batch* of example stimuli and use this information to dynamically improve our policy. Figure 10 shows the simulation statistics collected by our simulator for each benchmark in terms of spiking activity fraction of a layer relative to the whole network. We use statistics at a layer granularity to consolidate storage and computation required on-chip for this purpose. A higher granularity would provide better results, but incur more storage and computational overhead. Static or compile-time information is a functional requirement for network simulation in CyNAPSE. Therefore, it poses no extraneous computational cost to generate and store static kernel data. We just set up masked loads to ensure our adaptive techniques are applied to relevant layers and ranges. Next, for dynamic adaptivity, we have set up low-granularity layer-wise counters attached to the auxiliary queue that use fixed-point arithmetic once after every batch of examples. Given the average number of spikes per example and the average number of routes per spike, this presents little logical (power/area) overhead and performance cost relative to the amount of performance savings resulting from freeing up the cache for high reuse neurons taking advantage of our management policy. We propose two techniques to extend our scheme:

4.2.1 Cache Bypassing

Consider the SCWN benchmark. All LIF (internal) neurons in the network demonstrate extremely little activity relative to the input spike frequencies. CyNAPSE routes all internal

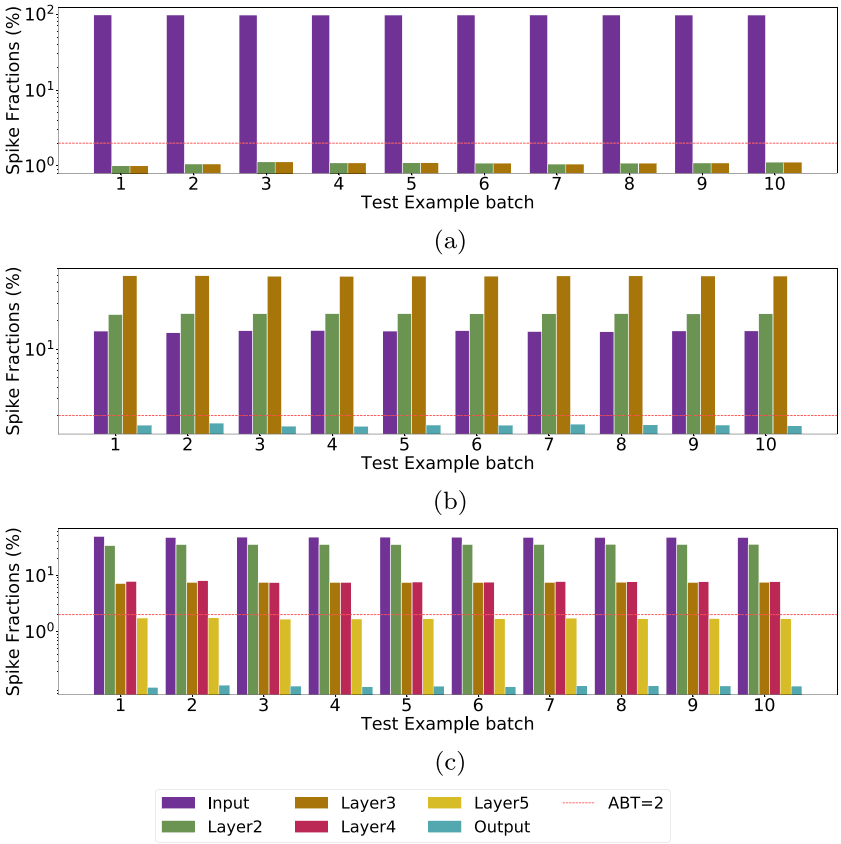


Figure 10 Spike statistics generated dynamically by the CyNAPSE simulator to adaptively configure cache requests.(a), (b) and (c) show layer-wise activity fractions for the SCWN, SDBN and SCNN respectively with time.

events into the auxiliary queue for further processing, including output events, since this is a hard requirement for recurrent topologies like the SCWN. Statically, therefore, we have no information that can benefit our scheme. However, as we will see experimentally, the distribution of network activity is highly skewed in favor of the input layer when compared to the processing neurons. Not only does this help our basic strategy, but also gives us a clear path to an adaptive extension that we can apply. For feed forward benchmarks like the SDBN and SCNN, the distribution is not so obviously favorable. There is considerable activity in the deeper layers and these neurons usually pollute cache blocks by occupying them at the cost of high reuse conflicting neurons which could otherwise save energy. Therefore, we propose a mechanism to bypass all memory accesses pertaining to sparse activity neurons by collecting information on a layer-by-layer granularity. Information can be statically provided to the cache at compile-time (for e.g. output neurons for feed forward networks can be bypassed etc.) or dynamically generated (for e.g. low activity layers in any network can be bypassed etc.). For dynamically arriving information to the controller, we maintain an *activity bypass threshold (ABT)*, which is the minimum activity fraction a layer needs to maintain on average for its neurons to allocate data in the cache. On a bypassed request at route-time, the memory controller does not allocate a cache block and directly retrieves the requested data from the main memory or next level.

4.2.2 Line protection

As opposed to low activity LIF neurons in SCWN, Layers 2 and 3 of SDBN and Layer 2 of SCNN show very high activity among processing neurons (see Figure 10). These layers can hurt our management scheme greatly if not accounted for. To that end, we propose a protection scheme for processing neurons in high-activity layers by dynamically providing them with a *probable reuse score* based on network activity statistics collected over time. Figure 11 shows the mean reuse distances of neurons in each layer for each benchmark. We put a probable reuse score which is inversely proportional to the reuse distance of a neuron so as to account for all reuses expected within a certain window of time.

5 Leakage Power Mitigation

Although we established in Figure 8 that the on-chip power consumption in its entirety (including Logic, SRAM and FIFO) is a small percentage of the system, it multiplies when we scale the system up to multiple logical cores to take

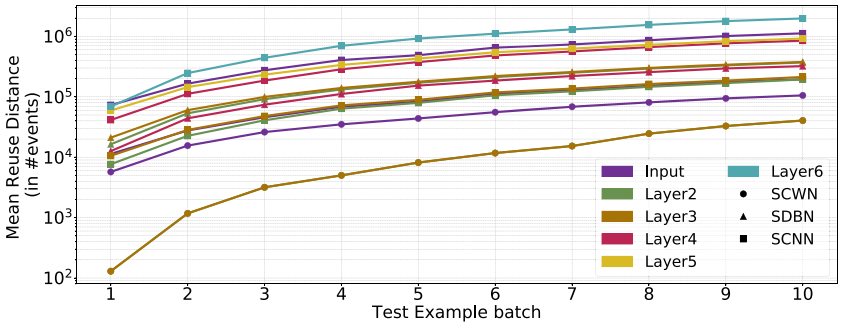


Figure 11 Layer-wise mean reuse distances of neurons in each benchmark.

advantage of parallel updates to a layer's inference. Since multiple such systems end up sharing higher level memory hierarchies, core power consumption becomes more and more accountable vis-a-vis off-chip power consumption.

Figure 12 shows the distribution of power consumption among the system's on-chip resources including the routing systems, memory controller, L1 cache, physical neuron array and dendritic SRAM stacks besides the control circuitry associated with each. The key takeaway is that as we scale up each core's individual inference capacity and boost performance (by affording more physical neurons in the array), more of its energy is consumed in the idle phase. This is not surprising since the dataflow of SNN inference is essentially event-driven in nature. Therefore, a large amount of hardware resources spend most of their time dissipating leakage power while doing no meaningful work. Therefore, we propose to employ microarchitectural techniques to control runtime leakage in sparsely active circuits and regions.

5.1 Gated V_{DD} and Starved V_{DD}

Power gating is a technique that shuts off power supply to an execution unit dynamically when the onset of a long idle period is detected [31, 53]. [28] prescribes a time-based finite state machine for gating execution units using activity cycle counters. While gated V_{DD} techniques have produced some success for cache lines [37, 64], it depends entirely on the detection of data with sparse temporal reuse. The same cannot be used for SRAM cells in the dendritic stacks because they carry their state throughout the simulation and have complicated temporal reuse as we have already discussed. [2, 18] introduced a starved V_{DD} technique to put lines of a cache sub-bank into a *drowsy* state by dynamically increasing threshold voltage of devices for predicted idle periods. The dendritic stacks in CyNAPSE have large well-defined idle periods that can take advantage of a starved V_{DD} leakage control strategy.

We start by setting up the parameters required by the Runtime Leakage Mitigation (RLM) controller module (see Figure 6) that controls the *header* devices to every module to gate or starve power supply to these hardware regions by monitoring their switching activity over time. Following from [28] and using technology-specific attributes from the 65nm CMOS library, we set the parameters accordingly. The user-configurable parameter $T_{idleDetect}$ represents the number of idle cycles that the controller waits for before declaring a certain module to be idle. A conservative management policy uses a sufficiently high $T_{idleDetect}$ value to avoid large number of *sleep/starve* and *wake-up* cycles each of which is associated with an overhead energy $E_{overhead}$ which is directly proportional to the width of the

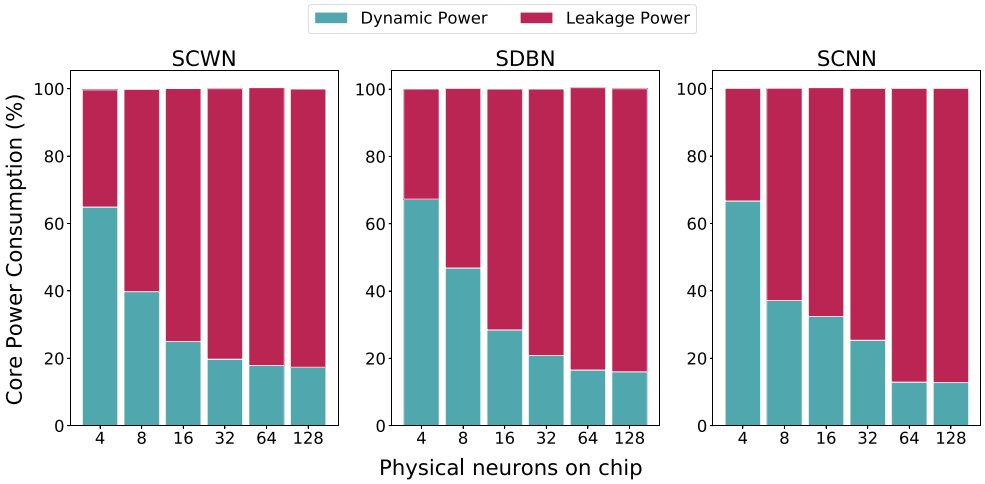


Figure 12 Core power consumption of CyNAPSE broken down for each bench-mark shows significant leakage percentage.

header device and internal gate capacitance of the block in question [28]. While the latency of sleep cycles T_{sleep} can be hidden behind the actual execution timeline, the latency of waking up a module on an access during its idle period incurs a performance penalty T_{wakeUp}. Therefore, low overhead and low performance impact from unwanted wake-up cycles is also expected from more conservative approaches. On the other hand, gradually lowering the $T_{idleDetect}$ threshold will ideally result in greater percentage of idle cycles $\%idleCycles$ but at the cost of higher switching overhead and percentage performance penalty $\%perfPenalty$.

We classify the CyNAPSE hardware macros in a coarse-grained manner and sweep the $T_{idleDetect}$ values across each of them to study this effect and derive experimental optimal values. For each benchmark, module and $T_{idleDetect}$, we first calculate the average number of cycles elapsed per example inference with LPM settings C_{LPM} and without it C_{leak}. From our PrimeTime reports, we collect the average leakage power consumption P_{leak} over an example inference and derive the required leakage energy E_{leak} as:

$$E_{leak} = P_{leak} * C_{leak} * T_{cyc} \tag{7}$$

where T_{cyc} is the cycle time. In presence of LPM, we count idle cycles only at the end of a T_{sleep} latency and until the start of the next T_{wakeUp} cycle. We calculate the leakage energy consumption E_{LPM} as:

$$E_{LPM} = (P_{leak} * C_{LPM} * T_{cyc}) * \left(1 - \frac{\%idleCycles}{100}\right) + 2\sum_{W} E_{overhead} \tag{8}$$

where W is the total number of wakeUp or sleep cycles incurred on average. Accordingly the net leakage savings is calculated as:

$$E_{save}(\%) = \frac{(E_{leak} - E_{LPM}) * 100}{E_{leak}} \tag{9}$$

5.2 Application-specific RLM

As mentioned earlier, SNN benchmarks are different from general purpose CPU or GPU benchmarks because of a fixed dataflow and well-defined phases in the simulation where a particular module might be active. Using the timeline of SNN inference, we make further optimizations in the RLM strategy. For instance, simulation of every BT in CyNAPSE begins with a routing cycle during which the input router interfaces with the memory hierarchy through the controller and requires atomic operations on SRAM dendritic stacks. Therefore, for the entirety of this phase, the neuron array and internal router can be *force-gated*. Using similar reasoning, the input router and memory controller can be force-gated during the update and internal routing phases. Lastly, the dendritic SRAM stacks can be collectively force-starved during the internal routing phase only.

Further, unlike the memory controller and input router, the activity of the neuron array and internal router are limited to conspicuous periods within the simulation during which their operation is not intermittently shared with the operation of another module. This means they can have the lowest possible $T_{idleDetect}$ value without significant penalty in performance or overhead energy. The only exception to this is when the neuron array accesses an SRAM bank that is idle. Therefore, we configure the lowest possible $T_{idleDetect}$ to be only as long as required to tolerate the latency of waking up SRAM banks.

5.3 Pre-activated SRAM banks

SRAM bank accesses happen in a pipelined way through either the input router (routing phase) or the neuron array (update phase). Therefore, each bank can be pre-activated before the actual access takes place. As a result, we can hide the latency of waking up the subsequent bank within the latency of accessing the current active bank. This allows us to reduce the $Ti_{idleDetect}$ value of the SRAM banks to the lowest possible value. However, while the $\%idleCycles$ are expected to increase steadily with this reduction, it drives up the overhead energy of redundant sleep/starve and wakeup cycles. Therefore, we implement the pre-activation of SRAM banks and sweep $T_{idleDetect}$ towards very low latencies to study the effect of this recommendation.

6 Experimental Infrastructure

6.1 Low-Level Design

Figure 13 shows our experimental infrastructure and tool flow. The CyNAPSE core was designed in synthesizable Verilog HDL and functionally verified against all benchmarks[1]. The logic portion (excluding the on-chip dendritic SRAMs and FIFOs) was synthesized to a commercial 65nm TSMC library using a nominal supply voltage of 0.9V. Synthesis was done using the Cadence SOC Encounter RTL Compiler while pre and post-synthesis simulation were carried out in ModelSim. We then used the synthesized netlists to dump representative activities to VCD files. Synopsys PrimeTime was used to estimate power using a compatible SAIF format, easily convertible from the VCD for power estimation over each of our benchmark test-benches.

[1]Source code for the RTL implementation of the CyNAPSE neuromorphic accelerator is available at: https://github.com/saunak1994/CyNAPSEv11

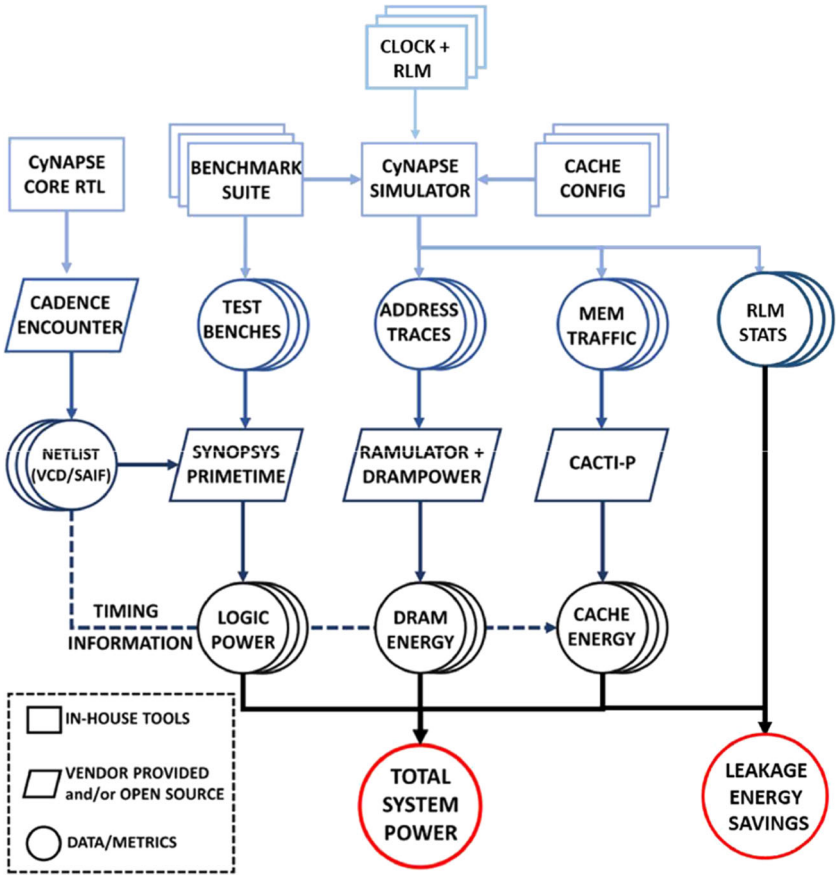


Figure 13 Experimental Infrastructure and flow.

All our on-chip SRAM structures were bypassed from the CAD process and modeled only for functional verification. For estimating timing and power of SRAMs we used CACTI-P [43].

6.2 High-Level Exploration

For high-level architectural exploration of the memory subsystem, our software simulator generates high-level statistics like memory accesses per spike, tag array and data array accesses per spike, hit rate, miss profile, etc. while maintaining a deterministic one-one equivalence with the hardware model thereby confirming accurate simulations. Additionally, the simulator's memory controller module also generates DRAM address traces for all synaptic lookups that go to main memory. These address traces are used by Ramulator [39] in appropriate organization, speed and timing configurations to dump JEDEC standard command traces in DRAMPower [11] format. We route these commands to DRAMPower 3.1 with consistent configurations to estimate the energy consumption of these traces. We use a 256MB DDR3 x8 configuration with a 1600MHz pin bandwidth which is more than sufficient to store all synaptic and meta data for our benchmarks. Although each network has varying tolerance to error and, thereby, have different precision requirements, we fix all synaptic data-widths to 8-bytes to have a fair comparison of memory footprint independent of any algorithmic optimization on top of them. Using the memory consumption of traces and CyNAPSE's timing information, we calculate the power consumption of the system. In a cached configuration, we use the same infrastructure to estimate energy consumption from the main memory. Additionally, we use high-level statistics like tag and data array read and write accesses to the cache and plug them into CACTI's UCA cache energy estimates to model net power consumption of the system for each of our benchmarks. Owing to very long simulation times, we simulate the MNIST dataset for a representative set of 100 examples containing a uniform distribution of all digits. For our experiments with dynamic-adaptive schemes, we use intermediately generated statistics from our simulator by dumping the contents of the FIFO queues after each batch of examples. We provide a simple routine to calculate these statistics, feed them into the simulator and restart simulation from the checkpoint with forwarded cache contents.

For exploring and evaluating the leakage power mitigation strategy, we use the RLM and global cycle and synchronization layer on our simulator to dump statistics representing elapsed clock cycles in presence and absence

of various degrees of runtime leakage control. The simulator also dumps module-wise idle cycle percentage and overhead energy incurred. We use the average leakage power obtained from PrimeTime reports and timing statistics from our simulator to calculate average savings in leakage energy by varying parameters and control strategy as desired.

7 Results

7.1 Adaptive Memory Management

After a binary search through three degrees of freedom in cache design: block size, associativity and number of cache blocks, we have selected a 256 KB 4-way set-associative cache with 64 byte blocks as our operating point. With conventional cache management policies, we found that this configuration gives us the best return-on-investment, on average, over our benchmarks within constrained memory and power budgets. In this section, all reported results use the same configuration as above to ensure fair comparison of similarly provisioned alternatives. We first validate our proposal for the correct read-time replacement philosophy by presenting experimental results and our interpretation of the same. Using the above verdict, we evaluate the effectiveness of simple conventional cache management policies vis-a-vis our proposed policy for the same configuration in reducing the power consumption of the system. We then evaluate the relative benefit of applying adaptive extensions to our policy. With these results, we attempt to explain the behavior of each benchmark with intuitive understanding and spike statistics obtained from our simulator.

7.1.1 Read-Time Replacement

In Section 4, we described the potential concerns with read-time replacement of neuron data. Our simulator provides hooks to dump and visualize cache contents at any given time in the simulation. Further, it can provide information on which block was replaced and the reuse score it was carrying at the time of replacement. Using these statistics, we fixed a minimum reuse threshold according to the frequency of reuse scores seen in replaced blocks for each benchmark to validate our intelligent approach towards the handling of read-time replacements. In Figure 14, we show the result of exploring all three read-time replacement policies. For all benchmarks, the intelligent approach outperforms conservative and aggressive approaches. Aggressive replacement defeats the purpose of generating maximum reuse by ignoring reuse scores at read-time. Conservative replacement has a similar effect by leading to unnecessary route-misses that could have been avoided. However, it does not lead to multiple unnecessary memory accesses at read-time which makes it better than the aggressive scheme. With benchmarks having short reuse distances (e.g., SCWN), there is a bigger loss while for benchmarks having larger reuse distances (e.g., SCNN), little difference is observed. All results declared hereafter in this paper use the intelligent approach for our policies.

7.1.2 LRU vs Random vs Our Policy

Figure 15 shows the power consumption of the CyNAPSE system as a function of test example batch for each benchmark. It covers our selected cache configuration running on LRU and Random replacement policies and draws a comparison with our policy.

The SCWN network is a relatively low activity network. It produces an average of 2.144 spikes every biological timestep or 2144 spikes per example. Each input neuron needs to produce multiple spikes in order to induce robust inference which makes it ideal for exploiting temporal locality of neuron data. LRU exploits reuse in short timescales, for instance, within the span of an example. For each example, some *winner* neurons will demonstrate heightened activity while inhibiting others. 85-90% of cache misses remain classified as capacity misses, so we know that it is not limited by associativity. Random replacement fails to fully capture the essence of intra-stimulus reuse but in a cache that has sufficient associativity to handle conflicts, it reaches close to LRU. SCWN is also an input-dominated network. As mentioned before, we have 97.8%

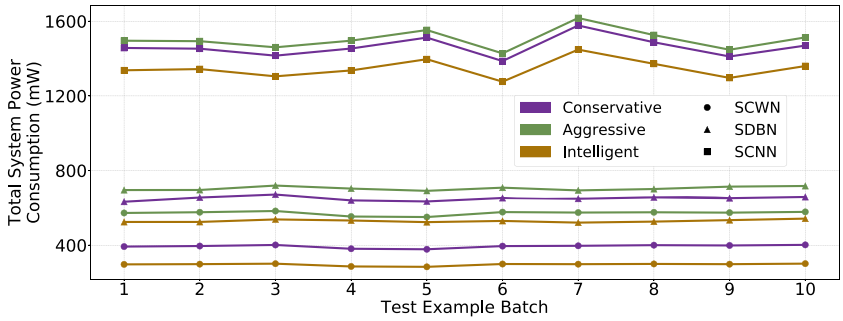


Figure 14 Comparative analysis of the various read-time replacement handling approaches.

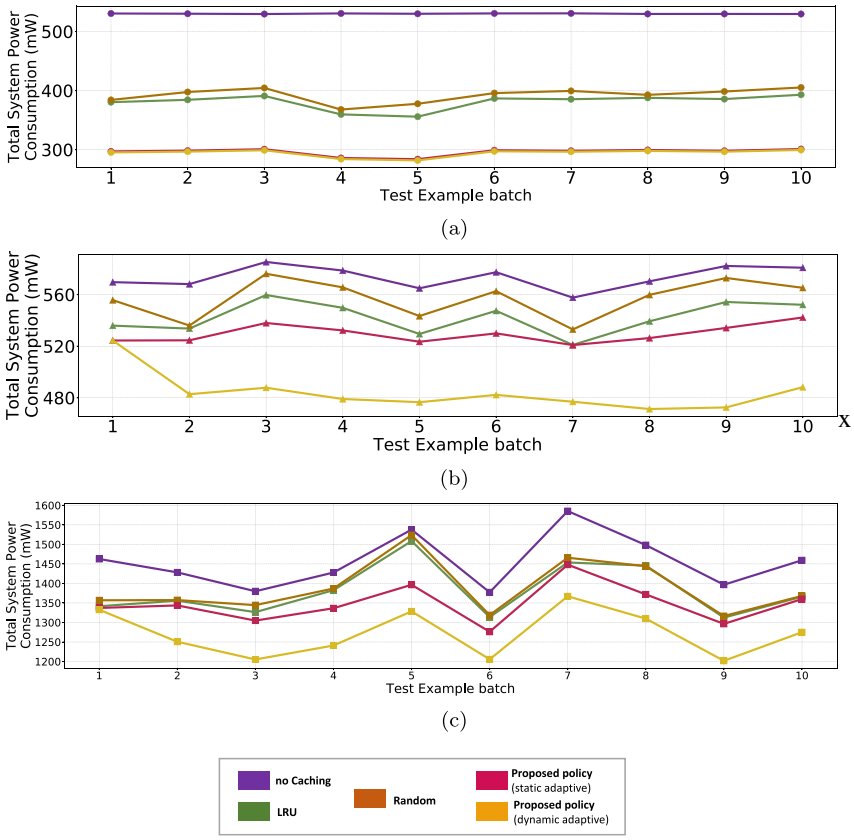


Figure 15 Comparative analysis of energy savings using our policy over conventional policies for the **a** SCWN, **b** SDBN and **c** SCNN benchmarks.

of the spiking activity in the input layer of the network. This means we have a good view of majority of the future events in the queue. Besides, neurons in the generally excited areas of the input field share activity across many stimuli. They also share locality in meta-data, especially in the page address meta data.

The SDBN network has a different activity profile. Its synaptic weights are unnormalized, which means that the input events, although moderate in activity, induce higher spike frequency in subsequent layers. Particularly high activity is observed in the third layer. With low input activity, most extra-stimulus reuse is arrested by LRU, so not much benefit comes from switching to our policy.

The SCNN network is a very high activity convolutional neural network. On an average, it produces 219 spikes per timestep. Since by definition of a biological timestep, a neuron cannot spike more than once in a single timestep, this network has a relatively much higher mean reuse

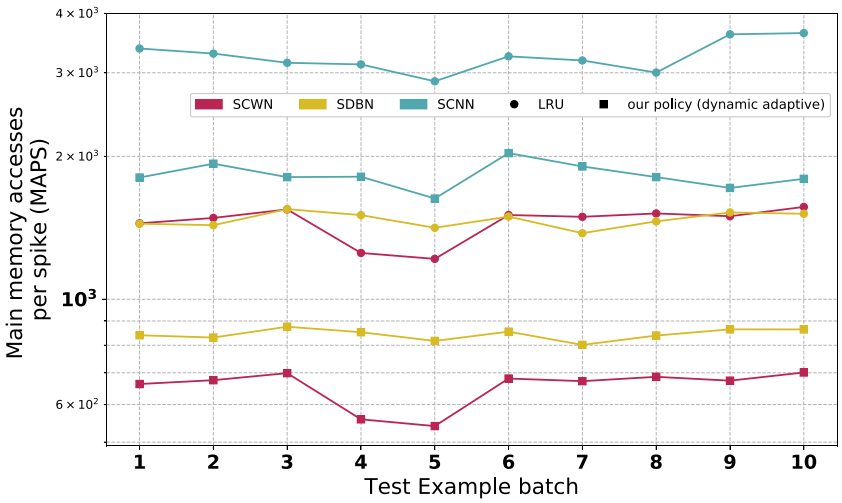


Figure 16 Reduction in off-chip memory traffic for our policy relative to LRU.

Table 2 Relative energy savings achieved using different policies.

Benchmark	LRU v/s baseline	Random v/s baseline	Our policy (static adaptive) v/s baseline	Our policy (dynamic adaptive) v/s baseline	Our policy v/s LRU
SCWN	28.13%	25.99%	44.13%	44.45%	22.71%
SDBN	5.46%	2.88%	7.65%	15.55%	10.67%
SCNN	5.12%	4.59%	7.4%	12.61%	7.9%

distance than other networks. With limited capacity, it is difficult to exploit any reuse for conventional cache policies. However, there is a good fraction of input activity which is effectively targeted by our policy. We were able to collect some reuse scores over the course of the simulation, enough to outperform both conventional policies.

7.1.3 Applying Network-Adaptive Enhancements

Our policy collects reuse scores in SCWN neurons from both intra- and extra-stimulus reuse distances and significantly outperforms conventional policies. We have set an activity bypasss threshold of 2% and added dynamic adaptation schemes on a layer granularity as mentioned before. This means that any layer having a mean spiking activity less than 2% switches to a bypass mode. However, our results show that this does not lead to much difference for SCWN because the network is dominated by input spikes and bypassing only affects 1.1% of spiking activity.

However, when we apply dynamically generated protecting schemes in SDBN, we notice great reduction in memory traffic. We repeatedly apply protecting reuse scores to processing neuron data inversely proportional to the mean reuse distances in that particular layer as discussed before. The smaller the mean reuse distance, the higher protection score we need to apply on importing the data. Most neurons in Layer 3 benefit from the scheme and we see marked reduction in weight access misses which brings down power consumption for this benchmark.

In SCNN, dynamically generated protection schemes provide us with a lot more energy savings than statically generated topological bypass requests. However, in the processing convolutional and subsampling layers, most neurons are dormant in nature, irrespective of the stimulus. With a few number of neurons requesting allocation under a protection scheme, SCNN benefits little from an adaptive extension relative to the SDBN benchmark.

Figure 16 shows the reduction in filtered off-chip memory requests for our dynamic adaptive policy relative to an LRU L1 cache. At least 42%, 36% and 29% reduction in off-chip memory traffic is observed for SCWN, SDBN and SCNN kernel respectively. With all cache configurations constant, this is directly in correlation with the mean reuse distances of neurons in each of these networks. The summary of results for our adaptive memory management strategy are listed in Table 2.

7.2 Runtime Leakage Mitigation

Figure 17 shows the variation of net savings in average leakage energy E_{save} as a percentage of baseline average leakage energy consumption for each hardware module with the respective average performance penalty incurred as a percentage of baseline single example inference cycles,

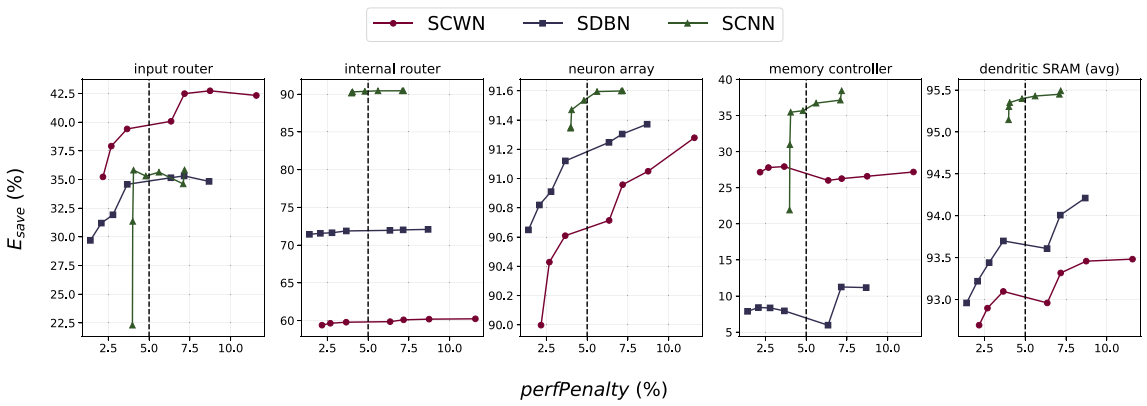


Figure 17 Net savings in leakage energy against net performance penalty incurred as $T_{idleDetect}$ is varied for each benchmark and hardware macro.

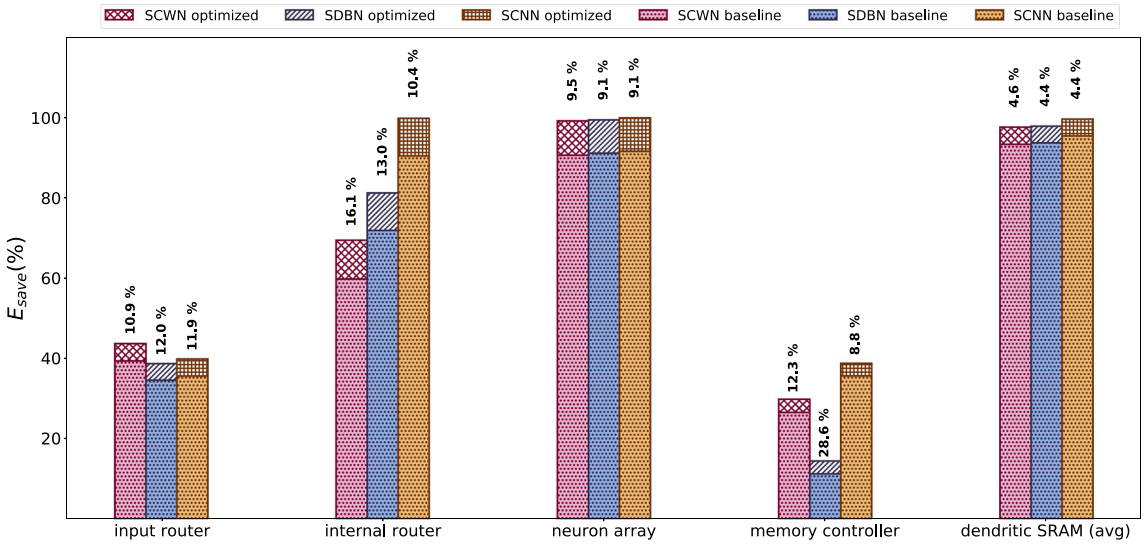


Figure 18 Improvement in net leakage savings in every module using its optimal $T_{idleDetect}$ for each benchmark.

swept across $T_{idleDetect} = 2^n$ with n ranging from 9 to 3 (from left to right). Note that the general trend is that as $T_{idleDetect}$ is gradually lowered, greater net savings is observed but at the cost of higher performance penalty. With lower $T_{idleDetect}$ values, greater $idleCycles$ percentage is attained. This contributes to more savings. However, depending on the size of the module as well as of the header device, $E_{overhead}$ resulting from the module can offset this contribution. Whenever we see a dip in the slope of these curves, this is because the $E_{overhead}$ from redundant sleep and wakeUp cycles overshoot the modest amount of savings gained from higher $idleCycles$. We use the experimental observations from this section to determine an *optimal* $T_{idleDetect}$ value per module per benchmark to ascertain the amount of net savings achievable in CyNAPSE for each benchmark. For this, we set a 5% upper limit on the $\%perfPenalty$ and select the highest $T_{idleDetect}$ value that gives us the best leakage savings. For instance, in the memory controller for SCNN, we use optimal $T_{idleDetect}$ as 128 and not 64 even though both are below the prescribed limit since 128 provides practically the same amount of savings with lower performance loss. For all our subsequent experiments, we use these optimal values to measure the incremental benefits of each recommendation.

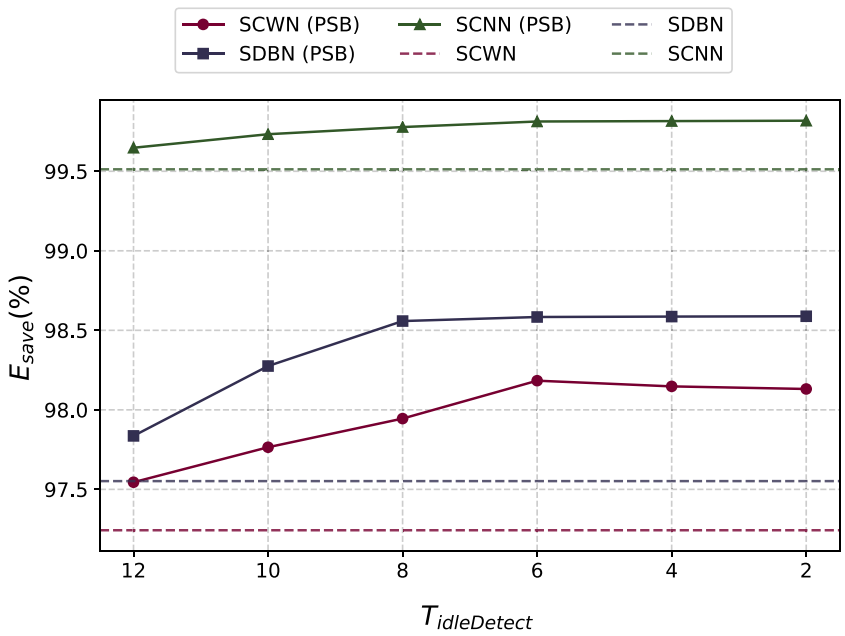


Figure 19 Average leakage energy savings in dendritic SRAM banks for very low $Ti_{idleDetect}$ values using pre-activation with no additional performance loss.

Table 3 Relative energy savings achieved using different policies.

Benchmarks	E_{save} for modules (%)				
	Input router	Internal router	Neuron array	Memory controller	Dendritic SRAM bank (avg.)
SCWN	**46.84**	69.43	99.45	31.04	98.18
SDBN	39.4	81.21	99.48	14.47	98.59
SCNN	40.12	**99.81**	**99.92**	**41.64**	**99.82**

Bold indicates best network

7.2.1 Application-specific RLM

Figure 18 shows the effect of application-specific enhancements on different modules for each benchmark. Each module improves leakage savings by either force-gating during large periods of idleness or by lowering $T_{idleDetect}$ to the lowest possible values as discussed in Section 5.2. However, force-starving the V_{DD} into a drowsy mode for dendritic SRAM banks generates the least amount of improvements. This is precisely because the period of time they can be force-starved is a short portion of the entire simulation period of a BT (the internal spike handling stage). Therefore, we propose the preactivation of banks to improve leakage savings in this region.

7.2.2 Pre-activated SRAM banks

Figure 19 shows the average leakage savings in SRAM banks obtained in each benchmark for a single example inference using pre-activated banks by sweeping $T_{idleDetect}$ to the lowest possible value. For reference, it shows the relative benefit over no pre-activation in each case. Note that the curves saturate at a point because further reducing $T_{idleDetect}$ increases the $E_{overhead}$ by orders of magnitude thereby reducing total attained savings. However, except for the SCWN, we can settle for the lowest possible value because unlike the initial situation in Figure 17, zero additional performance loss is now incurred for waking up the SRAM banks. For the SCWN, although, we notice a slight decay in savings with extremely aggressive leakage control. Therefore, we settle for the intermediate value.

The RLM results are summarized in Table 3 showing the final net leakage savings obtainable using our strategy for each benchmark. Note that since SCNN has the sparsest connectvity and topology, it is the most idle SNN out of the three and naturally, reaps the most benefit out of our strategy. The only exception is the input router module whose activity is skewed towards input intensity (average number of input spikes per BT) which is highest in SCNN and least in SCWN.

8 Conclusion

We have presented CyNAPSE, a reconfigurable architecture for accelerating SNNs. We showed that power dissipation in this system is dominated by off-chip memory. By using an application-specific caching strategy, we have achieved up to 44% power savings over the baseline and outperformed LRU by up to 22%. To further reduce the system's energy consumption, we attempt to conserve the on-chip static leakage energy dissipation. By progressively recommending general and application-specific strategies, we achieve over 99% leakage energy savings in sparsely active circuits or at least over 14% savings in busier modules. We strongly feel that a combination of these two approaches will greatly enhance the efficiency of massively multicore inference architectures sharing common off-chip main memories. Among possible avenues of future work, compiler driven optimizations could be valuable in trading off some performance for greater energy savings especially for benchmarks with reuse on larger timescales. Besides, our policy can be considered for any execution model that has a queue-based event processing in its front end. Any event-driven simulation platform such as embedded performance and energy counters [6], general purpose emulators [4] and others can possibly benefit from the adaptive scheme, if allocation latency at read time can be tolerated by individual instruction latency.

References

1. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557.
2. Allu, B., & Zhang, W. (2004). Static next sub-bank prediction for drowsy instruction cache. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems* (pp. 124–131): ACM.

3. Attwell, D., & Laughlin, S.B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, *21*(10), 1133–1145.
4. Bauer, J., Bershteyn, M., Kaplan, I., Vyedin, P. (1998). A reconfigurable logic machine for fast event-driven simulation. In *Proceedings 1998 Design and Automation Conference. 35th DAC.(Cat. No. 98CH36175)* (pp. 668–671): IEEE.
5. Belady, L.A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, *5*(2), 78–101.
6. Bellosa, F. (2000). The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system* (pp. 37–42): ACM.
7. Bi, G.q., & Poo, M.m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, *18*(24), 10464–10472.
8. Boahen, K. (2017). A neuromorph's prospectus. *Computing in Science & Engineering*, *19*, 14–28.
9. Brette, R., & Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, *94*(5), 3637–3642.
10. Cassidy, A., Andreou, A.G., Georgiou, J. (2011). Design of a one million neuron single fpga neuromorphic system for real-time multimodal scene analysis. In *2011 45Th annual conference on information sciences and systems* (pp. 1–6): IEEE.
11. Chandrasekar, K., Weis, C., Li, Y., Akesson, B., Wehn, N., Goossens, K. (2012). Drampower: Open-source dram power & energy estimation tool. http://www.drampower.info, 22.
12. Chen, Y.H., Krishna, T., Emer, J.S., Sze, V. (2016). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, *52*(1), 127–138.
13. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99.
14. Delbruck, T. (2016). Neuromorophic vision sensing and processing. In *2016 46Th european solid-state device research conference (ESSDERC)* (pp. 7–14): IEEE.
15. Diehl, P.U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*, 99.
16. Diehl, P.U., Neil, D., Binas, J., Cook, M., Liu, S.C., Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)* (pp. 1–8): IEEE.
17. Duong, N., Zhao, D., Kim, T., Cammarota, R., Valero, M., Veidenbaum, A.V. (2012). Improving cache management policies using dynamic reuse distances. In *2012 45Th annual IEEE/ACM international symposium on microarchitecture* (pp. 389–400): IEEE.
18. Flautner, K., Kim, N.S., Martin, S., Blaauw, D., Mudge, T. (2002). Drowsy caches: simple techniques for reducing leakage power. In *ACM SIGARCH Computer architecture news* (vol. 30, pp. 148–157): IEEE computer society.
19. Gerstner, W., & Kistler, W.M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge: Cambridge University Press.
20. Gerstner, W., & Naud, R. (2009). How good are neuron models? *Science*, *326*(5951), 379–380.
21. Goodman, D.F., & Brette, R. (2009). The brian simulator. *Frontiers in Neuroscience*, *3*, 26.
22. Han, S., Mao, H., Dally, W.J. (2015). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding coRR. arXiv:1510.00149.
23. Hinton, G.E., Osindero, S., Teh, Y.W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.
24. Hinton, G.E., Sejnowski, T.J., Poggio, T.A. (1999). *Unsupervised learning: foundations of neural computation*. Cambrdige: MIT press.
25. Hodgkin, A.L., & Huxley, A.F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544.
26. Hodgkin, A.L., Huxley, A.F., Katz, B. (1952). Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of Physiology*, *116*(4), 424–448.
27. Hopfield, J.J. (2007). Hopfield network. *Scholarpedia*, *2*(5), 1977.
28. Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H., Bose, P. (2004). Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 international symposium on Low power electronics and design* (pp. 32–37): ACM.
29. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, *18*(1), 6869–6898.
30. Izhikevich, E.M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural networks*, *15*(5), 1063–1070.
31. Jiang, H., Marek-Sadowska, M., Nassif, S.R. (2005). Benefits and costs of power-gating technique. In *2005 International conference on computer design* (pp. 559–566): IEEE.
32. Jiang, S., & Zhang, X. (2002). Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review*, *30*(1), 31–42.
33. Jolivet, R., Lewis, T.J., Gerstner, W. (2004). Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy. *Journal of Neurophysiology*, *92*(2), 959–976.
34. Jolivet, R., Rauch, A., Lüscher, H.R., Gerstner, W. (2006). Integrate-and-fire models with adaptation are good enough. In *Advances in neural information processing systems* (pp. 595–602).
35. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44Th annual international symposium on computer architecture (ISCA)* (pp. 1–12): IEEE.
36. Jug, F. (2012). *On competition and learning in cortical structures*. Ph.D. thesis, ETH Zurich.
37. Kaxiras, S., Hu, Z., Martonosi, M. (2001). Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proceedings 28th annual international symposium on computer architecture* (pp. 240–251): IEEE.
38. Khan, S.M., Tian, Y., Jimenez, D.A. (2010). Sampling dead block prediction for last-level caches. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 175–186): IEEE Computer Society.
39. Kim, Y., Yang, W., Mutlu, O. (2015). Ramulator: a fast and extensible dram simulator. *IEEE Computer Architecture Letters*, *15*(1), 45–49.
40. Kim, Y., Zhang, Y., Li, P. (2015). A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *11*(4), 38.

41. Koch, C., & Segev, I. (1998). *Methods in neuronal modeling: from ions to networks*. Cambridge: MIT press.
42. LeCun, Y., Cortes, C., Burges, C. (2010). Mnist handwritten digit database at&t labs.
43. Li, S., Chen, K., Ahn, J.H., Brockman, J.B., Jouppi, N.P. (2011). Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques. In *Proceedings of the International Conference on Computer-Aided Design* (pp. 694–701): IEEE Press.
44. Li, Y., & Pedram, A. (2017). Caterpillar: Coarse grain reconfigurable architecture for accelerating the training of deep neural networks. In *2017 IEEE 28Th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 1–10): IEEE.
45. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, *234*, 11–26.
46. Mahowald, M.A., & Mead, C. (1991). The silicon retina. *Scientific American*, *264*, 76–82.
47. Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636.
48. Moerland, P., & Fiesler, E. (1996). Hardware-friendly learning algorithms for neural networks: an overview. In *Proceedings of Fifth International Conference on Microelectronics for neural networks* (pp. 117–124): IEEE.
49. Neckar, A., Fok, S., Benjamin, B.V., Stewart, T.C., Oza, N.N., Voelker, A.R., Eliasmith, C., Manohar, R., Boahen, K. (2019). Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, *107*(1), 144–164.
50. Neil, D., & Liu, S.C. (2014). Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *22*(12), 2621–2628.
51. O'Connor, P., Neil, D., Liu, S.C., Delbruck, T., Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, *7*, 178.
52. Podili, A., Zhang, C., Prasanna, V. (2017). Fast and efficient implementation of convolutional neural networks on fpga. In *2017 IEEE 28Th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 11–18): IEEE.
53. Powell, M., Yang, S.H., Falsafi, B., Roy, K., Vijaykumar, T. (2000). Gated-v dd: a circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 international symposium on Low power electronics and design* (pp. 90–95): ACM.
54. Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., Indiveri, G. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in Neuroscience*, *9*, 141.
55. Qureshi, M.K., Jaleel, A., Patt, Y.N., Steely, S.C., Emer, J. (2007). Adaptive insertion policies for high performance caching. *ACM SIGARCH Computer Architecture News*, *35*(2), 381–391.
56. Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1985). *Learning internal representations by error propagation*. Technical report. California Univ San Diego La Jolla Inst for Cognitive Science.
57. Schuman, C.D., Potok, T.E., Patton, R.M., Birdwell, J.D., Dean, M.E., Rose, G.S., Plank, J.S. (2017). A survey of neuromorphic computing and neural networks in hardware. arXiv:1705.06963.
58. Shepherd, G.M. (2003). *The synaptic organization of the brain*. Oxford: Oxford University Press.
59. Wen, B., & Boahen, K. (2009). A silicon cochlea with active coupling. *IEEE Transactions on Biomedical Circuits and Systems*, *3*(6), 444–455.
60. Wijeratne, S., Jayaweera, S., Dananjaya, M., Pasqual, A. (2018). Reconfigurable co-processor architecture with limited numerical precision to accelerate deep convolutiosnal neural networks. In *2018 IEEE 29Th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 1–7): IEEE.
61. Yu, T., Park, J., Joshi, S., Maier, C. (2012). Cauwenberghs, g.: 65k-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing. In *2012 IEEE Biomedical circuits and systems conference (bioCAS)* (pp. 21–24): IEEE.
62. Zhao, R., Liu, S., Ng, H.C., Wang, E., Davis, J.J., Niu, X., Wang, X., Shi, H., Constantinides, G.A., Cheung, P.Y., et al. (2018). Hardware compilation of deep neural networks: an overview. In *2018 IEEE 29Th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 1–8): IEEE.
63. Zhao, W., Fu, H., Luk, W., Yu, T., Wang, S., Feng, B., Ma, Y., Yang, G. (2016). F-cnn: an fpga-based framework for training convolutional neural networks. In *2016 IEEE 27Th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 107–114): IEEE.
64. Zhou, H., Toburen, M.C., Rotenberg, E., Conte, T.M. (2003). Adaptive mode control: a static-power-efficient cache design. *ACM Transactions on Embedded Computing Systems (TECS)*, *2*(3), 347–372.