Journal of
Low Power Electronics and Applications

MDPI

Article

MB-CNN: Memristive Binary Convolutional Neural Networks for Embedded Mobile Devices

Arjun Pal Chowdhury [1,*,†], Pranav Kulkarni [2,†] and Mahdi Nazm Bojnordi [1]

[1] School of Computing, University of Utah, Salt Lake City, UT 84112, USA; bojnordi@cs.utah.edu
[2] Electrical & Computer Engineering, University of Utah, Salt Lake City, UT 84112, USA; u1069755@utah.edu
* Correspondence: arjunmax@cs.utah.edu
† These authors contributed equally to this work.

Received: 1 May 2018; Accepted: 11 October 2018; Published: 13 October 2018

Abstract: Applications of neural networks have gained significant importance in embedded mobile devices and Internet of Things (IoT) nodes. In particular, convolutional neural networks have emerged as one of the most powerful techniques in computer vision, speech recognition, and AI applications that can improve the mobile user experience. However, satisfying all power and performance requirements of such low power devices is a significant challenge. Recent work has shown that binarizing a neural network can significantly improve the memory requirements of mobile devices at the cost of minor loss in accuracy. This paper proposes MB-CNN, a memristive accelerator for binary convolutional neural networks that perform XNOR convolution in-situ novel 2R memristive data blocks to improve power, performance, and memory requirements of embedded mobile devices. The proposed accelerator achieves at least 13.26×, 5.91×, and 3.18× improvements in the system energy efficiency (computed by energy × delay) over the state-of-the-art software, GPU, and PIM architectures, respectively. The solution architecture which integrates CPU, GPU and MB-CNN outperforms every other configuration in terms of system energy and execution time.

Keywords: convolutional neural networks; binary convolutions; in-situ processing; RRAM technology; computer architecture; embedded systems

1. Introduction

Sensor equipped Internet of Things (IoT) devices are expected to impact the future of consumer electronics. According to a recent prediction of ABI research [1] and IDC forecast [2], by the end of this decade, approximately 480 million IoT wearable devices will be sold.Detecting different human behaviors and ambient contexts and producing appropriate reaction are the core application of any mobile IoT device. Deep learning has emerged as one of the key techniques to enable many IoT mobile applications—extracting sensor data, identifying meaningful context, and performing intelligent tasks such as face detection [3], image classification [4], and speech recognition [5,6]. However, the increased demand of computation, memory, and energy consumption creates serious challenges to its applicability in IoT mobile devices. Recent studies have shown that the memory requirements of neural networks can be reduced by applying various compression and quantization techniques [7,8]. It has been observed that full precision value of weights or inputs is not required to get state-of-the-art accuracy from various deep neural networks. For example, binary neural network [9] and XNOR-Net [10] replace the power consuming floating point multiplications with bitwise XNOR operations. As shown in Figure 1, one bit quantization helps to achieve significant performance improvements for state-of-the-art neural networks. In addition, energy consumption and memory footprint are significantly reduced. However, simulation results indicate that 66% of the total execution

time in Alex XNOR-Net inference is dedicated to XNOR convolution. The number further increases for a bigger network such as VGG16-net.

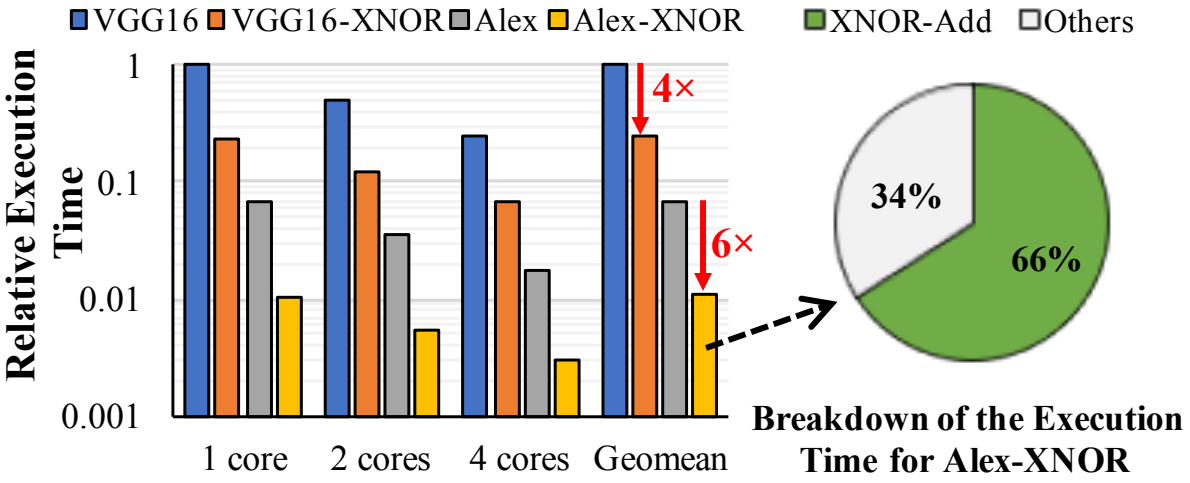

Figure 1. Impact of XNOR convolution in state-of-the-art neural networks.

This paper proposes a memory-centric hardware accelerator based on RRAM technology. Because of its high density and low read latency, all the parameters of a huge neural network can be stored without impacting the overall area or performance of the device. The accelerator exploits the bit-line and word-line architecture of a conventional memory cell to realize highly parallel in-situ computation. The in-memory computation eliminates the need for huge data movement between memory arrays and a computational unit. The integration of MB-CNN accelerator with a single core CPU device provides 4.14× improvement in performance and 4× improvement in energy over software based single core CPU solution. As compared with GPU based and PIM-like accelerators, the proposed accelerator achieves at least 5.91× and 3.18× improvements in the system energy-delay-product, respectively.

2. Background

This section provides the necessary background on the binary convolutional neural network (B-CNN), IoT applications, and resistive memory technology.

2.1. Convolutional Neural Networks

A convolutional neural network (CNN) is a deep learning architecture that has proven successful in image classification and recognition [4,11]. The CNN architecture comprises three main types of layers, namely *convolution*, *pooling*, and *classification*. Unlike a regular neural network, the layers of a CNN have neurons arranged in three dimensions, which are depth, width, and height. The initial stage of every CNN architecture is dedicated to feature extraction using multiple convolutional layers. Next, the dimensions of the extracted feature maps are reduced by retaining only the salient features of the image using pooling layers. The last stage performs image classification using fully connected layers—also known as classifiers. The convolutional layer comprises a set of three-dimensional filters, each of which converts a three-dimensional input feature map or image data into a two-dimensional feature map used by the next layer. Every filter realizes a small kernel applied to the input volume to compute the dot product between the filter weights and the data. The kernel is slid across width and height dimensions of the data to cover all of the features or image data.

A typical deep convolutional neural network may require millions of parameters to be learned and stored during the training pass and to be retrieved and used for inference. In addition to the stringent storage requirements, accessing these parameters by the convolutional layers necessitates consuming significant amounts of energy and time (more than 90%). For example, Alex-Net [4]

requires 61 million parameters to be processed by 1.5 billion floating point operations for classifying a single image. Larger networks, such as VGG-Net [12] and Deepface [3], require a significant number of parameters and operations to complete a classification task. Table 1 shows the amounts of memory consumed for maintaining the parameters in four widely known CNN architectures. Both the input data and the parameters of a typical CNN are real numbers that may be represented using 16-bit single precision fixed point. Interestingly, high-precision parameters are not important to achieve high accuracy in the outcome of a neural network; as a result, numerous optimization techniques have been proposed in the literature focusing on trading the precision of computations for gaining energy-efficiency and performance [8,13–16].

Table 1. Example CNN architectures.

	Alex-Net	VGG-16	Goggle-Net	Deepface	Res-Net 18
Memory Size	240 MB	560 MB	16 MB	480 MB	50 MB

2.2. Binary CNN: XNOR Network

Binary CNN is an active research topic mainly because of its advantage in performance and energy efficiency for embedded low power applications. In a binary CNN, both the inputs and weights are binarized. Prior work have shown comparable accuracy of binary network with respect to full precision networks [17]. This section describes XNOR-Net [10], a binary neural network that can achieve high accuracy on large datasets—such as ImageNet [18]. Figure 2 illustrates the difference between a typical CNN and XNOR-Net. In XNOR-Net training phase, binary weights are used during forward pass and backward propagation. Full precision weights are used only for the gradient calculation. The parameters and learning rates are updated based on the stochastic gradient descent (SGD) update with momentum or ADAM algorithm [19].

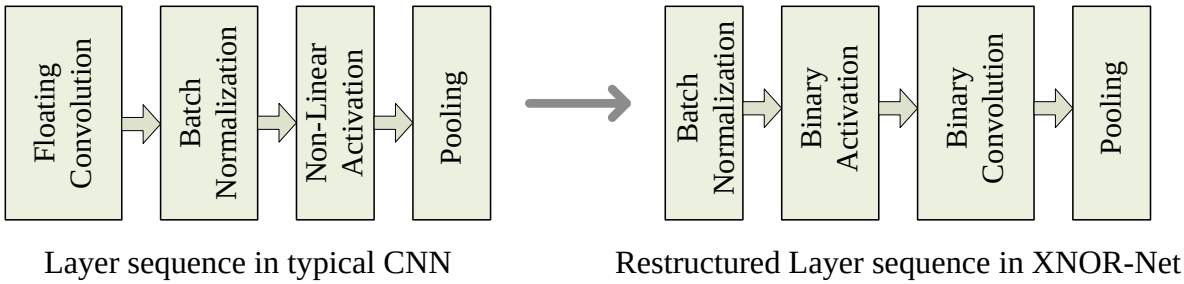

Figure 2. Illustrative examples of a typical CNN and an XNOR-Net architecture.

2.2.1. Deterministic Binarization

XNOR-Net relies on binarizing the input activation and weights by converting them into either $+1$ or -1 using a sign function defined as

$$x^b = sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \quad (1)$$

where x is a real-valued weight or activation and x^b is the binarized output. To directly use logical operations for computing XNOR-Net, all -1 s are encoded to 0 s.

2.2.2. Parameter Scaling

The binary values generated by the binarization layer are then scaled to better approximate the real-valued weights and to improve the accuracy of results. For example, a real-valued filter ($W \in \omega$) is approximated by $W \approx \alpha B$, where α is a weight scaling factor and B is an instance binary filter from

$\{+1, 0\}^{(c \times w \times h)}$. A consolidated scaling factor matrix (**K**) is generated for all of the input neurons in binary activation layer and is used to approximated the convolution between input (**I**) and weights (**W**) using the following binary equation

$$\mathbf{I} * \mathbf{W} \approx (sign(\mathbf{I}) \circledast sign(\mathbf{W})) \odot \alpha \mathbf{K} \quad (2)$$

where $*$ indicates the real-valued convolution, $\circledast$ is the binary convolution using bitwise XNOR and addition (bit-count), and $\odot$ represents the Hadamard product of two binary matrices. Notice that the outcome of every bit-count operation can be a multi-bit value, which is passed through a threshold comparison function to ensure producing binary output for convolution. The resultant binary matrix is then multiplied by $\alpha \mathbf{K}$ that computes a real-valued scale matrix to produce the output neurons of a binary convolution layer.

2.2.3. XNOR Convolution

Figure 3 depicts the two steps of an example XNOR convolution applied to an input data ($X^{c \times h \times w = 3 \times 3 \times 3}$). In the first step, $X^{3 \times 3 \times 3}$ is convolved with the filter $F^{3 \times 2 \times 2}$ using element-wise XNOR operations followed by a sum operation to compute the intermediate output $Y^{1 \times 2 \times 2}$. In the second step, the intermediate values are summed and the result is compared with $N/2$ to produce a single element of the output matrix. (N represents the total number of elements in the filter used for this particular layer.) Notice that each filter is stridden over the entire input image to produce a single output channel ($Z^{2 \times 2}$); as a result, an n-channel output can be produced using n filters convolved with the same input ($X^{3 \times 3 \times 3}$).

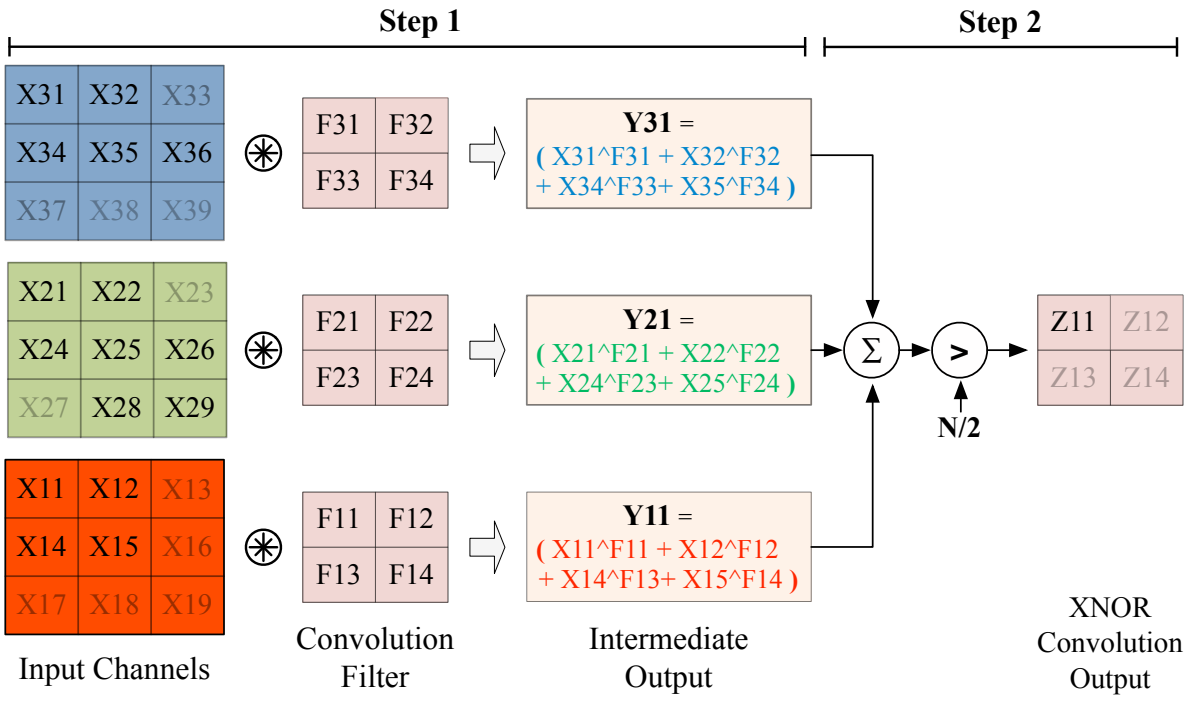

Figure 3. XNOR convolution.

2.3. Evolution in BCNN Algorithm

Hubara et al. [9] showed that it is possible to achieve state-of-the-art accuracy for various datasets—e.g., MNIST [20], CIFAR-10 [21], and SVHN [22]—using binarized parameters and activation. However, recent work by Rastegari et al. [10] and Tang et al. [17] report poor accuracies when applying BNN directly to large datasets such as ImageNet. Instead, they proposed XNOR-Net [10] by restructuring the convolutional layers and introducing weights and input scaling factors to achieve

comparable accuracy with Alex-Net on large datasets. A most recent work by Tang et al. [17] proposes new activation, normalization, and scaling methods that reduce the computation workload and provides better accuracy. This highlights that the algorithmic space of binary convolutional neural networks is still evolving. Therefore, a full fledged end to end inference ASIC accelerator for binary neural network may not be a viable solution in current scenario. However, XNOR Convolution is a primitive for most binary CNNs. This paper provides a complete solution where the proposed accelerator performs in-situ XNOR convolution within RRAM based non-volatile memory of embedded IoT devices and the rest of the operations—e.g., input normalization and weight scaling—are performed in software either by CPU or a GPU acceleration. By leaving these components in software, the proposed approach allows the algorithm to evolve.

2.4. IoT Applications

IoT devices are optimized for energy-efficiency. The stringent ultra low power requirements and energy efficiency problems of the existing IoT platforms render the realization of a full-fledged deep learning solution in IoT nodes largely impractical. Innovations at both hardware and software levels are required to alleviate these problems for future IoT systems. Figure 4 shows the generic system architecture of an IoT device (or edge node). Depending on the application objectives and the design constraints, a single- or multi-core processor is employed for executing programs. Typically, the memory system consists of both volatile and non-volatile modules to address the needs of a wide range of software applications. L1 and L2 caches as well as scratch-pad memory units may be used mainly for temporary data storage. Non-volatile memory is commonly used to permanently store data and software programs. Currently, FLASH is the widely used technology for wearable and mobile devices [23]. However, emerging non-volatile memory technologies, such as FeRAM [24,25] and RRAM [26–28], are expected to replace conventional memories [29,30].

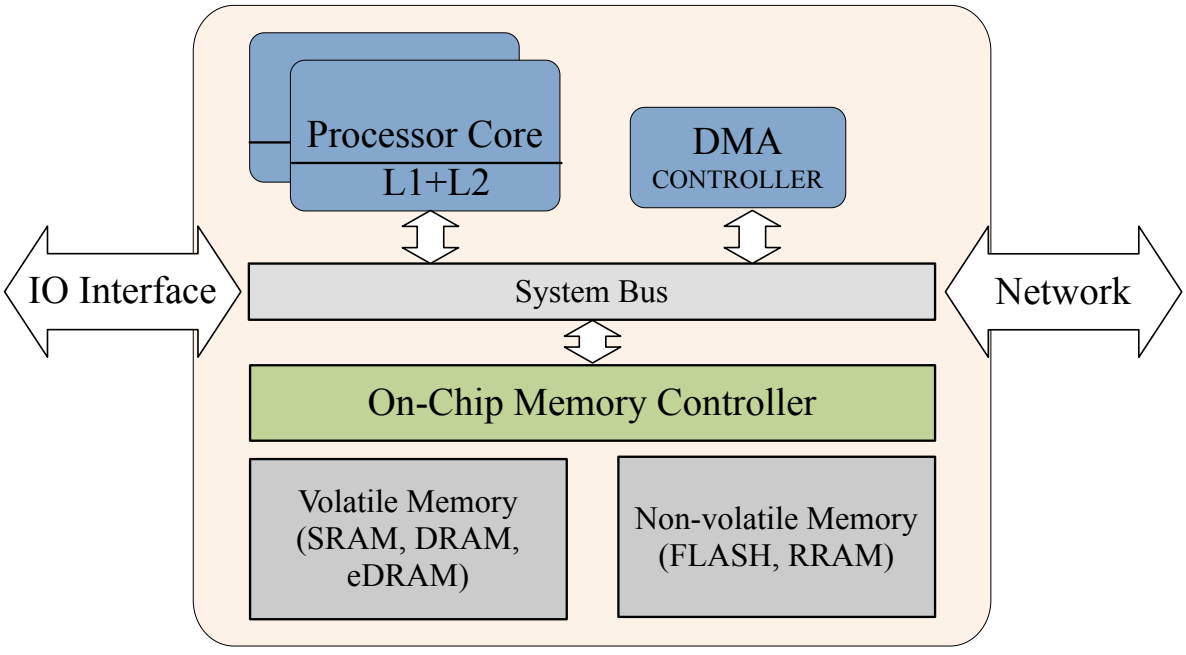

Figure 4. Illustrative example of a mobile IoT system.

2.5. Memristive Crosspoint Arrays

Resistive technologies have shown excellent characteristics for the future memory systems capable of storing large amounts of data and performing in-memory computation [31–33]. Resistive RAM (RRAM) is one of the most promising memristive devices currently under commercial development

that exhibits excellent scalability, high-speed switching, a high dynamic resistance range that permits multi-level cells (MLC), and low power consumption [34]. Figure 5a depicts an example RRAM crosspoint used for storing a two-dimensional data array. In this dense array organization, the memory elements are connected directly to the wordlines and bitlines. Accessing each cell requires activating the corresponding wordline, after which the data bits are read or written through the bitlines. A row decoder is used to activate the line drivers during every memory access. For example, a row of the data array is selected by applying a read voltage to a horizontal wordline. The contents of the resistive cells within the selected row are read using vertical bitlines and a set of sense amplifiers (SA). Similarly, by applying a write voltage with the appropriate polarity and magnitude across the bitline and the wordline, the cell can be switched into a desired resistance state: high or low.

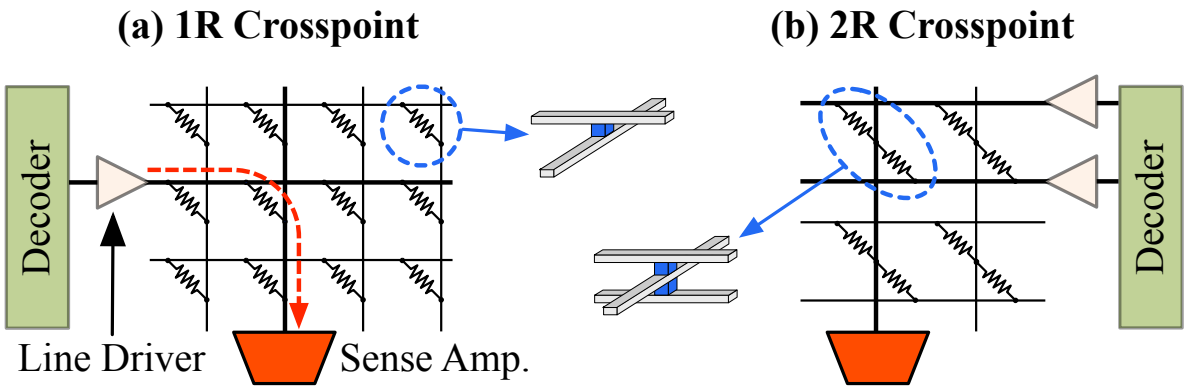

Figure 5. Illustrative examples of 1R (**a**) and 2R (**b**) crosspoint arrays.

Unlike the conventional 1T-1R array, the 1R crosspoint does not require access transistors per memory cells, which makes it possible to realize data arrays by placing memristive cells between every two metal layers for the bitlines and wordlines [35]. Therefore, multiple crosspoint layers can be stacked to build a dense, three-dimensional (3D) memory structure within a limited chip area. However, the absence of access transistors creates a set of significant challenges in designing large crosspoint arrays, such as half selected cells per access [36] and sneak current [37]. Numerous solutions have been proposed in the literature, of which the 2R crosspoint [38] significantly alleviates the problem and is suitable for ultra low power systems such as mobile IoT devices. As shown in Figure 5b, every bit and its complement are stored in a pair of memristive cells that are sandwiched between two wordline layers and one bitline layer. This paper employs the 2R crosspoint structure to realize a dense memory capable of storing data and computing in-situ XNOR convolution.

3. Memristive Binary Convolution

Designing a memory-centric accelerator for deep learning workloads in mobile IoT devices is a significant challenge. On the one hand, hardware design is significantly constrained by the stringent cost and power requirements. On the other hand, software applications demand for more computational capabilities. The proposed memristive framework addresses this problem by enabling in-situ binary convolution in the conventional 2R crosspoint arrays for accelerating the binary convolutional neural networks in low power computer systems. The key idea is to exploit the computational capabilities in 2R crosspoint arrays to realize the XNOR and bit-count operations. Using a hierarchical organization, the proposed memristive framework is capable of serving reads and writes, as well as performing XNOR convolution for the deep learning workloads.

3.1. System Overview

Figure 6 shows how the proposed memristive hardware is employed to accelerate B-CNN workloads on mobile IoT platforms. The computation platform includes an application processor, a volatile memory module (e.g., DRAM), and an RRAM-based non-volatile memory subsystem comprising crosspoint data arrays for storing data permanently. The proposed RRAM module is capable of performing the XNOR convolution—as explained in Section 2.2.3—with significantly better performance and lower energy. The accelerator module is interfaced with the system processor using an LPDDR3 standard bus [39]. Therefore, all of the accelerator-specific commands (e.g., start computation) are converted to valid LPDDR3 transactions. All of the relevant parameters (weights) of a trained B-CNN model are first written into the RRAM arrays, which can be later used for inference tasks many times (the accelerator module is capable of maintaining the parameters of multiple layers). Next, the software algorithm initiates an inference process by reading the input data and computing the necessary values for the next XNOR convolutional layer. Prior to performing an XNOR convolution, all of the values followed by a start command are transferred to the accelerator. Software collects the results once they are ready and continues executing the rest of the algorithm.

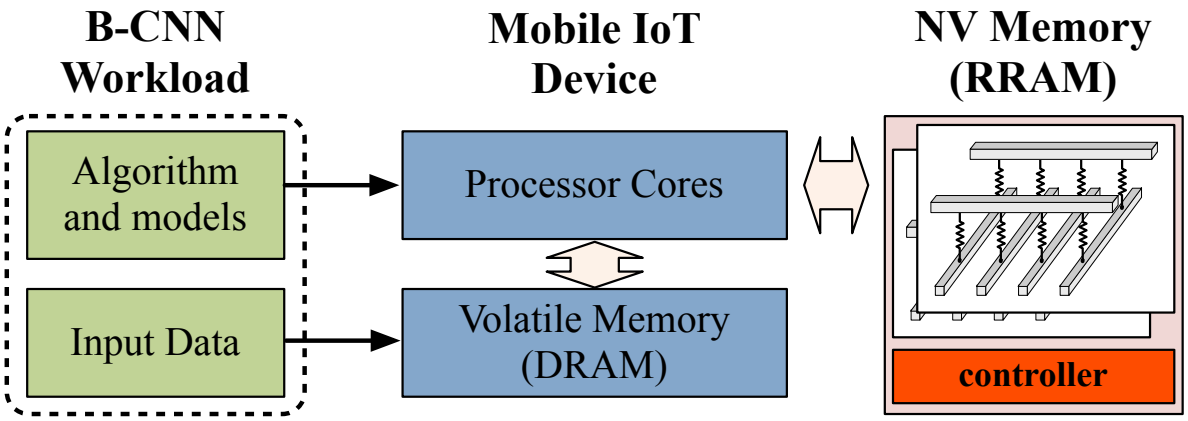

Figure 6. Accelerating B-CNN workloads with the proposed memristive binary convolutional architecture. The RRAM crosspoint arrays are used for storing data and performing in-situ XNOR convolution.

3.2. XNOR Convolution within 2R Crosspoint

Every XNOR convolution consists of binary XNOR operations between the elements of a filter and input channels followed by bit-counting and binary approximation to compute the results.

3.2.1. Memristive XNOR Operation

The proposed accelerator exploits the computational capabilities of the 2R crosspoint to perform all three operations within memory arrays. Figure 7 shows how the differential form of a bit stored in a 2R cell can help performing in-situ XNOR operation. The true and complement forms of every filter element are stored in a cell as b and $\overline{b}$, respectively, where logic 1 is represented by the low resistance state (LRS) of the RRAM cell, and the high resistance state (HRS) is used for 0. Notice that the filter weights are determined during the training phase and remain fixed for every B-CNN model. Therefore, the proposed in-situ XNOR convolution does not impose additional switching overheads such as energy, delay, and wear-out problems. Similarly, an input element is represented in differential form and is applied to the cell through two wordlines w and $\overline{w}$. The cell structure forms a simple resistive network developing an output voltage (out). Assuming that 1 and 0 are represented by the high voltage and ground, respectively, out indicates the logical XNOR between w and b.

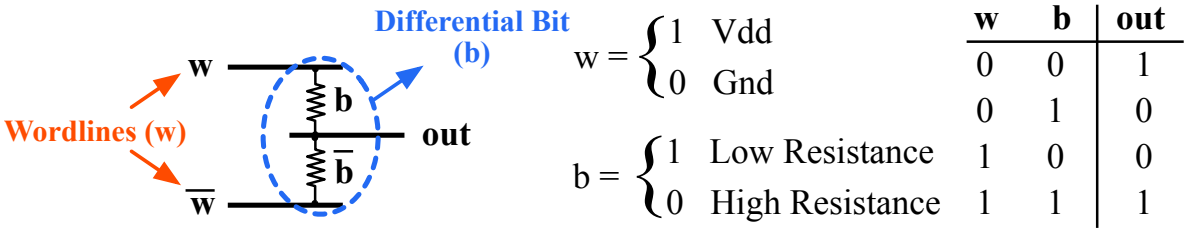

Figure 7. Performing XNOR operation using the 2R crosspoint cell.

3.2.2. Analog Bit-Count Operation

In a crosspoint array, every group of memory cells within a column are connected to a shared bitline. Figure 8 shows how the 2R memory cells can contribute in the bit-count operation. Every cell computes the binary XNOR between the corresponding elements of the filter and the input data ($\overline{w_i \oplus b_i}$). All of the XNOR results are summed up along the bitline to produce an output voltage (sum). Notice that the binary outcome of every XNOR operation is a high or low voltage indicating 1 or 0. Regardless of the operand's values, the high voltage (1) is produced for an XNOR operation only if Vdd is connected to the RRAM cell with low resistance; otherwise, a low voltage (0) appears at the output. For a bitline connected to n 2R cells, assume that m XNOR operations produce 1s in their outputs. The resultant bitline voltage can be computed by $sum = V_{dd}\frac{mH+(n-m)L}{n(H+L)}$; where, H and L are the amount of high and low resistances of the RRAM cells. The bitline voltage is linearly proportional to the number of 1s produced by the XNOR operations—i.e., the bit-count. By quantizing this voltage, the final binary value for a single convolution can be computed. For example, a comparator can be used to output a 1 if $sum \geq V_{dd}/2$ ($= m \geq n/2$) and 0 otherwise.

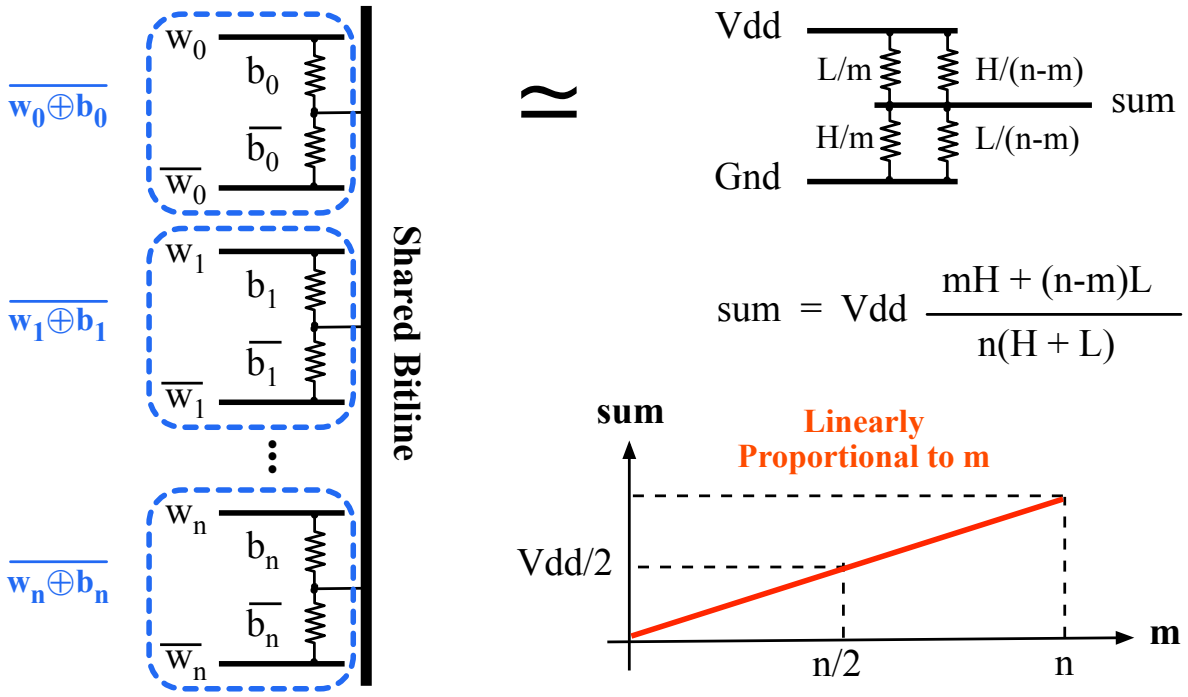

Figure 8. Performing bit-count operation within the 2R crosspoint array.

3.2.3. Hierarchical Bit-Counting

The proposed in-situ bit-counting requires having all of the operands connected to a single bitline during an XNOR convolution. As a result, large crosspoint arrays are required to realize the bit-counting for modern deep learning workloads. Implementing such monolithic data arrays

is largely impractical due to limitations in sensing circuits, significant power dissipation, excessive delay overheads, and serious reliability issues. Instead, the proposed architecture employs a novel hierarchical mechanism that allows for computing partial bit-counts in multiple arrays and quantizing the aggregated sum into a single bit. This hierarchical mechanism requires converting the bitline voltages (*sum*) into digital values and computing the final sum by adding all those numbers. Unlike the conventional single-level sensing, the proposed mechanism employs an analog to digital converter (ADC) circuit to produce each partial bit-count. Detailed explanation on converting the bitline voltage into a digital value is provided in Section 4.3. The accuracy of this voltage quantization is heavily dependent on the line slope ($\delta = \frac{V_{dd}}{n} \times \frac{H-L}{H+L}$) in Figure 9. A large slope value is desirable for realizing more accurate and reliable sensing, which can be achieved by using higher V_{dd}, decreasing n, and increasing the resistive ratio of the memristive device ($\gamma = \frac{H-L}{H+L}$). The proposed accelerator uses the nominal V_{dd} specified for the underlying technology node; existing RRAM technologies are studied to find an appropriate memristive device for realizing the memory cells; and the design space of bitline sensing circuits and the arrays sizes is explored to find an appropriate value for n (see Section 5).

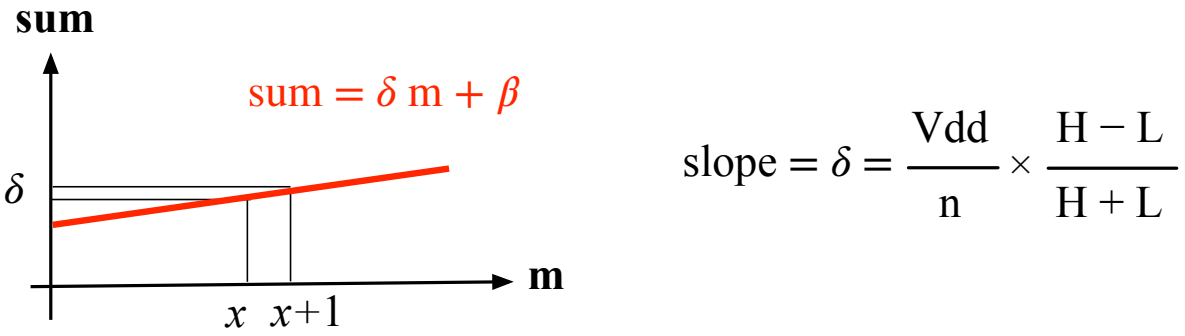

Figure 9. Impact of the technology parameters on the sensing mechanism.

4. The MB-CNN Architecture

The proposed hardware accelerator builds upon the existing memory system architectures. As shown in Figure 10, every accelerator chip comprises an external IO interface and a memory core with a chip controller and multiple banks. All of the data transfers among the IO interface and memory banks are managed by the chip controller. Every bank is capable of serving a memory request or performing the XNOR convolution, independently. Depending on the size of each layer, single or multiple banks may be involved in every XNOR convolution. Notice that the proposed hardware is used for inference tasks. Training is assumed to be carried out once in cloud and the resultant weights are deployed on mobile IoT devices.

4.1. On-Chip Control

Once the new edge weights are available, an MB-CNN chip can be reconfigured regarding the number and size of the convolutional layers. A single bank may be used to store the parameters of one or multiple small layers, while a large layer can occupy more than one bank. The chip controller includes local non-volatile RRAM arrays for tracking the banks that maintain the relevant parameters of each layer. Every B-CNN model includes multiple binary convolutional layers, each of which requires the software to make a call to the accelerator. First, the chip controller receives an initiate command specifying which layer is used next for computing the XNOR convolution; as a result, the relevant banks are configured accordingly. Then, the input data for the selected layer is streamed into the accelerator and is distributed among the relevant banks by the chip controller. Local buffers are used at the memory banks to collect the convolutional results. At the end of every computation, software is notified by the accelerator to read the results and process the subsequent layers of the B-CNN.

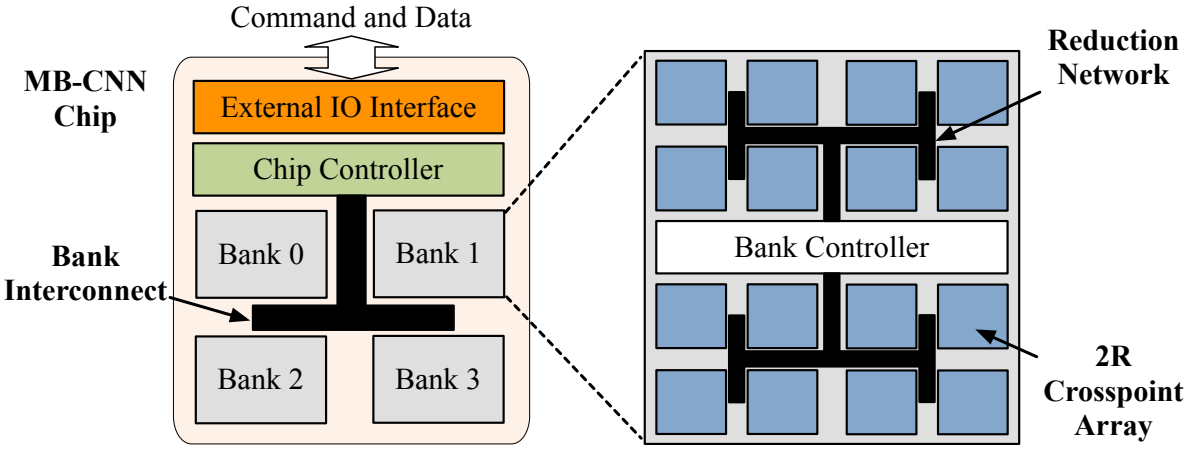

Figure 10. The proposed hierarchical organization for the accelerator.

4.2. Bank Organization

Every bank comprises a controller, an H-tree based reduction network, and a set of data arrays. The bank controller includes: (1) local memory for buffering the computed bit-count results; and (2) full adders and comparators for computing the final sum and quantizing the final result into a binary value. During an XNOR convolution, partial bit-counts are computed by the memory arrays and merged into a single bit-count when transferred over the reduction tree to the bank controller. Figure 11 shows how partial bit-counts are merged in the reduction tree. Every node employs a serial adder element to add two single-bit operands and store the carry bit locally. The serial addition allows for low cost and energy-efficient computation in the reduction tree. Similarly, at the bank controller, a serial adder is used to compute the difference between the final bit-count and the quantization threshold ($n/2$). The two's complement format of the threshold is used to compute the subtraction using the serial adder circuit, thereby realizing a serial comparator.The last bit to be computed by the serial comparator represents the sign in two's complement format that indicates whether the result is negative ($sum < n/2$) or positive ($sum \geq n/2$). The inverted version of this bit represents the binary result.

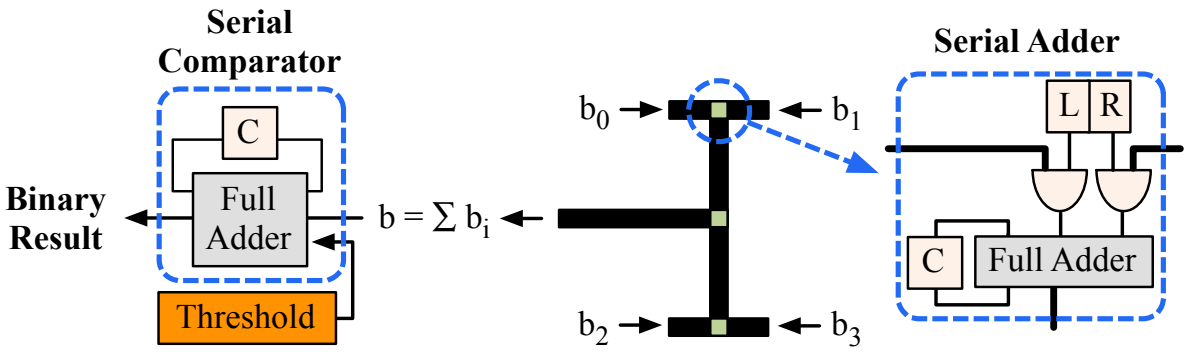

Figure 11. Merging bit-counts into a single result inside a bank.

Every bit serial adder at a node of the reduction tree is reconfigurable using two flip-flops R and L, each of which is used to masking a branch of the tree. Notice that the values of R and L determine whether the node performs a serial addition or copies the value of one branch to upstream root. Valid values for the R-L flip-flops are 1-1 for serial addition, 1-0 for transferring the right branch, and 0-1 for values of the left branch. After every XNOR convolution, the carry bit holder is reset. All of the R-L flip-flops are determined once for the convolutional layers and ate maintained by the chip controller. On computing an XNOR convolution, appropriate R-L bits are loaded into the reduction

tree. The latency and energy overheads for loading the configuration bit-stream and initializing the accelerator is accurately modeled in the evaluations.

4.3. Array Structure

Every memory array is implemented using an RRAM crosspoint with M rows and N columns (Figure 12). RRAM cells are programmed to represent the binary weights of a CNN layer. To enable in-situ XNOR convolution within the crosspoint arrays, a set of latches are provided at the periphery of every weight array to store the input data and apply them through the wordlines. Along the lines of prior proposals on using multi-bit sensing mechanisms for analog computation [33,40–42], a cost-efficient multi-bit sensing circuit is designed and used for quantizing the bit-line voltage (*sum*). The proposed sensing circuit comprises a differential amplifier [43], a sample and hold unit [44], and a digital to analog converter [45]. Due to the exponential increase in the complexity of this circuit with the number of output bits, its precision is limited to 5 bits. Therefore, every array is capable of computing the sum of 32 terms of every XNOR convolution.

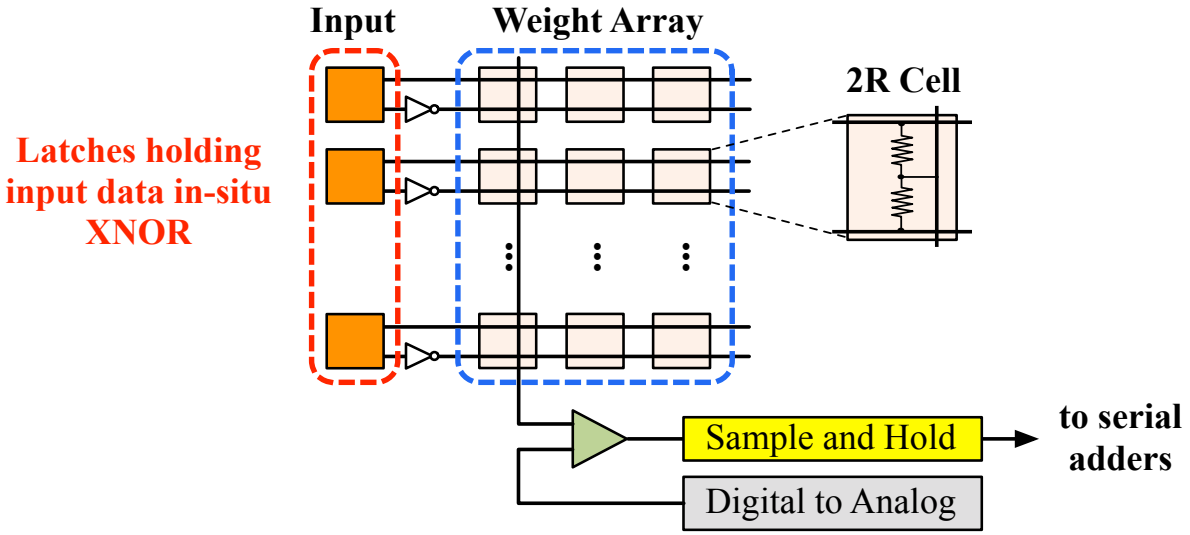

Figure 12. In-situ XNOR operations within the proposed memory array.

4.4. Data Organization

In this section, we describe how the inputs and weights of a convolution layer are mapped in the accelerator. Further, to simplify the illustration, an example mapping is shown in Figure 13, where the size of input feature map of convolution layer i is $I^{c\times h\times w} = 32 \times 7 \times 7$, where c denotes the input depth, while h and w are the height and width, respectively. The input is convolved with a kernel $K_0^{c\times h\times w} = 32 \times 2 \times 2$. If we have 128 such kernels then the convolution layer will produce 128 different output feature map of size $O^{h\times w} = 6 \times 6$. Let us assume a memristive crosspoint array of size 64×64; to map the entire convolution layer, we need four such crosspoint arrays namely a_0, a_1, a_2, a_3 in the bank $bank_0$. As shown in Figure 13, the kernel K_0 is distributed among the first column of all the arrays with a maximum of 32 elements per array. Similarly, all the other kernels are mapped into the entire bank. The kernel K_n is distributed among the n%64th columns of all four arrays, where n is the position of the bit line in the array.

For example, in Figure 13, K_0 has a total of 128 ($32 \times 2 \times 2$) elements $\{w0_0, w0_1 ... w0_{127}\} \in K_0$ where $w0_0 ... w0_{31}$ is mapped to n_0 of a_0. Similarly, $w0_{32} ... w0_{63}$ is mapped to n_0 of a_1 and so on. Again, K_{64} where $\{w64_0, w64_1 ... w64_{127}\} \in K_{64}$ is mapped to lower half of n_0 in all arrays. In a similar fashion, $K_1 ... K_{127}$ is mapped to $n_1 ... n_{63}$ of $a_0 ... a_3$. To convolve with these kernels, 128 elements of input I are fed to the bank by chip controller. On a XNOR convolution, the chip controller initiates streaming data to the crosspoint. The inputs are distributed among four arrays as shown in Figure 13. When particular

32 rows (cell segment) are driven by the input data, the other rows of the RRAM array remain inactive and do not contribute to in-situ computation.

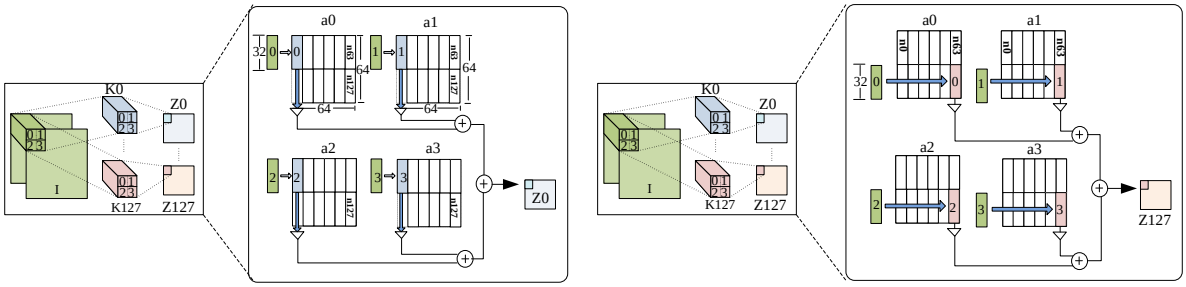

Figure 13. Distribution of a convolution layer to the crosspoint arrays: input feature map is convolved with kernel K0, and input feature map is convolved with kernel K127.

The five bit partial output of each array is given to a sequential adder tree present in H-tree reduction network to form the final output z_0 and $\{z_0, z_1 ... z_{35}\} \in Z_0$, where Z_n is output of each convolution between input I and kernel K_n. As 64 bit lines of each array are multiplexed to a single sense amplifier, every five cycles the multiplexer switches its input to the next bit line. The operation happens in the pipeline and in every seven cycles, the bank *bank*0 produces one output element of Z. After 7×128 cycles the first element of all output feature map Z_0 to Z_{127} is available at local output buffer.

Once the subset of input feature map is reused by all the filters for convolution, the chip controller feeds the next set of input to the arrays to convolve with the same kernel to produce next output element of all the output feature maps. In this way, it takes $7 \times 128 \times 6 \times 6 = 32{,}256$ cycles to produce the complete output of a convolution layer. The output of final adder is fed to a comparator, as shown in Figure 11, to quantize the convolution output to one bit. More simultaneous operations are possible by increasing the number of sense amplifiers in each array or by replicating the same kernel weights in multiple banks. However, that improved performance will cause more chip area and power consumption. In similar fashion, the FC layers are mapped to the proposed accelerator.

5. Experimental Setup

Simulations were conducted on Alex XNOR-Net [10] to assess the energy and performance potentials of the proposed accelerator. We used a bottom up approach for the power, performance and area evaluations all the way from circuit SPICE simulation of the 2R memory arrays, logic synthesis for the controlling circuits with the Cadence synthesis tool chain and 45 nm standard library cell [46] that consists of variety of cell types from flip flops and latches to multiplexers and bitwise logic gates, which are being used for simulation of MB-CNN. The results were then scaled to 22 nm [47]. NVSim [48] was used to estimate the area, delay, and power dissipation in the crosspoint arrays. We used CACTI IO [49] to assess the timing and power parameters for caches and memory interfaces. We used a modified version of the ESESC simulator [50] to evaluate the performance of single and multi-core processors for modern mobile IoT devices. Different images from Image Net dataset [51] were used as input dataset for evaluations. We used McPat [52] to estimate the core static and dynamic power consumption.

5.1. Neural Network Model

We chose Alex XNOR-Net [10] as the baseline architecture with single precision floating point tensors and weights. We trained the original XNOR-Net [10] in Torch7 [53] with image net dataset. According to prior work on XNOR-Net [10], we developed the required software kernels for Alex XNOR-Net. We changed the storage data structure for the outputs of binary activation layer such that the input matrices of binary convolution layers were organized in depth (channel) major instead

of row/column major. This technique helped us pack multiple 32 inputs into a single 32-bit value. The trained weights were also packed in the same fashion and stored in main memory prior to inference phase. With this bit packing mechanism, we employed SIMD behavior for data parallel XNOR operations in CPU cores. Figure 14 depicts the bit packing mechanism by arranging 3D input matrix in depth/channel major format. The binary operation was followed by a population count (*popcount*) and threshold comparison, as shown in Figure 3, to generate the output of XNOR convolution. The optimization showed a 19× performance improvement as compared with an unoptimized software where each binary operation is performed individually. We also developed an open-CL kernel for floating point convolution layers to run the application in a LP GPU based system.

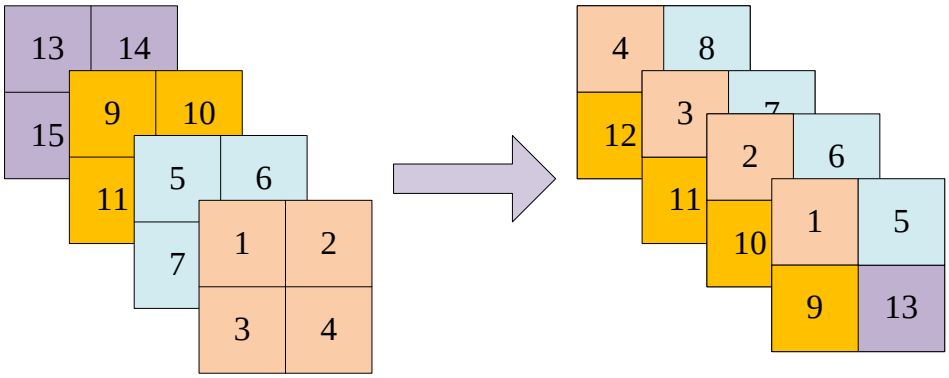

Figure 14. Input data representation.

5.2. Architecture

The proposed hardware accelerator can be integrated in both single- and multi-core mobile systems. We considered both systems for evaluations using the ESESC simulator [50] to model single- and multi-core out-of-order processors. All of the processor cores realized the MIPS64 ISA. We also considered GPU based and PIM-like ASIC accelerators for evaluating power and performance potentials of MB-CNN. For the GPU implementations, we considered the Nvidia Tegra X1 low power GPU platform with 256 cores [54]. The low power GPU was used to implement floating point convolutions in first and last layer of the end to end inference. We used Tegra X1 GPU for evaluation purposes only, which can be replaced by an efficient floating-point implementation run on CPU or any state-of-the-art low power GPU without incurring any performance impact. The GPU code was optimized to achieve high-performance similar to that of the industrial solution.

The PIM accelerator integrates additional gates for XNOR trees and pop-counts close to RRAM arrays to fetch data from arrays and compute XNOR convolution. The PIM accelerator we modeled to compare with our proposed accelerator comprises of memory subarray and hardware logic associated for XNOR convolution [55]. A hardware controller was employed to fetch operands from data arrays and compute the XNOR convolution. Similar to MB-CNN, the outcome of every XNOR convolution is transferred to software for generating the layer outputs.Unlike MB-CNN, the PIM baseline does not support in-situ RRAM arrays computation.The PIM-like hardware was optimized so that it occupies the same area as that of MB-CNN. Table 2 summarizes the key simulation parameters.

We optimized the PIM architecture to achieve high performance while consuming the same area as MB-CNN.

Table 2. Simulation parameters.

CPU	Single-core	Multi-core
Core Type	a2-issue OoO cores, 128 ROB entries 2.4 GHz	multiple 2-issue OoO cores, 128 ROB entries, 2.4 GHz
IL1 cache (per core)	32 KB, 2-way, LRU, 64B block, hit/miss delay 1/1	
DL1 cache (per core)	32 KB, 4-way, LRU, 64B block, hit/miss delay 2/2	
L2 cache shared	1 MB, 8-way, LRU, 64B block, hit/miss delay 8/6, MESI protocol	
Temperature	350 K (77 °C)	
DRAM Organization	LPDDR3-800, FR-FCFS, channel/rank/bank: 1/2/8, 8KB row buffer	
DRAM Timing (cycles)	RAS: 36, BURST: 4, FAW: 40 RCD: 15, CL: 12, WL: 9, RP: 15, RC: 48, WTR: 8, RTP: 8, RRD: 8	
GPU	Nvidia Tegra X1 low power GPU (256 cores)	
MB-CNN	Chip/Banks/Arrays: 1/128/2048, 1 GB, LPDDR3-800, Read Latency: 4.4 ns, Write Latency: 100 ns, Compute Latency: 8.5 ns, Read Voltage: 0.8 V, Write Voltage: 1.3 V	

5.3. Hardware–Software Integration

To achieve high energy-efficiency, careful mapping of the XNOR-Net layers onto hardware and software components was necessary. We mapped the compute and memory intensive XNOR convolution layers onto the proposed memristive accelerator, while other layers—i.e., pooling, relu, batch normalization, and softmax—were computed in the application software using CPU. We evaluated the end-to-end execution that includes loading images, preparing inputs to all layers, computing the outputs, and interpreting the results, for system power and performance evaluations in this paper. The memristive accelerator can be seen as a segment of main memory Figure 6 dedicated for XNOR convolution acceleration. The input and output data transaction between main memory and the accelerator was done using DMA. We accurately modeled the delay and energy overheads of this transaction in all of the evaluations in which data are moved between main memory and the accelerator. Once the software triggers the accelerator for XNOR convolution, it waits for an *XNOR-done* signal from the accelerator, which indicates the end of XNOR convolution. Then, a DMA request is generated to initiate a DMA transaction to retrieve XNOR convolution output and make them available in main memory for further computation. Table 3 details the required size of input and output data transfer from/to the accelerator for implementing various layers of Alex XNOR-Net and VGG-16 XNOR-Net. Alex XNOR-Net needs about 7 MB of permanent storage to store the trained weights, which were appropriately programmed into the accelerator memory cells prior to inference phase.

5.4. Design Space Exploration

We studied existing memory technologies to find an appropriate cell type that provides more reliable and energy efficient hardware for accelerating the XNOR convolution layers. Table 4 reports the γ ratio for different memory technologies. As mentioned in Section 3.2.3, we employed the highest ratio of γ from the existing memory technologies to achieve more accurate and reliable bit sensing. The RRAM and PCM technologies have higher γ than STT-MRAM. However, endurance of the RRAM technology is higher than PCM [56,57] for the highest values of γ. As a result, we chose to employ the RRAM memory technology proposed by Cheng et al. [56] for building the accelerator memory arrays.

Table 3. Network parameters for mapping XNOR convolutional layers to the hardware accelerator.

Model	No.	Total Input Data (bits)	Total Output Data (bits)	Total Filter Size (bits)	Total XNOR Operations
Alex XNOR Net	1	9.22×10^4	18.66×10^4	6.14×10^5	44.79×10^7
	2	5.76×10^4	6.49×10^4	8.85×10^5	14.95×10^7
	3	8.64×10^4	6.49×10^4	13.27×10^5	22.44×10^7
	4	8.64×10^4	4.33×10^4	8.85×10^5	14.95×10^7
	5	9216	4096	37.75×10^6	37.75×10^6
	6	4096	4096	16.78×10^6	16.78×10^6
Total		33.6×10^4	36.79×10^4	58.24×10^6	10.26×10^8
VGG16 XNOR-Net	1	32.69×10^5	32.11×10^5	36.66×10^3	18.50×10^8
	2	83.17×10^4	16.06×10^5	73.73×10^3	92.48×10^7
	3	16.635×10^5	16.06×10^5	14.75×10^4	18.50×10^8
	4	43.06×10^4	80.28×10^4	29.49×10^4	92.48×10^7
	5	86.12×10^4	80.28×10^4	58.98×10^4	18.50×10^8
	6	86.12×10^4	80.28×10^4	58.98×10^4	18.50×10^8
	7	23.04×10^4	40.14×10^4	11.8×10^5	92.48×10^7
	8	46.08×10^4	40.14×10^4	23.59×10^5	18.50×10^8
	9	46.08×10^4	40.14×10^4	23.59×10^5	18.50×10^8
	10	13.11×10^4	10.035×10^4	23.59×10^5	46.24×10^7
	11	13.11×10^4	10.035×10^4	23.59×10^5	46.24×10^7
	12	13.11×10^4	10.035×10^4	23.59×10^5	46.24×10^7
	13	25.09×10^3	4096	10.28×10^7	10.28×10^7
	14	4096	4096	16.78×10^6	16.78×10^6
Total		94.91×10^5	10.34×10^6	13.42×10^7	15.38×10^9

Table 4. Resistive characteristics of different memristive technologies.

Type	Material	Endurance	H/L	γ
RRAM	GeO/STO [56]	10^6	5×10^5	0.999
	Pt/HfO_2:Cu/Cu [58]	10^8	10^3	0.998
	Ni/GeOx/HfON/TaN [59]	10^6	9×10^2	0.998
	HfO_2 [60]	10^6	10^2	0.980
	HfO_x/PCMO [61]	10^5–10^7	58.3	0.966
PCM	GeTe/Sb [57]	6.3×10^6	10^2	0.980
	$Ge_2Sb_2Te_5$(GST) [62]	10^5	30	0.935
	Chalcogenide alloy [63]	10^{10}	16.7	0.890
	$Ge_2Sb_2Te_5$(GST) [64]	10^8	15.8	0.880
STT MRAM	Ultra thin MgO layer between two ferromagnetic layers or other tunneling oxide [65–67]	10^{15}	2	0.333
		10^{15}	2	0.333
		10^{15}	1.66	0.248

5.5. Input Datasets

We trained the XNOR-Net model on Torch7 [68] using ten classes from ImageNet dataset in ILSVRC 2015 [4,51]. We used ten images from the same classes in ILSVRC 2015 for inference. These images were scaled to dimensions of $3 \times 224 \times 224$ and given to the first layer in the RGB format. Table 5 depicts the datasets used for inference on the XNOR-Net model.

Table 5. Input datasets.

Input Class	Ball	Bird	Desktop	Cup	Fish
					
Input Class	Human Face	Piano	Sandwich	Snake	Voilin
					

6. Evaluations

We evaluated the overall system energy and performance of MB-CNN as compared with both software implementations and hardware accelerators for Alex XNOR-Net within single- and multi-core mobile systems. We calculated the geometric mean of the overall execution time and total system energy recorded for the ten input classes. The baseline software solutions for single and multi core systems are denoted by "SW". The accelerator based solutions are named by the type of host CPU (i.e., single-, dual-, and quad-core) and the hardware accelerator (i.e., "GPU", "PIM", and "MB-CNN"). The software solution wherein all the CNN layers are executed in a single core CPU, floating point convolutions in GPU, where as the binary convolution layers in MB-CNN accelerator is denoted by MB-CNN_GPU notation for comparing it with different configurations. GPU carries out all the floating point operations for GPU-based accelerators. We also compared the peak capabilities of the proposed accelerator with different state-of-the-art neural network accelerators.

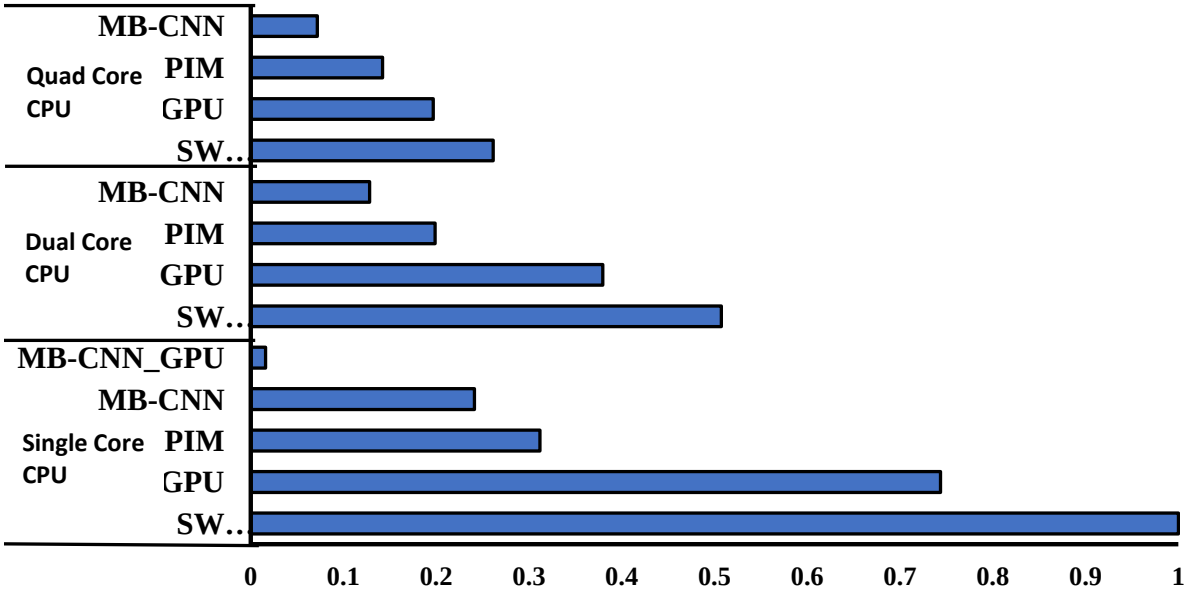

Figure 15. Execution time for input datasets for 13 different configurations all normalized to single-core CPU.

6.1. Performance

Figure 15 shows the total execution time of XNOR-Net inference with different system configurations. MB-CNN outperforms both software and accelerator-based systems in all types of single- and multi-core frameworks. As compared with the software implementations, MB-CNN achieved 4.17×, 4.25×, and 3.71× performance improvements for the single-, dual-, and quad-core systems, respectively. The integration of single-core CPU, along with GPU for floating point convolutions and MB-CNN for XNOR operationsGPU with MB-CNN achieved performance improvement of 62.7× over SW based single core CPU implementation. Although the PIM-like accelerators achieved better performance than GPU-based systems, MB-CNN outperformed the PIM-like ASICs by 1.29×, 1.58×, and 2× in the single-, dual-, and quad-core systems, respectively. The MB-CNN_GPU configuration outperformed PIM and GPU integrated based systems at least by 9× and 12.4× respectively. Notice that MB-CNN benefits from in-situ computation within memory arrays that enables massive parallelism and eliminates data movement, significantly.

6.2. Energy

Figure 16 presents the energy savings of obtained by MB-CNN compared with the baseline software and accelerator based implementations. MB-CNN achieved average energy reductions of 4×, 3.83×, and 3.64× over the software implementations on the single-, dual-, and quad-core systems, respectively. The single core MB-CNN_GPU configuration had the greatest energy savings. The total energy for this system was 24 *times* less than single core SW baseline. Among all of the evaluated configurations, the GPU based systems consumed chip area and power to improve performance. However, it could only save more energy than the PIM-like accelerator in the quad-core systems. This was mainly because of the reduced leakage energy in the GPU as the overall execution time on CPU is reduced. MB-CNN achieved better energy savings over the PIM and GPU based accelerators by respective 2.46×, 2.61×, and 2.63× for the single-, dual-, and quad-core systems, respectively. The single core MB-CNN_GPU configuration had the greatest energy savings. The total energy for this system was 24 *times* less than single core SW baseline.

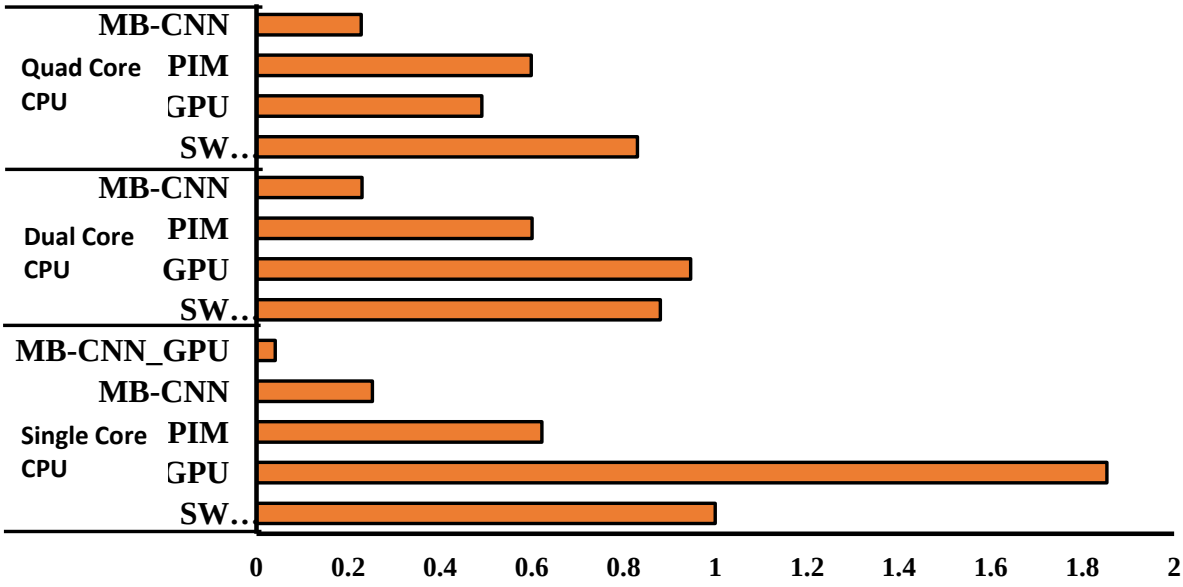

Figure 16. Overall system energy for 13 different configurations all normalized to single-core CPU.

6.3. Energy Efficiency

Figure 17 shows the total system energy vs. the overall execution time for MB-CNN and the baseline systems. All the results are normalized to single core CPU configuration. The quad-core CPU

solution with MB-CNN accelerator outperformed all other configurations in terms of performance. Notice that all of the MB-CNN systems lie on the Pareto frontier. We also calculated energy delay product (EDP) for all the systems. The results indicate that MB-CNN achieved at least 13.26×, 5.91×, and 3.18× system EDP over the software, GPU, and PIM baselines, respectively. In addition, the MB-CNN_GPU configuration had the best energy efficiency overall, as its EDP was at least 25× less than its other counterparts.

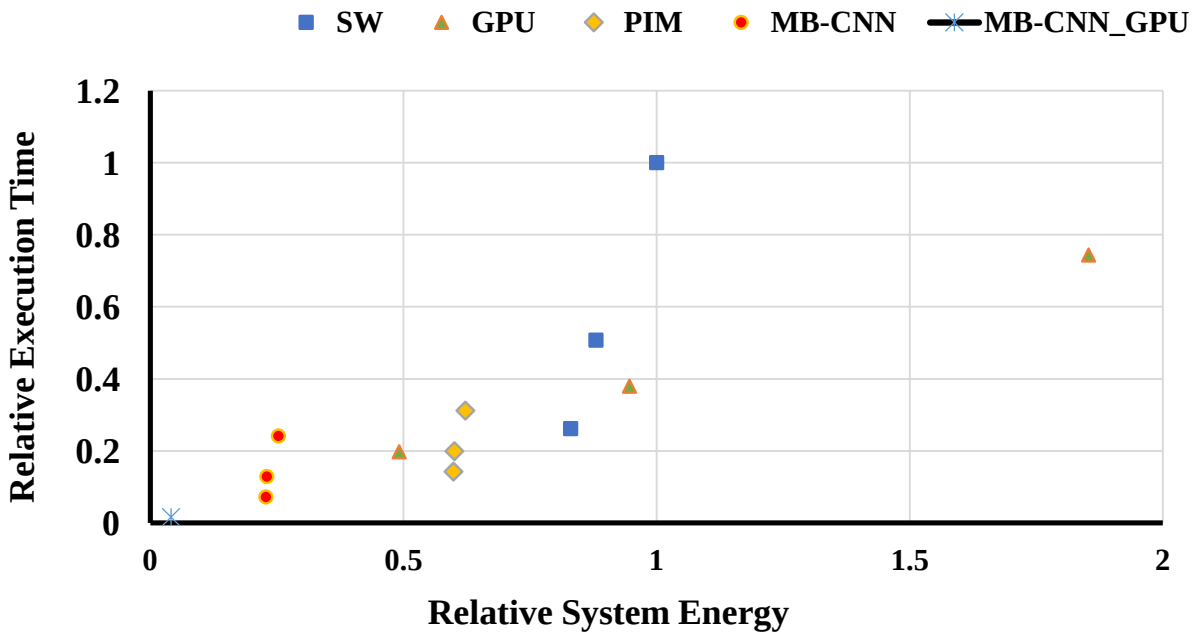

Figure 17. System energy vs. execution time relative to single core CPU configuration.

6.4. Comparison with Other Neural Network Accelerators

The performance and energy overheads of different low power ASIC accelerators were compared with MB-CNN. For this, we considered two of our MB-CNN configurations (single-core CPU and GPU), (Quad-core CPU) which offer best results. Figures 18 and 19 show the frames/s and energy consumed by system, both relative to the MB-CNN quad-core (4 CPU) configuration.

XNOR-POP [69], and XNORBIN [70] are accelerators for binary convolution network. However, neither takes into account necessary floating convolutions required at the beginning and end of XNOR net to provide improved accuracy [10]. To make a fair comparison to MB-CNN, we added a similar software interface as MB-CNN to perform floating point convolutions which adds to its actual frame processing time. There is further scope to reduce floating point convolution execution time by optimizing the software, however it will impact execution time of all binary accelerators in a similar fashion including MB-CNN. We considered energy and number of frames per second for all the accelerators from XNOR-POP [69] and calculated relative frames/s and energy consumed by system. In addition, we considered CPU, cache, and GPU powers to calculate energy number of MB-CNN, since those are integral part fo MB-CNN solution. MB-CNN with a single-core CPU and GPU configuration can process up to 65 frames per second in its end to end execution. As shown in Figure 18, this system is approximately 1.86× better than Eyeriss accelerator [71]. Figure 19 also shows energy consumed per frame for MB-CNN(single-core CPU and GPU) is almost comparable with Eyeriss. Clearly, MB-CNN configuration with a single core CPU and a GPU achieves better results than other accelerators when both metrics are considered for evaluations.

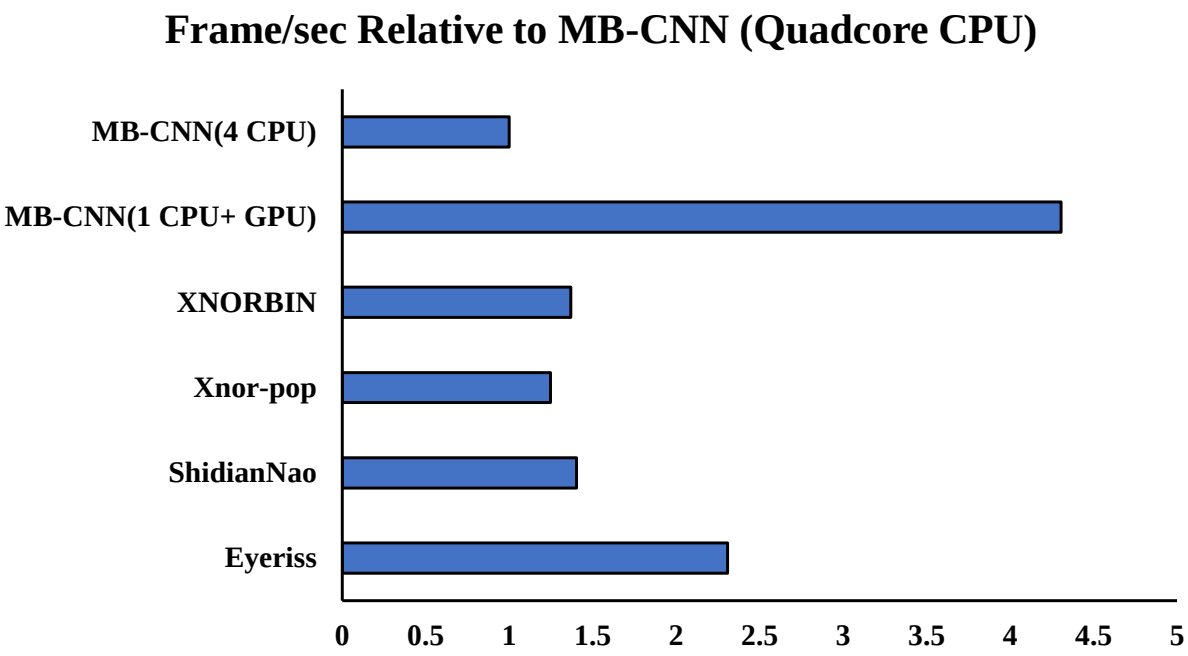

Figure 18. Number of frames per second for different state-of-the-art accelerators relative to MB-CNN.

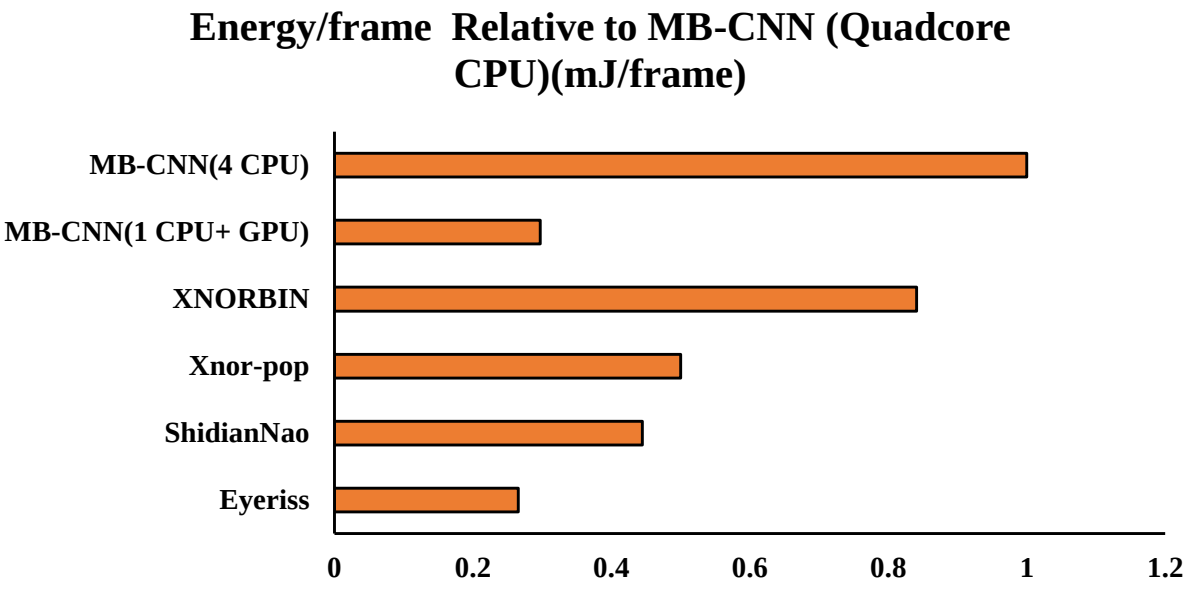

Figure 19. System Energy per frame for different state-of-the-art accelerators relative to MB-CNN.

7. Related Work

This section provides a summary of related work in hardware accelerators for neural network, compressed networks, processing in memory, and mobile IoT applications.

7.1. Neural Network Accelerators

There have been many works proposed for hardware neural network accelerators [72], where complex analog computations in different benchmarks like PARSEC [73] are being replaced by vector matrix multiplication, to reduce energy efficiency which is a key factor for embedded system applications. Many applications require large amount of datasets and real-time values for their computations but they do not always require exact precision in their results. This is proposed in BRAINIAC [74], a platform that combines precise accelerators with approximate neural network based

computing. Similarly, Esmaeilzadeh et al. [41] introduced the potential of energy and performance improvement in general purpose code which can tolerate small errors by using approximate neural network accelerators for computations. PRIME [75] proposes to perform matrix-vector multiplications of neural networks in RRAM arrays to accelerate the overall process. Moreover, it proposes a software/hardware interface to implement various NNs on their proposed accelerators. Unlike all prior work, the proposed accelerator can be used for wide range of binary neural networks not only to improve energy efficiency but also performance.

7.2. CNN and DNN Hardware Accelerators

High performance and energy efficient hardware accelerators have been proposed for compute and memory intensive neural network applications. Dian-Nao [76] exploits the scope of parallelism in CNN and DNN by introducing parallel MAC unit. It also addresses long latency in data movement between main memory and compute engine by introducing the concept of tiling and prefetch buffer. DaDian-nao [77] extends the previous architecture by addressing the problem of huge memory bandwidth requirement for CNN and DNN computations. However, both solutions limit the performance of large benchmarks due to excessive memory bandwidth. Eyeriss [71] introduces a new technique to feed input data and weight to different PEs to maximize their reuse. It also mentions hierarchical memory organization to minimize the cost of data movement from main memory to Process Engine. ShiDian-nao [78] proposes similar concepts and introduces systolic array based convolution computation to reuse input data and intermediate outputs. ISAAC [42] proposes memristive in situ computations of dot product to accelerate high performance computing of convolution and FC layers. Redeye [79] moves the convolution computation in analog domain to reduce power consumption of data movement between sensor and micro-controller and to minimize the cost of analog readout. Tianqi et al. [80] proposed RRAM based full fledged BNN accelerator that employs binarized hardware for all layers (including first and last), which limits the scope of introducing new algorithm [10]. Qiu et al. [81] proposed FPGA solution to accelerate convolution neural network in an embedded device. GPU based CNN acceleration for low power embedded device was proposed by Motamedi et al. [82]. IMCE [83] employs SOT-MRAM based accelerator to perform in memory convolution computations of low bit width CNNs. The IMCE sub-arrays along with some digital sub-blocks (Bit-Counter, Shifter and Adder) perform the convolution computation of CNN. However, MB-CNN is capable of performing XNOR, partial summation or bit counting operations of multiple kernels in parallel inside a single RRAM arrays with efficient data movement due to pre-programmed kernels. This further ameliorates XNOR computation speed. In addition, the highly dense RRAM based MB-CNN accelerator evaluated in this work has a memory capacity of 1 GB and can support much larger networks such as VGG-16 [12] and DeepFace [3] with less area.

7.3. Compressing and Quantizing Network Parameters

Numerous optimization techniques have been proposed in the literature focusing on trading the precision of computations for gaining energy-efficiency and performance. Recent work [13] exploits redundancy inherent in deep CNNs and applies linear compression (singular value decomposition) technique on a pre-trained model to speedup convolution operation during inference. Han et al. [84] exploited sparsity in network parameters and applies pruning to reduce the number of redundant weights and represented them in compressed CSR matrix format for efficient storage. They further extended it to deep-compression [7] by quantizing weights and applying Huffman encoding to reduce the memory footprint to 49× for VGG-16 Net. They designed a hardware accelerator EIE [8] for the compressed network to achieve substantial speedup and energy saving by doing a sequential operation on compressed data set. It is proven that high precision weights are not very important to achieve high accuracy in deep neural network. DaDian-Nao [77] has shown that using 16 bit fixed point weights during inference has no impact in network accuracy. Liu et al. [85] proposed to quantize the weights of FC layer using vector quantization technique at the expense of 1% accuracy

loss on state-of-the-art CNN models. Binary Connect [86] quantizes the weights into binary form during forward pass and backward propagation of training phase while retaining full precision weights for gradient accumulation and parameter updates. BNN [9] and XNOR-Net [10] extend the binarization further by binarizing both input and weights with massive reducing storage and runtime for convolution neural network.

7.4. Processing in Memory

Convolutional Neural Networks require many data in the form of synapses which obtained from training. Thus, there is lot of data movement from main memory to processor for obtaining the synapses which increase power consumption. This introduced the concept for in situ memory computations which could increase the energy efficiency and make it possible to use CNNs for mobile devices where energy consumption matters the most. Initial works proposed near memory computations where processors along with programmable arrays can be placed near DRAM for fast processing [87–89]. Neural Network applications which run on mobile devices can tolerate some errors in results which can be later improved by off line intensive training algorithm. However, what matters most in such applications is performance and energy consumption of the application. Memristive Memory technologies have been recently used for in situ computations. Energy Efficient architectures performing computations like neural branch predictors, using memristive crossbars have been explored [90]. Yakopcic et al. [91] studied Memristive Crossbar arrays using SPICE Models to how that Neural Network algorithms can be computed on such memories to reduce area and power overhead. Bojnordi et al. [33] proposed an accelerator using RRAM memory technology to perform in situ operations on combinatorial logic and deep learning algorithm thereby eliminating the need for data exchange between main memory and processor. Many works propose to perform analog computations in convolution layers of CNNS and DNNS by accelerating them using memristive crossbar arrays [8,42,75]. None of the above works are targeted towards performing binary computations in convolution layers using crossbar arrays, which we propose in this paper to eliminate the need of ADCs and reduce the total number of operations by further optimizing the binarized weights and inputs using bit packing.

7.5. Mobile IoT Applications

To accelerate deep learning tasks such as speech recognition (speech-to-text and speech-to-command) and computer vision (face detection and image classification), low power mobile GPUs accelerated software library have been proposed [82,92]. Although efficient, continuous deep learning inference in GPU contributes to the overall power consumption of the device. Sanyam et al. in their recent research [93] on wear bench application showed that high performance can be achieved for deep learning along with other wearable applications (e.g., image compression, audio play back and video rendering) using Out of Order 32 bit CPU core with SIMD feature (e.g., ARM-A15 with NEON). Non-volatile processor architecture has been proposed [94] for energy harvesting IoT device to maximize the forward progress of IoT application in unstable and intermittent power supply environment. This paper considers out-of-order core as a base line processor for IoT wearable applications and introduces memristive accelerator to offload compute intensive CNN calculations from the core not only to reduce core dynamic power consumption but also to limit data movement between memory and core. At the same, we also achieve improved performance in comparison to other existing low power solutions.

8. Conclusions

Convolutional neural networks have emerged as one of the important techniques in the field of computer vision and are being used in various image classification algorithms. However, the convolution operations are memory intensive and require a lot of data movement between main memory and processor, which degrades the overall system performance and increases energy

dissipation. As CNN algorithms are used in mobile applications, energy-efficiency becomes a significant challenge. MB-CNN enables in-situ computation of XNOR computations within memristive RRAM crossbars, thereby eliminating data movement between memory arrays and processor cores. The proposed accelerator was studied in single-core and multi-core mobile systems. For both platforms, we observed significant improvements in energy-efficiency as compared with CPU, GPU, and PIM based solutions. MB-CNN has total memory of 1 GB in its current design, which can be used for storing parameters of larger CNNs such as VGG-16. Using the MB-CNN solution on different state-of-the-art CNNs and evaluating its performance is definitely a future work in this field. Developing the MB-CNN hardware and evaluating the solution for practical implementations on different platforms (e.g., FPGA boards and cellphones) is another area which can be explored.

Author Contributions: Conceptualization and methodology is done by A.P.C. and M.N.B. Software, validation, formal analysis, investigation is done by P.K. and A.P.C. Writing original draft preparation is done by A.P.C. Writing review and editing is done by P.K. and M.N.B. Visualization, supervision, project administration, funding acquisition by M.N.B.

Funding: This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1755874 and in part by The University of Utah Seed.

Conflicts of Interest: The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Wearable Computing Devices. Available online: https://www.abiresearch.com/press/wearable-computing-devices-like-apples-iwatch-will/ (accessed on 21 February 2013).
2. Wearable Device Prediction. Available online: http://www.idc.com/getdoc.jsp?containerId=prUS41530816 (accessed on 15 June 2016).
3. Taigman, Y.; Yang, M.; Ranzato, M.; Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014.
4. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*; Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2012; pp. 1097–1105.
5. Povey, D.; Ghoshal, A.; Boulianne, G.; Burget, L.; Glembek, O.; Goel, N.; Hannemann, M.; Motlicek, P.; Qian, Y.; Schwarz, P.; et al. The Kaldi speech recognition toolkit. In Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, Waikoloa Village, HI, USA, 11–15 December 2011; Number EPFL-CONF-192584; IEEE Signal Processing Society: Piscataway, NJ, USA, 2011.
6. Lei, X.; Senior, A.W.; Gruenstein, A.; Sorensen, J. Accurate and compact large vocabulary speech recognition on mobile devices. In Proceedings of the Interspeech, Lyon, France, 25–29 August 2013; ISCA: Singapore, 2013; Volume 1.
7. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
8. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient inference engine on compressed deep neural network. In Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; IEEE Press: Piscataway, NJ, USA, 2016; pp. 243–254.
9. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized Neural Networks. In *Advances in Neural Information Processing Systems 29*; Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016; pp. 4107–4115.
10. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv* **2016**, arXiv:1603.05279.
11. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
12. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.

13. Denton, E.L.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2014; pp. 1269–1277.
14. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; ACM: New York, NY, USA, 2015; pp. 161–170.
15. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
16. Chen, W.; Wilson, J.T.; Tyree, S.; Weinberger, K.Q.; Chen, Y. Compressing Neural Networks with the Hashing Trick. In Proceedings of the ICML, Lille, France, 6–11 July 2015; pp. 2285–2294.
17. Tang, W.; Hua, G.; Wang, L. *How to Train a Compact Binary Neural Network with High Accuracy?* AAAI: Menlo Park, CA, USA, 2017; pp. 2625–2631.
18. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR09, Miami, FL, USA, 20–25 June 2009.
19. Kingma, D.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
20. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
21. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images;* Technical Report; University of Toronto: Toronto, ON, Canada, 2009.
22. Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; Ng, A.Y. Reading digits in natural images with unsupervised feature learning. In Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Granada, Spain, 12–17 December 2011; Volume 2011, p. 5.
23. Li, Y.; Lee, S.; Oowada, K.; Nguyen, H.; Nguyen, Q.; Mokhlesi, N.; Hsu, C.; Li, J.; Ramachandra, V.; Kamei, T.; et al. 128Gb 3b/Cell NAND flash memory in 19nm technology with 18MB/s write rate and 400Mb/s toggle mode. In Proceedings of the 2012 IEEE International on Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 19–23 February 2012; pp. 436–437.
24. Takashima, D.; Nagadomi, Y.; Ozaki, T. A 100 MHz ladder FeRAM design with capacitance-coupled-bitline (CCB) cell. *IEEE J. Solid-State Circuits* **2011**, *46*, 681–689. [CrossRef]
25. Hoya, K.; Takashima, D.; Shiratake, S.; Ogiwara, R.; Miyakawa, T.; Shiga, H.; Doumae, S.M.; Ohtsuki, S.; Kumura, Y.; Shuto, S.; et al. A 64-Mb chain FeRAM with quad BL architecture and 200 MB/s burst mode. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2010**, *18*, 1745–1752. [CrossRef]
26. Simpson, R.; Krbal, M.; Fons, P.; Kolobov, A.; Tominaga, J.; Uruga, T.; Tanida, H. Toward the ultimate limit of phase change in Ge2Sb2Te5. *Nano Lett.* **2009**, *10*, 414–419. [CrossRef] [PubMed]
27. Chien, T.K.; Chiou, L.Y.; Sheu, S.S.; Lin, J.C.; Lee, C.C.; Ku, T.K.; Tsai, M.J.; Wu, C.I. Low-Power MCU with Embedded ReRAM Buffers as Sensor Hub for IoT Applications. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2016**, *6*, 247–257. [CrossRef]
28. Kawahara, A.; Azuma, R.; Ikeda, Y.; Kawai, K.; Katoh, Y.; Hayakawa, Y.; Tsuji, K.; Yoneda, S.; Himeno, A.; Shimakawa, K.; et al. An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput. *IEEE J. Solid-State Circuits* **2013**, *48*, 178–185. [CrossRef]
29. Benoist, A.; Blonkowski, S.; Jeannot, S.; Denorme, S.; Damiens, J.; Berger, J.; Candelier, P.; Vianello, E.; Grampeix, H.; Nodin, J.; et al. 28 nm advanced CMOS resistive RAM solution as embedded non-volatile memory. In Proceedings of the 2014 IEEE International Reliability Physics Symposium, Waikoloa, HI, USA, 1–5 June 2014; p. 2E-6.
30. Ueki, M.; Akeuchi, K.; Yamamoto, T.; Tanabe, A.; Ikarashi, N.; Saitoh, M.; Nagumo, T.; Sunamura, H.; Narihiro, M.; Uejima, K.; et al. Low-power embedded ReRAM technology for IoT applications. In Proceedings of the 2015 Symposium on VLSI Circuits (VLSI Circuits), Kyoto, Japan, 16–18 June 2015; pp. T108–T109.
31. Yang, J.J.; Pickett, M.D.; Li, X.; Ohlberg, D.A.; Stewart, D.R.; Williams, R.S. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nat. Nanotechnol.* **2008**, *3*, 429–433. [CrossRef] [PubMed]

32. Xu, C.; Niu, D.; Muralimanohar, N.; Balasubramonian, R.; Zhang, T.; Yu, S.; Xie, Y. Overcoming the challenges of crossbar resistive memory architectures. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 7–11 February 2015; pp. 476–488.
33. Bojnordi, M.N.; Ipek, E. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona, Spain, 12–16 March 2016; pp. 1–13.
34. Pan, F.; Gao, S.; Chen, C.; Song, C.; Zeng, F. Recent progress in resistive random access memories: Materials, switching mechanisms, and performance. *Mater. Sci. Eng. R: Rep.* **2014**, *83*, 1–59. [CrossRef]
35. Niu, D.; Xu, C.; Muralimanohar, N.; Jouppi, N.P.; Xie, Y. Design trade-offs for high density cross-point resistive memory. In Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, Redondo Beach, CA, USA, 30 July–1 August 2012; ACM: New York, NY, USA, 2012; pp. 209–214.
36. Xu, C.; Dong, X.; Jouppi, N.P.; Xie, Y. Design implications of memristor-based RRAM cross-point structures. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 14–18 March 2011; pp. 1–6.
37. Zidan, M.A.; Fahmy, H.A.H.; Hussain, M.M.; Salama, K.N. Memristor-based memory: The sneak paths problem and solutions. *Microelectron. J.* **2013**, *44*, 176–183. [CrossRef]
38. Chiu, P.F.; Nikolić, B. A Differential 2R Crosspoint RRAM Array With Zero Standby Current. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *62*, 461–465. [CrossRef]
39. Dumas, S. Mobile Memory Forum: LPDDR3 and WideIO. JEDEC Mobile Forum; 2011; Volume 201. Available online: https://www.jedec.org/sites/default/files/Sophie_Dumas_11%2006%20Mobile%20Memory%20Forum.pdf (accessed on 13 october 2018).
40. Qureshi, M.K.; Franceschini, M.M.; Lastras-Montaño, L.A.; Karidis, J.P. Morphable memory system: A robust architecture for exploiting multi-level phase change memories. In *ACM SIGARCH Computer Architecture News*; ACM: New York, NY, USA, 2010; Volume 38, pp. 153–162.
41. Esmaeilzadeh, H.; Sampson, A.; Ceze, L.; Burger, D. Neural acceleration for general-purpose approximate programs. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, Vancouver, BC, Canada, 1–5 December 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 449–460.
42. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, R.; Strachan, J.P.; Hu, M.; Williams, R.S.; Srikumar, V. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; IEEE Press: Piscataway, NJ, USA, 2016; pp. 14–26.
43. Allen, P.E.; Geiger, R.L.; Strade, N.S. *VLSI Design Techniques for Analog and Digital Circuits*; McGraw-Hill Publishing Company: New York, NY, USA, 1990.
44. Razavi, B. *Principles of Data Conversion System Design*; Wiley-IEEE Press: New York, NY, USA, 1995.
45. Kester, W.; Analog Devices, I. *Data Conversion Handbook*; Analog Devices, Inc.: Norwood, MA, USA, 2005; ISBN 0-916550-27-3.
46. Free PDK 45 nm Open-Access Based PDK for the 45 nm Technology Node. Available online: http://www.eda.ncsu.edu/wiki/FreePDK (accessed on 1 March 2017).
47. Esmaeilzadeh, H.; Blem, E.; Amant, R.S.; Sankaralingam, K.; Burger, D. Dark Silicon and the End of Multicore Scaling. In Proceedings of the 38th International Symposium on Computer Architecture (ISCA'11) , San Jose, CA, USA, 4–8 June 2011.
48. Dong, X.; Xu, C.; Xie, Y.; Jouppi, N.P. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2012**, *31*, 994–1007. [CrossRef]
49. Jouppi, N.P.; Kahng, A.B.; Muralimanohar, N.; Srinivas, V. CACTI-IO: CACTI with off-chip power-area-timing models. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 1254–1267. [CrossRef]
50. Ardestani, E.K.; Renau, J. ESESC: A fast multicore simulator using time-based sampling. In Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013), Shenzhen, China, 23–27 February 2013; pp. 448–459.
51. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis. (IJCV)* **2015**, *115*, 211–252. [CrossRef]

52. Li, S.; Ahn, J.H.; Strong, R.D.; Brockman, J.B.; Tullsen, D.M.; Jouppi, N.P. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-42, New York, NY, USA, 12–16 December 2009; pp. 469–480.
53. A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT. Available online: https://github.com/torch/torch7 (accessed on 1 February 2017).
54. GPU-Based Deep Learning Inference: A Performance and Power Analysis. Available online: https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf (accessed on 1 March 2017).
55. Rupesh, Y.K.; Behnam, P.; Pandla, G.R.; Miryala, M.; Bojnordi, M.N. Accelerating k-Medians Clustering Using a Novel 4T-4R RRAM Cell. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, 1–14. [CrossRef]
56. Cheng, C.; Chin, A.; Yeh, F. Novel ultra-low power RRAM with good endurance and retention. In Proceedings of the 2010 Symposium on VLSI Technology (VLSIT), Honolulu, HI, USA, 15–17 June 2010; pp. 85–86.
57. Hu, Y.; You, H.; Zhu, X.; Zou, H.; Zhang, J.; Song, S.; Song, Z. Superlattice-like GeTe/Sb thin film for ultra-high speed phase change memory applications. *J. Non-Cryst. Solids* **2017**, *457*, 141–144. [CrossRef]
58. Liu, S.; Wang, W.; Li, Q.; Zhao, X.; Li, N.; Xu, H.; Liu, Q.; Liu, M. Highly improved resistive switching performances of the self-doped Pt/HfO2: Cu/Cu devices by atomic layer deposition. *Sci. China Phys. Mech. Astron.* **2016**, *59*, 127311. [CrossRef]
59. Cheng, C.; Chin, A.; Yeh, F. Ultralow switching energy Ni/GeOx/HfON/TaN RRAM. *IEEE Electron Device Lett.* **2011**, *32*, 366–368. [CrossRef]
60. Luo, Q.; Xu, X.; Liu, H.; Lv, H.; Gong, T.; Long, S.; Liu, Q.; Sun, H.; Banerjee, W.; Li, L.; et al. Super non-linear RRAM with ultra-low power for 3D vertical nano-crossbar arrays. *Nanoscale* **2016**, *8*, 15629–15636. [CrossRef] [PubMed]
61. Lee, J.; Jo, M.; Seong, D.J.; Shin, J.; Hwang, H. Materials and process aspect of cross-point RRAM. *Microelectron. Eng.* **2011**, *88*, 1113–1118. [CrossRef]
62. Ahn, C.; Fong, S.W.; Kim, Y.; Lee, S.; Sood, A.; Neumann, C.M.; Asheghi, M.; Goodson, K.E.; Pop, E.; Wong, H.S.P. Energy-efficient phase-change memory with graphene as a thermal barrier. *Nano Lett.* **2015**, *15*, 6809–6814. [CrossRef] [PubMed]
63. Zhou, P.; Zhao, B.; Yang, J.; Zhang, Y. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH Computer Architecture News*; ACM: New York, NY, USA, 2009; Volume 37, pp. 14–23.
64. Pellizzer, F.; Pirovano, A.; Ottogalli, F.; Magistretti, M.; Scaravaggi, M.; Zuliani, P.; Tosi, M.; Benvenuti, A.; Besana, P.; Cadeo, S.; et al. Novel/spl mu/trench phase-change memory cell for embedded and stand-alone non-volatile memory applications. In Proceedings of the 2004 Symposium on VLSI Technology, Honululu, HI, USA, 15–19 June 2004; Digest of Technical Papers; IEEE: Piscataway, NJ, USA, 2004; pp. 18–19.
65. Ohashi, T.; Yamaguchi, A.; Hasumi, K.; Inoue, O.; Ikota, M.; Lorusso, G.; Donadio, G.L.; Yasin, F.; Rao, S.; Kar, G.S. Variability study with CD-SEM metrology for STT-MRAM: Correlation analysis between physical dimensions and electrical property of the memory element. In Proceedings of the SPIE Metrology, Inspection, and Process Control for Microlithography XXXI, San Jose, CA, USA, 26 February–2 March 2017; p. 101450H.
66. Chen, Y.; Wang, X.; Li, H.; Xi, H.; Yan, Y.; Zhu, W. Design margin exploration of spin-transfer torque RAM (STT-RAM) in scaled technologies. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2010**, *18*, 1724–1734. [CrossRef]
67. Zhang, Y.; Zhang, L.; Wen, W.; Sun, G.; Chen, Y. Multi-level cell STT-RAM: Is it realistic or just a dream? In Proceedings of the 2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), DresDen, Germany, 12–16 March 2012; pp. 526–532.
68. Collobert, R.; Bengio, S.; Mariéthoz, J. Torch: A Modular Machine Learning Software Library; Technical Report; Idiap, 2002. Available online: https://infoscience.epfl.ch/record/82802/files/rr02-46.pdf (accessed on 13 October 2018).
69. Jiang, L.; Kim, M.; Wen, W.; Wang, D. XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs. In Proceedings of the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Taipei, Taiwan, 24–26 July 2017; pp. 1–6.
70. Bahou, A.A.; Karunaratne, G.; Andri, R.; Cavigelli, L.; Benini, L. XNORBIN: A 95 TOp/s/W Hardware Accelerator for Binary Convolutional Neural Networks. *arXiv* **2018**, arXiv:1803.05849.

71. Chen, Y.H.; Emer, J.; Sze, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 367–379.
72. Chen, T.; Chen, Y.; Duranton, M.; Guo, Q.; Hashmi, A.; Lipasti, M.; Nere, A.; Qiu, S.; Sebag, M.; Temam, O. BenchNN: On the broad potential application scope of hardware neural network accelerators. In Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC), San Diego, CA, USA, 4–6 November 2012; pp. 36–45.
73. Bienia, C.; Kumar, S.; Singh, J.P.; Li, K. The PARSEC benchmark suite: Characterization and architectural implications. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, Toronto, ON, Canada, 25–29 October 2008; ACM: New York, NY, USA, 2008; pp. 72–81.
74. Grigorian, B.; Farahpour, N.; Reinman, G. BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), San Francisco, CA, USA, 7–11 February 2015; pp. 615–626.
75. Chi, P.; Li, S.; Xu, C.; Zhang, T.; Zhao, J.; Liu, Y.; Wang, Y.; Xie, Y. PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; IEEE Press: Piscataway, NJ, USA, 2016; pp. 27–39.
76. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*; ACM: New York, NY, USA, 2014; Volume 49, pp. 269–284.
77. Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; et al. Dadiannao: A machine-learning supercomputer. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 13–17 December 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 609–622.
78. Du, Z.; Fasthuber, R.; Chen, T.; Ienne, P.; Li, L.; Luo, T.; Feng, X.; Chen, Y.; Temam, O. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*; ACM: New York, NY, USA, 2015; Volume 43, pp. 92–104.
79. LiKamWa, R.; Hou, Y.; Gao, J.; Polansky, M.; Zhong, L. RedEye: Analog ConvNet image sensor architecture for continuous mobile vision. In Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, Korea, 18–22 June 2016; IEEE Press: Piscataway, NJ, USA, 2016; pp. 255–266.
80. Tang, T.; Xia, L.; Li, B.; Wang, Y.; Yang, H. Binary convolutional neural network on RRAM. In Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, Japan, 16–19 January 2017; pp. 782–787.
81. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. Going deeper with embedded fpga platform for convolutional neural network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; ACM: New York, NY, USA, 2016; pp. 26–35.
82. Motamedi, M.; Fong, D.; Ghiasi, S. Fast and Energy-Efficient CNN Inference on IoT Devices. *arXiv* **2016**, arXiv:1611.07151.
83. Angizi, S.; He, Z.; Parveen, F.; Fan, D. IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network. In Proceedings of the 23rd Asia and South Pacific Design Automation Conference, Jeju, Korea, 22–25 January 2018; IEEE Press: Piscataway, NJ, USA, 2018; pp. 111–116.
84. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 1135–1143.
85. Gong, Y.; Liu, L.; Yang, M.; Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv* **2014**, arXiv:1412.6115.
86. Courbariaux, M.; Bengio, Y.; David, J.P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 3123–3131.
87. Gokhale, M.; Holmes, B.; Iobst, K. Processing in memory: The Terasys massively parallel PIM array. *Computer* **1995**, *28*, 23–31. [CrossRef]

88. Elliott, D.G.; Stumm, M.; Snelgrove, W.M.; Cojocaru, C.; McKenzie, R. Computational RAM: Implementing processors in memory. *IEEE Des. Test Comput.* **1999**, *16*, 32–41. [CrossRef]
89. Oskin, M.; Chong, F.T.; Sherwood, T. *Active Pages: A Computation Model for Intelligent Memory*; IEEE Computer Society: Washington, DC, USA, 1998; Volume 26.
90. Wang, J.; Tim, Y.; Wong, W.F.; Li, H.H. A practical low-power memristor-based analog neural branch predictor. In Proceedings of the 2013 International Symposium on Low Power Electronics and Design, Beijing, China, 4–6 September 2013; IEEE Press: Piscataway, NJ, USA, 2013; pp. 175–180.
91. Yakopcic, C.; Hasan, R.; Taha, T.; McLean, M.; Palmer, D. Memristor-based neuron circuit and method for applying learning algorithm in SPICE. *Electron. Lett.* **2014**, *50*, 492–494. [CrossRef]
92. Latifi Oskouei, S.S.; Golestani, H.; Hashemi, M.; Ghiasi, S. CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android. In Proceedings of the 2016 ACM on Multimedia Conference, Amsterdam, The Netherlands, 15–19 October 2016; ACM: New York, NY, USA, 2016; pp. 1201–1205.
93. Mehta, S.; Torrellas, J. WearCore: A core for wearable workloads? In Proceedings of the 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), Haifa, Israel, 11–15 September 2016; pp. 153–164.
94. Ma, K.; Li, X.; Swaminathan, K.; Zheng, Y.; Li, S.; Liu, Y.; Xie, Y.; Sampson, J.J.; Narayanan, V. Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power. *IEEE Micro* **2016**, *36*, 72–83. [CrossRef]