Published in the Proceedings of 28th IFIP/IEEE International Conference on Very Large Scale Integration, 5-9 October 2020, Salt Lake City, Utah, USA, which should be cited to refer to this work.

A Hybrid Cache HW/SW Stack for Optimizing Neural Network Runtime, Power and Endurance

William Andrew Simon*, Alexandre Levisse*, Marina Zapater†* and David Atienza*
*Embedded Systems Laboratory (ESL), Swiss Federal Institute of Technology Lausanne (EPFL)
†University of Applied Sciences Western Switzerland (HEIG-VD / HES-SO)
Email: {william.simon, alexandre.levisse, marina.zapater, david.atienza}@epfl.ch

***Abstract*—Hybrid caches consisting of both SRAM and emerging Non-Volatile Random Access Memory (eNVRAM) bitcells increase cache capacity and reduce power consumption by taking advantage of eNVRAM's small area footprint and low leakage energy. However, they also inherit eNVRAM's drawbacks, including long write latency and limited endurance. To mitigate these drawbacks, many works propose heuristic strategies to allocate memory blocks into SRAM or eNVRAM arrays at runtime based on block content or access pattern. In contrast, this work presents a HW/SW Stack for Hybrid Caches (SHyCache), consisting of a hybrid cache architecture and supporting programming model, reminiscent of those that enable GP-GPU acceleration, in which application variables can be allocated explicitly to the eNVRAM cache, eliminating the need for heuristics and reducing cache access time, power consumption, and area overhead while maintaining maximal cache utilization efficiency and ease of programming. SHyCache improves performance for applications such as neural networks, which contain large numbers of invariant weight values with high read/write access ratios that can be explicitly allocated to the eNVRAM array. We simulate SHyCache on the gem5-X architectural simulator and demonstrate its utility by benchmarking a range of cache hierarchy variations using three neural networks, namely, Inception v4, ResNet-50, and SqueezeNet 1.0. We demonstrate a design space that can be exploited to optimize performance, power consumption, or endurance, depending on the expected use case of the architecture, while demonstrating maximum performance gains of 1.7/1.4/1.3x and power consumption reductions of 5.1/5.2/5.4x, for Inception/ResNet/SqueezeNet, respectively.**

***Index Terms*—eNVRAM, STT-MRAM, hybrid caches, neural networks, low-power systems**

I. Introduction

In recent years, Neural Networks (NNs) have gained popularity for performing a variety of tasks such as image recognition [1], object detection [2], and natural language processing [3]. In an effort to enable NNs on as many devices as possible, many optimizations to reduce NN memory and compute overhead have been proposed, such as quantization, pruning, and custom layers [4]–[6]. Even so, the memory footprint of "small" NNs often still measure in the order of MBs [7]; therefore, memory enhancements that exploit the invariant nature of these weights can continue to improve NN performance on area restricted devices. In this regard, Figure 1 displays the read access to write access (read/write) ratios of the memory blocks that account for 98% of total inference-time read accesses for the SqueezeNet neural net [6]. As can be seen, the memory blocks with the highest read/write ratio contain weight values, as these blocks are only written during line fills from lower memory levels during inference, never from the processor. Weight values also account for almost 40% of all memory accesses performed at runtime. It can be inferred that improving processor read access to these weight values will result in overall application performance gain.

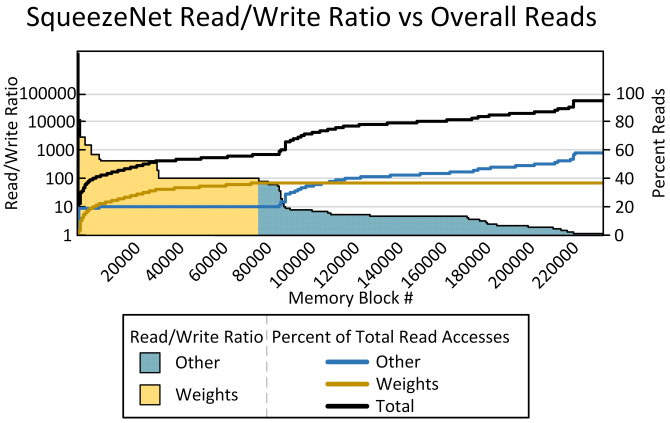

Fig. 1: Read/write access ratios in relation to total read accesses. Weight accesses account for nearly 40% of all reads.

In this context, Hybrid Caches (HCs), consisting of SRAM and emerging Non-Volatile Random Access Memories (eNVRAMs), can be used to accelerate NNs. eNVRAM's low area footprint and leakage energy enable more efficient execution of memory intense algorithms by increasing cache capacity with little area overhead, while simultaneously reducing power consumption. However, eNVRAMs also incur a high write energy cost and have limited endurance. It is therefore necessary to optimize write strategies to avoid unnecessary writes. Many works have proposed heuristical, predictive placement strategies. In contrast, a deterministic cache allocation strategy enables the utilization of eNVRAM allocated variables to choose which values are written to eNVRAM and avoid unnecessary transfer between SRAM and eNVRAM, thus providing maximum cache usage efficiency. In the case of NNs, invariant weight values are an excellent candidate for eNVRAM storage.

To this end, we present a HW/SW Stack for Hybrid Caches (SHyCache), consisting of a HC architecture and deterministic

cache allocation strategy, supported via a programming model reminiscent of those utilized to enable GP-GPU computation, illustrated in Figure 2-a. SHyCache enables precise control over data placement within the cache, and is compatible with heuristical hybrid cache strategies. Then, we explore the HC design space by considering various eNVRAM/SRAM HC ratios, and benchmark SHyCache in the gem5-X [8] architectural simulator, on a range of NNs of varying computational complexity and memory footprint.

The contributions of this paper are as follows:

- We introduce SHyCache, an HC architecture with a deterministic allocation strategy allowing for precise data allocation within an HC. SHyCache's allocation strategy is compatible with other hybrid cache allocation strategies.
- We develop a programming model with a C++ support library allowing easy integration of SHyCache support into any existing application.
- We implement SHyCache in the gem5-X architectural simulator and explore the HC design space to optimize for performance, power, and endurance, demonstrating performance gains of 1.7/1.4/1.3x and power consumption reductions of 5.1/5.2/5.4x for the Inception v4, ResNet-50, and SqueezeNet 1.0 NNs, respectively.

The rest of this paper is organized as follows. Section II explores related state-of-the-art work. Section III details SHyCache's HC architecture. Section IV details SHyCache's programming model and support library and discusses tandem implementation with other allocation strategies. Section V details our benchmarking methodology, while Section VI discusses results. Finally, Section VII concludes this work.

II. Related Work

A. Resistive Random Access Memory

Emerging nonvolatile memories, including phase change [9], resistive [10] and spin-torque transfer [11] memories, have gained popularity in recent years thanks to their small size, up to 4x smaller than 6T SRAM cells [12], and low leakage energy resulting from their nonvolatility. However, eNVRAM also suffers from long/high-energy write operations, and low endurance due to the underlying physics of the technology. In order to efficiently utilize eNVRAM within an architecture, eNVRAM-specific optimizations must be implemented to magnify their advantages while mitigating or masking drawbacks.

B. Hybrid Cache Design and Allocation Strategies

One implementation of eNVRAM within the memory hierarchy involves placement alongside standard SRAM cache arrays, creating a Hybrid Cache (HC) hierarchy, as illustrated in Figure 2-b. This architecture increases cache capacity while also reducing power consumption [13]. However, HCs also inherit eNVRAM's disadvantages as described above. Further, a naive HC implementation may magnify these disadvantages, as the frequency of cache writes, and therefore cache lifetime, is highly variant depending on the application [14], as well as reducing performance even while not in use due to slower access time. Many works have therefore proposed memory management strategies [15] for allocating blocks in either SRAM or eNVRAM depending on a variety of factors. The majority of these strategies are heuristic [14], [16], [17] or compiler based [18], [19]. In contrast, this work presents an application driven allocation strategy which obviates the need for heuristics and takes advantage of cases in which an application's data is constant, such as neural networks.

C. Neural Networks

Neural networks are a class of applications that accept inputs in various forms such as images, text, or audio, process them through the use of consecutive compute layers, and return an output, for example, the class of the input. Each hidden layer consists of one or more "neurons" of various function. The two most widely used neuron layers are the fully connected and convolutional layer. Both layers perform multiply-and-accumulate operations between the outputs of the previous layer and an array of previously trained weights. These layers require a massive number of weight values; the classical Alexnet neural network utilizes 3.78M weights (144MB for floating point weights) in its first fully connected layer [1]. Convolutional layers reduce memory footprint by using small (ex. 3x3) weight kernels that are convolved with the layer input. While convolutional layers greatly reduce the NN's memory footprint, they are generally still large in an absolute sense; for example, the SE-ResNeXt-50 NN achieves the highest Top-1 and Top-5% accuracy on the ImageNet-1k database at a low operational complexity, yet still contains over 10MB of weights [7]. Managing such large quantities of weights is imperative for efficient NN execution.

III. Hybrid Cache Architectural Design

SHyCache's hybrid cache consists of arrays of two memory types, one being standard 6T SRAM based memory and the other a flavor of eNVRAM, as illustrated in Figure 2-b. Each bitcell array is indexed by a separate tag array. The combined area of the tag array memory macros is equivalent to a single tag array of an equivalently sized monolithic cache memory, plus overhead for tag array periphery. As our data placement strategy is deterministic, as described in Section IV, only one data/tag array needs be accessed per read/write, reducing power consumption in comparison to heuristic strategies that must check both arrays for the data as its location is not known beforehand. In regards to cache access latency, it is important to note that, as only either the SRAM or eNVRAM is accessed, SHyCache's allocation strategy does not impact access latency of programs not utilizing the eNVRAM, i.e. the system kernel, and thus does not impact standard system performance. This is not necessarily the case if other heuristic or compiler-based allocation strategies are implemented alongside SHyCache's allocation strategy, as discussed in Section IV-C.

As illustrated in Figure 2-b, we consider HC configurations at both the L1 and L2 levels. We utilize an inclusive cache policy for reasons explained in Section VI. The L1 cache utilizes parallel tag/data access to reduce access time, while the L2 uses sequential tag/data access to reduce power consumption.

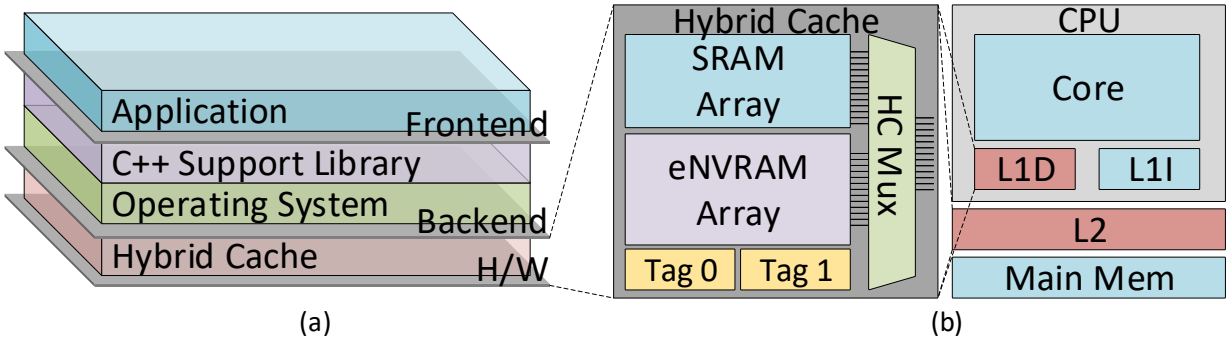

Fig. 2: SHyCache is a HW/SW stack (a) that enables efficient use of a hybrid cache (b).

```
using namespace SHyCache;
void loadWeights(string weightsFile, size_t len) s{
    // Declare var to be stored to eNVRAM portion of
    // cache. Allocation handled by helper library.
    float32_nv *weightsPtr = new float32_nv[len];
    // Open file containing pre-calculated weights.
    ifstream wIn(weightsFile);
    // Store weights to previously allocated memory.
    wIn.read((char *)weightsPtr,len);
    //...Perform inference...
    // Clean up
    delete weightsPtr;
}
```

Listing 1: Allocating the hybrid cache is done by allocating the variable pointer within the memory mapped region reserved for eNVRAM.

IV. Integrating SHyCache's Programming Model into Neural Network Frameworks

Several characteristics of NN weights enable NNs to be accelerated by HCs. The first is that, as previously mentioned, most NNs that achieve >80% Top-5% accuracy utilize large amounts (in the order of MBs) of weights. Second, weight values are calculated at training time and not modified during inference. Finally, fully connected and convolutional layers result in spatially local data accesses. These characteristics make eNVRAM suitable for storing NN weights. High eNVRAM bitcell density allows more weights to be stored without the need for eviction, while the long write latency of eNVRAM is mitigated by the read-only nature of weights.

A. Enabling HC allocation at the Operating System Level

Most previous HC works utilize heuristic strategies to allocate data either in the SRAM or eNVRAM bitcell arrays depending on various factors. In contrast, because the location and value of NN weight values are deterministic, no heuristic strategy is necessary for weight allocation in SHyCache. This is accomplished at the system level by reserving a portion of memory at operating system startup that can be mapped by an application in the same manner that peripherals can be mapped and accessed by user applications. When variables allocated to the memory range reserved for eNVRAM caching are fetched into the cache hierarchy, an address predecoder analyzes the MSBs of the incoming address. Addresses within the reserved memory region will be automatically cached in eNVRAM array when accessed. Such a strategy does not require any compiler modification and minimal application modification. Architectural modifications will depend on the nature of the architectures virtual-physical memory address translation. If the reserved memory is virtual, when address translation occurs the processor can tag the memory access with a bit to indicate if it is a standard or eNVRAM memory access before passing the access to the cache hierarchy. If the reserved memory is physical, or there is no virtual-physical translation, for example in embedded systems that use tightly coupled memory [20], the type of memory access will be attained as a byproduct of the address decoding that occurs during cache access, hence, no modification to the processor architecture is necessary.

B. Enabling HC allocation at the Application Level

At the application level, the programmer utilizes SHyCache's C++ data types to instantiate variables that will be allocated to the eNVRAM, as seen in the example function in Listing 1. The support library then facilitates the allocation of variables to the eNVRAM memory region without further programmer intervention by allocating the variables to the memory mapped region described in Section IV-A. Current NN frameworks such as Tensorflow, Caffe, and the ARM Compute Library perform several preprocessing stages upon weights before storing them in their final tensor, after which this tensor is not modified during inference. Framework extension to support HCs consists therefore of redirecting the output of the final preprocessing stage to store weight values in a tensor stored in the eNVRAM cache, resulting in no extra data movement overhead. In this work, we extend the ARM Compute Library, a neural network framework optimized for ARM processors [21], with SHyCache's C++ support library, however such extensions could be applied to any of the aforementioned frameworks to enable HC support. It should be noted that the use of a support library obviates the need for any language compiler modifications, simplifying the deployment process.

C. Co-Implementation with Other Hybrid Cache Allocation Strategies

One advantage of SHyCache is that it does not preclude the use of other heuristical [14] or compiler-driven [18] HC

TABLE I: Simulator Parameters

Processor	2GHz, 4 stage pipeline, ARMv8 ISA in-order core, 7 entry LSQ
NEON Co-processor	128 bit registers 16 parallel 8-bit operations
L1-I Cache	32kB, 4-way, 2 cycle access
L1-D Cache	32/0kB SRAM, 0/128kB STT-MRAM 4-way, 2 cycle access
L2 Cache	1024/0kB SRAM, 0/4096kB STT-MRAM mostly-inclusive, 16-way, 20 cycle access
STT-MRAM Write Time	50ns [11]
Memory	DDR3 2133MHz, 4GB

allocation strategies. Such strategies can be implemented in tandem by excluding the memory region utilized by SHyCache from the data migration scheme. Even a heuristical allocation strategy with oracle prediction abilities would benefit from SHyCache, as, in order to maintain fast access times, the tag array (and data array in the case of simultaneous tag/data access) of both the SRAM and eNVRAM portions of the HC cache must be accessed simultaneously, as the location of the data is unknown prior to access. On the other hand, SHyCache determines the location of the data at compile time, and the address decoding process routes data access to only the portion of cache in which the data is located, reducing power consumption.

V. Experimental Setup

To assess SHyCache's application level performance, we extend the gem5-X architectural simulator [8] [22] to support HC caches. We then simulate three NNs of differing computational complexity and memory footprint, and extract performance, power, and endurance trends across a range of HC geometries.

A. gem5-X Simulator Parameters and Hybrid Cache Access Latency Simulation

We emulate an ARMv8 A53 in-order core by calibrating gem5-X with the simulation parameters illustrated in Table I, and simulating an Ubuntu 18.04 LTS software environment. CPU and interconnect power statistics are extracted via the McPAT power estimation framework [23]. SRAM timing and power values are extracted from an implemented subarray in 28nm using TSMC's high performance technology PDK [24]. We draw eNVRAM power values from literature, considering STT-MRAM [11] for this work, however the allocation strategy is technology independent. In order to illustrate SHyCache's performance and power trends, we extract performance and power statistics across multiple HC hierarchies, in addition to SRAM-only baseline simulations. Hybrid cache geometries are defined by assuming a 4x area ratio between SRAM and eNVRAM bitcell arrays [12], and then sweeping eNVRAM capacity between 0-128kB and 0-4096kB for the L1/L2 caches, respectively, while maintaining an equivalent area footprint.

In order to accurately simulate HC access, SRAM and STT-MRAM access latency is defined in cycles, as documented in Table I. This access latency represents the time to access a cache block through the decoding logic and H-tree, and is pipelined

TABLE II: Neural Network Benchmark Parameters

Benchmark	# Parameters	Weight Memory Footprint (MB)
Inception v4	41.1M	156.8
ResNet-50	23.5M	89.6
SqueezeNet v1.0	1.25M	4.76

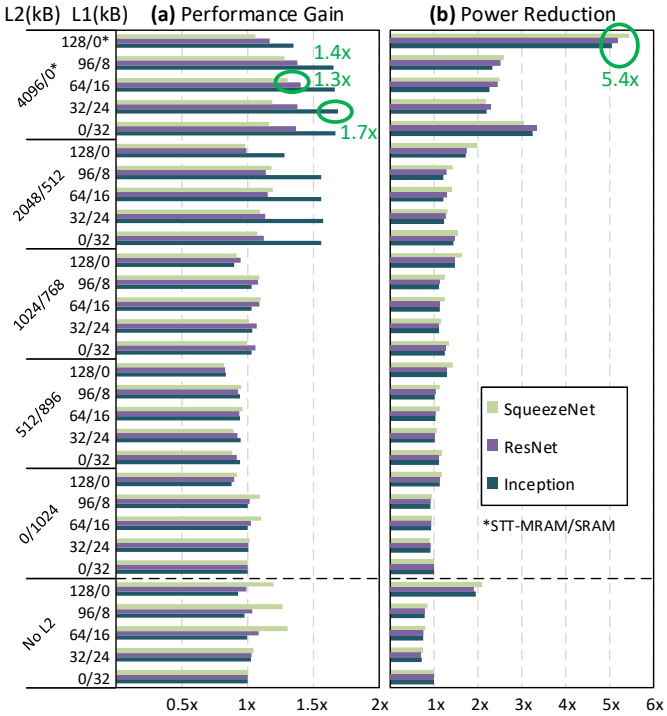

Fig. 3: Performance gain (a) and power reduction (b) across the L1/L2 design space. HC label format is STT-MRAM/SRAM Capacity (kB.)

in this implementation, allowing consecutive cache accesses to overlap without blocking. Additionally, STT-MRAM write latency includes an additional write time measured in *ns*, representing the time taken to write a line of data to STT-MRAM. During this time, the subarrays being written to cannot be accessed; therefore, this time is not pipelined and subsequent accesses to busy subarrays are blocked. As SHyCache is deterministic in that only the SRAM or eNVRAM portions of the memory need to be accessed for any given cache block, this added latency is not present in standard SRAM accesses, and hence does not impact system performance in cases where the eNVRAM is not accessed.

B. Neural Network Benchmarks

In order to benchmark SHyCache, we utilize three modern neural networks of differing sizes, namely, Inception v4 [25], ResNet-50 [26] and SqueezeNet v1.0 [6], whose parameters are outlined in Table II. We choose these networks to benchmark SHyCache under a wide range of network complexities and memory footprints. All weights and inputs are in floating point, and input batch sizes are set to one. We use the ARM Compute Library (ACL) [21] as our software framework. ACL is a graph dataflow framework, specially designed to optimally utilize the ARM NEON SIMD co-processor to accelerate neural networks.

VI. Experimental Results and Analysis

Our experimental results reveal trends in relation to runtime performance, power consumption, and eNVRAM endurance.

A. Performance Results

To observe SHyCache's impact on NN runtime, summarized in Figure 3-a, we perform inference with a batch size of one for the three neural networks, normalizing the results to pure SRAM cache hierarchies. The No-L2 portion is normalized to a pure SRAM cache of 32kB, while all other portions are normalized to a 32/1024kB L1/L2 pure SRAM cache hierarchy.

As can be seen, runtime acceleration varies widely across cache geometries. On one hand, if we consider solely the L1 cache in Figure 3-a, we measure performance gains of up to 1.31/1.09/1.03x for Inception/ResNet/SqueezeNet, respectively, as we increase the STT/SRAM HC ratio up to 64kB/16kB. However, increasing the size of the STT-MRAM array past this point degrades performance, as less SRAM cache space is allocated for the remainder of the application. Generally, a 128kB pure STT-MRAM L1 cache results in a steep decrease in performance as all memory accesses, including those to memory with low read/write ratios, are relegated to STT-MRAM. It should also be noted that performance gain attributable to L1 STT-MRAM decreases as computational complexity and memory footprint increases, as the tiny L1 cache becomes insignificant in comparison to the size of the weights.

On the other hand, if we consider the L2 we find a very different trend. Increasing the HC ratio consistently improves performance for all NN benchmarks, up to a pure eNVRAM array of 4096kB. The larger cache size results in fewer weight evictions, and the mostly-inclusive cache policy mitigates the effects of constant L1 evictions. Additionally, having such a large ratio between L2 and L1 STT-MRAM capacity (64 in the case of a 64kB L1 and 4096kB L2), reduces the negative effects of data repetition that results from an inclusive cache policy. Overall, we achieve maximum possible performance gains of 1.7/1.4/1.3x for Inception/ResNet/SqueezeNet, respectively, when normalized against pure SRAM L1/L2 cache hierarchies.

B. Power Results

Next, we consider SHyCache's implications on power consumption. Figure 3-b summarizes the results of the HC design space, from which two trends can be drawn. First, regardless of the L2 cache, a spike in power reduction is seen at a 128kB pure SRAM cache. This is because in a pure STT-MRAM cache the power-hungry SRAM bitcell array is replaced with a low leakage STT-MRAM bitcell array. A similar, more pronounced power reduction occurs when replacing the L2 SRAM array entirely with STT-MRAM. Figure 4 provides an in-depth breakdown of the power consumption of pure SRAM, pure L1 STT-MRAM, and pure L1/L2 STT-MRAM cache hierarchies. As can be seen, while L1/L2 STT-MRAM write power is substantial, eliminating the energy-leaking SRAM caches provides an excellent reduction in power consumption. STT-MRAM read energy is on par with SRAM read energy, and is too low to be visible in Figure 4. Overall, we see a maximum possible power reduction of 5.1/5.2/5.4x, for Inception/ResNet/SqueezeNet, respectively.

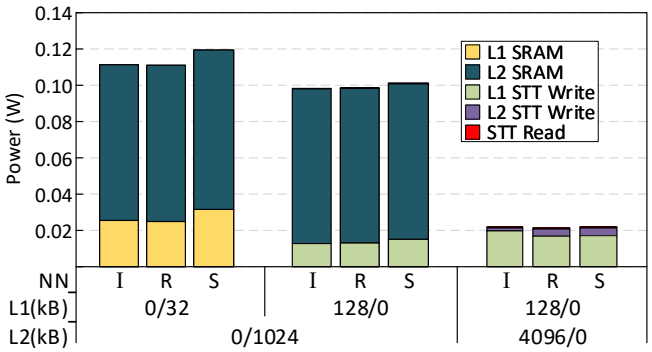

Fig. 4: Power consumption of all-SRAM, SRAM+eNVRAM, and all-eNVRAM caches for Inception (I), ResNet (R) and SqueezeNet (S) NNs.

C. Endurance Results

Lastly, we analyze the number of bitflips that occur within the STT-MRAM array at different cache geometries. eNVRAM life expectancy is tied to its endurance with respect to bitcell value flips, or bitflips. This is measured by counting every $1\rightarrow0/0\rightarrow1$ flip during writes to the STT-MRAM arrays. As eNVRAM technologies have significantly lower endurance compared to CMOS-based memories, it is imperative to consider bitflip frequency of any architecture utilizing eNVRAM.

Figure 5-a illustrates the STT-MRAM bitflip count at all L1 HC geometries with no L2. Consistent with the performance results and reasoning presented in Section VI-A, bitflip count drops for 64 and 96kB STT-MRAM caches, before increasing again for pure STT-MRAM caches, with the bitflip reduction more pronounced in the smaller SqueezeNet NN.

Meanwhile, Figure 5-b presents STT-MRAM bitflip count for all L2 HC geometries with a pure SRAM L1 cache. The first point of note is that the geometry with the highest bitflip count is not a pure STT-MRAM cache, but in fact an HC of 512/896kB. This is consistent with the performance drop seen across all NNs at this geometry in Section VI-A, and is a result of cache thrashing due to the small cache size in relation to the number of weights. Bitflip count then drops as the HC ratio increases and less cache blocks are evicted. Finally, at a pure STT-MRAM cache, the bitflip count for the smaller SqueezeNet NN spikes, as the whole application utilizes STT-MRAM. ResNet and Inception's larger weight footprints dilute this effect, as they gain more from keeping weights in-cache.

In this paper, we consider only overall bitflip count, not flip counts for individual bits. We do observe a drop in average flip per bit as cache capacity increases; however, this metric does not account for uneven intra-word flips skewed toward the LSBs. Many works have explored various eNVRAM wear reducing and leveling optimizations to alleviate this skew. These optimizations are out of this paper's scope, however, and have not been applied in this work; hence, the numbers demonstrated here are worst case values, with room for future optimization.

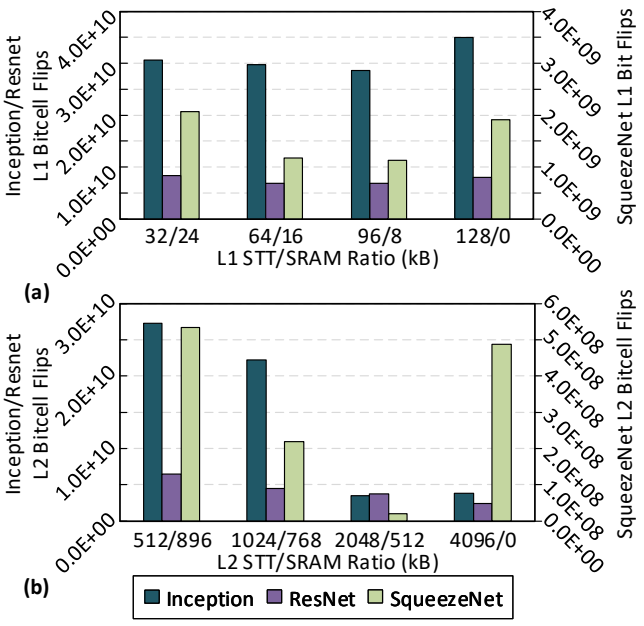

Fig. 5: STT-MRAM bitcell flips across varying L1 (a) and L2 (b) cache sizes.

D. Optimizing HCs for Performance, Power, or Endurance

As seen in Sections VI-A-C, proper selection of HC geometry for the L1 and L2 caches depends on the system's expected use case. Different geometries optimize either performance, power, or endurance. For example, performance is maximized with a 64/16kB L1 HC cache and a pure STT-MRAM 4096kB L2 cache. However, such a configuration may have a poor endurance when small NNs are the target application. In terms of power, a pure STT-MRAM L1 and L2 provides significant power reduction; however, endurance suffers greatly from such a configuration. From an endurance perspective, the highly active L1 cache accounts for nearly half of all bitflips during inference; a good trade-off between performance, power, and endurance, therefore, may be a pure SRAM L1 with a 2048/512kB L2 HC cache. This architecture provides performance and power improvements of 1.6/1.1/1.1x and 1.5/1.5/1.5x, respectively, while incurring the lowest bitflip count of any architecture.

VII. Conclusion

In this work, we presented SHyCache, a hybrid cache with a deterministic allocation strategy and supporting programming model designed to improve NN runtime while reducing power consumption. SHyCache enables NN frameworks to explicitly allocate weight values to the eNVRAM cache, eliminating data transitions between SRAM and eNVRAM arrays and providing maximal cache efficiency. In this work, we explained how SHyCache can be implemented at the system and application level and in tandem with other HC allocation strategies, we have developed a C++ support library allowing implementation in current applications, and we benchmarked SHyCache on three neural network applications of varying computational complexity and memory footprint. Our experimental results have demonstrated a maximum performance gains of 1.7/1.4/1.3x and power consumption reductions of 5.1/5.2/5.4x, for our Inception/ResNet/SqueezeNet benchmarks, respectively. Finally, we have considered the implications of our results for optimizing an architecture based on expected use case, and propose a middle-ground solution that provides optimal trade-off between performance, power, and endurance.

Acknowledgments

This work has been partially supported by EC H2020 RECIPE project (GA No. 801137), EC H2020 WiPLASH project (GA No. 863337), ERC Consolidator Grant COMPUSAPIEN (GA No. 725657), and by the Swiss NSF ML-Edge Project (GA No. 200020_182009).

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, 2017.
[2] S. Ren, K. He *et al.*, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NIPS 28*, 2015.
[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," *ICML*, 2008.
[4] A. Zhou, A. Yao *et al.*, "Incremental network quantization: Towards lossless cnns with low-precision weights," *CoRR*, 2017.
[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *CoRR*, 2015.
[6] F. N. Iandola, M. W. Moskewicz *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *CoRR*, 2016.
[7] S. Bianco, R. Cadene *et al.*, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, 2018.
[8] Y. M. Qureshi, W. A. Simon *et al.*, "Gem5-x: A gem5-based system level simulation framework to optimize many-core platforms," *HPC*, 2019.
[9] G. W. Burr, M. J. Brightsky *et al.*, "Recent progress in phase-change memory technology," *JETCAS*, vol. 6, no. 2, pp. 146–162, June 2016.
[10] R. Fackenthal, M. Kitagawa *et al.*, "19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *ISSCC*, Feb 2014.
[11] Q. Dong, Z. Wang *et al.*, "A 1mb 28nm stt-mram with 2.8ns read access time at 1.2v vdd using single-cap offset-cancelled sense amplifier and in-situ self-write-termination," in *ISSCC*, Feb 2018.
[12] L. Wei, J. G. Alzate *et al.*, "13.3 a 7mb stt-mram in 22ffl finfet technology with 4ns read sensing time at 0.9v using write-verify-write scheme and offset-cancellation sensing technique," in *ISSCC*, Feb 2019.
[13] J. Li, C. J. Xue, and Yinlong Xu, "Stt-ram based energy-efficiency hybrid cache for cmps," in *VLSI-SOC 19*, Oct 2011.
[14] Y. Li, Y. Chen, and A. K. Jones, "A software approach for combating asymmetries of non-volatile memories," in *ISLPED*, 2012.
[15] D. Atienza, J. M. Mendias *et al.*, "Systematic dynamic memory management design methodology for reduced memory footprint," *ACM TODAES*, Apr. 2006.
[16] J. Ahn, S. Yoo, and K. Choi, "Prediction hybrid cache: An energy-efficient stt-ram cache architecture," *TC*, March 2016.
[17] Z. Wang, D. A. Jiménez *et al.*, "Adaptive placement and migration policy for an stt-ram-based hybrid cache," in *HPCA 20*, Feb 2014.
[18] Y.-T. Chen, J. Cong *et al.*, "Static and dynamic co-optimizations for blocks mapping in hybrid caches," in *ISLPED*, 2012, p. 237–242.
[19] Q. Li, J. Li *et al.*, "Compiler-assisted stt-ram-based hybrid cache for energy efficient embedded systems," *TVLSI*, vol. 22, no. 8, Aug 2014.
[20] F. Conti, D. Rossi *et al.*, "Energy-efficient vision on the pulp platform for ultra-low power parallel computing," in *SiPS*, Oct 2014, pp. 1–6.
[21] (2018). [Online]. Available: https://developer.arm.com/technologies/compute-library
[22] (2019). [Online]. Available: https://github.com/esl-epfl/gem5-X
[23] S. L. Xi, H. Jacobson *et al.*, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *HPCA 21*, 02 2015.
[24] W. A. Simon, Y. M. Qureshi *et al.*, "An in-cache computing architecture for edge devices," *TC*, 2020.
[25] C. Szegedy, S. Ioffe *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," *AAAI 31*, 2017.
[26] K. He, X. Zhang *et al.*, "Deep residual learning for image recognition," in *CVPR*, June 2016.