Designing Efficient and High-Performance AI Accelerators With Customized STT-MRAM

Kaniz Mishty and Mehdi Sadi, *Member, IEEE*

***Abstract*— We demonstrate the design of efficient and high-performance artificial intelligence (AI)/deep learning accelerators with customized spin transfer torque (STT)-MRAM (STT-MRAM) and a reconfigurable core. Based on model-driven detailed design space exploration, we present the design methodology of an innovative scratchpad-assisted on-chip STT-MRAM-based buffer system for high-performance accelerators. Using analytically derived expression of memory occupancy time of AI model weights and activation maps, the volatility of STT-MRAM is adjusted with process and temperature variation aware scaling of thermal stability factor to optimize the retention time, energy, read/write latency, and area of STT-MRAM. From the analysis of AI workloads and accelerator implementation in 14-nm technology, we verify the efficacy of our AI accelerator with STT-MRAM (STT-AI). Compared to an SRAM-based implementation, the STT-AI accelerator achieves 75% area and 3% power savings at isoaccuracy. Furthermore, with a relaxed bit error rate and negligible AI accuracy tradeoff, the designed STT-AI Ultra accelerator achieves 75.4% and 3.5% savings in area and power, respectively, over regular SRAM-based accelerators.**

***Index Terms*— Artificial intelligence (AI) accelerator, deep learning, spin transfer torque (STT)-MRAM (STT-MRAM).**

NOMENCLATURE

$N_{\text{in\_ch}}$	Number of input channels.
$N_{\text{out\_chn}}$	Number of input channels.
N_{bat}	Number of images per minibatch.
k_h	Kernel height.
k_w	Kernel width.
P_s	PE internal size.
W_A	Accelerator array width (PEs).
H_A	Accelerator array height (PEs).
W_{SA}	Systolic array width (# of MACs), $P_s * W_A$.
$N_{\text{ofmap\_rw}}$	Number of rows in ofmap.
$N_{\text{ofmap\_cl}}$	Number of columns in ofmap.
$N_{\text{cyc\_per\_stp}}$	Total clk cycles per iteration in conv./systolic mode of accelerator.
n_{fc}	Number of neurons in the input FC layer.
m_{fc}	Number of neurons in the output FC layer.
T_{clk}	Clock cycle time.

Manuscript received April 5, 2021; revised July 6, 2021; accepted August 11, 2021. Date of publication September 1, 2021; date of current version October 6, 2021. This work was supported by the Auburn University IGP grant. *(Corresponding author: Mehdi Sadi.)*

The authors are with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: kzm0114@auburn.edu; mehdi.sadi@auburn.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3105958.

Digital Object Identifier 10.1109/TVLSI.2021.3105958

I. INTRODUCTION

THE demand for deep learning and artificial intelligence (AI) is growing at a rapid pace across a wide range of applications, such as self-driving vehicles, image and voice recognition, medical imaging and diagnosis, finance and banking, and defense operations. Because of these data-driven analytics and AI boom, demands in deep learning and AI will emerge at both data centers and the edge [1]–[4]. In a recent market research [1], it has been reported that AI-related semiconductors will see a growth of about 18% annually over the next few years—five times greater than the rate for non-AI applications. By 2025, AI-related semiconductors could account for almost 20% of all semiconductor demand, which would translate into about $67 billion in revenue [1]. As a result, significant R&D efforts in developing AI accelerators—optimized to achieve much higher throughput in deep learning compared to GPUs—are underway from academia, big techs, and startups [2].

On-chip memory capacity plays a significant role in the performance and energy efficiency of AI tasks [2], [3], [5], [6]. In an AI accelerator, off-chip dynamic random access memory (DRAM) accesses can take 200 times and ten times more energy compared to the local register file and global buffer (GLB) memory, respectively [5]. Larger on-chip buffer memory is needed to minimize DRAM accesses, and it can improve the energy efficiency and speed of the accelerator. However, conventional static random access memory (SRAM)-based solutions suffer from area constraints and leakage power at advanced technology nodes [7], [8], which is a major concern for the energy-constraint IoT domain. Spin transfer torque (STT)-MRAM (STT-MRAM) has the potential to replace SRAM as the GLB in high-performance AI accelerators that require large on-chip memory [3], [9]. As conventional embedded flash storage suffers from scalability and reliability issues at advanced nodes [8], emerging memory-based solutions are required for AI accelerators. As analyzed in detail in [10], because of weight reuse in deep learning, radiation-induced soft errors in the memory block of the accelerator can impact the accuracy of AI models. This is especially a concern for safety-critical applications, such as autonomous vehicles with rigid FIT requirements [10], and STT-MRAM can be a better option for these types of applications.

At scaled technologies (e.g., 10 nm and newer), static energy loss from the high leakage current dominates the overall energy dissipation in DRAM and SRAM technologies [8]. Although Trench cap-based embedded DRAM (eDRAM) has a higher density compared to SRAM, the leakage power and

scaling challenges of eDRAM at advanced process nodes make it less competitive in the future technology roadmap [8]. Beyond the 28-nm node, eFlash faces scaling challenges, and eMRAM technology becomes superior over eFlash because of its lower write voltage and energy, higher endurance, lower area, and faster read/write time [11]. The emerging resistive RAM (RRAM) and phase change (PCM)-based cross-point memory suffer from endurance, reliability, and variability problems [8], [12]. Among all the emerging embedded memory technologies, STT-MRAM is one of the most promising due to its high energy efficiency, write endurance (e.g., more than 1 million cycles), high cell density, high-temperature data retention capability, operating voltage comparable with CMOS logic, and immunity to soft errors [7], [8], [13]–[17]. Moreover, STT-MRAM is highly compatible with CMOS and requires only two to six extra masks in the backend-of-the-line (BEOL) process [7], [14]. Because of the leakage power issue, beyond a certain memory size, embedded MRAM becomes more energy efficient compared to SRAM [7].

While performing deep learning/AI tasks, the throughput of the AI accelerator primarily depends on: 1) the number of processing elements (PEs) and 2) the size of on-chip buffer memory [2], [3]. As a result, area efficiency is of paramount importance for AI accelerators, and the critical design goal is to increase PE density and on-chip memory capacity. Because of compact size ($6F^2$ of STT-MRAM versus $100F^2$ of SRAM [18], [19]), STT-MRAM has the potential to outperform conventional SRAM as the on-chip memory in accelerators. At isomemory capacities, the MRAM module occupies a much lower area compared to SRAM [7]. In addition, for power constraint mobile/edge/IoT applications, STT-MRAM-based AI accelerators can significantly minimize static power compared to SRAMs. However, the higher write energy and write latency of conventional eMRAM can be a deterrent in their full adoption in AI accelerators. In this article, we present a methodology to design efficient AI accelerators with customized STT-MRAM that can provide high bit-cell density while still ensuring fast write speed and decreased write energy. We achieve this feat by analyzing the volatility requirement of weight and input/output feature-map (ifmap/ofmap) data on-chip and scaling the eMRAM's retention time accordingly without incurring unacceptable bit error rates (BERs). In contrast to crossbar-based in-memory inference hardware, in this article, our objective is to analyze the memory constraints and energy efficiency problem of regular AI accelerator hardware.

To the best of our knowledge, this is the first extensive work on designing AI/deep learning accelerators with STT-MRAM-based on-chip memory systems. The key contributions and highlights of this article are given as follows.

1) We present an innovative runtime reconfigurable core design that can be optimized for both dot products of convolution layers and matrix multiplications of fully connected (FC) layers.
2) We derive the analytical expressions of occupancy times of weights and input/output feature maps in the global memory of the AI accelerator between different stages (i.e., Conv. layer followed by Conv., Conv. layer followed by FC layer, and FC-FC) of AI/deep learning operation. Guided by this data activity duration, we scale the retention time of STT-MRAM and customize the design for application as the GLB memory in an energy-efficient AI accelerator. We consider process variations and runtime temperature fluctuations in this scaling procedure to ensure negligible read/write BERs and retention failures (RFs) across all corners.
3) Based on detailed design space exploration using state-of-the-art AI/deep learning models, an AI accelerator system and MRAM technology codesign framework are presented with the key innovations.
    a) It optimizes STT GLB size to minimize DRAM accesses.
    b) A novel scratchpad-assisted STT-MRAM-based GLB architecture is presented to minimize the writes to the MRAM by bypassing writes of the partial ofmaps to the scratchpad.
    c) For inference-only tasks, to store the trained weights, a specially customized embedded STT-MRAM—as a flash replacement—with optimized retention time (e.g., three to four years) and robust BER is used.
4) To further improve the energy and area efficiency, we exploited the inherent error tolerance of deep learning/AI models and created two STT-MRAM banks for the GLB. For the first bank, the thermal stability factor is scaled further to a relaxed BER, and the less critical half of the weights/fmap bits (e.g., LSB groups) is assigned to this memory block. The second bank has scaled retention time with a robust BER, and the other remaining half of the bits (e.g., MSB groups) is stored in this bank.

The rest of this article is organized as follows. The background is discussed in Section II. STT-MARM-based optimum AI/deep learning accelerator design methodology is presented in Section III. The AI accelerator-aware eMRAM technology codesign methodology is presented in Section IV. We present simulation results in Section V, related work in Section VI, and conclusions in Section VII, respectively.

II. Background

A. Deep Neural Networks

At the core of deep learning/AI is the deep neural network (DNN). Modern state-of-the-art DNN consists of stacks of convolution layers to extract the objects' features and a few FC layers at the end to classify them. In convolution, kernels convolve over input feature maps (ifmap) to extract embedded features and generate the output feature maps (ofmap) by accumulating the partial sums (psums), as shown in Fig. 1. Each fmap and filter is a 3-D structure consisting of multiple 2-D planes, and a batch of 3-D fmaps is processed by a group of 3-D filters in a layer. Activation functions (e.g., ReLU) operate on the results before they go to the maxpooling layer. The computations of a convolutional layer

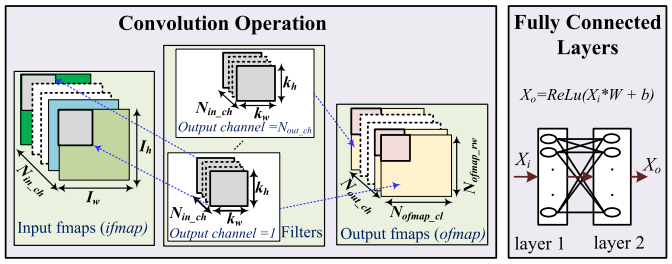

Fig. 1. Convolution and FC layer operations.

can be expressed as

$$\begin{aligned}\text{Ofmap}[z][u][x][y] = \text{ReLU} \\ \times \left(B[u] + \sum_{v=1}^{N_{\text{in\_ch}}} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} I[z][v][Sx+i] \right. \\ \left. \times [Sy+j] \times F[u][v][i][j] \right) \\ 0 \le z < N, \quad 0 \le u < N_{\text{out\_ch}} \\ 0 \le y < N_{\text{ofmap\_rw}}, \quad 0 \le x < N_{\text{ofmap\_cl}} \\ N_{\text{ofmap\_rw}} = ((I_h - k_h + 2P)/S) + 1 \\ N_{\text{ofmap\_cl}} = ((I_w - k_w + 2P)/S) + 1 \end{aligned} \tag{1}$$

where Ofmap, N, F, I, P, S, and B represent output feature maps, the number of inputs in a Batch, filter weights, input feature maps, padding, stride size, and bias, respectively [5].

Unlike the convolution layers, each neuron of the FC layer is generally connected to every other neuron of its previous/next layer with a specific weight (0 for no connection). The computations of FC layers are matrix/vector–matrix multiplications, where the output activation (X_o) of a layer is obtained by multiplying the input activation (X_i) matrix/vector with the weight matrix (W) followed by the addition of a bias term and, finally, passing the result through a nonlinear function, $X_o = \text{ReLu}(X_i * W + b)$.

B. Deep Learning/AI Hardware Accelerators

SIMD, or systolic array-based hardware, is the present state-of-the-art hardware to accelerate AI operations [2], [3]. The systolic array is only optimized for matrix–matrix multiplication. Mapping the convolution dot products into the matrix multiplications by converting the activation maps into the Toeplitz matrix and the kernel weights into a row vector is a popular solution to address this problem. Nonetheless, it involves redundant data in the input feature map, which gives rise to inefficient memory storage and complex memory access pattern [2]. More recently, heterogeneous architectures are evolving, which have optimized cores for convolution and FC layers [6]. While this solves the complications regarding Toeplitz matrix conversion, it incurs area overhead. Because when convolution core is active, FC core remains idle, wasting circuit area. In response to the existing issues, in this article, we propose a novel concept of a reconfigurable core capable of efficiently performing both convolution dot products and matrix multiplications based on the operation-dependent (i.e., convolution or FC) control signal.

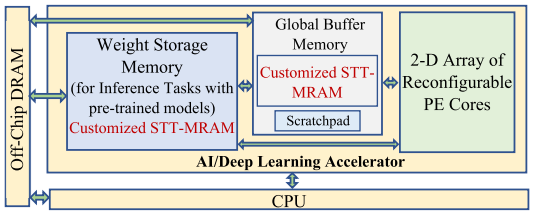

Fig. 2. AI accelerator with reconfigurable cores optimized for both Conv. and FC layers, and STT-MRAM based on-chip memory.

C. Memory System in AI/Deep Learning Hardware

The memory system is one of the vital metrics in determining the performance of AI hardware. Each off-chip DRAM access is 100–200 times more energy costly than any ALU operation or a local memory (e.g., register file/scratchpad) access [2]. Energy-constraint AI hardware leverages a memory hierarchy of register files, GLB, and DRAM. Moreover, a significant amount of memory is required to store the pretrained weights for inference-only applications. The larger the GLB memory, the more energy-efficient the AI hardware is due to lower DRAM access. Most of the existing DNN hardware uses SRAM both as GLB and register file and eFlash as weight storage [2]. Because of the large size of SRAM and static energy loss due to high leakage at scaled nodes (e.g., 10 nm and newer), the GLB size cannot be increased beyond a certain threshold energy-efficiently. eFlash starts to suffer from scaling challenges even at earlier technology, such as 28 nm [11]. The benefits of our proposed Δ-customized STT-MRAM as a replacement of both the SRAM-based GLB and the eFlash-based weight storage memory are many-fold: higher memory capacity, lower read-write latency and energy, and higher endurance against soft errors.

III. Efficient AI/Deep Learning Hardware

Fig. 2 depicts the top-level architecture of the accelerator containing the proposed reconfigurable core and MRAM-based memory system. The following sub-sections describe the dataflow in convolution and systolic mode and formulate the memory occupancy time in each mode.

A. Reconfigurable Core

The architecture and workflow of our proposed reconfigurable core are quite simple but powerful enough to support both matrix multiplication and convolution dot product at run-time configuration. The reconfigurable core consists of three MAC modules and four multiplexers. Each MAC contains a BFloat16 multiplier and an FP32 adder [20], [21] to accommodate both training and inference. If only inference is desired, the hardware can be 8-bit int8 type [2], [3]. The multipliers take input feature maps and filter weights as inputs and pass the results to their neighboring adders to be added with the previous partial sum results. The multiplexers act as mode selectors of the core. When Mode is deasserted, the MACs are disconnected from each other, and their outputs are collected downward to reflect the systolic array architecture [see Fig. 3(b)]. On the other hand, when Mode is asserted, three MACs collectively act as a convolution block that

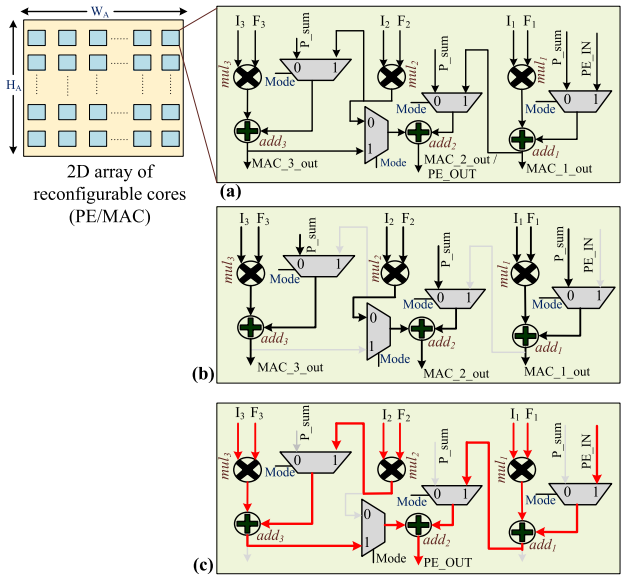

Fig. 3. (a) Reconfigurable core. (b) Reconfigurable core acting as building block of systolic array when mode is low. (c) Reconfigurable core acting as convolution PE when mode is high.

performs three dot products parallelly and produces one partial sum [see Fig. 3(c)]. In this case, adder_3 adds the outputs of multiplier_3 and multiplier_2 to produce the intermediate sum. Meanwhile, adder_1 adds the multipler_1 output with the previous partial sum. These operations occur concurrently, provided that the input activations and filter weights are assigned to the multipliers parallelly. Once the outputs from adder_3 and adder_1 are ready, adder_2 sums them up to produce the PE_OUT. The building block containing three MACs and four Muxes is defined as a PE block for convolution in this work. Fig. 3 illustrates the functionality of the reconfigurable core in systolic array mode (b) and convolution mode (c). Since most state-of-the-art DNN architecture has a kernel size of 3, we chose the number of blocks as 3 in the reconfigurable core. However, designers can adjust this parameter to achieve the highest possible computation/utilization efficiency of the reconfigurable core as needed based on use cases.

B. STT-MRAM Based On-Chip Memory System

The prime criteria of memory to sustain as an on-chip memory are high density, low read/write latency, and energy. Conventional STT-MRAM suffers from high read/write energy and latency. However, in the case of on-chip memory/GLB, the intrinsic nonvolatility property of STT-MARM can be compromised to minimize the read/write energy and latency by adjusting the thermal stability factor (Δ). Considering the data retention time in the GLB, Δ can be scaled down to achieve a significant reduction in read/write energy, latency, and increase in cell density. This section will formulate necessary expressions to calculate the data retention time in the GLB for the most time-consuming AI operations, such as the convolution layer and FC layer operations. The derived expressions will help us to precisely determine the maximum data retention time in GLB and, thus, help to scale down Δ.

Deep learning/AI operations are layerwise sequential operations (i.e., the current layer's output acts as input to the following layer). To formulate the data retention time between two consecutive layers, in the inference mode, we define T_1 as the time required by the accelerator to generate the ofmap of one layer. Once the ofmap of a layer is generated, it goes through pooling and activation functions to serve as the input to the following layer. We refer $T_{\text{pool\_relu}}$ as the time required to perform the maxpooling and ReLU operations. The time required to generate the ofmap of the following layer is termed T_2. Finally, T_{ret} is the data retention time in memory between two consecutive convolution (or FC) layers.

1) Retention Time for Conv–Conv Layers: In the convolution mode, each PE block of the array performs the dot product between the input feature maps (ifmaps) and weights. Each unit PE block's size is defined as P_s, where P_s represents the number of elements that the MAC module can process. The ifmaps and kernel weights are loaded into the PE array from GLB memory, and the PE array computations occur in parallel. Without loss of generality, in our analysis, we adopted the row stationary dataflow where kernel rows are loaded into the PE blocks and kept stationary, and ifmaps are shifted according to the stride size [2], [5]. The partial sums are accumulated vertically to generate the ofmaps. This process is repeated until a complete ofmap is generated. Setting Mode = 1 in the Muxes of the PE blocks (Fig. 3) ensures that the reconfigurable core is acting as a convolution core.

To calculate T_1, we formulate an expression to estimate the time required to generate the output of a convolution layer. We assume that the operations related to the next output channel will be assigned in the accelerator array only after all the MAC operations related to the previous output channel have been completed. In other words, in an iteration of the accelerator array, the input channels present in it are all related to the same output channel. In addition to simplifying the PE scheduling procedure, this assumption also aligns with our goal of obtaining a convolution layer's worst case completion time.

For layer $n-1$, a single row of a partial ofmap (i.e., ofmap corresponding to one kernel and one input channel) will require $(k_h * \lceil k_w/P_s \rceil)$ PE blocks (symbol meanings are given in the Nomenclature), implying that a partial ofmap for a single input channel will require, $N_{\text{ofmp\_rw}} * k_h * \lceil k_w/P_s \rceil$ PEs. (The $\lceil\rceil$ symbol means ceil operation where the result is rounded to the nearest larger integer.) An example of convolution operation inside the core in Conv. mode is shown in Fig. 4.

Next, we find out how many partial ofmaps (i.e., how many input channels) can be fit in the full accelerator array in one step. This number is obtained by dividing the total available PEs in the accelerator array, $W_A * H_A$, by the number of PEs required for one input channel. The total number of required steps (i.e., number of times the complete accelerator array will be used) for all input channels ($N_{\text{in\_ch}}$) for one 3-D filter (i.e., one output channel), $N_{\text{steps\_per\_out\_ch}}$, can be expressed as

$$N_{\text{steps\_per\_out\_ch}} = \left\lceil \frac{N_{\text{in\_ch}} * k_h * N_{\text{ofmp\_rw}} * \left\lceil \frac{k_w}{P_s} \right\rceil}{W_A * H_A} \right\rceil. \quad (2)$$

The details of the symbols used in (2)–(6) can be found in the Nomenclature. Time for each of the above

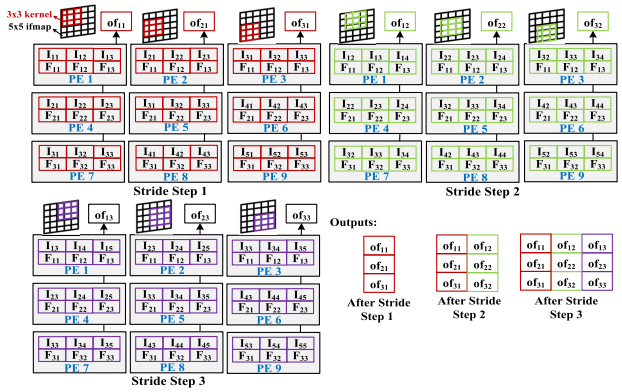

Fig. 4. 3×3 kernel ($k_h = k_w = 3$) is convolved (with stride $= 1$) over a 5×5 ifmap to produce 3×3 ofmap ($N_{\text{ofmp\_rw}} = N_{\text{ofmp\_cl}} = 3$). The size of unit PE block is $P_s = 3$. Total nine PE blocks are required.

steps is given as

$$t_{\text{per\_step}} = T_{\text{clk}} \ * \ N_{\text{cyc\_per\_stp}} \ * \ N_{\text{ofmp\_cl}} \ * \ N_{\text{bat}}. \tag{3}$$

$N_{\text{cyc\_per\_stp}}$ refers to the total clock cycles required in the accelerator, for one image of the batch, to perform: 1) dot products between the kernel and ifmap elements; 2) partial sum accumulation of the dot products; and 3) partial sum of ofmap of previous input channel with current channel. This term depends on the circuit-level implementation of accelerator hardware. The term $N_{\text{ofmap\_cl}}$ appears in (3) because the kernel needs to be shifted (i.e., according to the stride parameter of convolution) this many times to generate the partial ofmap for each input channel. N_{bat} appears in the equation since each image from the minibatch will be serially processed [5]. Between each input channel operations, the partial sum of ofmaps of the input channels will be stored in the scratchpad to be accumulated to the next input channel's partial ofmaps to finally create the full ofmap output for that particular output channel and filter. The total time required to generate each output channel/ofmap, $t_{\text{per\_out\_ch}}$, is given by

$$t_{\text{per\_out\_ch}} = N_{\text{steps\_per\_out\_ch}} \ * \ t_{\text{per\_step}}. \tag{4}$$

If there are a total of $N_{\text{out\_chn}}$ output channels, then the total time required to generate the full ofmap (i.e., for all output channels) is $T_1 = t_{\text{per\_out\_ch}} * N_{\text{out\_chn}}$. Using (2)–(4), the T_1 term can be expressed with the following equation. All parameters in (5) are for Conv. layer $n-1$

$$T_1 = \left\lceil \frac{N_{\text{in\_ch}} \ * \ k_h \ * \ N_{\text{ofmp\_rw}} * \left\lceil \frac{k_w}{P_s} \right\rceil}{W_A \ * \ H_A} \right\rceil * \ T_{\text{clk}} * N_{\text{cyc\_per\_stp}} \ * \ N_{\text{ofmp\_cl}} \ * \ N_{\text{bat}} * N_{\text{out\_chn}}. \tag{5}$$

The ofmap of layer $n-1$ will act as ifmap to next layer n after passing through the ReLU and MaxPool layers. The ifmap_n should be retained in the memory until layer n has finished reading it to generate its output ofmap_n. A closer look into the situation will reveal that the input data read time for layer n is related to the ofmap_n generation time. This implies that the ifmap_n data need to be in the memory for a maximum duration of time that is equal to the time required for complete ofmap_n generation. Considering the above facts, we first calculate ofmap_n generation time using the similar

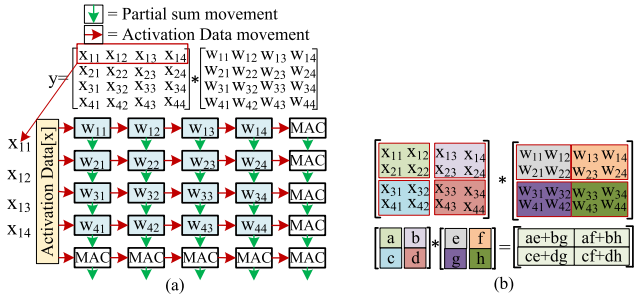

Fig. 5. (a) Dataflow inside systolic array and (b) larger matrices can be divided into smaller submatrices to fit in the systolic array. An example of dividing two 4×4 matrix into four 2×2 matrices.

methods of (2)–(5) and then assign it as T_2. All parameters in (6) are for Conv. layer n

$$T_2 = \left\lceil \frac{N_{\text{in\_ch}} \ * \ k_h \ * \ N_{\text{ofmp\_rw}} * \left\lceil \frac{k_w}{P_s} \right\rceil}{W_A \ * \ H_A} \right\rceil * \ T_{\text{clk}} * \ N_{\text{cyc\_per\_stp}} * N_{\text{ofmp\_cl}} \ * \ N_{\text{bat}} * N_{\text{out\_chn}}. \tag{6}$$

ReLU and MaxPool layers take relatively much shorter time compared to Conv. layers do. Therefore, we can directly estimate $T_{\text{pool\_relu}}$ from hardware implementation of ReLU and MaxPool layers. Combining T_1, T_2, and $T_{\text{pool\_relu}}$, we can estimate the required data retention time, T_{ret}, in memory between two consecutive Conv. layers in inference phase

$$T_{\text{ret\_conv-conv}} = T_1 + T_{\text{pool\_relu}} + T_2. \tag{7}$$

2) Retention Time for FC-FC Layers: To perform the computations associated with FC layers, the reconfigurable core transforms into the systolic array. This is achieved by disabling the mode signal of the Muxes present in the PE blocks. In this section, we formulate an expression to estimate the time required to compute the output of an FC layer. The systolic array shown in Fig. 3(b) has $H_A \times W_{\text{SA}}$ MAC modules. Because of the reconfigurable feature of the core, $W_{\text{SA}} = P_s * W_A$. The weights are loaded into the array according to the capacity of the systolic array implying that the number of weights can be loaded into the array in one step is equal to the number of MACs present in the array, $N_{\text{wt\_per\_step}} = H_A * W_{\text{SA}}$. If the total number of weights is greater than the number of weights, the array can accommodate in one step, i.e., $N_{\text{tot\_wt}} > N_{\text{wt\_per\_step}}$, using the concept of *divide & conquer* in matrix multiplication [see Fig. 5(b)], we find out how many steps (i.e., number of times we need to load new weights to the accelerator array) are required to complete the computation with all elements of the weight matrix. The number of steps required to complete the computation with all weights is $\lceil m_{\text{fc}}/H_A \rceil \ * \ \lceil n_{\text{fc}}/W_{\text{SA}} \rceil$. (For symbol meanings, see the Nomenclature.) In every step, the array is loaded with $N_{\text{wt\_per\_step}}$ weights, inputs are streamed from left to right, and the partial sums move downward to be collected in accumulators [3]. The clock cycles required to complete each step are $N_{\text{cyc\_per\_stp}}$ that depends on the circuit-level implementation of systolic array hardware and the dimension of systolic array core. Combining the above terms, and considering there are N_{bat} images in the minibatch, the time required to generate the

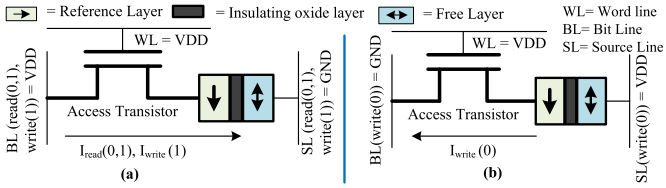

Fig. 6. Bit-cell of STT-MRAM. (a) Reading from it and writing 1. (b) Writing 0.

output of FC layer $(n-1)$, T_1, is expressed in the following equation:

$$T_1 = \left\lceil \frac{m_{\text{fc}}}{H_A} \right\rceil * \left\lceil \frac{n_{\text{fc}}}{W_{\text{SA}}} \right\rceil * T_{\text{clk}} * N_{\text{cyc\_per\_stp}} * N_{\text{bat}} \quad (8)$$

where all parameters are for FC layer $(n-1)$.

FC layer n will consider the output of previous FC layer $(n-1)$ as its input. The output of FC_{n-1} should be stored in the memory until FC_n has completed reading it for generating its output. With this reasoning, we calculate the output generation time for FC_n following the above method and assign it as T_2. All parameters of (9) are for FC layer n

$$T_2 = \left\lceil \frac{m_{\text{fc}}}{H_A} \right\rceil * \left\lceil \frac{n_{\text{fc}}}{W_{\text{SA}}} \right\rceil * T_{\text{clk}} * N_{\text{cyc\_per\_stp}} * N_{\text{bat}}. \quad (9)$$

Two consecutive FC layers do not have maxpooling layer in between. Therefore, we can find the data retention time, $T_{\text{ret\_fc-fc}}$, for an FC layer followed by another FC as

$$T_{\text{ret\_fc-fc}} = T_1 + T_2. \quad (10)$$

C. Retention Time of Convolution Layer Followed by FC Layer

The retention time between a convolution layer followed by an FC layer is also expressed as

$$T_{\text{ret\_conv-fc}} = T_1 + T_{\text{pool\_relu}} + T_2 \quad (11)$$

where T_1 is the time required to generate the Conv. layer ofmap and T_2 is the time required to generate the FC layer output.

Using the above expressions of weight, ifmap, and ofmap occupancy times in GLB memory, for a particular accelerator hardware architecture and the operating clock frequency, we can estimate the maximum retention time required for STT-MRAM-based GLBs. The MRAM write and read times will be added with the above retention time expressions. As the MRAM read/write times are orders of magnitude lower (i.e., less than 10 ns) compared to the retention times T_1 and T_2 that are in the ms or s range, as explained in Section IV, we did not explicitly add the MRAM read/write time with the above retention time (T_{ret}) expressions.

IV. Optimizing STT-MRAM for AI Accelerators

A bit-cell of STT-MRAM consists of a magnetic tunnel junction (MTJ) for storing the bit and an access transistor to read/write the bit. The MTJ contains two ferromagnetic layers: 1) fixed magnetic orientation and 2) free layer whose orientation can be switched externally by an applied current. The orientation of the free layer relative to the reference layer represents the state of the stored bit. Fig. 6 depicts the cell schematic, read, and write operations.

A. Critical Design and Performance Parameters of MTJ

1) Thermal Stability Factor and Critical Current: The energy barrier E_b, in which the free layer magnetization must overcome to switch its stable state, is defined as the thermal stability factor (Δ) and is expressed as [22], [23]

$$\Delta = \frac{E_b}{k_B T} = \frac{H_K M_S V}{2 k_B T} \quad (12)$$

where E_b is the energy barrier of free layer, k_B is Boltzmann's constant, T is the temperature, H_K is the anisotropy field, M_S is the saturation magnetization, and V is the volume of MTJ.

The critical current, I_c, is defined as the minimum current required to flip the state of the free layer [19], [22], [23]. The critical switching current is modeled as [22], [23]

$$I_c = \left(\frac{4 e k_B T}{h}\right) * \frac{\alpha}{\eta} * \Delta * \left(1 + \frac{4\pi M_{\text{eff}}}{2 H_K}\right) \quad (13)$$

where e is the electron charge, k_B is Boltzmann's constant, T is the temperature, h is Plank's constant, α is the LLGE damping constant, η is the STT-MRAM efficiency parameter, $4\pi M_{\text{eff}}$ is the effective demagnetization field, and H_K is the anisotropy field.

2) Retention Time and Retention Failure: Once data are written, MTJ should retain its state, even if the power source is removed, until any external force is applied to flip the state. However, due to thermal noise, the logic state might get flipped unintentionally. The maximum time MTJ can retain its nonvolatility is known as the data retention time. The RF probability for a given time period t_{ret} is [22], [23]

$$P_{\text{RF}} = 1 - \exp\left[-\frac{t_{\text{ret}}}{\tau * \exp(\Delta)}\right] \quad (14)$$

where t_{ret} is the retention time and τ is the technology constant.

3) Read Disturbance (RD): To read a bit from STT-MRAM, read current I_r, much less than the critical current I_c, is flown from bitline through the access transistor and MTJ. Fig. 6 shows that writing 1 and reading (both 0 & 1) share the same current path. This can sometimes cause the unintentional switching of the bit-cell content resulting in RD. For read current I_r and read latency t_r, the probability of RD can be modeled as [22], [23]

$$P_{\text{RD}} = 1 - \exp\left[-\frac{t_r}{\tau * \exp\left(\Delta\left(1 - \frac{I_r}{I_c}\right)\right)}\right]. \quad (15)$$

4) Write Error Rate (WER): Writing a bit-cell requires a write current I_w, larger than I_c, to be flown between BL to SL, as shown in Fig. 6. Because of the stochastic nature of the write operation, the switching time of MTJ varies from access to access [19], [22], [23]. If the write current is terminated before the free layer has successfully changed its state, the write operation can be erroneous. For write pulsewidth t_w, the WER is [22], [23]

$$\text{WER}_{\text{bit}} = 1 - \exp\left[\frac{-\pi^2 \Delta\left(\frac{I_w}{I_c} - 1\right)}{4\left[\frac{I_w}{I_c} * \exp\left\{\frac{t_w}{\tau}\left(\frac{I_w}{I_c} - 1\right)\right\} - 1\right]}\right]. \quad (16)$$

B. Customizing STT-MRAM for AI Accelerators

1) Scaling Thermal Stability Factor: To achieve typical retention period of ten years, the thermal stability factor, $\Delta \geq 60$, is required [11], [14], [19], [22], [23]. However, such a long retention time may be unnecessary depending on the application. For example, if MRAM is used as the GLB memory in AI accelerators, then the retention time can be significantly scaled depending on the weight and input/output feature-map data occupancy time (e.g., ms to s ranges) in that memory. If STT-MRAM is used as an eFlash replacement for pretrained weight storage for AI inference tasks, then three to five years retention might be enough instead of ten years. From (12), it is seen that, by adjusting the volume (i.e., area and/or thickness) of the MTJ, the thermal stability factor (Δ) can be scaled. In other words, considering the target operating temperature range of the AI accelerator and the expected lifetime of the data, scaling down of thermal stability factor will improve area efficiency by increasing the memory bit-cell density. Moreover, with scaled Δ and bit-cell area, the cell would require a lower operating current, thus saving energy.

2) Optimizing Read/Write Latency and Energy at Target WER and RD: Recent state-of-the-art STT-MRAMs can compete or outperform SRAMs in all aspects except write energy and write latency [7], [14], [15]. However, for AI accelerator applications, by scaling Δ and the retention time of STT-MRAM, we can circumvent the write energy and latency limitations. Equation (16) implies that write latency is $t_{\text{pw}} \propto \ln(\Delta)$ at constant WER. We can exploit this relationship to reduce the write latency with a scaling down of Δ. From (14), we infer that retention time t_{ret} is exponentially proportional to Δ. Thus, depending on the desired retention period of STT-MRAM in the AI accelerator, we can optimally scale down Δ and also minimize write latency at that target retention time. However, (16) also implies an inverse relationship between write latency and WER, which hinders us from aggressively scaling down write latency at the desired Δ. Fortunately, to boost the writing speed at the scaled Δ, we can keep I_w higher (e.g., close to the prescaled value), and this can assist in designing an STT-MRAM with high write-speed [19]. Recently, high-speed write has been experimentally demonstrated in [17] by optimizing the free layer materials. We can identify the optimum Δ and I_w that minimize write latency and write energy while still satisfying the WER and retention time requirements for the AI accelerator. As depicted in (13), with scaling of Δ the critical current I_c decreases linearly, and hence, the read current I_r also decreases. At this scaled Δ and I_r, the read latency can also be scaled by adjusting the sense amplifier reference voltage [7], [19]. Equation (15) implies that, at scaled Δ, the shortened read pulse duration will also ensure that the read disturb rate is within the acceptable target.

C. Addressing Process and Temperature Variation

Recent experimental results show that, by adjusting MTJ free-layer material properties [16] or by dimension scaling [6], the retention time and write speed can be adjusted. However, the performance of MRAM can degrade due to the process and temperature variations [7], [14], [24]. Process-induced

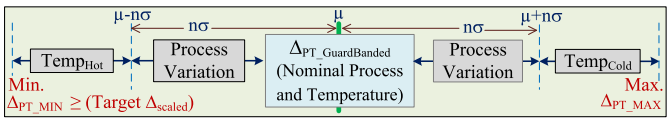

Fig. 7. Impact of process and temperature variation on thermal stability factor (Δ).

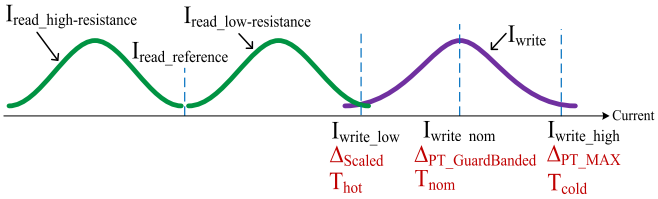

Fig. 8. Distribution of read/write currents with PT variation. Worst case occurs when worst process corners experience T_{hot} or T_{cold}.

variations in free layer thickness in MTJ and in access transistor channel length/width and threshold voltage contribute to the performance variations in MRAM. From silicon measurement data in [7], the standard deviation (σ) of MTJ diameter variation was reported to be 2.1% of the mean. To ensure that the adjusted Δ accounts for process variations, we ensure that enough guard band (e.g., 4σ) is added, as shown in (17) and (18).

The bit-cells are placed compactly on the layout; as a result, the bit-cell to bit-cell variations within the same die/chip are minimal, and the process variation is dominated by the chip-to-chip variations. Δ increases with an increase in MTJ diameter and H_K due to process variation and a decrease in temperature from the nominal value [see Fig. 7 and (12)]. An increase in Δ increases the critical current (I_c), which eventually increases the write current (I_w) [see (13) and (16)]. Given the smaller write pulse, write failure occurs when supplied I_w is less than the required I_w. Worst case occurs when both: 1) the supplied I_w decreases due to the access transistor being in the slow process corner and 2) the required I_w increases due to an increase in Δ resulting from Process and runtime Temperature (PT) variations. On the other hand, a decrease in Δ beyond a minimum due to PT variation will result in RF [see (14)].

To protect the desired Δ_{scaled} against the worst case PT variation, appropriate guard band needs to be added. The $\Delta_{PT\_\text{GuardBanded}}$ is chosen to cover both the worst case 4σ range (i.e., 99.993% of the samples) of process variation and high temperature operating scenario, as shown in (17)

$$\Delta_{\text{scaled}} \leq (\Delta_{\text{PT\_GuardBanded}} - 4\sigma) * (T_{\text{nom}}/T_{\text{hot}}) \quad (17)$$

$$\Delta_{\text{PT\_MAX}} = (\Delta_{\text{PT\_GuardBanded}} + 4\sigma) * (T_{\text{nom}}/T_{\text{cold}}). \quad (18)$$

The chips located on the right-hand side of the process variation distribution (i.e., $\mu + n * \sigma$, where $n \geq 1$) of $\Delta_{\text{PT\_GuardBanded}}$ will experience larger Δ, as shown in Figs. 7 and 8. In addition, at cold temperatures, Δ will further increase to $\Delta_{\text{PT\_MAX}}$, as shown in (18). Although the higher $\Delta_{\text{PT\_MAX}} > \Delta_{\text{scaled}}$ will be benign for retention time, the required write current will increase in this scenario to confine the write time and WER of these samples within the nominal bound. Designing the write driver for this worst case scenario will dissipate unnecessary power for all other nonworst case samples. To address this, we propose a

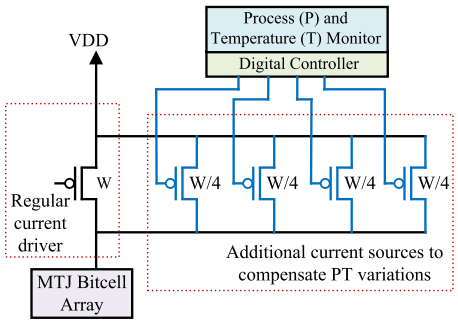

Fig. 9. Modified write driver.

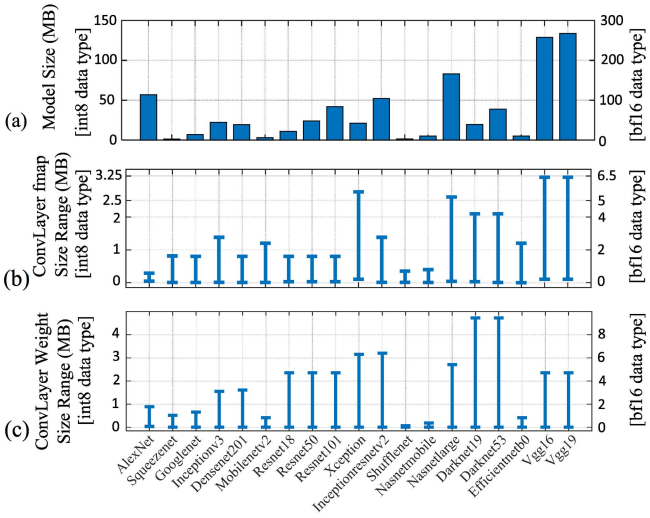

Fig. 10. (a) Complete sizes of widely used AI models. (b) Activation map (ofmap/ifmap) sizes. (c) Weight sizes for Conv. layers.

dynamically adjustable write driver depicted in Fig. 9. In addition to adjusting the write current according to the process variation profile of the MRAM chip/die, a PT monitor senses the runtime temperature and adjusts the current dynamically by turning on/off the pMOS transistors (see Fig. 9). For example, at the nominal process and temperature, the extra transistors can be OFF; however, at PT-induced rise in Δ, the transistors can be individually activated to ensure successful write.

In summary, we design the MTJ with higher $\Delta_{\text{PT\_GuardBanded}}$ than the desired Δ_{scaled} to accommodate potential degradation in thermal stability from worst case 4σ process variation and runtime high temperature and proposed a write-driver with the controllable current drive to address the high write-current demand at cold temperature and slow process corner.

D. MRAM Write Energy Optimization With ScratchPad

Since the write energy of STT-MRAM is higher than the read energy, we propose an innovative scratchpad-assisted MRAM GLB architecture to minimize the write frequency in STT-MRAM and, thus, further optimize the energy. Reduced write-frequency is achieved by using a small global SRAM scratchpad, typically in the kB range (details in Section V), in addition to a large (i.e., MB range) global STT-MRAM buffer. When the accelerator PE array generates partial ofmaps (i.e., ofmap corresponding to each input channel), they need to be stored somewhere in memory to be added to the next partial ofmap to produce the complete ofmap of an output channel.

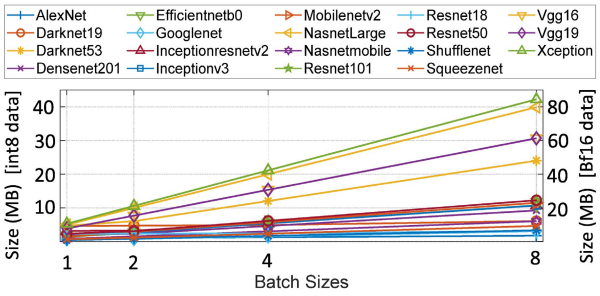

Fig. 11. Required capacity of GLB with varying batch sizes to avoid DRAM access during inference.

The reason behind these partial-ofmap writes is that the accelerator might not produce the complete ofmap in one step. Between the subsequent steps, the partial ofmap result from the previous step needs to be written in the memory to be subsequently read and accumulated with the partial ofmap result from the following step. Adding this small SRAM scratchpad memory (for intermediate ofmap writes) with MRAM GLB further improves the energy efficiency. In summary, our proposed scratchpad-assisted MRAM memory architecture provides energy efficiency by: 1) minimizing the STT-MRAM write frequency and 2) in addition, at a smaller size, SRAM is more energy-efficient than STT-MRAM [7].

V. Results and Analysis

A. Design Space Exploration for Selecting Memory Capacity

Nineteen widely used state-of-the-art deep learning models were analyzed to design and validate our STT-MRAM-based AI accelerator with the reconfigurable core. Fig. 10(a) shows the model sizes both in 8-bit int8 (left Y-axis) and 16-bit BrainFloat16 (BF16) [20], [21] (right Y-axis) datatypes. For inference-only accelerator int8 datatype and hardware suffice, however, if full-scale training or transfer learning is desired, then BF16 hardware and data-type are necessary [20], [21]. The models' sizes imply that around 280 and 140 MB of STT-MRAM are required as non-volatile (NVM) weight storage memory to store the pre-trained models using BF16 and int8 datatypes, respectively. The STT-MARM NVM weight storage memory can replace the currently used eFlash memory as an efficient alternative. Fig. 10(b) and (c) represents the input/output feature map and weight size ranges of all models for convolution layers both in int8 (left Y-axis) and BF16 (right Y-axis) formats, and these data help us to estimate the maximum required GLB memory size to avoid DRAM accesses during each convolution layer operation. In the cases of FC layer operations, only the feature maps, usually in the kB range for most of the models, are stored in the GLB, and the weights, around 200 MB in size for the largest model in BF16, are directly assigned from DRAM (or weight-storage NVM) to the systolic array for matrix multiplications. Hence, we ignored the FC layers' weight and activation sizes from design space analysis for selecting on-chip GLB memory capacity.

To fit a Conv. Layer data completely into the GLB, it needs the capacity to store: 1) input fmap (ifmap); 2) filter weights; and 3) output fmap (ofmap) of the current layer. Fig. 11 shows

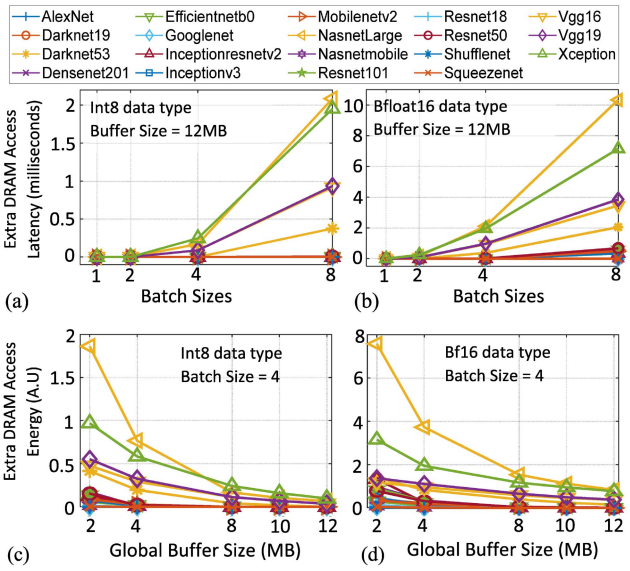

Fig. 12. For Conv. layers and total extra DRAM access latency for varying batch sizes: (a) int8 and (b) BF16 datatypes; total extra DRAM access energy for varying GLB size: (c) int8 and (d) BF16 datatypes.

the required GLB size for 19 widely used deep learning models in int8 (left Y-axis) and BF16 (right Y-axis) for different batch sizes. For a smaller batch size (i.e., $\leq$2), a maximum of 12 MB of GLB would be enough for the int8 datatype. With 12-MB on-chip GLB memory, most of the models, except a few (e.g., Darknet53, VGG19, Nasnetlarge, and Xception), can support larger batch sizes, such as 8. For BF16, 12 MB would suffice for batch size 1 for all models. If pruned models [2] are used, the batch of more images can be fit into the GLB. For high-performance accelerators that operate with larger batches of data, the GLB size can be further increased.

When a Conv. layer data—ifmap, weight, and ofmap—does not fit into GLB at one attempt, extra DRAM accesses are needed, incurring extra energy and latency. Fig. 12(a) shows that, if a GLB of 12 MB is used, even larger batch sizes, such as 8, the extra DRAM access-related latency is zero for most of the models (int8 case) and around 2 ms for few models. For the BF16 datatype, the extra DRAM access latency increases slightly but is within 10 ms. Fig. 12(c) depicts that, if the GLB size is 12 MB, for most of the models in int8 datatype, extra DRAM access-associated energy reduces to zero. For BF16 datatype, most models would need a few extra DRAM accesses [see Fig. 12(d)]. The DRAM access energy and latency were calculated for dual-channel DDR4-2933 DRAM with 64-bit data bus.

B. Memory Retention Time Estimation for AI Models

The data retention time in GLB for the models (in BF16 datatype) is calculated using (5)–(11) (see Section III) and the postlayout timing results from the implementation of our proposed reconfigurable accelerator core at 14-nm technology (see Table I). The results for 42 $\times$ 42 MAC array and batch size 16, as presented in Fig. 13, show that the maximum data retention time in GLB for all models is less than 1.5 s where most models have retention time less than 0.5 s. The retention time goes even smaller (in ms range) for

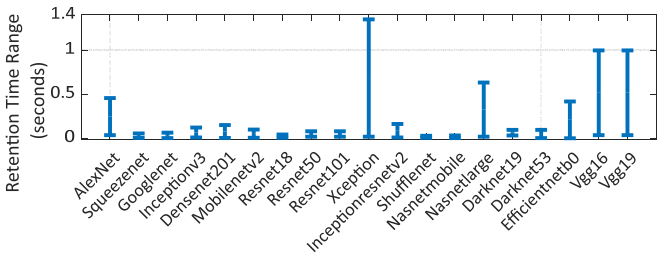

Fig. 13. GLB retention time range for 42 $\times$ 42 MAC array (Bfloat16 hardware and CLK details in Table I) and a batch size of 16.

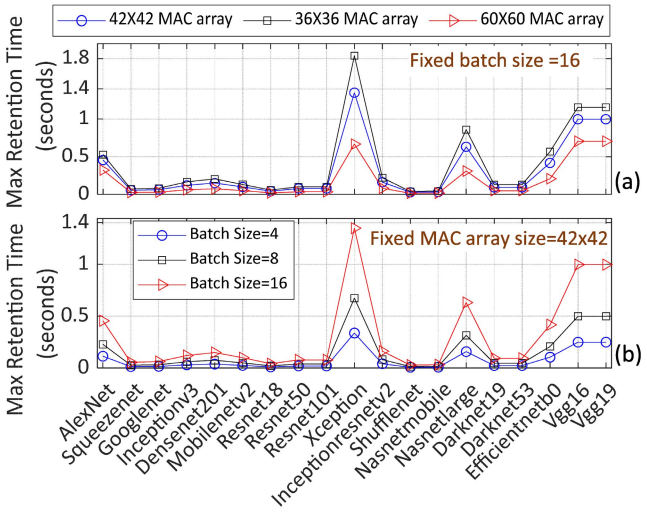

Fig. 14. Required retention time of MRAM GLB for Bfloat16 hardware (CLK cycles and frequency given in Table I). (a) Varying MAC array capacity. (b) Varying batch sizes.

int8 datatype as the clock cycle reduces significantly (usually 1–2 clock cycles) in int8 version hardware. Fig. 14(a) shows the maximum retention time for all models (in BF16 datatype) for fixed batch size 16 and varying MAC array sizes, whereas Fig. 14(b) shows the maximum retention time needed for a fixed MAC array size of 42 $\times$ 42 for varying batch sizes. From the figures, it is evident that a further reduction in retention time can be achieved by using the proper combination of batch and MAC array sizes.

C. Customizing STT-MRAM for AI Accelerator

Using (14), we analyzed the impact of the thermal stability factor (Δ) on retention time within certain BERs. To identify the target BER of STT-RAM for applications in pretrained weight storage and GLB memory, we first analyzed the size of modern AI models. From Fig. 10(a), it can be seen that a few hundred MBs would be enough to store the pretrained weights; within this memory capacity, we choose BER in the order of 10^{-9} (i.e., one bit-flip per one billion bits). Given the worst case cumulative BER can occur from RF, RD, and write error (WE), the worst case bit-flips for VGG16 at this BER is about 12 bits. This BER is negligible and cannot make any impact on the AI task's accuracy [25]. Fig. 15(a) shows that, with $\Delta = 39$, we can ensure that the loaded pretrained weight will successfully remain in the accelerator for about three years at this target BER, which is enough, given that AI models are replaced frequently with better ones.

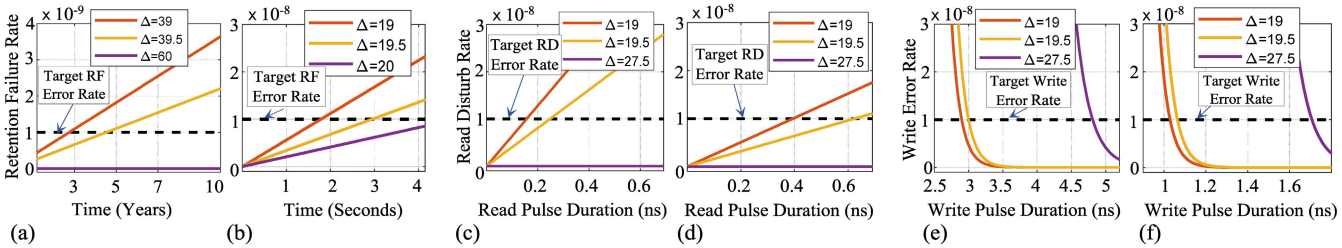

Fig. 15. (a) Thermal stability (Δ) scaling for three-year retention time (for pretrained weight storage NVM application). (b) Δ and retention time scaling for accelerator's GLB memory design. (c) and (d) With scaled Δ and read pulsewidth scaling while ensuring that RD BER is within limit. (e) and (f) Write latency scaling with Δ and within target WER. Note that (c) and (e) use base-case (10-year ret. time) from [7], and (d) and (f) from [14]. Target BER is chosen to ensure no accuracy impact on AI tasks [25].

To obtain device-level process/temperature variation profile, we varied MTJ physical parameters. In our SPICE Monte Carlo simulation flow, the sample size was 10 000. We chose process variation $\sigma = 2.1\%$ of mean (experimentally observed in [6]), $T_{\text{hot}} = 120$ °C (393 K), and $T_{\text{cold}} = -20$ °C (253 K) in 17 and 18. We adjust $\Delta = 39$ to $\Delta_{\text{PT\_GB}} = 55$ after guard-banding. After performing device- and circuit-level simulations to verify the functionality of the bit-cell and memory array, we performed a system-level evaluation of the design.

For GLB memory, we can lower Δ and retention time according to the average occupancy time of weight and input/output fmaps in the accelerator's GLB memory. Also, since this memory size is within few tens of MB (e.g., 12 MB), we can increase the BER to 10^{-8}, which will cause less than three bit-flips in the worst case (i.e., considering BER from RF, RD, and WE) at this memory size. The accuracy impact of deep learning models at this BER and memory size is negligible [25]. In Fig. 15(b), at scaled $\Delta = 19.5$ (after PT guard band $\Delta_{\text{PT\_GB}} = 27.5$), we can achieve 3 s of retention time (which is much higher than the minimum required, as shown in Fig. 14) at the target BER of 10^{-8}. In [6] the fabricated diameter reported was 38 nm for $\Delta = 60$. In our case to scale it to $\Delta = 27.5$, we estimate the scaled diameter to be 17.2 nm. Moreover, if both free-layer material property adjustment [16] and scaling are done, then we can achieve the desired Δ at a relatively higher MTJ diameter.

Next, we analyze the impact of scaling Δ on the read pulsewidth. If the read pulsewidth is large, then the chances of RD increase. Moreover, with scaling $\Delta = 19.5$ (after guard-banding $\Delta_{\text{PT\_GB}} = 27.5$), the required read current also decreases. As a result, a significant reduction in read energy is also possible. In our study of Δ scaling impact of read/write latency, as the base-case STT-MARM parameters, we used the chip-implemented data of [7] and [14]. Fig. 15(c) and (e) uses base-case (i.e., $\Delta = 60$) from [7], and (d) and (f) from [14]. With the scaling of retention time, the write latency only scales as a factor of $\ln(\Delta)$; to further decrease the write latency, we can use the write current as another knob, as discussed in Section IV. The write latency scaling is shown in Fig. 15(e) and (f).

We used the destiny memory modeling tool [18] to compare STT-MRAM area and energy with SRAM, while Δ is scaled down. Although theoretically, STT-MRAM has a minimum area of $6F^2$; however, silicon results show that the MRAM area is scaled by 70% compared to SRAM at 14-nm node [7].

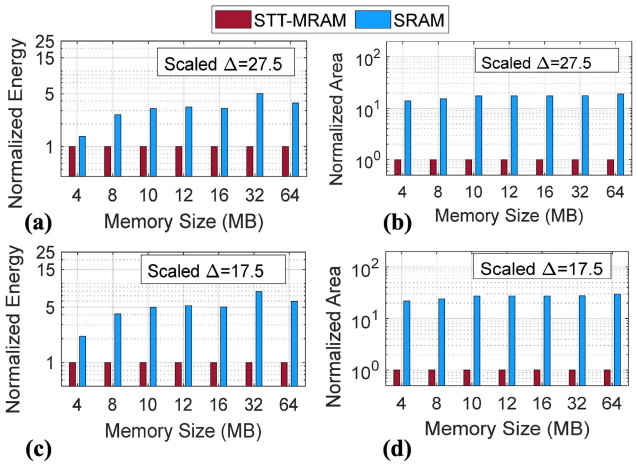

Fig. 16. Energy and area comparison of SRAM and STT-MRAM for various sizes. Δ scaled: (a) and (b) for GLB and (c) and (d) for eMRAM banks to store lower half (i.e., LSB groups) of weight/fmap bits.

We modified the destiny tool to incorporate this silicon observation. The results for scaled Δ at the 14-nm technology node are shown in Fig. 16. We see a significant advantage from STT-MRAM beyond 4-MB capacity. Compared to SRAM, the area scales by more than ten times at isomemory capacity [see Fig. 16(b) and (d)]. Similarly, for STT-MRAM, the relative energy efficiency improves as the memory capacity increases [see Fig. 16(a) and (c)]. These results imply that STT-MRAM can offer significant performance gains at future high-performance AI accelerators that will use large on-chip buffer memory.

D. Energy Optimization With Variable Retention MRAM Banks

We further improved the efficiency in STT-AI Ultra accelerator with two separate MRAM banks of $\Delta = 27.5$ and $\Delta = 12.5$ ($\Delta_{\text{PT\_GB}} = 17.5$). The first half of the weight/fmap bits is considered significant (MSB group) and stored in $\Delta_{\text{PT\_GB}} = 27.5$ bank and the rest of the LSB groups in $\Delta_{\text{PT\_GB}} = 17.5$ bank. For the LSB group at $\Delta_{\text{PT\_GB}} = 17.5$, we relaxed the BER to 10^{-5}, as shown in Fig. 17. The relative gains in energy and area are shown in Fig. 16(c) and (d).

E. Optimizing Energy With Scratchpad for Partial Ofmaps

Our simulation results show that, for STT-MRAM, the write energy is about 70% more than the read energy at scaled Δ.

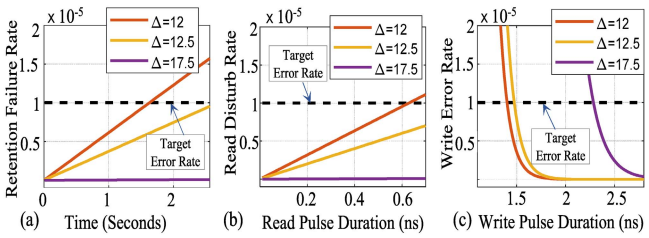

Fig. 17. Δ Scaling with relaxed BER for LSB bit groups. (a) Retention, (b) read, and (c) write latency within target BER. (Base case, $\Delta = 60$, and data modeled after [14].)

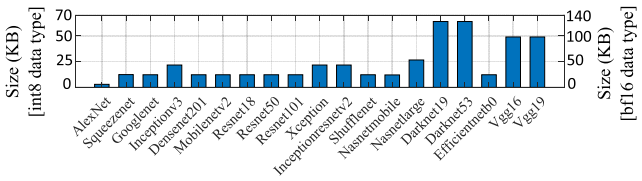

Fig. 18. Maximum size of partial ofmaps.

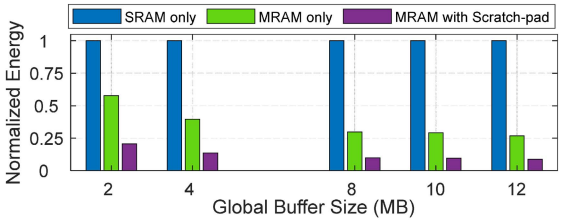

Fig. 19. Comparison of buffer memory energy dissipation for: 1) SRAM; 2) MRAM; and 3) MRAM with scratch pad architectures.

As described in Section IV-D, using a small SRAM scratchpad for writing the intermediate partial ofmaps instead of the MRAM can significantly reduce the write frequency and save energy. Fig. 18 shows the partial ofmap size distribution. For the BF16 data type, we see that 52-kB (26 kB for int8) scratchpad will fit most of the models in one attempt. The normalized energy improvements of the proposed scratchpad-assisted MRAM system are shown in Fig. 19 for ResNet-50 model and 14-nm technology.

F. Accelerator Implementation

We implemented our AI accelerator architecture with reconfigurable cores (i.e., in Fig. 3), at the RTL level using BF16 hardware as BF16 can support both inference and training. We used Synopsys 14-nm standard cell library [26] to complete synthesis, and place and route of the design. The postlayout CLK cycle data for the PE/MAC are shown in Table I. The top-level view of physical design from ICC2 tool [26] is shown in Fig. 20. We used the Synopsys 14-nm memory compiler to create the SRAMs for our baseline accelerator. Results from postlayout and timing-closed accelerator design are shown in Table II, where row 7 shows the area and power for the baseline accelerator with 12-MB GLB memory. Next, to implement the MRAM-based STT-AI accelerator, we estimated areas and power data from the destiny [18] tool at scaled Δ and modeled those as a black box in the physical design part in Synopsys ICC2 [26] for 14-nm node. The 52-kB SRAM scratchpad is divided into two banks with individual CLK/power gating. Rows 4 and 8 show

TABLE I

RECONFIGURABLE PE CORE DETAILS (BFLOAT16 HARDWARE, AND SYNTHESIZED WITH 14-nm STANDARD CELL LIBRARY [26])

Reconfigurable Core Mode	*CLK Freq*	*Required CLK Cycles*
Systolic Core (1 MAC)	1 GHz	11
Conv. Core (3 MAC)	1 GHz	17

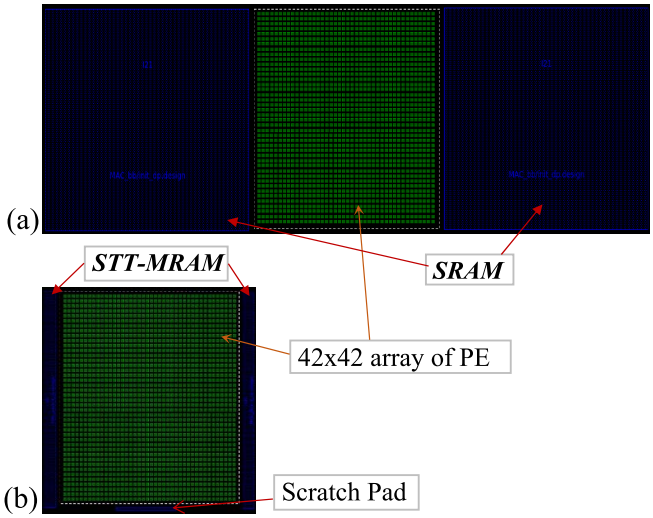

Fig. 20. Top-level floorplan view from ICC2. Accelerator designed with (a) 12-MB SRAM and (b) 12 MB-STT-MRAM with scratchpad.

TABLE II

ACCELERATOR DESIGN DETAILS AT 14 nm

Module	Details	Area (mm^2)	Dynamic Power (mW)	Leakage Power (mW)
Functional Core	Reconfigurable core with 42x42 MACs	4.08	954	0.91
SRAM Block	12 MB SRAM global memory	16.2	48.98	0.21
STT-MARM (Δ=27.5)	12 MB MRAM global memory	1.01	17.61	0.08
STT-MRAM (Δ=17.5, Δ=27.5)	6 MB MRAM (Δ=17.5) 6 MB MRAM (Δ=27.5)	0.93	13.75	0.06
SRAM ScratchPad (for MRAM)	52 KB (two 26KB blocks with CLK/power gating)	0.069	0.2	8E-4
Baseline Accelerator (SRAM-based)	Functional Core and SRAM (Row 3 above)	20.28	1003	1.13
STT-AI Accelerator	Functional core and STT-RAM (Row 4 above)	5.09	972	0.99
STT-AI Ultra Accelerator	Functional Core and STT-RAM (Row 5 above)	5.0	968	0.98

that the STT-AI accelerator offers significant area and leakage energy savings. The STT-AI Ultra accelerator achieves further improvements in power and area, as shown in row 9 in Table II.

G. Accelerator Performance With ImageNet Dataset

Next, we modeled our hardware and STT-MRAM BERs in PyTorch [27] and ran inference for pretrained AlexNet, VGG16, and ResNet-50 models with ImageNet benchmark to obtain top-one and top-five accuracy results. With STT-MRAM having $\Delta_{PT\_GB} = 27.5$, there was no accuracy loss compared to the baseline SRAM version. However, for STT-AI Ultra accelerator, with BER $= 10^{-5}$ in half of the bits (LSB group in lower Δ STT-MRAM bank), we observed

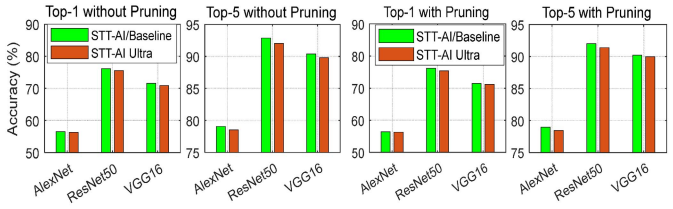

Fig. 21. Top-one and top-five accuracy comparisons for STT-AI/Baseline and STT-AI Ultra cases. No accuracy change for STT-AI/baseline cases, and negligible (less than 1% normalized) accuracy change occurs on STT-AI Ultra accelerator. Both original and pruned (at 50% pruing rate) [2] model results are shown.

negligible (less than 1% normalized) accuracy loss, as shown in Fig. 21.

VI. Related Work

Over the last decade, STT-MRAM technology has been extensively researched for its high-endurance, radiation hardness, nonvolatility, and high-density memory properties [28]–[33]. The prior works on STT-MRAM applications can be broadly categorized into two domains: 1) its use as the last-level cache memory in processors and 2) its application in emerging process-in-memory (PIM)-based computing paradigm.

Several studies [28]–[32] have used STT-MRAM in PIM setting due to complex and tunable resistance dynamics achieved through its spin-transfer torque mechanism and simultaneous access to multiple word-lines of the same array. Yan *et al.* [29] proposed crossbar arrays where the internal resistance states—which were used to mimic the weights of the models—of STT-MRAM were tuned to support nonuniform quantization. Some studies, such as [34], used the PIM architecture, where the MAC operation was simplified to addition/subtraction and bit-wise operation by manipulating the models' parameters. Shi *et al.* [31] mapped the LeNet5 model to a synaptic cross-bar array of STT-MRAM memory cells for inference. However, major challenges of PIM over conventional deep learning/AI are: 1) requirements of additional circuitry, such as DAC and ADC, which results in area overhead; 2) quantization of weights to be represented with fewer bits resulting in lower precision; 3) in digital PIM, extra manipulation of models' algorithm is needed to replace MAC with the bit-wise operation; and 4) in most of the cases, PIM is only suitable for inference-only applications. Although PIM-based analog architectures provide fast execution, the performance, energy efficiency, and reliability of analog PIM still lag behind the state-of-the-art AI accelerators [2], [3], [12].

While some research leveraged the scalable property of thermal stability factor of STT-MRAM to replace SRAM-based cache memory, others used the error tolerance property of certain applications and designed STT-MRAM-based energy-efficient cache with approximate storage. In [35], the retention time of STT-RAM was scaled to implement cache memory that can compete with SRAM-based caches, and DRAM-like refresh was used to compromise the ultralow retention time. In [36], the STT-MRAM-based approximate cache was proposed to exploit the error-resilience property of some specific applications. Unlike previous studies, Sayed *et al.* [37] proposed a hybrid STT-MRAM design for cache for different applications depending on the run-time requirements without compromising reliability.

In [38], a hybrid of SRAM and 3-D-stacked STT-MRAM-based AI accelerator was proposed for real-time learning where eMRAM acted as weight storage memory for infrequently accessed and updated layers, such as all convolutional layers and first few FC layers for a transfer learning followed by the reinforcement learning algorithm. However, due to the use of typical slow and write-power-hungry STT-MRAM, this study could not completely exploit STT-MRAM to substitute SRAM and eventually used SRAM for storing weights of the last few FC layers that are accessed and updated frequently.

In summary, prior notable research on STT-MRAM applications has focused on implementing last-level cache memory and designing in-memory computing architectures. Our work is the first to present a detailed analysis on the feasibility of using STT-MRAM as high bandwidth on-chip buffer memory in DNN/AI accelerator hardware that can offer much larger capacity at lower energy and area costs compared to SRAM. Moreover, for complete DNN model storage in edge inference devices, the nonvolatility relaxed STT-MRAM design is presented as an alternative to eFlash, which suffers from scaling limitations at advanced technology nodes.

VII. Conclusion

In this article, we demonstrated the design of highly efficient AI/deep learning accelerators that utilize emerging STT-MRAMs. Based on detailed design space exploration, we designed the STT-MRAM-based GLB to minimize DRAM access latency and energy, and reduce the area and power of the MRAM buffer. We presented an innovative runtime-reconfigurable core optimized for multiplication in convolution and FC layers. A scratchpad-assisted STT-MRAM GLB design has been demonstrated that reduces the frequency of energy-dominant write operations of the partial ofmaps during convolution. Using actual data occupancy times in memory for AI tasks, we guide the STT-MRAM thermal stability factor scaling. We showed that, with the STT-AI accelerator, 75% area and 3% power savings are possible at isoaccuracy. Furthermore, with STT-AI Ultra, 75.4% and 3.5% savings are possible in area and power, respectively, over regular SRAM-based accelerators at minimal accuracy tradeoff.

References

[1] G. Batra *et al.*, "Artificial-intelligence hardware: New opportunities for semiconductor companies," McKinsey & Company, Tech. Rep., 2019.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[3] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *ACM Commun.*, vol. 61, no. 9, pp. 50–59, 2018.

[4] M. Sadi and U. Guin, "Test and yield loss reduction of AI and deep learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Jan. 14, 2021, doi: 10.1109/TCAD.2021.3051841.

[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[6] D. Shin, J. Lee, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture," *IEEE Micro*, vol. 38, no. 5, pp. 85–93, Sep. 2018.
[7] S. Sakhare *et al.*, "J_{SW} of 5.5 MA/cm^2 and RA of 5.2-$\Omega \cdot \mu m^2$ STT-MRAM technology for LLC application," *IEEE Trans. Electron Devices*, vol. 67, no. 9, pp. 3618–3625, Aug. 2020.
[8] H.-S. P. Wong *et al.* (2020). *Stanford Memory Trends*. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends
[9] S. Moore, "Cerebras's giant chip will smash deep learning's speed barrier," IEEE Spectrum, Tech. Rep., 2020.
[10] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12.
[11] A. Antonyan, S. Pyo, H. Jung, and T. Song, "Embedded MRAM macro for eFlash replacement," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–4.
[12] H. Li, M. Bhargava, P. N. Whatmough, and H.-S.-P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *Proc. 56th Annu. Design Automat. Conf.*, Jun. 2019, pp. 1–6.
[13] Q. Dong *et al.*, "A 1 Mb 28 nm STT-MRAM with 2.8 ns read access time at 1.2 V VDD using single-cap offset-cancelled sense amplifier and *in-situ* self-write-termination," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 480–482.
[14] L. Wei *et al.*, "A 7 Mb STT-MRAM in 22 FFL FinFET technology with 4 ns read sensing time at 0.9 V using write-verify-write scheme and offset-cancellation sensing technique," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 214–216.
[15] Y.-D. Chih *et al.*, "A 22 nm 32 Mb embedded STT-MRAM with 10 ns read speed, 1M cycle write endurance, 10 years retention at 150°C and high immunity to magnetic field interference," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 222–224.
[16] J. Park *et al.*, "A novel integration of STT-MRAM for on-chip hybrid memory by utilizing non-volatility modulation," in *IEDM Tech. Dig.*, Dec. 2019, pp. 2–5.
[17] G. Hu *et al.*, "Spin-transfer torque MRAM with reliable 2 ns writing for last level cache applications," in *IEDM Tech. Dig.*, Dec. 2019, pp. 2–6.
[18] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3D NVM and eDRAM caches," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Mar. 2015, pp. 1543–1546.
[19] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De, "Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances," in *IEDM Tech. Dig.*, Dec. 2009, pp. 1–4.
[20] S. Wang and P. Kanwar, "BFloat16: The secret to high performance on cloud TPUs," Google Cloud Blog, Tech. Rep., 2019.
[21] D. Kalamkar *et al.*, "A study of BFLOAT16 for deep learning training," 2019, *arXiv:1905.12322*. [Online]. Available: https://arxiv.org/abs/1905.12322
[22] A. V. Khvalkovskiy *et al.*, "Basic principles of STT-MRAM cell operation in memory arrays," *J. Phys. D, Appl. Phys.*, vol. 46, no. 7, Feb. 2013, Art. no. 074001.
[23] Z. Diao *et al.*, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *J. Phys., Condens. Matter*, vol. 19, no. 16, Apr. 2007, Art. no. 165209.
[24] J. M. Iwata-Harms *et al.*, "High-temperature thermal stability driven by magnetization dilution in CoFeB free layers for spin-transfer-torque magnetic random access memory," *Sci. Rep.*, vol. 8, no. 1, pp. 1–7, Dec. 2018.
[25] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.
[26] *Synopsys*. Accessed: Mar. 2021. [Online]. Available: https://www.synopsys.com//
[27] *PyTorch*. Accessed: Mar. 2021. [Online]. Available: https://pytorch.org/
[28] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 470–483, Dec. 2017.
[29] H. Yan, H. R. Cherian, E. C. Ahn, X. Qian, and L. Duan, "ICELIA: A full-stack framework for STT-MRAM-based deep learning acceleration," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 408–422, Feb. 2020.
[30] A. Anwar, A. Raychowdhury, R. Hatcher, and T. Rakshit, "XBAROPT–enabling ultra-pipelined, novel STT MRAM based processing-in-memory DNN accelerator," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 36–40.
[31] Y. Shi, S. Oh, Z. Huang, X. Lu, S. H. Kang, and D. Kuzum, "Performance prospects of deeply scaled spin-transfer torque magnetic random-access memory for in-memory computing," *IEEE Electron Device Lett.*, vol. 41, no. 7, pp. 1126–1129, Jul. 2020.
[32] H. Zhuang *et al.*, "A second-order noise-shaping SAR ADC with passive integrator and tri-level voting," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1636–1647, Jun. 2019.
[33] F. Ferdaus, B. M. S. B. Talukder, M. Sadi, and M. T. Rahman, "True random number generation using latency variations of commercial MRAM chips," in *Proc. 22nd Int. Symp. Qual. Electron. Design (ISQED)*, Apr. 2021, pp. 510–515.
[34] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.
[35] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 50–61.
[36] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "STAxCache: An approximate, energy efficient STT-MRAM cache," in *Proc. Design, Automat. Test Europe Conf. Exhib. (DATE)*, Mar. 2017, pp. 356–361.
[37] N. Sayed, L. Mao, R. Bishnoi, and M. B. Tahoori, "Compiler-assisted and profiling-based analysis for fast and efficient STT-MRAM on-chip cache design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 4, pp. 1–25, Jul. 2019.
[38] I. Yoon, M. A. Anwar, R. V. Joshi, T. Rakshit, and A. Raychowdhury, "Hierarchical memory system with STT-MRAM and SRAM to support transfer and real-time reinforcement learning in autonomous drones," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 3, pp. 485–497, Sep. 2019.

Kaniz Mishty received the B.S. degree in electronics and communication engineering from Khulna University of Engineering & Technology, Khulna, Bangladesh, in 2018. She is currently working toward the Ph.D. degree in electrical and computer engineering (ECE) at Auburn University, Auburn, AL, USA.

As a Summer Intern, she worked on incorporating AI/machine learning in ASIC design flows at Qualcomm, Santa Clara, CA, USA. Her current research interests are energy- and area-efficient VLSI system design, artificial intelligence (AI)/neuromorphic hardware design, and AI/machine learning (ML) in computer-aided design (CAD).

Mehdi Sadi (Member, IEEE) received the B.S. degree from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2010, the M.S. degree from the University of California at Riverside, Riverside, CA, USA, in 2011, and the Ph.D. degree in electrical and computer engineering (ECE) from the University of Florida, Gainesville, FL, USA, in 2017.

He was a Senior Research and Development SoC Design Engineer with the Xeon Server CPU Design Team, Intel Corporation, Hillsboro, OR, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering (ECE), Auburn University, Auburn, AL, USA. His research focus is on developing algorithms and computer-aided-design (CAD) techniques for implementation, design, reliability, and security of artificial intelligence (AI), and brain-inspired computing hardware. He has published more than 20 peer-reviewed research articles. His research also spans into developing machine learning-/AI-enabled design flows for System-on-Chip (SoC) and design-for-robustness for safety-critical AI hardware systems.

Dr. Sadi was a recipient of the Semiconductor Research Corporation Best in Session Award and the Intel Xeon Design Group Recognition Awards.