An Adaptive Memory Management Strategy Towards Energy Efficient Machine Inference in Event-Driven Neuromorphic Accelerators

Saunak Saha, Henry Duwe, and Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa, USA
Email:{saha, duwe, zambreno}@iastate.edu

***Abstract*—Spiking neural networks are viable alternatives to classical neural networks for edge processing in low-power embedded and IoT devices. To reap their benefits, neuromorphic network accelerators that tend to support deep networks still have to expend great effort in fetching synaptic states from a large remote memory. Since local computation in these networks is event-driven, memory becomes the major part of the system's energy consumption. In this paper, we explore various opportunities of data reuse that can help mitigate the redundant traffic for retrieval of neuron meta-data and post-synaptic weights. We describe CyNAPSE, a baseline neural processing unit and its accompanying software simulation as a general template for exploration on various levels. We then investigate the memory access patterns of three spiking neural network benchmarks that have significantly different topology and activity. With a detailed study of locality in memory traffic, we establish the factors that hinder conventional cache management philosophies from working efficiently for these applications. To that end, we propose and evaluate a domain-specific management policy that takes advantage of the forward visibility of events in a queue-based event-driven simulation framework. Subsequently, we propose network-adaptive enhancements to make it robust to network variations. As a result, we achieve 13-44% reduction in *system* power consumption and a 8-23% improvement over conventional replacement policies.**

***Index Terms*—Neuromorphic, Spiking Neural Networks, Reconfigurable, Accelerator, Memory, Caching, Energy efficiency**

I. 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 [1]. 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 [2]. 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 [3]–[6], training [7], [8] as well as compilers to bridge the gap between software simulation and hardware acceleration [9]. 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 [10].

The major inefficiency of these networks result from continuous activation of analog neurons and expensive MAC operations 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*. Spiking neurons activate only when they reach a certain threshold and, thereby, only sparsely communicate their post-synaptic weights [11]. As a result, SNNs are extremely energy efficient, fast, noise-invariant and give great insight into neuroscientific understanding. Furthermore, SNNs are inherently sensitive to temporal distribution of input data and therefore, are universally suited to all kinds of spatiotemporal pattern recognition. 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 [12], 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 [13]. While analog hardware can emulate biological realism and energy efficiency to a much greater extent [14]–[16], they are plagued by process, voltage and temperature (PVT) variations and is especially difficult to scale to today's technology nodes [17]. Digital implementations provide these advantages and are suitable for integration to embedded systems and software ecosystems [18]–[20]. 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 perfor-

TABLE I
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%

mance. However, the usefulness of a digital neural accelerator is strongly dependent on its energy efficiency.To that end, this paper makes the following contributions:

- We present a digital SNN accelerator with reconfigurable network topology and neural dynamics.
- We study the memory access patterns of several SNN workloads and observe that spike processing is predominantly memory-intensive with respect to power consumption.
- 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.

II. SPIKING NEURAL NETWORK BENCHMARKS

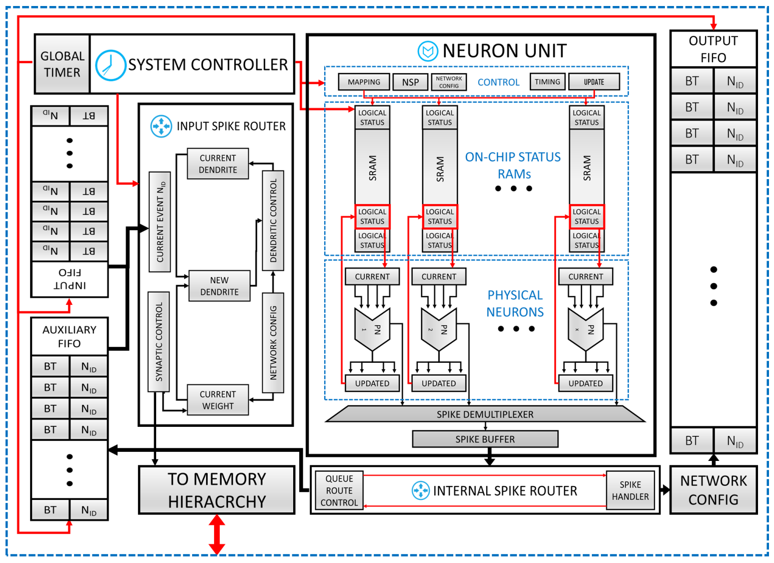
We have selected three different SNNs as our benchmarks for this work. They bring significant diversity to our case study by using different neural dynamics and training algorithms to solve the standard task of MNIST digit recognition [21].

The first network is a winner-take-all network with lateral inhibition trained using a biologically plausible learning rule known as *spike timing dependent plasticity* (STDP) [22]. It is built from *leaky integrate and fire* (LIF) neurons that integrate leaky conductances instead of constant current which makes these neurons more biologically realistic than usual LIF neurons. This network achieves a maximum accuracy of 95% on MNIST as shown in [23]. 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* [24], an unsupervised learning technique, and has been converted to an event-based network post-training, achieving a maximum accuracy of 92% [25]. This network has no recurrent connections, is built from simple LIF neurons and is referred to as the *spiking deep belief network* (SDBN). The third network follows a convolutional topology and has been trained using standard *error backpropagation* [26]. Thereafter, it has been converted into the spiking domain following the methodology advised by [27]. 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 unlike LIF and have much simpler dynamics. 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).

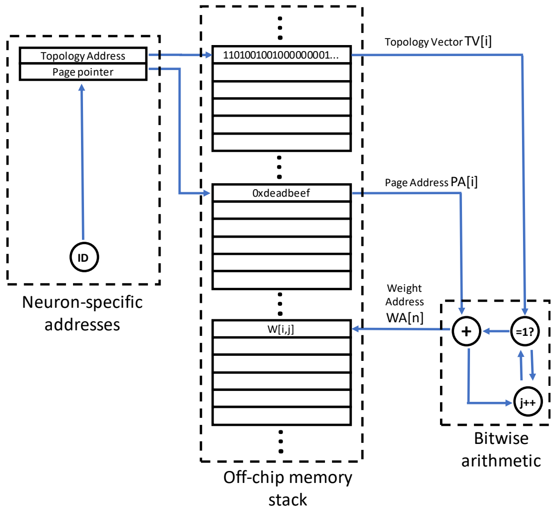
The activity of each of these networks along with their internal connectivity will determine the energy footprint of these networks. Activity of a network depends on a large number of tunable *hyperparameters*. SNNs have an inherent temporal nature to them which is tunable by controlling the maximum 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 I lists succinct details of the benchmarks.

III. THE CYNAPSE MICROARCHITECTURE

While existing digital SNN accelerators like [28] can simulate SNNs that use fixed neural dynamics like LIF, they cannot account for other phenomenonological extensions such as leaky conductance-based synapses, spike-frequency adaptation, excitatory and inhibitory populations. CyNAPSE is a similar neural inference architecture with support for reconfigurable neural dynamics. It implements a *generalized* integrate and fire model of neurons [29] that provide for a wide variety of representative neural behaviors. By reducing this model of spiking neurons, simpler models can be obtained which allows universal acceleration of SNNs. It works in a co-processor configuration with a master CPU that can run spike-supply and handling routines. Alternatively, it can be



(a)



(b)

Fig. 1. (a) The CyNAPSE microarchitecture. Broadly, it consists of a neuron unit with on-chip neuron circuits and dendritic status SRAMs, an input event router, a Spike handler, FIFO priority queues and a system controller. (b) The control-flow of a single synaptic lookup in the CyNAPSE routing phase

employed in an embedded environment assisted by a spiking retina or dynamic vision sensor and motor control. Fig. 1a shows the baseline microarchitecture of the system.

A single CyNAPSE core can simulate a maximum number of N logical neurons and N^2 logical synapses. It uses a smaller number X of physical on-chip neuron units, each time-multiplexed to support the entire promised ensemble. This also requires X on-chip dendritic trees implemented in SRAMs storing all N intermediate neuron statuses (N and X being design-time configurable parameters). Input neurons of an SNN that produce the first wave of stimulus response, or *input* spikes, are modeled off-chip in software while processing neurons with spiking neural dynamics producing *internal* spikes are modeled in multiplexed hardware. CyNAPSE uses an Address Event Representation (AER) protocol [30] to encode all spike events from a real-time representation to an *in situ* biological time representation. Each individual spike is encoded into an AER packet containing the biological timestep and the ID of the logical neuron that produced it. These AER spike packets are read into an input FIFO priority queue in a streaming manner as they are produced. The input queue pops the top event to route if its timestep matches the current biological time. The input router performs a full synaptic lookup for that particular neuron's weights and routes them to appropriate logical dendritic trees. The neuron unit updates itself after all events of the current timestep have been routed and refreshed statuses corresponding to all logical neurons appear in all the on-chip scratchpad memories. All *internal* spikes produced in this timestep are filtered into a spike buffer and routed to the auxiliary queue for handling in the next timestep. This marks the end of all computation within the timestep and a barrier synchronization allows the system to tick the global timer to the next timestep. The details of the design and evaluation process are mentioned in section VI.

IV. Memory power consumption

Depending on the average connectivity of a network, CyNAPSE spends most of its time fetching weights for the long routing process. Since the amount of storage required in large networks is infeasible for realization on-chip, the entire process becomes dominated by off-chip memory traffic. This leads to a compromise of efficiency as a significant amount of energy is consumed in fetching synaptic data from a remote DRAM storage.

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 Fig. 2, we show the full power consumption profile of the CyNAPSE system (see Section VI 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 [31]. Regardless, the opportunity and importance of memory power savings in such a system is clear and calls for architectural investigation to explore possible solutions.

While algorithmic optimizations like pruning and quantization of weights [32], [33] 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.

V. Energy Efficient Memory Management

Nominally, a fixed table of neuron weights requires $O(N^2)$ memory where N is the number of supported logical neurons

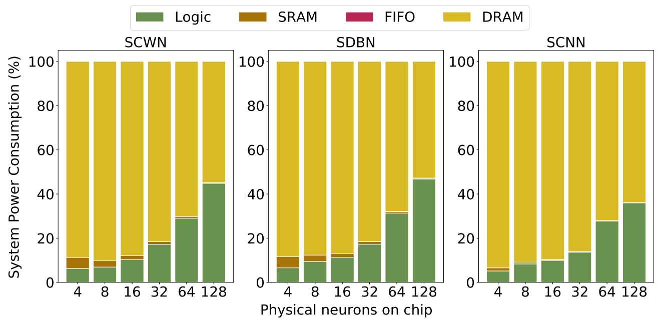


Fig. 2. System power consumption of CyNAPSE broken down for each benchmark shows significant DRAM percentage

on chip. However, often with deep networks and sparser topologies, this is highly inefficient. We eliminate both the hard storage requirement and the naive memory lookup strategy in CyNAPSE by storing weights in *pages*. This adds a second layer of indirection via a translation table that directs the router to the relevant page that belongs 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 number of weights of a certain neuron) and, thus, are marked by the starting address of the page, or a *page pointer*. The number of weights in a 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 Fig. 1b i.e. a topology vector, a page address, and all the weights in that page via a simple bitwise arithmetic scheme. We observe a number of opportunities to exploit locality and reuse as a result of this data flow.

A. 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 [34]. While Belady's optimal replacement policy [35] requires an infeasible view of the future, policies like DIP [36], RRIP [37] and LIRS [38] 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:

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 VI) 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.

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.

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) 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.

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.

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. Fig. 3 summarizes the baseline memory control scheme.

B. 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 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

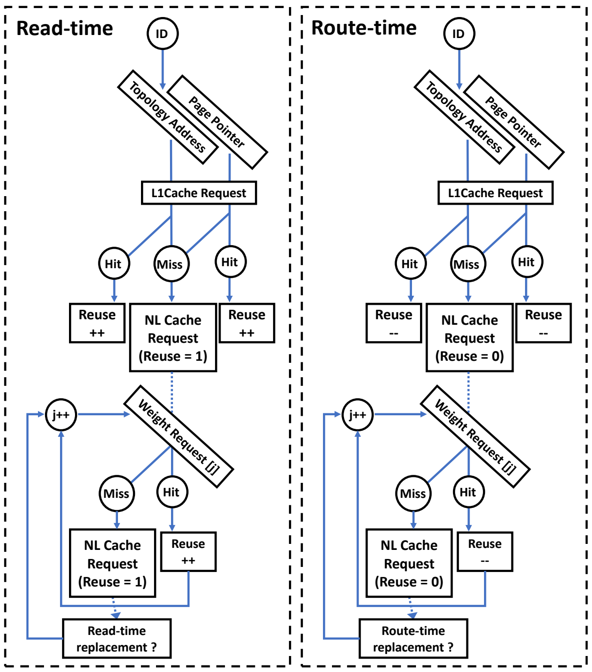


Fig. 3. Baseline read-time and route-time memory control schemes

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. Fig. 4 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. We propose two techniques to extend our scheme:

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 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

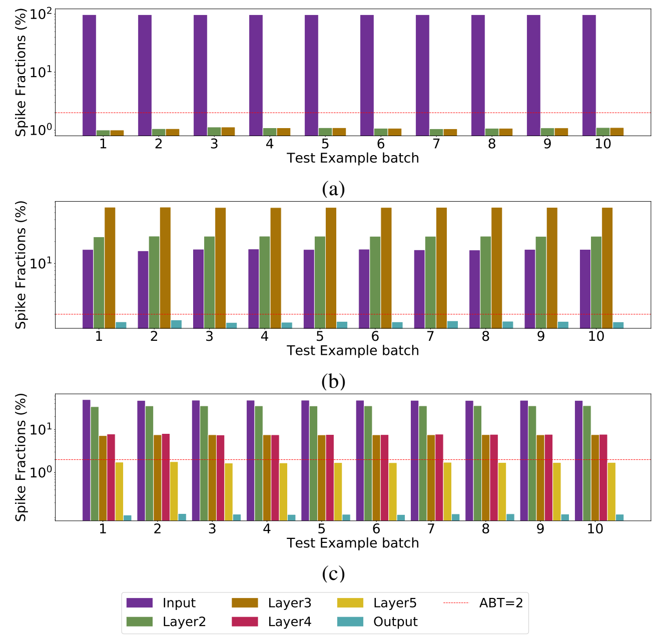


Fig. 4. Spike statistics generated dynamically by the CyNAPSE simulator to adaptively configure cache requests. (a), (b) and (c) show layer-wise activity fractions for the SCWN, SDBN and SCNN respectively with time.

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.

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 Fig. 4). 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. Fig. 5 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.

VI. Design Methodology

A. Low-level design

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

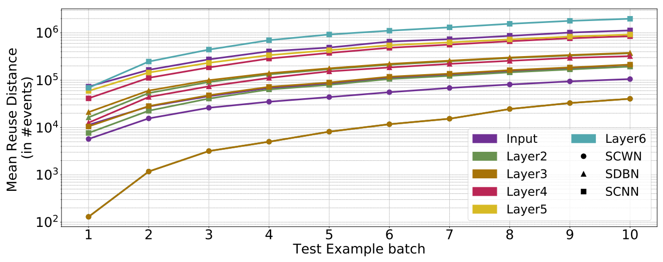


Fig. 5. Layer-wise mean reuse distances of neurons in each benchmark

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. 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 [39].

B. High-level exploration

For high-level architectural exploration of the memory subsystem, we also built a software simulation of the CyNAPSE core and an associated cache simulator. 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 [40] in appropriate organization, speed and timing configurations to dump JEDEC standard command traces in DRAMPower [41] 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

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

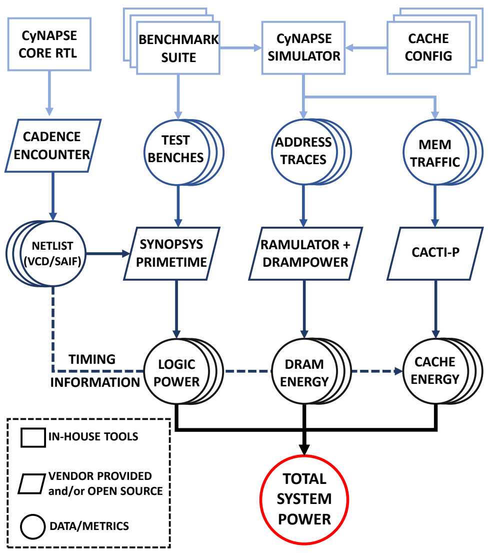


Fig. 6. Experimental Infrastructure and flow

of all digits. Fig. 6 shows our experimental infrastructure and tool flow. 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.

VII. Results

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.

A. Read-time replacement

In Section V, 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 Fig. 7, 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 distance (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.

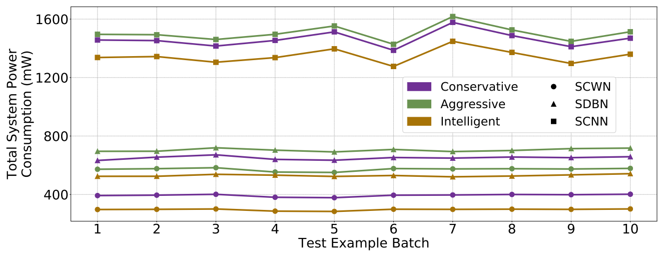


Fig. 7. Comparative analysis of the various read-time replacement handling approaches

B. LRU vs Random vs Our Policy

Fig. 8 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% 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

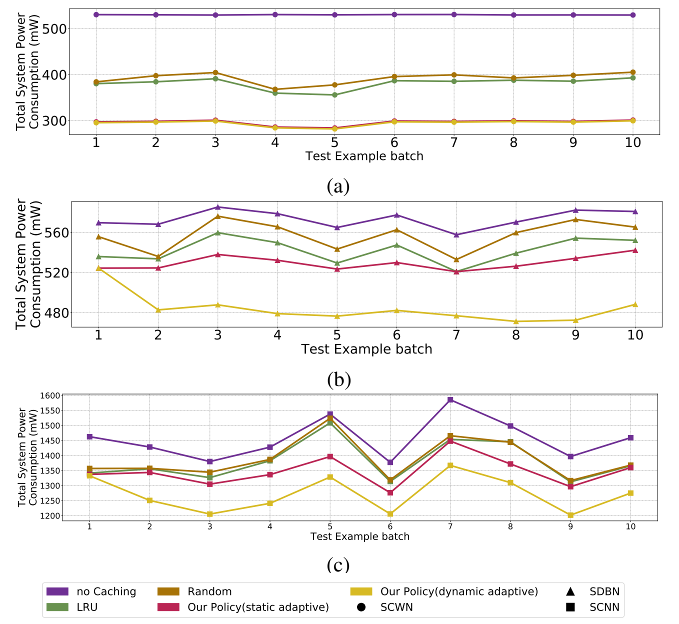


Fig. 8. Comparative analysis of energy savings using our policy over conventional policies for the (a) SCWN, (b) SDBN and (c) SCNN benchmarks

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 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.

C. 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.

TABLE II
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%

In SCWN, 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. The results are summarized in Table II.

VIII. Conclusion

We have presented CyNAPSE, a reconfigurable architecture for accelerating SNNs. We showed that power dissipation in this system is dominated by 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%. A possible avenue of future work within this area could be towards core power dissipation reduction using leakage control techniques since majority of its logic power consumption is in the idle state. For benchmarks with reuse on larger-timescales, compiler driven optimizations could be valuable in trading off some performance for greater energy savings. 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 [42], general purpose emulators [43] and others can possibly benefit from this scheme, if allocation latency at read time can be tolerated by individual instruction latency.

Acknowledgements

This work is supported in part by the National Science Foundation (NSF) under award CCF-1149539.

References

[1] K. Boahen, “A neuromorph’s prospectus,” *Computing in Science & Engineering*, vol. 19, pp. 14–28, 2017.

[2] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[5] A. Podili, C. Zhang, and V. Prasanna, "Fast and efficient implementation of convolutional neural networks on fpga," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 11–18.

[6] S. Wijeratne, S. Jayaweera, M. Dananjaya, and A. Pasqual, "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)*. IEEE, 2018, pp. 1–7.

[7] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "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)*. IEEE, 2016, pp. 107–114.

[8] Y. Li and A. Pedram, "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)*. IEEE, 2017, pp. 1–10.

[9] R. Zhao, S. Liu, H.-C. Ng, E. Wang, J. J. Davis, X. Niu, X. Wang, H. Shi, G. A. Constantinides, P. Y. Cheung *et al.*, "Hardware compilation of deep neural networks: An overview," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–8.

[10] D. Attwell and S. B. Laughlin, "An energy budget for signaling in the grey matter of the brain," *Journal of Cerebral Blood Flow & Metabolism*, vol. 21, no. 10, pp. 1133–1145, 2001.

[11] W. Gerstner, "Spiking neurons," MIT-press, Tech. Rep., 1998.

[12] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[13] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[14] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in neuroscience*, vol. 9, p. 141, 2015.

[15] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.

[16] T. Yu, J. Park, S. Joshi, C. Maier, and G. Cauwenberghs, "65k-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing," in *2012 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2012, pp. 21–24.

[17] Y. Kim, Y. Zhang, and P. Li, "A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 4, p. 38, 2015.

[18] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "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*, vol. 34, no. 10, pp. 1537–1557, 2015.

[19] D. Neil and S.-C. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

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

[21] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database. at&t labs," 2010.

[22] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[23] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[25] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in neuroscience*, vol. 7, p. 178, 2013.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[27] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[28] D. Ma, J. Shen, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *Journal of Systems Architecture*, vol. 77, pp. 43–51, 2017.

[29] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner, "Integrate-and-fire models with adaptation are good enough," in *Advances in neural information processing systems*, 2006, pp. 595–602.

[30] K. Boahen, "Communicating neuronal ensembles between neuromorphic chips," *Neuromorphic Systems Engineering*, pp. 229–259, 1998.

[31] A. Cassidy, A. G. Andreou, and J. Georgiou, "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*. IEEE, 2011, pp. 1–6.

[32] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[33] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2016.

[34] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 389–400.

[35] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[36] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, 2007.

[37] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 175–186.

[38] S. Jiang and X. Zhang, "Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[39] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2011, pp. 694–701.

[40] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[41] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, vol. 22, 2012.

[42] F. Bellosa, "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*. ACM, 2000, pp. 37–42.

[43] J. Bauer, M. Bershteyn, I. Kaplan, and P. Vyedin, "A reconfigurable logic machine for fast event-driven simulation," in *Proceedings 1998 Design and Automation Conference. 35th DAC.(Cat. No. 98CH36175)*. IEEE, 1998, pp. 668–671.