Implementing Binary Neural Networks in Memory with Approximate Accumulation

Saransh Gupta
sgupta@ucsd.edu
University of California, San Diego

Mohsen Imani
moimani@ucsd.edu
University of California, San Diego
University of California, Irvine

Hengyu Zhao
h6zhao@ucsd.edu
University of California, San Diego

Fan Wu
f2wu@ucsd.edu
University of California, San Diego

Jishen Zhao
jzhao@ucsd.edu
University of California, San Diego

Tajana Šimunić Rosing
tajana@ucsd.edu
University of California, San Diego

ABSTRACT

Processing in-memory (PIM) has shown great potential to accelerate the inference tasks of binarized neural networks (BNNs) by reducing data movement between processing units and memory. However, existing PIM architectures require analog/mixed-signal circuits that do not scale with the CMOS technology. On the contrary, we propose BitNAP (Binarized neural network acceleration with in-memory ThreSholding), which performs optimization at operation, peripheral, and architecture levels for an efficient BNN accelerator. BitNAP supports row-parallel bitwise operations in crossbar memory by exploiting the switching of 1-bit bipolar resistive devices and a unique hybrid tunable thresholding operation. In order to reduce the area overhead of sensing-based operations, BitNAP presents a memory sense amplifier sharing scheme and also, a novel operation pipelining to reduce the latency overhead of sharing. We evaluate the efficiency of BitNAP on the MNIST and ImageNet datasets using popular neural networks. BitNAP is on average 1.24× (10.7×) faster and 185.6× (10.5×) more energy-efficient as compared to the state-of-the-art PIM accelerator for simple (complex) networks.

CCS CONCEPTS

• **Hardware → Emerging architectures**; *Non-volatile memory*; • **Computer systems organization →** *Neural networks.*

KEYWORDS

binary neural networks, memristors, processing in memory

ACM Reference Format:
Saransh Gupta, Mohsen Imani, Hengyu Zhao, Fan Wu, Jishen Zhao, and Tajana Šimunić Rosing. 2020. Implementing Binary Neural Networks in Memory with Approximate Accumulation. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '20), August 10–12, 2020, Boston, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3370748.3406562

ISLPED '20, August 10–12, 2020, Boston, MA, USA

ACM ISBN 978-1-4503-7053-0/20/08.
https://doi.org/10.1145/3370748.3406562

1 INTRODUCTION

Artificial neural networks, in particular deep learning, have a wide range of applications in different areas such as detection [1], self-driving cars, and translation [2]. In some specific tasks, such as AlphaGo [3] and ImageNet Recognition [4], deep learning algorithms are providing human-level performance. Convolutional neural networks (CNNs) are the most commonly used deep learning applications which are both compute and memory intensive [1]. Prior work tried to reduce the CNN computation cost by binarizing the input and weight of each neuron during the training process [5–7]. Binarized neural networks (BBNs) represent weights and inputs with 1-bit values, replacing multiplications with XNOR [5, 8].

Although BNNs reduce the computation cost, they are still memory intensive, thus processing them on conventional cores is slow and inefficient [9]. The inefficiency comes from the limited on-chip cache of conventional cores, which does not have enough capacity to store all BNN weights. This results in a large amount of data movement between the memory and processing cores [10, 11]. Processing in-memory (PIM) is a promising solution to address the data movement issue [12]. Low leakage power and high density of emerging non-volatile memories, e.g., resistive random access memory (ReRAM), make these technologies a great candidate for PIM. Work in [10, 11] exploit the analog characteristic of non-volatile memory to support matrix multiplication in memory. These architectures transfer the digital input data into an analog domain and compute matrix multiplication by passing an analog signal through a crossbar ReRAM. However, the state-of-the-art PIM architectures have two main disadvantages: (i) they use digital-to-analog converter (DAC), and analog-to-digital converter (ADC) blocks to transfer data between analog and digital domain [10]. These blocks take the majority of the chip area and do not scale with technology. (ii) Analog processing results in a large amount of internal data movement, as data points cannot be processed where they are already stored. The slow write operation in PIM architecture can significantly degrade the PIM efficiency in practice.

In this paper, we propose BitNAP, a highly parallelized and efficient in-memory BNN accelerator. BitNAP maps the computation of different BNN layers into a digital crossbar memory, removing the necessity of the data conversion to the analog domain. BitNAP exploits the switching characteristics of NVMs to implement row-parallel operations internally in a crossbar memory. BitNAP

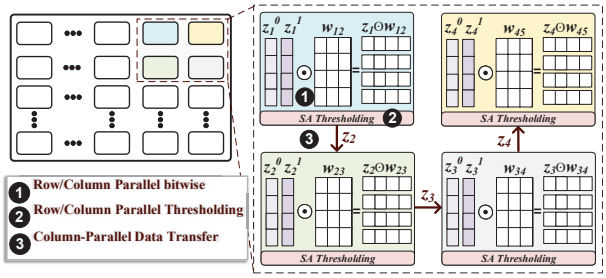

Figure 1: BitNAP overview architecture.

also proposes a hybrid tunable thresholding operation to implement accumulation and activation in a parallel and efficient way by using switching and sensing PIM operations. To fully exploit the high parallelism provided by the switching based operations, BitNAP presents new implementations of different neural network layers. BitNAP also eliminates the overhead of adding thresholding capability to sense amplifiers by sharing them between different blocks. To further reduce the latency bottleneck due to sharing, BitNAP uses a novel scheme to pipeline sense amplifier operations. We evaluate BitNAP on popular large-scaled classification datasets such as MNIST and ImageNet. BitNAP is on average 1.24× (10.7×) faster and 185.6× (10.5×) more energy-efficient as compared to the state-of-the-art PIM accelerator for simple (complex) networks.

2 RELATED WORK

Neural networks are computationally expensive due to their large amount of vector-matrix multiplication. Prior work improves the efficiency of CNNs by computing with binary inputs and weights [5, 13, 14]. A binary neural network is significantly smaller than an equivalent network with single-precision weight values. A prominent advantage of a BNN as compared to a CNN is that BNN does most calculations with bit-level operators. The state-of-the-art work in [5] implemented BNNs by representing weights/inputs by +/-1, thereby converting multiplication into XNOR. As a result, the convolutions can be estimated by XNOR and bit-counting operations.

Prior work accelerated BNN computation on GPU, FPGA, and ASICs [8, 15–19]. However, a large amount of data movement between off-chip memory and processing cores slows down the computation. Processing in-memory architectures are a suitable candidate for accelerating BNN applications. PIM architectures provide significant parallelism while reducing data movement between the memory and processing cores, resulting in high energy-efficiency. The work in [15, 16, 20] proposed a distributed in-memory computing architecture using ReRAM. It accelerated binary vector-matrix multiplication in the crossbar, which is the base operation in any BNN. The design proposed in [9] accelerates the whole BNN using ReRAM crossbar. All these approaches depend heavily on their peripheral circuits (large sense amplifiers or ADCs/DACs) for computation, which restricts the possible parallelism. Similarly, the work in XNOR-POP [8] implemented a DRAM-based architecture for BNNs. Although it exploits the DRAM row buffers to use a large number of peripherals in parallel, the performance of these peripherals becomes a bottleneck for large networks. Neurocube [17] attempted to accelerate neuromorphic computing using HMC based 3D-DRAM, limited by the inefficient HMC interface.

In contrast, in this work, we exploit the idea of digital PIM architecture. Here, functions are applied to values stored in memory without transferring them into analog domain, eliminating the necessity of using ADC/DAC blocks. We reduce the latency of in-memory accumulation with new thresholding techniques. We map entire layers to memory crossbars. We also share sense amplifiers to reduce the SA area and energy consumption of BitNAP.

3 BitNAP: PIM-BASED BNN

3.1 BitNAP Overview

Figure 1 shows the overview of the BitNAP consisting of several memory blocks. Each memory block in BitNAP models the functionality of a single BNN layer. The pre-trained weights of BNNs are pre-stored in each crossbar memory. Each memory block enables computation by performing a row-parallel XNOR operation between the inputs and weights. XNOR can be implemented in parallel over all rows of the memory. BitNAP first accumulates the results of XNOR operation and then passes through an activation function. BitNAP fuses these two operations by thresholding the outputs of XNOR operations. BitNAP uses a hybrid thresholding approach, utilizing both in-memory and sense amplifier based thresholding. Thresholding compares the number of ones in the XNOR results with a threshold value. Each memory block can also implement convolution using these operations. Finally, BitNAP implements pooling using a series of OR-based or AND-based operations. One major issue with the current PIM architectures is the size of the sense amplifier. BitNAP proposes a sense amplifier sharing and novel pipelining scheme to obtain high performance while significantly reducing the area of the accelerator.

3.2 BitNAP Block Computation

BNNs require bitwise operations and thresholding functionality. For bitwise operations, the XNOR is required in both fully connected and convolution layers to support the binary multiplication between the input and weight matrix. The OR/AND operation is required to implement MIN/MAX pooling. The thresholding functionality is used to implement the accumulation and activation functions.

PIM Bitwise Operations: BitNAP utilizes switching based PIM operations to implement bitwise operations in memory. The work in [21–23] implements different logic gates like `NOR, NAND, OR, XNOR`, etc in the digital memory. Figure 2a explains the execution proposed in [22] where, by applying a voltage, V_0, at the input devices and grounding the output device, many bitwise operations are performed. The output device switches its state whenever the voltage developed across it is higher than the threshold of the device. A sequence of these logic gates can implement other operations such as XNOR ($\odot$). As shown in Figure 2a, these PIM operations can be applied over multiple rows in parallel.

PIM Thresholding: After window-wise XNORing the input and weight matrices, thresholding is applied to all the windows independently. BitNAP uses a hybrid approach and combines sense amplifier based thresholding (*sa-THR*) with memristor switching based thresholding (*mem-THR*).

sa-THR: In this paper, we integrate thresholding functionality with the sense amplifiers of each memory block. In a typical memristive crossbar block or array, each memory column has a separate

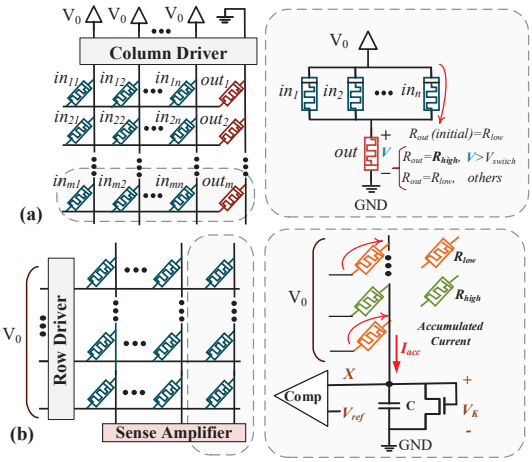

Figure 2: (a) Row-parallel bitwise PIM operations and (b) sense amplifier based thresholding (sa-THR).

sense amplifier. Activating multiple rows at the same time provides the opportunity to do computation on the data in those rows. Figure 2b shows the structure of the sa-THR. The total current injected into a sense amplifier depends on the number of '1's (low resistance devices) present in the activated rows (r_1, r_2, r_3) at the corresponding column (for example, r_{1m}, r_{2m}, r_{3m} at bit position m). This incoming current develops a voltage at node 'X', given by the relation $V = I * t/C$. More bits with '1' value result in a higher voltage at node 'X.' Finally, a voltage comparator compares the voltage at node 'X' and the reference voltage, v_{ref}, and generates a single bit output. BitNAP configures different threshold values by changing v_{ref}. This output corresponds to the function $THR(r_{1m}, r_{2m}, r_{3m})$ and is generated in parallel for all the columns.

The sa-THR circuit can execute all the required threshold operations in a BNN, but the number and placement of sense amplifiers may restrict the parallelism and direction of thresholding. Also, since all the results are generated at the periphery of the memory, it leads to unnecessary reads and writes of intermediate data. Hence, BitNAP implements a hybrid approach which combines sa-THR with a novel in-memory threshold technique, called mem-THR. It is based on the memristor switching and decouples data read-out and thresholding to provide highly parallel thresholding.

mem-THR: As discussed before, FELIX [22] can implement different bitwise PIM operations in-memory by changing the voltage, V_0, in the circuit in Figure 2a. We extend this concept to propose a new switching based configurable thresholding. A NOR operation is equivalent to thresholding at one '1' in the inputs, where the output device switches in the presence of one or more '1's. For the device threshold parameter $v_{off} = 0.5V$ [22], a V_0 of 1V is required to implement NOR operation. If we lower the voltage to 0.75V, the output switches when two or more inputs are '1.' Hence, the circuit implements in-memory configurable thresholding, where a voltage of $v_{off} \times (1 + 1/n)$ is required to detect the presence of n '1's. Our circuit-level simulations show that the detection works without error in the presence of 10% variations in applied voltage and device resistance. We call this way of performing in-memory thresholding as mem-THR. BitNAP uses a combination of sa-THR and mem-THR functions to achieve maximum efficiency while thresholding. For

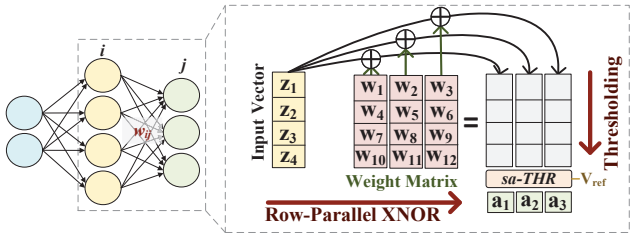

Figure 3: A fully-connected layer of BNN in BitNAP.

example, in the case of a 2D window, mem-THR is first performed over rows, reducing each row to a single element and 2D input to a column vector. Then, sa-THR is performed over the column to obtain the final output of the layer.

4 BitNAP ARCHITECTURE

In-memory bitwise operations are in general slower than the corresponding CMOS-based implementations because memristor devices are comparatively slower in switching. However, PIM architecture can potentially provide significant performance gains over CMOS accelerators when running applications with extensive parallelism. Section 3.2 showed how PIM supports different BNN operations in parallel, irrespective of the number of rows. It takes the same amount of time for PIM to process a PIM operation in a single row or all the rows of memory. On the other hand, the processing time in conventional cores highly depends on the data size. In this section, we explain how BitNAP parallelizes the functionality of different BNN layers in a memristive crossbar memory block.

4.1 Fully Connected

Figure 3 shows a row-parallel implementation of a fully connected layer in a crossbar memory. BitNAP stores the input vector vertically along with the weight matrix stored its adjacent columns (W_{ij}), where a column of the matrix contains the weights corresponding to the same neuron. BitNAP first computes the XNOR of the input vector with all columns of the weight matrix. It writes the column-wise XNOR results in separate columns of the same memory block. Next, BitNAP needs to accumulate the stored XNOR results column-wise. Finally, the result of accumulation passes through an activation function, which compares the output of the accumulated neuron with fixed threshold values. BitNAP fuses the accumulation and activation function by enabling analog configurable thresholding. As explained in Section 3.2, BitNAP uses sa-THR (thresholding enabled sense amplifier scheme) to implement thresholding in fully connected layers due to the presence of a large number of inputs. The result of thresholding is a binary vector (one bit per column) available as the sense amplifier output. It acts as the input vector for the next BNN layer and is written to the next memory block.

To estimate the execution time of a fully connected layer in BitNAP, let us assume that each XNOR operation takes $R = 2$ cycles (1 cycle: 1.1ns). Owing to the row-parallel bitwise operations supported by BitNAP, it can XNOR all the inputs (stored as a column vector) with a column of weight matrix in R cycles. Hence, for a fully connected BNN layer with M inputs and N neurons, we require $R \times N$ cycles to perform the multiplication and a single cycle to perform thresholding. The computation of a fully connected layer in BitNAP is independent of the number of inputs in the layer.

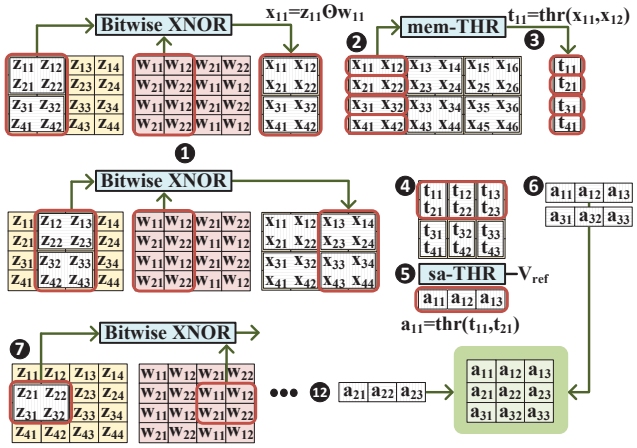

Figure 4: BitNAP implementing a convolution layer of BNN.

4.2 Convolution

Figure 4 shows the implementation of convolution layer in BitNAP. BitNAP stores input matrix (z matrix in the figure) in memory. The inputs are convoluted with weights by performing element-wise XNOR between weight window and subsets of input and then accumulating the XNOR output for each window.

Weight storage and XNOR in a convolution window: The weight kernel ($[w_{11}\ w_{12};\ w_{21}\ w_{22}]$) is flattened out and stored as a row vector adjacent to each row of the input matrix. It transforms the original weight matrix into a matrix containing as many copies of weights as the rows in the input matrix. Each row of the flattened weight kernel is left-rotated by $((I_r - 1)\%W_h) \times W_w$, where I_r is the input row number and W_h and W_w are the height and width of the weight matrix. XNOR happens in the same way as in FC layer (❶).

Sliding over columns: Let the widths of weight kernel (before flattening) and input matrix be W_w and I_w respectively. The first W_w columns (1 to W_w) of input matrix are XNORed with the first W_w columns of weight matrix. Then, the second W_w columns (2 to $W_w + 1$) of input matrix are XNORed with the first W_w columns of weight matrix and so on (❶), requiring $(I_w - W_w + 1) * W_w$ XNOR cycles in total. At this point, we have the bitwise XNOR outputs for a convolution window slid across columns (❷). Now, BitNAP applies hybrid thresholding on these outputs window-wise. First, BitNAP applies row-wise thresholding on outputs of each convolution window (currently 2D) using mem-THR (❸), generating vectors of length W_h per window (❹). This requires $(I_w - W_w + 1)$ mem-THR cycles. Then, sa-THR is performed on these vectors. The sa-THR processes $(I_w - W_w + 1)$ windows in parallel (❺). Hence, it requires I_h/W_h sa-THR cycles to perform thresholding on all the windows (❻). In total, sliding over column generates outputs for $(I_w - W_w + 1) * (I_h/W_h)$ windows, requiring $(I_w - W_w + 1) * W_w$ XNOR, $(I_w - W_w + 1)$ mem-THR, and I_h/W_h sa-THR cycles. Rest of the outputs are generated by sliding over rows.

Sliding over rows: Sliding over rows is equivalent to moving the windows vertically. The process of sliding over columns (❶-❻) is carried for each of the W_w columns of weight matrix (❼-⓬). This results in sliding over rows while sliding over columns for each row. In total, sliding over rows is performed for W_h weight rows. Each of these slides includes sliding over columns.

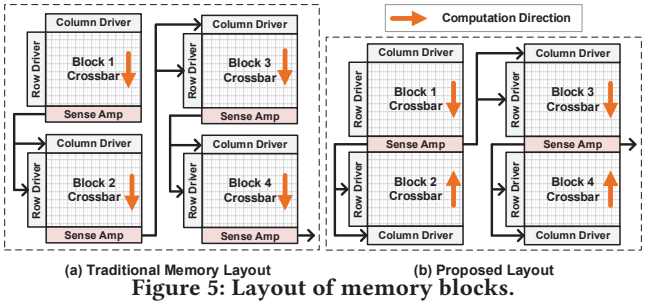

Figure 5: Layout of memory blocks.

4.3 Pooling

BitNAP supports the most popular pooling layers in memory, namely MIN/MAX/average. The MIN/MAX pooling in BNNs is implemented using OR/AND operation [5]. BitNAP supports these bitwise operations, as explained in Section 3.2. THR performs the average pooling with a threshold of 50%. Application of OR/AND/THR on 2D pooling windows happen in the same way as THR (mem-THR + sa-THR) in convolution. In pooling, OR/AND/mem-THR (for MIN/MAX/average pooling) are first applied similar to mem-THR in a convolution layer, reducing windows to vectors. Then, sa-THR is applied over these vectors.

4.4 Block Layout & SA Sharing

BitNAP has a large within-memory compute to SA access ratio. For our implementation of ResNet-18 network, the latency due to SA is just 3.1% of the total latency of the network. Hence, to reduce the area overhead due to SAs, BitNAP shares SAs between blocks, as shown in Figure 5. To prevent the interference of currents from different blocks, we add isolation transistors (ISO1 and ISO2) between SAs and the corresponding bit-lines of the two blocks. At a time, either ISO1 or ISO2 is asserted to connect one block to the SA, leaving the other block isolated. This approach eliminates half the SAs from BitNAP. Although adding the isolation transistors increases the area overhead of each sa-THR enabled SA by 6.7%, the reduced number of SAs results in an effective SA area reduction of 46.7% as compared to the BitNAP with no SA sharing.

4.5 Pipelining SA Operations

The SA sharing technique proposed in Section 4.4 reduces the total area overhead of adding processing capabilities to the sense amplifiers. However, this comes at the cost of sequential reads/sa-THR from the blocks sharing the SA. Figure 6a shows that the sharing of SAs serializes the sa-THR and data-write stages of the two blocks, say $block_a$ and $block_b$. This doubles time taken by the two stages as compared to BitNAP with no SA sharing. Since data-write doesn't require SAs it can happen in parallel for the two blocks. This is illustrated in Figure 6b as *Pipeline* 1. Also, a data write can happen in parallel with a sa-THR operation, shown as *Pipeline* 2. A quick analysis of *Pipeline* 2 shows that it isn't suitable as it results in multiple stalls due to the delayed availability of input data to odd (upper) blocks. For *Pipeline* 1, the corresponding implementation for three block pairs in parallel is shown in Figure 6c. It reduces the number of steps by 25% as compared to the sequential design, requiring 1.5× the steps taken with no SA sharing. Every data generated in odd blocks as a result of sa-THR has an idle cycle before being written to the next block, requiring extra registers to store them.

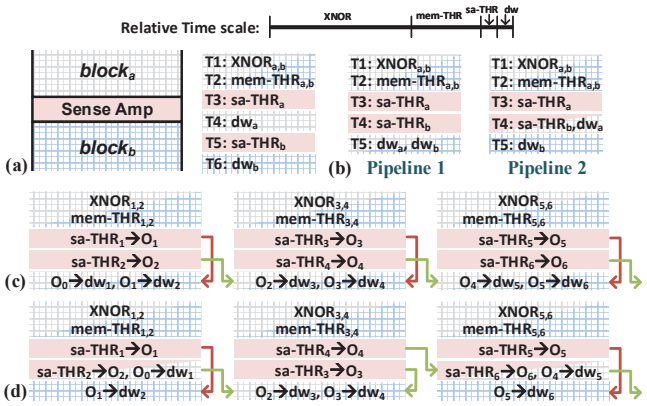

Figure 6: (a) Sequence with SA sharing, (b) pipeline schemes, (c) BitNAP with *pipeline*1, (d) BitNAP with hybrid pipeline.

Table 1: Energy for Operations in BitNAP

SET	$23.8fJ$	**sa-THR**	$0.5pJ$	**mem-THR-4**	$41.64fJ$
RESET	$0.32fJ$	**mem-THR-2**	$24.11fJ$	**mem-THR-5**	$49.24fJ$
XNOR	$34.97fJ$	**mem-THR-3**	$41.64fJ$	**mem-THR-6**	$49.24fJ$

*mem-THR-X detects the presence of X or more '1's

We propose a hybrid pipelining scheme by combining *Pipeline* 1 and *Pipeline* 2 as shown in Figure 6d. Here, an even block and the following odd block perform the same operation at the same time. When ($block_2$, $block_3$) perform sa-THR, ($block_4$, $block_5$) perform data-write and vice-versa. Also, the block pairs implement *Pipeline* 1 and *Pipeline* 2 alternatively. The data generated by alternate odd-blocks is written to the next stage in the following cycle, requiring only 50% of the registers used in case of *Pipeline* 1.

5 EVALUATION

5.1 Experimental Setup

We use an in-house cycle-accurate C++ simulator which emulates BitNAP functionality. We used HSPICE for circuit-level simulations and calculate energy consumption and performance of all the BitNAP operations in 45nm process node. The robustness of all proposed circuits has been verified by considering 10% process variations on the size and threshold voltage of transistors using 5000 Monte Carlo simulations. A maximum of 25.6% reduction in resistance noise margin was observed for RRAM devices. However, this did not affect BitNAP operations due to a high R_{OFF}/R_{ON} of $10M/10k\Omega$. Here, we adopt RRAM device with VTEAM model [24]. The device parameters are chosen to fit practical devices [21, 25] with switching delay of 1.1ns (= cycle time in BitNAP). The experiments were run with BitNAP chip size of 4.79 mm^2, containing 60 MB of RRAM memory, which is enough to run the largest of the four tested networks. Each block in BitNAP has 1024×1024 cells. We perform our experiment on popular MNIST and ImageNet datasets. For MNIST, we use LeNet [26]. For ImageNet, we exploit AlexNet, ResNet-18, and VGG-16.

5.2 Analysis of BitNAP Optimizations

Table 1 lists the energy of basic operations in BitNAP. Out of all these operations, XNOR takes 2 cycles (2.2ns) for execution, while the rest take just one cycle (1.1ns). The energy consumed by mem-THR increases with the number of inputs due to the finer precision required in execution voltage V_0. To visualize the application level

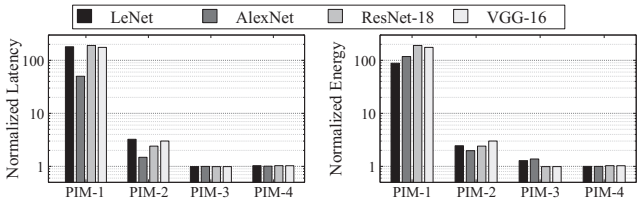

Figure 7: Latency and energy consumption for different configurations in Section 5.2 normalized to BitNAP.

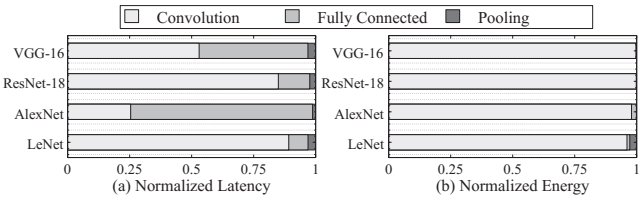

Figure 8: (a) Latency and (b) energy breakdown for different networks in BitNAP normalized to total network latency.

benefits of BitNAP from different optimizations, we compare it with different baseline PIMs. The first baseline, PIM-1, is BitNAP without sa-THR. PIM-2 is BitNAP without mem-THR. PIM-3 is BitNAP without SA sharing and no pipeline. PIM-4 is BitNAP with SA sharing but no SA pipelining. Figure 7 shows the execution time and energy consumption of BitNAP in these configurations. We observe that BitNAP derives major benefits from sa-THR. Over four networks, BitNAP is, on average 149× faster and 122.6× more energy efficient than the design without sa-THR. This happens because, in the absence of sa-THR, column accumulation requires a sequential addition of all 1024 rows in a block. BitNAP without mem-THR is 2.5× slower and consumes 1.9× more energy. Here, each row accumulation within a convolution window uses sequential addition. When compared with PIM-3, BitNAP is less than 1% slower and 39.2× more energy efficient. This is majorly due to the reduction in static energy consumption by using only half the number of sense amplifiers. Finally, comparing with PIM-4, BitNAP is 2.4% faster and consumes 0.2% more energy. Although pipelining reduces the delay of the sa-THR stages by 25%, the impact on overall application is not significant as bitwise computations dominate the latency.

5.3 Behavior of BitNAP for Different Layers

Figure 8 shows the impact of different types of layers of the tested networks on execution time and energy consumption while running on BitNAP. The energy is almost entirely consumed by convolution layers. For all the networks, most computation is performed in convolution layers, and hence, they impact the energy the most. On the other hand, for networks with large (more neurons) fully connected (FC) layers (both AlexNet and VGG-16 have FC layers with 4096, 4096, and 1000 neurons), the execution time is dominated by them. FC layers in AlexNet consume 73.3% of the total network latency. Since VGG-16 is a deeper network, with more convolution layers, the time consumed by FC layers reduces to 43.8% of the total time. While in LeNet, only 7.8% of the total time is spent in executing FC layers because the time required for FC layers in BitNAP is a direct function of the number of neurons in the layer. This shows BitNAP's high parallelism for convolution layers.

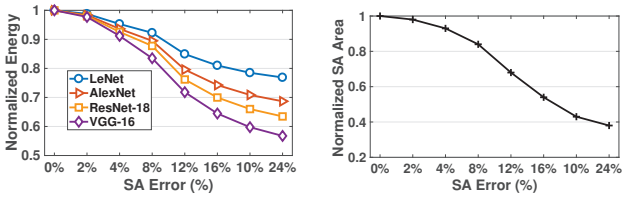

Figure 9: Effect of SA error on energy consumed and SA area.

5.4 BitNAP Tolerance to SA Noise

All PIM operations are accurate, but in the case of sa-THR, computations may be affected by the noise in either the bitline current or the sense amplifier (SA) circuit itself. SA with high tolerance to noise requires bigger transistors. This results in higher energy as well as area requirement. Instead, we use comparatively smaller transistors to reduce the total energy and area of the chip, resulting in SA with less tolerance to noise. We observe that sa-THR in BitNAP tolerates the loss in SA precision. Table 2 shows the change in the accuracy of BNN applications running on BitNAP with different SA errors. Here, the baseline SA results in just 1% error. Whereas, the SA used in BitNAP has a maximum error of 20%. We observe that BitNAP SA reduces the average ideal accuracy of classification by less than 2%. Figure 9 shows the change in energy of networks and area of SA with an error. Moreover, BitNAP over four networks is 31.2× more energy-efficient and takes 57% less area than baseline SA.

5.5 BitNAP vs Prior Designs

We compare BitNAP with state-of-the-art in-memory BNN accelerators. We observe that for AlexNet (LeNet) in BitNAP is 10.7× (1.32×) faster than XNOR-POP [8], which implements BNNs in DRAMs, because XNOR-POP has serial popcount operations. As compared to XNOR-POP, BitNAP is 188× more energy efficient for LeNet because all operations in XNOR-POP are implemented in sense amplifiers. However, for complex networks, the improvement is only 8.5× on average over AlexNet and ResNet-18. As compared to ReBNN [27], BitNAP is 1.16× faster for LeNet. However, ReBNN uses two memory cells per data bit, which incur significant overhead while writing data to memory. Due to the limited data from previous work, we only compare our energy consumption with IMC [28] and BCNN [9]. BitNAP is 12.4× and 1.28× more energy-efficient than IMC for LeNet and AlexNet, respectively. IMC uses SOT-MRAM devices for bitwise AND operations with a bitcount circuit for accumulation. However, the final output of convolution in IMC is approximated by CPU using the partial results obtained from memory, leading to lower accuracy. As compared to BCNN, BitNAP is 356.6× and 21.7× more energy efficient for LeNet and AlexNet, respectively. BCNN uses DACs and ADCs, which results in really high energy consumption. In the end, we also compare BitNAP with highly optimized in-memory binary vector-matrix multipliers, XNOR-RRAM [20] and XIMA [15]. BitNAP is 177× faster and 195× more energy efficient as compared to XIMA. XIMA uses sense amplifiers for all the operations, making it sequential with high energy consumption. BitNAP is 11.35× slower but 21× more energy efficient than parallel XNOR-RRAM for vector-matrix multiplication. XNOR-RRAM enables parallel computation across all the memory cells in a block (crossbar) while BitNAP parallelizes one row of a block at a time. However, XNOR-RRAM uses peripherals that act as scaling and latency bottleneck for large networks.

Table 2: Effect of SA error rate on the classification accuracy.

SA Error	*1%*	*2%*	*4%*	*8%*	*12%*	*16%*	*20%*	*24%*
LeNet	97.1	97.1	97.1	96.56	96	95.4	94.5	93.3
AlexNet	69.2	69.2	69.2	69	68.7	67.9	67.5	65.7
ResNet-18	71.9	71.9	71.8	71.3	71.0	70.4	69.8	69.0
VGG-16	89.4	89.4	89.4	89.25	89.04	88.3	88.0	87.1

6 CONCLUSION

BitNAP presents a novel efficient BNN architecture to take advantage of the efficient row-parallel PIM operations. BitNAP performs optimization at operation, peripheral, and architecture level to accelerate BNNs in a crossbar memory. In order to reduce the area overhead of sensing based operations, BitNAP presents a memory sense amplifier sharing scheme and also, a novel operation pipelining to reduce the latency overhead of sharing.

ACKNOWLEDGMENTS

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration grant, and also NSF grants # 1527034, #1730158, #1826967, and #1911095.

REFERENCES

[1] C. Dong *et al.*, "Learning a deep convolutional network for image super-resolution," in *ECCV*, pp. 184–199, Springer, 2014.
[2] L. Deng *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
[3] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[5] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, pp. 525–542, Springer, 2016.
[6] S. Gupta, M. Imani, H. Kaur, and T. S. Rosing, "Nnpim: A processing in-memory architecture for neural network acceleration," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1325–1337, 2019.
[7] M. Imani, M. S. Razlighi, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Deep learning acceleration with neuron-to-memory transformation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020.
[8] L. Jiang *et al.*, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *ISLPED*, pp. 1–6, IEEE, 2017.
[9] T. Tang *et al.*, "Binary convolutional neural network on rram," in *ASP-DAC*, IEEE, 2017.
[10] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, pp. 14–26, IEEE Press, 2016.
[11] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, pp. 27–39, IEEE Press, 2016.
[12] S. Gupta, M. Imani, and T. Rosing, "Exploring processing in-memory for different technologies," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 201–206, 2019.
[13] I. Hubara *et al.*, "Binarized neural networks," in *NIPS*, 2016.
[14] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv:1602.02830*, 2016.
[15] L. Ni *et al.*, "An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary rram crossbar," in *DASP-DAC*, pp. 280–285, IEEE, 2016.
[16] L. Ni *et al.*, "Distributed in-memory computing on binary rram crossbar," *JETC*, 2017.
[17] D. Kim *et al.*, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *ISCA*, IEEE, 2016.
[18] Y. Li *et al.*, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," in *FPGA*, pp. 290–291, 2017.
[19] R. Andri *et al.*, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights.," in *ISVLSI*, pp. 236–241, 2016.
[20] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *DATE*, IEEE, 2018.
[21] S. Kvatinsky *et al.*, "Magic memristor aided logic," *TCAS II*, vol. 61, no. 11, pp. 895–899, 2014.
[22] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.
[23] S. Gupta, M. Imani, J. Sim, A. Huang, F. Wu, M. H. Najafi, and T. Rosing, "Scrimp: A general stochastic computing architecture using reram in-memory processing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020.
[24] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *TCAS II*, 2015.
[25] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 802–815, 2019.
[26] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, p. 20, 2015.
[27] L. Song *et al.*, "Rebnn: in-situ acceleration of binarized neural networks in reram using complementary resistive cell," *CCF Transactions on HPC*, 2019.
[28] S. Angizi and D. Fan, "Imc: energy-efficient in-memory convolver for accelerating binarized deep neural network," in *Proceedings of the Neuromorphic Computing Symposium*, ACM, 2017.