ParaPIM: A Parallel Processing-in-Memory Accelerator for Binary-Weight Deep Neural Networks

Shaahin Angizi, Zhezhi He and Deliang Fan
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
{angizi,elliot.he}@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

Recent algorithmic progression has brought competitive classification accuracy despite constraining neural networks to binary weights (+1/-1). These findings show remarkable optimization opportunities to eliminate the need for computationally-intensive multiplications, reducing memory access and storage. In this paper, we present ParaPIM architecture, which transforms current Spin Orbit Torque Magnetic Random Access Memory (SOT-MRAM) sub-arrays to massively parallel computational units capable of running inferences for Binary-Weight Deep Neural Networks (BWNNs). ParaPIM's in-situ computing architecture can be leveraged to greatly reduce energy consumption dealing with convolutional layers, accelerate BWNNs inference, eliminate unnecessary off-chip accesses and provide ultra-high internal bandwidth. The device-to-architecture co-simulation results indicate ∼4× higher energy efficiency and 7.3× speedup over recent processing-in-DRAM acceleration, or roughly 5× higher energy-efficiency and 20.5× speedup over recent ASIC approaches, while maintaining inference accuracy comparable to baseline designs.

ACM Reference Format:
Shaahin Angizi, Zhezhi He and Deliang Fan. 2019. ParaPIM: A Parallel Processing-in-Memory Accelerator for Binary-Weight Deep Neural Networks. In *ASPDAC '19: 24th Asia and South Pacific Design Automation Conference (ASPDAC '19), January 21–24, 2019, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3287624.3287644

1 INTRODUCTION

Deep Convolutional Neural Network (CNN) has obtained remarkable success owning to outstanding performance in image recognition over large scale data-sets such as ImageNet [6]. Following current trend, by going deeper in CNNs (e.g. ResNet employs 18-1001 layers), memory/computational resources and their communication have faced inevitable limitations. This has been interpreted as "CNN power and memory wall" [2], leading to the development of different approaches to improve CNN efficiency at either algorithm or hardware level. Model pruning [14], parameters quantization [24] and binarization [20] are among the widely-explored algorithmic approaches to mitigate above challenges. Meanwhile, it has been proven that convolutional layers consume up to ∼90% [5, 6] of execution time and computational energy of whole CNN in both CPUs and GPUs, with the main purpose of feature extraction.

In hardware design domain, the isolated memory and processing units (GPU or CPU) interconnected via buses has faced serious challenges, such as long memory access latency, significant congestion at I/Os, limited memory bandwidth, huge data communication energy and large leakage power consumption for storing network parameters in volatile memory [8]. To address these concerns, Processing-in-Memory (PIM) CNN accelerators, as a potentially viable way to address memory wall challenge, have been widely explored [4, 5, 8, 16, 23]. The main idea of PIM is to embed logic units within memory to process data by leveraging the inherent parallel computing mechanism and exploiting large internal memory bandwidth. It could lead to remarkable saving in off-chip data communication energy and latency. The proposals for exploiting SRAM-based PIM architectures can be found in recent literature [12]. However, PIM in context of main memory (DRAM- [16]) has provided more benefits in recent years owing to the larger memory capacity and off-chip data communication reduction as opposed to SRAM-based PIM. However, the existing DRAM-based PIM architectures encounter several inevitable drawbacks, such as high refresh/leakage power, multi-cycle logic operations, operand data overwritten, etc.

The PIM architectures have recently become even more popular when integrating with emerging Non-Volatile Memory (NVM) technologies, such as Resistive RAM (ReRAM) [8]. ReRAM offers more packing density ($\sim 2-4\times$) than DRAM, and hence appears to be competitive alternatives to DRAM. However, it still suffers from slower and more power hungry writing operations than DRAM [15]. Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) [13] and recent Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) [5] technologies are other promising candidates for both the last level cache and the main memory, due to their low switching energy, non-volatility, superior endurance, excellent retention time, high integration density and compatibility with CMOS technology. Meanwhile, MRAM technology is undergoing the process of commercialization [9]. Hence, PIM in the context of different NVMs, without sacrificing memory capacity, can open a new way to realize efficient PIM paradigms [8, 17].

In this work, we take a significant step towards the energy-efficiency and performance by exploiting recent researches on Binary-Weight CNNs (BWNNs) [10, 20]. We present *ParaPIM* as a parallel in-situ accelerator, which transforms current SOT-MRAM sub-arrays to massively parallel computational units capable of running fast and energy-efficient inferences for BWNNs. We demonstrate that *ParaPIM* can accelerate BWNNs using new mapping methods, maximize resource utilization and minimize memory access leveraging the PIM technique.

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287644

2 BINARY-WEIGHT NETWORKS

CNN works in two distinct modes, first, training mode in which the configuration values of layers are calculated by training the network on pre-classified training images, and second, inference mode where new test images are examined. In both modes, Multiplication and Accumulation (MAC) operations are the most computationally-expensive arithmetic operations [2]. To eliminate the need for massive multiplication operations and memory usage, researchers have come up with various BWNNs by forcing the weights to be binary specifically in forward propagation. BinaryConnect [10] trains deep neural networks with binary weights (-1, +1) and shows near state-of-the-art results on MNIST and CIFAR-10 data-sets. This approach has the potential to bring great benefits to CNN hardware implementation by enabling the replacement of multiplication operations with much simpler complement addition/subtraction operations [2, 20], and by drastically reducing weight storage requirements. XNOR-NET [20] offers simple and accurate BWNN and achieves almost similar results with full-precision AlexNet on ImageNet. The following equation [10] shows the deterministic and stochastic binarization functions for floating point weights w_{fp}:

$$w_{b,De} = \begin{cases} +1, & w_{fp} \geq 0 \\ -1, & w_{fp} < 0 \end{cases}, \; w_{b,St} = \begin{cases} +1, & p = \sigma(w_{fp}) \\ -1, & 1-p \end{cases} \quad (1)$$

where σ is a hard sigmoid function to determine the probability distribution:

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1) = max(0, min(1, \frac{x+1}{2})) \quad (2)$$

In the next section, we present *ParaPIM* architecture to accelerate BWNNs in the PIM context.

3 PARAPIM ARCHITECTURE

The architectural diagram of the proposed *ParaPIM* accelerator is shown in Fig. 1a consisting of Image and Kernel Banks, SOT-MRAM based computational sub-arrays and a Digital Processing Unit (DPU) including three ancillary units (i.e. Binarizer, Batch Normalization and Activation Function). The accelerator can be adjusted by Ctrl unit to process entire BWNN. Assume Input feature maps (I) and Kernels (W) are initially stored in Image Bank and Kernel Bank of memory, respectively. Except for the inception block, kernels need to be constantly binarized before mapping into computational sub-arrays which are designed to handle the computational load of *ParaPIM* employing PIM techniques. However, binarized shared kernels can be utilized for different inputs. This operation is basically performed using DPU's Bin. (see DPU in Fig. 1a) and then results are mapped to the parallel sub-arrays (1st step). In the 2nd and 3rd steps, the parallel sub-arrays extract the features using combining and parallel computation schemes of *ParaPIM*, as will be explained in the following subsections. Finally, DPU's Active. activates the generated feature map to complete 4st step by producing output fmaps.

3.1 Computational Sub-arrays

Fig. 1b depicts the presented PIM sub-array architecture based on SOT-MRAM. This architecture mainly consists of Write Driver (WD), Memory Row Decoder (MRD) (elaborated in Fig. 1c Ⓐ), Memory Column Decoder (MCD), reconfigurable Sense Amplifier (SA) (Fig. 1c Ⓒ), and can be adjusted by Ctrl unit (Fig. 1c Ⓑ) to work in dual mode that perform both memory write/read and bit-line computing (using two distinct methods). SOT-MRAM device is a composite structure of spin Hall metal (SHM) and Magnetic Tunnel Junction (MTJ). The resistance of MTJ with parallel magnetization in both magnetic layers (data-'0') is lower than that of MTJ with anti-parallel magnetization (data-'1'). Each SOT-MRAM cell located in computational sub-arrays is associated with the Write Word Line (WWL), Read Word Line (RWL), Write Bit Line (WBL), Read Bit Line (RBL), and Source Line (SL) to perform operations based on reconfigurability of memory SAs.

New Reconfigurable SA: The key idea to perform memory read and bit-line computing is to choose different thresholds (references) when sensing the selected memory cell(s). The proposed reconfigurable SA, as depicted in Fig. 1c Ⓒ, consists of two sub-SAs and totally four reference-resistance branches that can be selected by Enable bits ($EN_M, EN_{OR2}, EN_{MAJ}, EN_{AND2}$) by the sub-array's Ctrl to realize the memory and computation schemes as tabulated in Table 1. Such reconfigurable SA could implement memory read and one-threshold based logic functions only by activating one enable at a time e.g. by setting EN_{AND2} to '1', 2-input AND/NAND logic can be readily implemented between operands located in the same bit-line. Meanwhile, by activating two enables at a time e.g. EN_{OR2}, EN_{AND2}, two logic functions can be simultaneously implemented and further used to generate two-threshold based logic functions like XOR/XNOR as explained accordingly.

Table 1: Configuration of enable bits for different functions.

In-memory Operations	read	OR2/ NOR2	AND2/ NAND2	MAJ/ MIN	XOR2/ XNOR2
EN_M	1	0	0	0	0
EN_{OR2}	0	1	0	0	1
EN_{MAJ}	0	0	0	1	0
EN_{AND2}	0	0	1	0	1

Memory Mode: To *write* a bit in any of the SOT-MRAM cells, e.g. in the cell of 1st row and 1st column, write current should be injected through the heavy metal substrate of SOT-MRAM. To activate this write current path, WWL1 is activated by MRD and SL1 is grounded, while all the other lines are kept floating. Now, in order to write '1' (/'0'), the WD (V1) connected to WBL1 is set to positive (/negative) write voltage. This allows sufficient charge current flows from V1 to ground (/ground to V1), leading to MTJ resistance in High-R_{AP} (/Low-R_P). For typical memory *read*, a read current flows from the selected cell to ground, generating a sense voltage (V_{sense}) at the input of SA, which is compared with memory mode reference voltage activated by EN_M ($V_{sense,P} < V_{ref,M} < V_{sense,AP}$). Now, if the path resistance is higher (/lower) than R_M (memory reference resistance) , i.e. R_{AP} (/R_P), then the SA produces High (/Low) voltage indicating logic '1' (/'0'). The idea of voltage comparison for memory read is shown in Fig. 2a.

Bit-line Computing Mode: The computational sub-array of *ParaPIM* is designed to perform bulk bit-wise in-memory logic operations between two or three operands located in the same bit-line. In the 2-input in-memory logic scheme, every two bits stored in the identical column can be selected and sensed simultaneously employing the MRD [17], as depicted in Fig. 1b. Then, the equivalent resistance of such parallel connected SOT-MRAMs and their cascaded access transistors are compared with a programmable reference by SA. Through selecting different reference resistances (R_{AND2}, R_{OR2}), the SA can perform basic 2-input in-memory Boolean functions (i.e. AND and OR) e.g. to realize AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). Consider the data organization shown in Fig. 1b L.H.S., where A and B operands

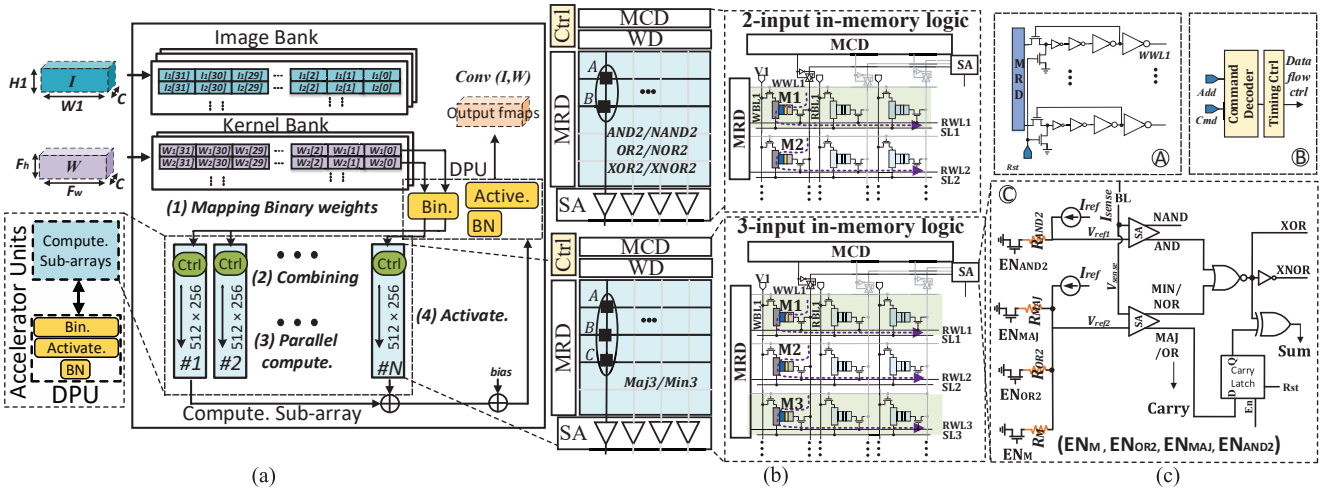

Figure 1: (a) ParaPIM accelerator architecture, (b) Computational sub-array of ParaPIM and its 2-input and 3-input in-situ logic operations, (c) Peripherals of SOT-MRAM computational sub-arrays to support computation.

correspond to M1 and M2 memory cells in Fig. 1b R.H.S., respectively, 2-input in-memory logic method generates *AB* after SA in a single memory cycle. The idea of voltage comparison between V_{sense} and V_{ref} for 2-input in-memory logic is shown on Fig. 2b. It is worth pointing out that only one sub-SA is used during one-threshold logic operations to reduce the power consumption of sensing. Owing to the complementary outputs of sub-SAs, the reconfigurable SA can also provide 2-input NOR, NAND functions. The XOR logic is realized with two SAs (i.e. performing AND and NOR logic, simultaneously) and an additional CMOS NOR gate as shown in SA circuit in Fig. 1c.

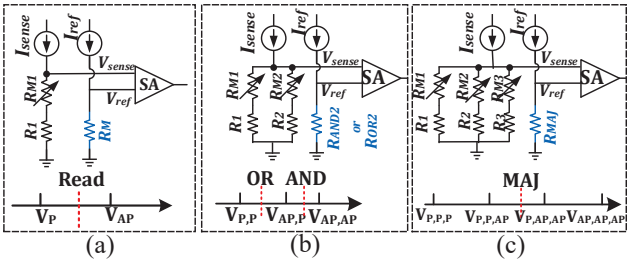

Figure 2: The idea of voltage comparison between V_{sense} and V_{ref} for (a) memory read, (b) 2-input and (c) 3-input in-memory logic operations.

In the 3-input in-memory logic scheme, every three cells located in an identical column can be selected by MRD and sensed simultaneously to realize 3-input majority/minority functions (Maj/Min) in a single sensing cycle. Consider the data organization shown in Fig. 1b where *A*, *B* and *C* operands correspond to M1, M2 and M3 memory cells, respectively, the computational sub-array can perform $AB + AC + BC$ Boolean function by setting EN_{MAJ} to '1'. As shown in Fig. 2c, to perform MAJ operation, R_{MAJ} is set at the midpoint of $R_P//R_P//R_{AP}$ ('0','0','1') and $R_P//R_{AP}//R_{AP}$ ('0','1','1').

In order to validate the variation tolerance of the sensing circuit, we have performed Monte-Carlo simulation with 10000 trials. A σ = 2% variation is added to the Resistance-Area product (RA_P), and a σ = 5% process variation is added on the Tunneling MagnetoResistive (TMR). The simulation result of sense voltage (V_{sense}) distributions in Fig. 3 shows the sense margin for conventional memory read, two fan-in in-memory logic and 3 fan-ins operation. It can be seen that sense margin gradually reduces when increasing the number of fan-ins (selected SOT-MRAM cells for computation). To avoid logic failure and guarantee the SA output's reliability, we have limited the number of sensed cells to three. Note that, such sense margin could be even improved by increasing the sense current, but by sacrificing the operation's energy-efficiency. Parallel computing/read is implemented by using one SA per bit-line.

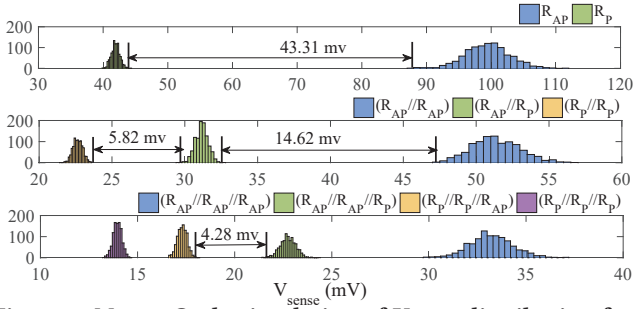

Figure 3: Monte-Carlo simulation of V_{sense} distribution for (top) memory read operation, and bit-line computing with (middle) two selected cells (down) three selected cells.

ParaPIM's sub-arrays can also perform addition/subtraction (*add*/*sub*) operation quite efficiently. With a careful observation on full-adder Boolean logic, we notice that carry-out can be directly produced by MAJ function (Carry in Fig. 1c Ⓒ) just by setting EN_{MAJ} to '1'. Accordingly, we considered a carry latch at this point to store intermediate carry outputs to be used in summation of next bits. Meanwhile, Sum output can be obtained by inserting a 2-input XOR gate in reconfigurable SA. Now, assume *A*, *B* and *C* operands (in Fig. 1b), the 2- and 3-input in-memory logic schemes can generate Sum(/Difference) and Carry(/Borrow) bits as will be elaborated in the next subsection.

3.2 In-Memory Convolver

From hardware implementation perspective, there are two types of convolution operations in BWNNs that need to be taken into account. The first one is massive binary-weight convolution (*add*/*sub*)

of middle layers between binary kernels and quantized inputs. The second one is bit-wise convolution located in inception layer and fully-connected (FC) layers in which convolution between different bit-width inputs and kernels requires bulk bit-wise operations. Both types can be readily implemented and accelerated with *ParaPIM*'s convolution methods.

Bit-wise Adder: As the main operation of BWNNs, *add*/*sub* is the most critical unit of the accelerator, as it is in charge for the most iterative layers which take up the vast majority of the run-time in the network. These units must keep high throughput and resource efficiency while handling different input bit-widths at run-time. Therefore, here we propose a parallel in-memory adder (/subtractor) based on 2- and 3-input in-memory logic schemes of *ParaPIM* to accelerate multi-bit *add*/*sub* operations. While there are few designs for in-memory adder/subtractor in literature [3, 16], to the best of our knowledge, this work is the first which presents a fast and fully parallel design in MRAM domain.

Fig. 4a elaborates the requisite data organization and computation steps of binary-weight layers with a straightforward and intuitive example in Fig. 4b only considering *add* operations. Obviously *sub* can be implemented based on *add*.

(1) Initially, c channels (here, 4) in the size of $kh \times kw$ (here, 3×3) are selected from input batch and accordingly produce a combined batch w.r.t. the corresponding binary {-1,+1} kernel batch. This combination is readily accomplished by changing the sign-bit of input data w.r.t. kernel data ($f1 * -1 = -f1$).

(2) The combined batch's channels are transposed and mapped to the designated computational sub-arrays. Considering n-activated sub-arrays with the size of x×y, each sub-array can handle the parallel *add*/*sub* of up to x elements of m-bit ($3m + 2 \leq y$) and so *ParaPIM* could process $n \times x$ elements to maximize the throughput. Here, Ch-1 to Ch-4 are respectively transposed and mapped to sub-array #1.

(3) After mapping, the parallel in-memory adder of *ParaPIM* accelerator operates to produce the output feature maps. The memory sub-array organization for such parallel computation is delineated in Fig. 4a R.H.S. Two reserved rows for Carry results initialized by zero and m (here, 4) reserved rows are considered for Sum results. We have shown the current state (Q) as well as the next state (Q*) of SA's latch after being enabled for further clarification. We use the *add* operation of two matrices of 4-bit elements (Ch1 and Ch2) in Fig. 4b to elaborate how addition operates in the *ParaPIM*. Every two corresponding elements that are going to be added together have to be aligned in the same bit-line. Here, Ch1 and Ch2 should be aligned in the same sub-array. Ch1 elements take the first 4 rows of the sub-array followed by Ch2 in the next 4 rows.

The addition algorithm starts bit-by-bit from the LSBs of the two words and continues towards MSBs. There are 2 cycles for every bit-position computation divided into four steps indicated by S1, S2, C1 and C2. In step 1 of Sum (S1), 2 RWLs (accessing to LSBs of 4 elements) and Latch (storing zero) are enabled to generate the sum. The SAs use the 2 bit cells located in the same bit-lines as input operands and carry latch's data as carry-in to generate sum based on the method explained in the previous subsection. During step 2 of Sum (S2), a WWL is activated to save back the Sum bit. In step 1 of Carry (C1), the same 2 operands in conjunction with one of the carry's reserved rows are enabled to generate the carry-out

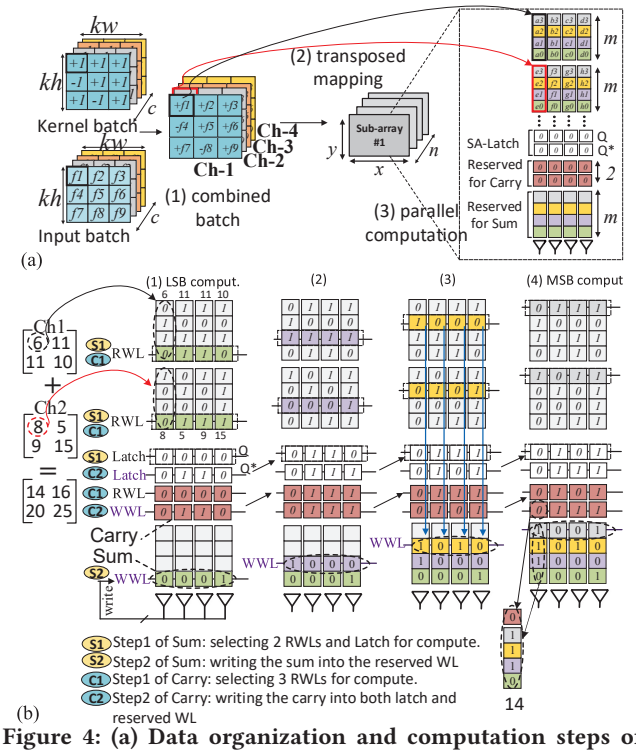

Figure 4: (a) Data organization and computation steps of binary-weight layers, (b) Parallel in-memory addition steps for generating sum and carry-out logic.

leveraging bit-line computing mode of *ParaPIM* accelerator. During step 2 of Carry (C2), a WWL is activated to save back the carry-out bit into a reserved row and also in latch. This carry-out bit overwrites the data in the carry latch and becomes the carry-in of the next cycle. This process is concluded after $2 \times m$ cycles, where m is number of bits in elements.

Bit-wise Convolver: Besides binarized layers, there are other layers in BWNNs, such as first conv. layer (directly taking image as inputs, not replaced by *add*/*sub*) and FC layers. Note that, FC layers can be equivalently implemented by convolution operations using 1×1 kernels [24]. Thus, the rest layers could be accelerated all by convolution computation by exploiting *logic AND* and *bitcount* operations as explained in [24]. The proposed 2-input in-memory logic scheme of *ParaPIM* is readily leveraged to accelerate bulk bit-wise AND logic required for this layer. Besides bitcount operation is translated to addition of bits. Due to the lack of space, we refer the readership to a hardware implementation of such method [5].

4 PERFORMANCE EVALUATIONS

In this section, we compare *ParaPIM* with other possible BWNN acceleration solutions on DRAM, ASIC, ReRAM and GPU. Obviously, enlarging the chip area brings a higher performance for *ParaPIM* and other designs due to the increased number of sub-arrays or computational units, though the die size directly impacts the chip cost. Therefore, to have a fair comparison, the area-normalized results (performance/energy per area) will be reported henceforth.

4.1 Experiment's setup

Accelerators: ***ParaPIM:*** We configure the *ParaPIM*'s memory sub-array organization with 256 rows and 512 columns per mat organized in a H-tree routing manner, 2×2 mats (with 2/2 and 2/2 as

Row and Column Activations) per bank, 8×8 banks (with 1/8 and 8/8 as Row and Column Activations) per group; in total 16 groups and 512Mb total capacity. To assess the performance of *ParaPIM* as a new PIM platform, a comprehensive device-to-architecture evaluation framework along with two in-house simulators are developed. First, at the device level, we jointly use the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) with spin Hall effect equations to model SOT-MRAM bitcell [5, 13]. For the circuit level simulation, a Verilog-A model of 2T1R SOT-MRAM device is developed to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE. 45nm North Carolina State University (NCSU) Product Development Kit (PDK) library [1] is used in SPICE to verify the proposed design and acquire the performance. Second, an architectural-level simulator is built based on NVSim [11]. Based on the device/circuit level results, our simulator can alter the configuration files (.cfg) corresponding to different array organization and report performance metrics for PIM operations. The controllers and add-on circuits are synthesized by Design Compiler [22] with an industry library. Third, a behavioral-level simulator is developed in Matlab calculating the latency and energy that *ParaPIM* spends on BWNNs. In addition, it has a mapping optimization framework to maximize the performance according to the available resources based on method explained in Section III.B. ***DRAM:*** We developed a DRISA-like [16] accelerator for BWNNs. *1T1C-adder* method of DRISA was selected for comparison which exploits a large *n*-bit adder circuit for *n*-bit BLs after SAs. We modified CACTI [7] for evaluation of DRAM's solutions. Similar to [16], the controllers and adders were synthesized in Design Compiler [22]. ***ASIC:*** We developed a YodaNN-like [2] ASIC accelerator. To have a fair comparison, we select two versions with either 8×8 tiles or 16×16 tiles. Accordingly, we synthesized the designs with Design Compiler [22] under 45 nm process node. The eDRAM and SRAM performance were estimated using CACTI [18]. ***ReRAM:*** A Prime-like [8] accelerator with two full functional (FF) sub-arrays and one buffer sub-array per bank (totally 64 sub-arrays) were considered for evaluation. In FF subarrays, for each mat, there are 256×256 ReRAM cells and eight 8-bit reconfigurable SAs. For evaluation, NVSim simulator [11] was extensively modified to emulate Prime functionality. Note that the default NVSim's ReRAM cell file (.cell) was adopted for the assessment. ***GPU:*** We used the NVIDIA GTX 1080Ti Pascal GPU. It has 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance). The energy consumption was measured with NVIDIA's system management interface. Similar to [16], we scaled the achieved results by 50% to exclude the energy consumed by cooling, etc.

Model: We select an eight-bit configuration for the inputs in BWNN. The SVHN [19] as a real-world image data-set consisting of photos of house numbers in Google Street View images, is selected for evaluation. The cropped format of colored images (32×32) centered around each single digit is selected. Accordingly, the images are re-sized to 40×40 and fed to the model. Our model is a CNN with 6 binary-weight convolutional layers, 2 (average) pooling layers and 2 FCs that cost about 80 FLOPs for a 40×40 image. To avert prediction accuracy degradation, we don't quantize the first and last layers [5, 20, 24].

4.2 Energy & Performance

Fig. 5a shows the *ParaPIM*'s energy-efficiency results (frames per joule) on the BWNN model compared to different accelerators for performing a similar task with a batch size of 8 and 32. As can be seen, the larger the batch is, the lower energy-efficiency is obtained. We observe that *ParaPIM* solution offers the highest energy-efficiency normalized to area compared to others owning to its fast, energy-efficient and fully-parallel operations. It shows on average 4× and 13.5× higher energy-efficiency than that of DRAM 1T1C-adder and ReRAM-based accelerators, respectively. In addition to large refresh power of DRAM-based PIM accelerators [16], they are dealing with a destructive *data-overwritten* issue due to the charge sharing characteristic of capacitors. It means that the result of computation will ultimately overwrite the operands. To solve this issue in the context of DRAM, *multi-cycle operations* [21] are set forth which has further degraded PIM performance. Note that, the *n*-bit adder located after SAs in DRAM-1T1C-adder solution doesn't necessarily provide higher performance due to its non-parallel operations and so it has limited its energy-efficiency. Meanwhile, we observe that *ParaPIM* solution is approximately 5× more energy-efficient than that of ASIC64 solution. This energy reduction basically comes from two sources: first, the standard convolution that are replaced and accelerated by energy-efficient *add*/*sub* in *ParaPIM* and second, limited and multiplexed computation resources of ASIC solution compared to the large number of available computational sub-arrays in *ParaPIM* distributed across different banks. Fig. 5 also shows that *ParaPIM* obtains ~42× saving in energy compared to GPU solution.

Fig. 5b depicts the *ParaPIM*'s performance (frames per second) results. We observe that *ParaPIM* solution is ~7.3× faster than that of DRAM solution (1T1C-adder) and 20.5× faster than ASIC64 solution. This is mainly because of (1) ultra-fast and parallel in-memory operations of *ParaPIM* compared to multi-cycle DRAM operations and (2) the existing mismatch between computation and data movement in ASIC designs and even 1T1C-adder solution. As a result, ASIC256 with more tiles does not demonstrate a higher performance. We can also observe that the larger the batch is, the higher performance is obtained for *ParaPIM* solution compared DRAM owning to the its more paralleled computations. Additionally, *ParaPIM* shows 14.7× higher performance than that of ReRAM solution. Note that ReRAM design employs matrix splitting due to intrinsically limited bit levels of ReRAM device so multiple sub-arrays are

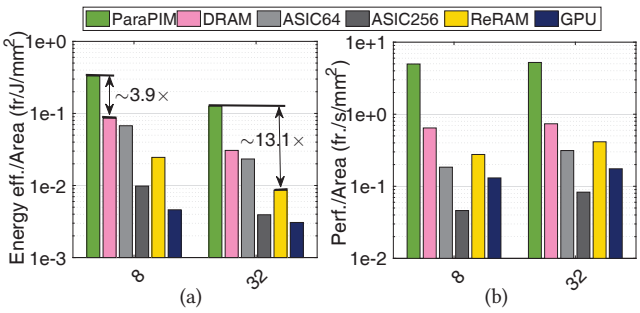

Figure 5: (a) Energy-efficiency and (b) Performance evaluation of different accelerators for running BWNNs normalized to area (Y-axis=Log scale).

occupied. Besides. ReRAM crossbar has a large peripheral circuit's overhead such as buffers and DAC/ADC which contribute more than 85% of area [8]. Compared to GPU-based solution, *ParaPIM* can obtain roughly 33× higher performance normalize to area.

4.3 Memory Bottleneck

ParaPIM's in-situ computing architecture can be leveraged to accelerate BWNNs inference, eliminate unnecessary off-chip accesses and provide ultra-high internal bandwidth. Fig. 6a depicts the memory bottleneck ratio i.e. the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens). The evaluation is performed according to the peak performance and experimentally extracted results for each platform considering number of memory access. The results show the *ParaPIM*'s efficiency for solving memory wall issue. We observe that *ParaPIM*, DRAM and ReRAM solutions spend less than ~16% time for memory access and data transfer owning to the PIM acceleration schemes. However, ASIC accelerators spend more than 90% time waiting for the loading data. Note that, DRAM-based solution shows higher memory bottleneck ratio compared with *ParaPIM* due to its unbalanced computation and data movement. GPU data could not be accurately reported for this evaluation. The less memory bottleneck ratio can be translated as the higher resource utilization ratio for the accelerators plotted in Fig. 6b. We observe that *ParaPIM* has the highest resource utilization with up to ~70%. Overall, PIM solutions demonstrate high ratio (>60%) which reconfirms the results reported in Fig. 6a.

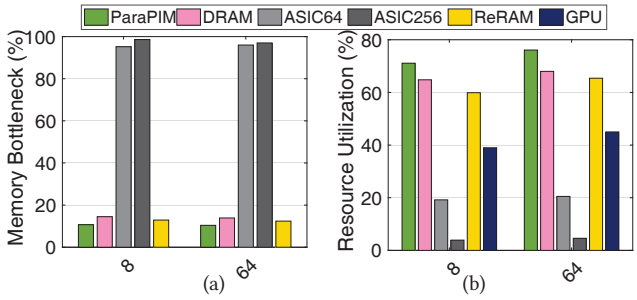

Figure 6: (a) The memory bottleneck ratio and (b) resource utilization ratio.

4.4 NVM-PIM Comparison

We evaluate the energy and area of BWNN accelerators implemented by two promising resistive memories (i.e. ReRAM [23] and SOT-MRAM [5]) for inference of one single image over three well-known data-sets under a 45nm technology node. Based on Table 2, *ParaPIM* can process BWNN quite efficiently. The energy and area reported in Table 2 are the convolution computation energy and area of all layers as an accelerator. We observe that *ParaPIM* processes binary-weight AlexNet [20] for ImageNet favorably with 561.3μJ/img and reduces energy and area by ~4× and 5×, respectively, compared to the ReRAM-based design. Meanwhile, it shows ~1.4× energy saving compared to IMCE with smaller area.

5 CONCLUSIONS

In this work, we presented *ParaPIM* architecture to transform current SOT-MRAM sub-arrays to massively parallel computational units capable of running inferences for Binary-Weight Deep Neural Networks (BWNNs). This architecture is leveraged to greatly reduce energy consumption dealing with convolutional layers and also accelerate BWNN inference. The simulation results show ~4× higher energy-efficiency and 7.3× speedup over recent processing-in-DRAM-based acceleration, or roughly 5× higher energy-efficiency and 20.5× speedup over recent ASIC approaches.

Table 2: Energy-area of BWNN accelerators.

Designs	ImageNet		SVHN		MNIST	
	Energy (μJ/img)	Area (mm^2)	Energy (μJ/img)	Area (mm^2)	Energy (μJ/img)	Area (mm^2)
ReRAM [23]	2275.34	9.19	425.21	0.085	13.55	0.060
IMCE [5]	785.25	2.12	135.26	0.01	0.92	0.009
ParaPIM	561.3	1.8	91.65	0.021	0.85	0.013

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] 2011. NCSU EDA FreePDK45. http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[2] Renzo Andri et al. 2016. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In *ISVLSI*. IEEE, 236–241.
[3] Shaahin Angizi et al. 2017. Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device. In *ISVLSI*. IEEE, 45–50.
[4] Shaahin Angizi et al. 2018. CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator. In *55th DAC*. ACM, 105.
[5] Shaahin Angizi et al. 2018. IMCE: energy-efficient bit-wise in-memory convolution engine for deep neural network. In *Proceedings of the 23rd ASP-DAC*. IEEE Press, 111–116.
[6] Lukas Cavigelli et al. 2015. Accelerating real-time embedded scene labeling with convolutional networks. In *DAC, 2015 52nd ACM/IEEE*.
[7] Ke Chen et al. 2012. CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory. In *DATE, 2012*. IEEE, 33–38.
[8] Ping Chi et al. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA*. IEEE Press.
[9] S-W Chung et al. 2016. 4Gbit density STT-MRAM using perpendicular MTJ realized with compact cell structure. In *IEDM*. IEEE.
[10] Matthieu Courbariaux et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.
[11] Xiangyu Dong et al. 2014. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*. Springer, 15–50.
[12] Charles Eckert et al. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *arXiv preprint arXiv:1805.03718* (2018).
[13] Xuanyao Fong et al. 2016. Spin-transfer torque devices for logic and memory: Prospects and perspectives. *IEEE TCAD* 35 (2016).
[14] Song Han et al. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR'16*.
[15] Benjamin C Lee et al. 2009. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM.
[16] Shuangchen Li et al. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *Micro*. ACM, 288–301.
[17] Shuangchen Li, Cong Xu, et al. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*. IEEE.
[18] Naveen Muralimanohar et al. 2009. CACTI 6.0: A tool to model large caches. *HP Laboratories* (2009), 22–31.
[19] Yuval Netzer et al. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop*, Vol. 2011. 5.
[20] Mohammad Rastegari et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.
[21] Vivek Seshadri et al. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Micro*. ACM, 273–287.
[22] Synopsys Design Compiler. Product Version 14.9.2014. Synopsys, Inc. [n. d.]. ([n. d.]).
[23] Tianqi Tang et al. 2017. Binary convolutional neural network on RRAM. In *22nd ASP-DAC*. IEEE, 782–787.
[24] Shuchang Zhou et al. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint:1606.06160* (2016).