A Product Engine for Energy-Efficient Execution of Binary Neural Networks Using Resistive Memories

João Vieira*, Edouard Giacomin*, Yasir Qureshi†, Marina Zapater†, Xifan Tang*,
Shahar Kvatinsky‡, David Atienza†, and Pierre-Emmanuel Gaillardon*

*LNIS, University of Utah, USA
†ESL, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
‡Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion - Israel Institute of Technology, Israel

***Abstract*—The need for running complex *Machine Learning* (ML) algorithms, such as *Convolutional Neural Networks* (CNNs), in edge devices, which are highly constrained in terms of computing power and energy, makes it important to execute such applications efficiently. The situation has led to the popularization of *Binary Neural Networks* (BNNs), which significantly reduce execution time and memory requirements by representing the weights (and possibly the data being operated) using only one bit. Because approximately 90% of the operations executed by CNNs and BNNs are convolutions, a significant part of the memory transfers consists of fetching the convolutional kernels. Such kernels are usually small (e.g., 3×3 operands), and particularly in BNNs redundancy is expected. Therefore, equal kernels can be mapped to the same memory addresses, requiring significantly less memory to store them. In this context, this paper presents a custom *Binary Dot Product Engine* (BDPE) for BNNs that exploits the features of *Resistive Random-Access Memories* (RRAMs). This new engine allows accelerating the execution of the inference phase of BNNs. The novel BDPE locally stores the most used binary weights and performs binary convolution using computing capabilities enabled by the RRAMs. The system-level gem5 architectural simulator was used together with a C-based ML framework to evaluate the system's performance and obtain power results. Results show that this novel BDPE improves performance by 11.3%, energy efficiency by 7.4% and reduces the number of memory accesses by 10.7% at a cost of less than 0.3% additional die area, when integrated with a $28\,\mathrm{nm}$ Fully Depleted Silicon On Insulator ARMv8 in-order core, in comparison to a fully-optimized baseline of YoloV3 XNOR-Net running in a unmodified Central Processing Unit.**

***Index Terms*—Machine Learning, Edge Devices, Binary Neural Networks, RRAM-based Binary Dot Product Engine**

I. Introduction

The role played by *Internet of Things* (IoT) in the advent of Big Data [1], which requires the execution of complex *Artificial Intelligence* (AI) algorithms in latest smart embedded systems (also called edge devices [2]), has created the necessity of developing new *Machine Learning* (ML) algorithms that can target highly energy and computing power constrained systems. Naturally, this led to the creation of new *Convolutional Neural Network* (CNN) models capable of being executed efficiently in such systems [3], and because approximately 90% of the operations realized in CNNs are convolutions [4], convolutional layers became the main targets for optimization. Consequently, two main approaches aiming at optimizing the execution of such layers were adopted: using dedicated hardware accelerators; and reducing the precision of the operands.

The use of custom accelerators attempts to optimize the execution of CNNs by exploiting hardware level parallelism [5], [6] and offloading workload to near-data accelerators [7], [8]. However, most of these solutions do not comply with the limitations of edge devices since they require a significant amount of hardware resources to be implemented, which is hardly feasible in the context of such energy- and cost-constrained systems. Furthermore, accelerators do not provide performance benefits whenever the workload associated to the dataset is not enough to overcome the communication overhead with main memory [9]. For that, the datasets have to be significantly big, and since edge devices are usually used for processing rather small datasets, accelerators may also not be suited for such systems.

CNN models using low-precision operands, such as Intel bfloat16 [10], were also created to speed computation in convolutional layers. Ultimately, *Binary Neural Networks* (BNNs) [11], [12] only use one bit to represent the weights. Moreover, in XNOR-Net BNNs [11] both the input and the weights of the convolutional layers are binary, thus convolutions are performed by simply executing the bit-wise XNOR of the input and kernel followed by a bitcount. Although accuracy is sacrificed to some level by such heuristics [11], the resultant memory savings and performance improvements allow some XNOR-Net CNNs to be executed by edge devices. This allows such systems to execute complex workloads, such as facial recognition, which is a real-time task that has to be performed efficiently [13]. Moreover, in most BNNs, the convolutional kernels are rather small (typically 3×3, 5×5 or 7×7), thus it is expected that a significant part of their data is redundant.

Exploiting data redundancy in BNNs, this paper proposes a *Binary Dot Product Engine* (BDPE) that locally stores the most used kernels and efficiently implements binary convolution to accelerate convolutional layers. Thus, whenever the kernels are stored in the unit, there is no need for transferring them from memory, reducing the overall memory accesses. Since *Resistive Random-Access Memories* (RRAMs) excel by their ability of enabling in-memory computing capabilities while providing storage support [14], the novel BDPE uses a robust and energy efficient RRAM-based convolutional block based on [15]. Contrarily to previous analog proposals (e.g., [16]) whose accuracy can be severely degraded due to process variation, this block operates in the digital domain. Since the BDPE is meant to be integrated into the pipeline of a *Central Processing Unit* (CPU), it does not introduce communication overheads, which is one of the main drawbacks of using dedicated accelerators.

Overall, the contributions of this paper are the following:

1) We propose a RRAM-based BDPE that efficiently performs the dot product between an input and a kernel stored in the RRAM array or between two inputs;
2) We integrate the devised BDPE with the pipeline of a conventional ARMv8 CPU;
3) We present an ARMv8 *Instruction Set Architecture* (ISA) extension to support the novel BDPE;
4) We evaluate the impact of the novel mechanism in the execution of a state-of-the-art BNN using an optimized ML framework, using the system-level gem5 architectural simulator.

Results show that, even for a modest RRAM usage rate, the BDPE allows a speedup of approximately 11.3% over a fully optimized version of YoloV3 XNOR-Net running on an ARM Cortex-A53, while reducing the number of memory accesses during the inference phase by 10.7%. Additionally, the energy spent during the execution of the BNN is reduced by 7.4%, and the die area required to implement the BDPE is less than 0.3% of the total CPU.

The rest of this paper is organized as follows. Section II provides insights on the necessary context for this work. Section III details the devised architecture. Sections IV and V present the evaluation methodology and the experimental results, respectively. Sections VI and VII summarize the key point of discussion and conclusions about the work, respectively.

II. CNN Background

CNNs relate to a class of *Neural Networks* (NNs) that are commonly applied to image analysis. Such networks are dominated by convolutional layers, where the input is convoluted by a kernel, and the produced output is passed to the next layer. The operation of CNNs is divided into two phases: the training phase and the inference phase. Although the devised mechanism can be applied to both phases, the computing power and energy required by the training phase goes beyond edge devices. Therefore, only the inference phase is targeted by this work.

In the image analysis domain, CNNs can be used for multiple purposes, such as image classification [17], [18], or object detection [19], such as pedestrians detection [20], [21] or face recognition [22]. YoloV3 [23], [24] is a state-of-the-art CNN for real-time object detection. It divides the image into regions and predicts bounding boxes and associated probabilities for each region. By sizing the network, YoloV3 also allows trading accuracy for performance by reducing the number of network layers. This represents an advantage for edge devices characterized by low computing power. When binarized using the definitions in [11], the XNOR-Net version of a given configuration of YoloV3 shows approximately the same *mean Average Precision* (mAP) as the full-precision network, while reducing the size of the weights approximately 32× and thus executing up to 58× faster. For the *Street View House Numbers* (SVHN) dataset [25], the YoloV3 XNOR-Net shows a mAP decrease of only 0.43%, while increasing the number of detections by 75%. YoloV3 is a widely accepted benchmark, thus we use it to evaluate the proposed BDPE.

From the same authors of YoloV3, Darknet [26] is a C-based ML framework compatible with a large number of NN models. When executed in Darknet, BNNs can be accelerated using CPU primitives that can efficiently perform the binary convolution of two 64-bit vectors. For that purpose, the elements of the small kernels are shifted and combined in vectors of 64 bits, thus reducing the data redundancy that can be found between combinations. For the trimmed version of YoloV3 XNOR-Net trained with the SVHN dataset, all the generated combinations of small kernels are different, making it impossible to take advantage of data redundancy. However, as shown in Table I, a small percentage of combinations is frequently used. For instance, 0.07% of the combinations (designated from now on as kernels) are used in 9.74% of the total number of convolutions. Thus it is possible to significantly accelerate the total amount of convolutions only storing a small percentage of kernels locally. Due to Darknet's performance, it is used in this work for evaluation purposes.

TABLE I: Profile of a trimmed version of YoloV3 regarding the frequency of use of each convolutional kernel during the inference phase.

Execution frequency	Kernels	Total executions	Percentage
169	115,200	19,468,800	45.64
676	18,432	12,460,032	29.21
2,704	1,152	3,115,008	7.30
10,816	320	3,461,120	8.11
43,264	96	4,153,344	9.74

III. System architecture

In this section, the architectural details of the *Binary Dot Product Engine* (BDPE) proposed in this paper are presented alongside the modifications done to an ARMv8 to integrate it as a new functional unit. Additionally, the operation of the devised BDPE is demonstrated with a practical example.

A. Architecture of the Binary Dot Product Engine

The devised BDPE aims at improving the performance of binary convolution while reducing the number of data transfers by locally storing the most used kernels. Accordingly, the base block used in this work, proposed in [15], implements the binary dot product using the compute capabilities of RRAM, while also providing storage. The operands involved in the binary dot product are a constant binary kernel stored in the RRAM array in the form of resistance values and a binary vector supplied as an input. By selecting the appropriate kernel local address and applying the input data to the RRAM array, the XNOR phase of the convolution is performed as a memory readout using custom XNOR sense amplifiers [15]. Then, a fully-digital combinational circuit counts the number of logic ones to determine the result of the binary dot product operation. The block diagram of the proposed RRAM-based convolutional block is depicted in Fig. 1b. Using the proposed mechanism has two main advantages. First, by locally storing the most used kernels (e.g., in YoloV3, 0.07% of the kernels are used in 9.74% of the convolutions), the data movements are substantially reduced, thus decreasing the overall energy consumption. Second, since it is capable of performing a dot product in one clock cycle, it also increases system's performance.

To use the devised engine in a CPU, a control path is added to the original block, which also unlocks the possibility of

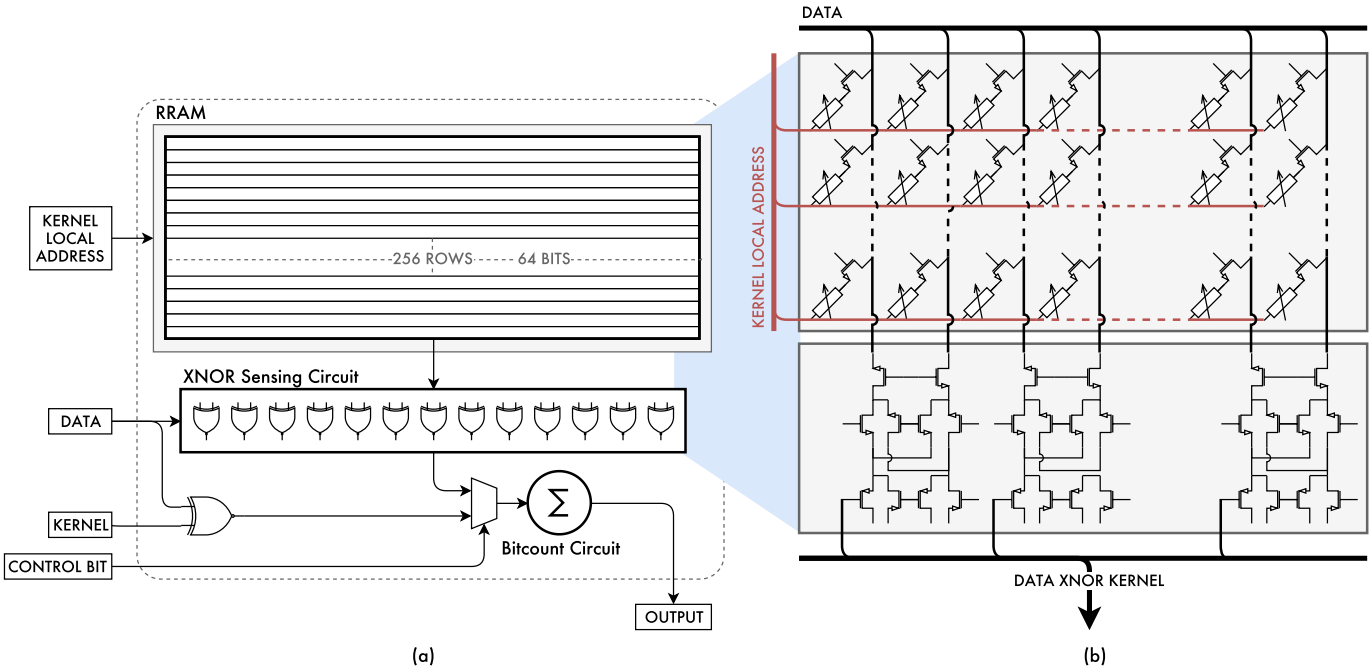

Fig. 1: Block diagram of the proposed BDPE. Fig. 1a illustrates the control logic and the alternative mechanism to perform the bit-wise XNOR of the data and the kernel (both supplied as an input), in case the kernel is not stored in the RRAM array. Fig. 1b depicts the RRAM-based convolutional block used in this work inspired by [15].

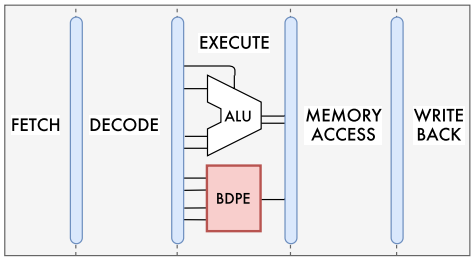

Fig. 2: Simplified block diagram of a generic processor pipeline integrating the Binary Dot Product Engine.

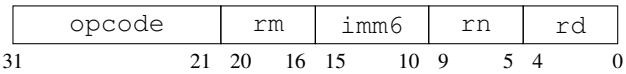

Fig. 3: Format of the new instructions added to the ARMv8 ISA to allow the processor to issue instructions to the BDPE. The *opcodes* `10000011000` and `11000011000` were re-purposed to specify the custom instructions. `rm` and `rn` specify addresses of 64-bit registers; `imm6` represents a 6-bit immediate; and `rd` specifies the address of the destination 64-bit register.

performing the XNOR of two inputs as an alternative to using the primary RRAM-based XNOR mechanism. Fig. 1a illustrates the block diagram of the BDPE.

The secondary XNOR mechanism is built from regular CMOS gates. Depending on the *control bit* obtained from the decoded *opcode*, the output of the alternative XNOR mechanism can be used as input to the *Bitcount Circuit*. In that case, the result is based on the kernel coming from a processor register rather a kernel stored in the RRAM. Such a functionality is particularly useful when the required convoluting kernel is not stored in the RRAM array.

B. Integration with Central Processing Unit

The integration of the proposed BDPE with a conventional ARMv8 core is divided into three phases: (1) the integration of the new functional unit with the processor's pipeline; (2) the creation of new instructions in the ISA to use the BDPE; (3) compiler support to use the new ISA instructions on the software side.

The new functional unit is integrated into the processor's pipeline in the *Execution* stage. It receives the operands from the *Decode* stage, similarly to the *Arithmetic and Logic Unit* (ALU), and passes the result to the *Execute/Memory Access* pipeline register, as shown in Fig. 2. According to the ARM Architecture Reference Manual for the ARMv8-A architecture profile [27], the ARMv8 ISA has unused *opcodes* that can be re-purposed to expand the functionality of the CPU. Using two of the unused *opcodes*, two instructions were created and assigned to the novel BDPE.

Fig. 3 illustrates the format of the new instructions and denotes the purpose of each distinct set of bits. Each of the new instructions is decoded in the *Decode* stage such that the content of the register specified by `rm` serves as input data of the BDPE; the content of the register represented by `rn` is the input kernel; `imm6` specifies the address of the kernel stored in the RRAM array; and the second *Most Significant Bit* (MSB) of the *opcode* designates the control bit.

After determining which kernels to store in the RRAM and the respective RRAM addresses, an additional phase is added to the compilation workflow to replace regular binary dot products using the processor's ALU with the corresponding instruction using the BDPE. By controlling the *opcode* (and

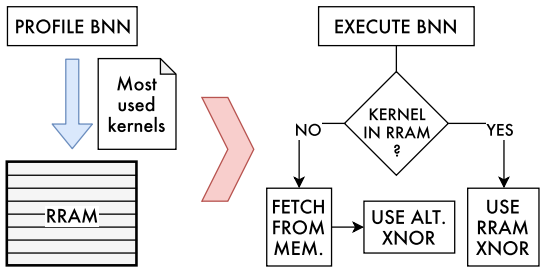

Fig. 4: Simple example that illustrates the process of storing the most used kernels inside the RRAM and running a BNN using the novel BDPE.

consequently the *control bit*), either the output of the RRAM array or the output of the alternative XNOR mechanism is used to calculate the final result of the binary dot product operation. To be able to use `imm6` to represent the kernel address in the RRAM, some compiler-level work would be required, specifically at *Low Level Virtual Machine* (LLVM) level. However, for a careful validation of the BDPE, in this work no modifications to LLVM have been made. Alternatively, a procedure that produces similar results that can be applied in a gem5 architectural simulation context was used (as described in section IV-A).

C. Operation example

As shown in Fig. 4, the workflow for running a BNN using the novel BDPE is divided into profiling and execution.

During the profiling, the BNN is used to perform a single inference while the kernel space is profiled, selecting the most frequently used kernels. The selected kernels are stored in the RRAM, and a configuration file is generated containing the information about the content of the RRAM. Then, the CNN is recompiled, and the code responsible for implementing the binary convolution is replaced by custom code that utilizes the BDPE. If the kernel being used is stored in the RRAM, the compiler inserts a special instruction to perform the binary convolution using the RRAM array. Otherwise, the compiler inserts a load instruction to fetch the kernel from memory, followed by a special instruction that performs the binary convolution using the two data inputs of the BDPE.

IV. Evaluation Methodology

In this section, the tools used to evaluate the performance, energy efficiency improvements and area requirements are presented alongside the methodology and considered assumptions.

A. Performance Evaluation

The Darknet framework and the gem5 architectural simulator [28], together with gem5-X [29] (which allowed to reduce the usual 10% error margin provided by gem5 to less than 4%), were used to evaluate the performance of the modified ARMv8 architecture. Darknet was set to operate as a trimmed version of the YoloV3 XNOR-Net CNN, and gem5 was configured to emulate the ARMv8 Cortex-A53 with and without the devised BDPE. Darknet was adapted to allow support for gem5 *System Emulation* (SE) mode by compiling all the inputs of the network (the configuration files, the previously trained weights for the SVHN [25] dataset and the input image) into a single executable binary file. Additionally, the Darknet framework was modified at assembly level to use the custom BDPE instead of the processor's ALU when performing 64-bit binary convolutions.

To determine the most used kernels and populate the RRAM array, the following two-step procedure was used: (1) Darknet was ran using gem5 and the kernel space was profiled; (2) The most used kernels were selected and stored in the RRAM. After populating the RRAM, the gem5 module responsible for emulating the BDPE was rebuilt. Because the framework was not recompiled, gem5 in SE mode mapped the data structures to the same addresses used in (1), and the application flow was kept the same except for the binary convolutions involving the most frequently used kernels stored in the RRAMs. In those cases, the RRAM array was used instead of the alternative XNOR mechanism to perform the XNOR operation. The complete system featuring the modified ARM Cortex-A53 and four DRAM ranks of 1 GB each operating at 2400 MHz was emulated and the entire workflow of Darknet was executed.

As a result of running the modified version of Darknet, gem5 produced timing results, statistics on memory accesses and usage of the CPU's several modules. Such results were used to estimate the energy consumption, as described in Section IV-C.

B. Circuit-level Implementation

Circuit-level metrics were obtained through electrical simulations using a commercial 28 nm *Fully Depleted Silicon On Insulator* (FD-SOI) design kit to assess the hardware requirements and the power demand of the devised BDPE. The detailed experimental procedure is described in [15]. Delay and power results were extracted from Eldo simulations, to be used in models for the architectural evaluation, as explained in Section IV-C. These metrics were extracted for the two possible cases (*control bit=0* and *control bit=1*) to consider when the XNOR is performed using a kernel locally stored in the RRAM or a kernel coming from the processor's registers, respectively. In order to consider an average case, it was assumed that half of the data inputs, as well as the kernels, are zeros and the other half is ones. For the area estimation, the full-custom layout of the RRAM array and its associated control path were modeled using Cadence Virtuoso. The bitcount circuit was generated with Synopsys Design from *Register Transfer Level* (RTL) netlists and integrated into a Place & Route flow using Cadence Innovus to obtain the complete layout of a 256×64 RRAM-based BDPE.

C. Power Models

For the energy efficiency and power assessment of the proposed architecture, 28 nm FD-SOI power models for ARMv8 in-order cores were used, as proposed by [29] and [30]. The power models include active core power, *Wait-For-Memory* (WFM) power, and the static core power. They also include the *Last Level Cache* (LLC) power consumption as well as the power for memory accesses to the DRAM in the system. These power numbers were combined with gem5 statistics to calculate the overall energy consumed by the system, including also the BDPE energy values, as explained in Section IV-B. McPAT [31] was not used for the power estimation of the core, as it does not have power models for 28 nm FD-SOI.

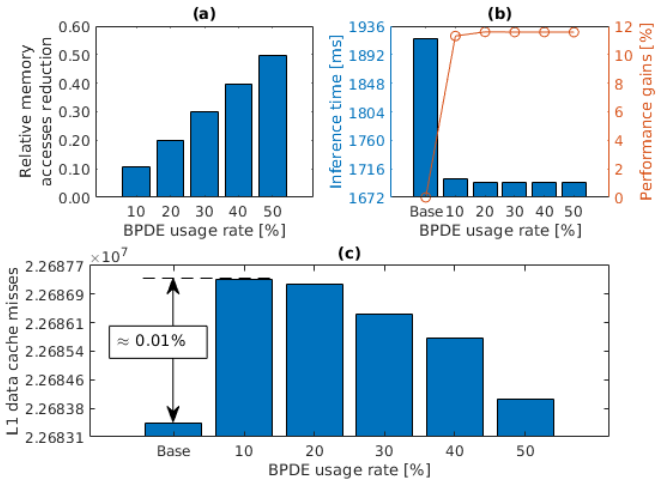

Fig. 5: Results showing the performance improvements due to using BDPE. Figure 5a shows the relative reduction on memory accesses during the inference phase of YoloV3 for five runs where the usage of the BDPE varies between 10% and 50%. Figure 5b depicts the execution time of the inference phase of YoloV3 and the relative performance improvements varying the usage of the BDPE. Figure 5c illustrates the total number of cache misses for the five considered scenarios and the baseline system.

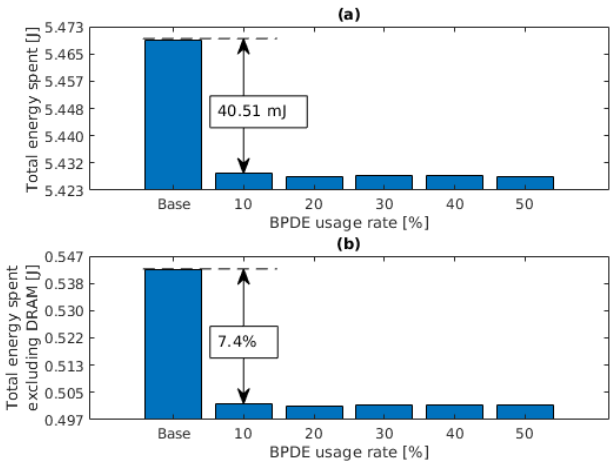

Fig. 6: Results showing the energy efficiency improvements due to using BDPE. Figure 6a shows the total energy spent when executing the baseline and five runs where the usage of the BDPE varies between 10% and 50%. Figure 6b shows, for the same scenarios the energy spent by the system excluding the main memory (DRAM).

V. Experimental Results

In this section, experimental results on performance, energy savings, and hardware requirements are presented.

To better evaluate the impact of the BDPE in the performance of the targeted ARMv8 CPU, five scenarios were considered where the RRAM usage rate (percentage of convolutions that use kernels locally stored in the RRAM) varies between 10% and 50%, when executing YoloV3 XNOR-Net.

A. Performance Analysis

By offloading the execution of binary convolutions to the BDPE, the kernels are not requested from the main memory when they are locally stored in the RRAM array. Therefore, a reduction in memory accesses equal to the RRAM usage rate is observed, as shown in Fig. 5a. Moreover, over 99% of the memory accesses reduction happens at the L1 cache. Thus, the system counts with the maximum benefits of caching effects.

However, avoiding the transfer of sequential kernels to the processor whenever the RRAM array is used produces irregularities in the memory access patterns. This situation leads to more evictions and cache collisions, thus causing additional cache misses, as shown in Fig. 5c. Nevertheless, the increase of the cache misses is lower than 0.01% relative to the total number of memory accesses. Hence, this is negligible and does not affect the overall performance.

All in all, as illustrated in Fig. 5, for a usage rate of 10% the performance improvement is 11.3%. Also, the performance gains show no significant variation with the RRAM usage rate. This effect has two main causes: (1) both the data paths in the BDPE take exactly one cycle to perform a binary convolution; (2) due to caching effects, the convolutional kernels are stored in the L1 cache 94% of the time, substantially reducing the time required to fetch them. Consequently, using the alternative method for performing the XNOR of the kernel and the input data takes approximately the same time as using the RRAM array and does not impact negatively the overall performance.

B. BDPE Energy and Hardware Resources

Table II shows the hardware requirements, power demand and delay for the BDPE obtained through simulation, as described in Section IV-B. In practice, since a 10% RRAM usage rate allows achieving the best trade-off between hardware requirements, performance improvements and energy savings, that scenario was used to obtain the results in this section.

The die area required to implement the novel mechanism is only $3.845\,\mu m^2$ per CPU core, using a FD-SOI $28\,nm$ process, while a dual-core ARM Cortex-A53 in an equivalent process requires $2.8\,mm^2$ [32]. Therefore, the BDPE represents less than 0.3% of the total CPU area. The energy spent for a single operation when using the RRAM array (*control bit=0*) is reduced by 37% comparing to using the alternative mechanism (*control bit=1*). This is allowed by the intrinsic energy efficiency of the RRAM array [15]. Although this advantage comes at a cost of a delay overhead at circuit level, the maximum operating frequency allowed is still $2.5\,GHz$. Thus, as the target platform is the ARM Cortex-A53 with an operating frequency of $2\,GHz$, the BDPE can be integrated with the system without constraining its overall frequency.

C. Energy Efficiency Analysis

The total energy spent by the baseline system (ARM Cortex-A53) and the five scenarios using the BDPE is illustrated in Fig. 6a. Then, Fig. 6b shows the energy consumption for the same circumstances subtracted by the energy spent by the DRAM. As shown in Table III, the total energy spent by the

TABLE II: Hardware resources and average power demand of the BDPE considering the two possible paths data paths considering a RRAM usage rate of 10%. When *control bit=0*, the RRAM array is used to implement the XNOR operation. Otherwise, the alternative XNOR mechanism is used.

	Area/Hardware resources [μm^2]	**Power** [mW]	**Delay** [ps]
control bit=0	3,845	1.24	408
control bit=1		3.23	214

TABLE III: Total energy spent by the BDPE and the CPU during the inference phase of YoloV3 XNOR-Net.

RRAM usage rate [%]	Baseline	10	20	30	40	50
BDPE [µJ]	0	0.870	0.279	0.271	0.263	0.255
CPU [µJ×10^6]	0.542	0.502	0.501	0.501	0.501	0.502

BDPE is negligible when compared with the rest of the system, and so the energy savings are mostly due to the reduction in the execution time. As the execution time is approximately constant regardless of the RRAM usage rate, so are the energy savings. When considering only the processing system (excluding the DRAM main memory), the use of the BDPE allows for average energy savings of 7.4%.

VI. Discussion

The advantages allowed by the devised BDPE are tightly coupled with the considered baseline CPU and the used CNN model. Since this work uses an ARM Cortex-A53, which is a high-efficiency CPU, the compute power and energy efficiency enabled by the baseline puts it among the most efficient new edge devices. Nevertheless, the use of the devised BDPE allows achieving significant performance improvements and energy savings at the cost of a minor area overhead. It is also worth saying that should the baseline be a more rudimentary processing system (e.g., an ultra-low power embedded system), the novel BDPE would allow for bigger improvements.

The ML framework Darknet, optimized to achieve the best performance on CPUs, eliminated the possibility of taking advantage of kernel redundancy, which would increase the RRAM usage rate and consequently allow for higher performance improvements and energy savings. To circumvent this and increase redundancy among kernels, the use of techniques such as weight clustering are proven to be efficient, while sacrificing little accuracy [16]. Nevertheless, since YoloV3 XNOR-Net uses a small subset of the kernels in most convolutions, significant data redundancy was still achieved.

VII. Conclusions

This paper presented a novel RRAM-based BDPE suited for accelerating the inference phase of BNNs meant to be integrated within the pipeline of a CPU. The power demand, hardware resources and propagation delay of the devised mechanism were modeled, and its impact on the considered base system was comprehensively evaluated using the Darknet framework and gem5, together with gem5-X. Results showed that even for a modest RRAM usage rate the novel BDPE achieved performance improvements of 11.3% and 7.4% energy savings. Furthermore, the integration of the novel mechanism requires only few modifications to the baseline CPU, while representing less than 0.3% of the total die area, and does not lower the operation frequency of the system.

Acknowledgements

This work was supported by the grants 2016016 from the United States-Israel Binational Science Foundation, ERC Consolidator Grant COMPUSAPIEN (GA No. 725657), ERC starting grant Real-PIM-System (GA No. 757259), and partially by the EC H2020 EUROLAB4HPC2 project (GA No. 800962).

References

[1] U. Ahsan and A. Bais. A Review on Big Data Analysis and Internet of Things. In *MASS*, pages 325–330. IEEE Computer Society, 2016.
[2] M. Ammar *et al*. Internet of Things: A survey on the security of IoT frameworks. *J. Inf. Sec. Appl.*, 38:8–27, 2018.
[3] A. Howard *et al*. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.
[4] J. Cong *et al*. Minimizing Computation in Convolutional Neural Networks. In *ICANN*, volume 8681 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2014.
[5] W. Chen *et al*. An Asynchronous Energy-Efficient CNN Accelerator with Reconfigurable Architecture. In *A-SSCC*, pages 51–54. IEEE, 2018.
[6] B. Sun *et al*. Ultra Power-Efficient CNN Domain Specific Accelerator With 9.3TOPS/Watt for Mobile and Embedded Applications. In *CVPR Workshops*, pages 1677–1685. IEEE Computer Society, 2018.
[7] M. Mao *et al*. A Versatile ReRAM-based Accelerator for Convolutional Neural Networks. In *SiPS*, pages 211–216. IEEE, 2018.
[8] K. Guo *et al*. RRAM Based Buffer Design for Energy Efficient CNN Accelerator. In *ISVLSI*, pages 435–440. IEEE Computer Society, 2018.
[9] A. Prakash *et al*. Modelling communication overhead for accessing local memories in hardware accelerators. In *ASAP*, pages 31–34. IEEE Computer Society, 2013.
[10] Intel. BFLOAT16 – Hardware Numerics Definition, 2018.
[11] M. Rastegari *et al*. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV (4)*, volume 9908 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2016.
[12] M. Courbariaux and Y. Bengio. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.
[13] Y. Shen *et al*. CS-CNN: Enabling Robust and Efficient Convolutional Neural Networks Inference for Internet-of-Things Applications. *IEEE Access*, 6:13439–13448, 2018.
[14] Haitong Li *et al*. Resistive ram-centric computing: Design and modeling methodology. *IEEE Trans. on Circuits and Systems*, 64-I(9):2263–2273, 2017.
[15] E. Giacomin *et al*. A Robust Digital RRAM-Based Convolutional Block for Low-Power Image Processing and Learning Applications. *IEEE Trans. on Circuits and Systems*, 66-I(2):643–654, 2019.
[16] S. Gupta *et al*. NNPIM: A Processing In-Memory Architecture for Neural Network Acceleration. *IEEE Transactions on Computers*, 2019.
[17] J. Guérin *et al*. CNN features are also great at unsupervised classification. *CoRR*, abs/1707.01700, 2017.
[18] U. Chester and J. Ratsaby. Machine Learning for Image Classification and Clustering Using a Universal Distance Measure. In *SISAP*, volume 8199 of *Lecture Notes in Computer Science*, pages 59–72. Springer, 2013.
[19] Z. Zhao *et al*. Object Detection with Deep Learning: A Review. *CoRR*, abs/1807.05511, 2018.
[20] E. Ohn-Bar and M. Trivedi. To boost or not to boost? On the limits of boosted trees for object detection. In *ICPR*, pages 3350–3355. IEEE, 2016.
[21] P. Dollár *et al*. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, 2012.
[22] K. Sung and T. Poggio. Example-Based Learning for View-Based Human Face Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(1):39–51, 2002.
[23] J. Redmon *et al*. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR*, pages 779–788. IEEE Computer Society, 2016.
[24] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.02767, 2018.
[25] Y. Netzer *et al*. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
[26] J. Redmon. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/, 2013–2016.
[27] ARM. ARM Architecture Reference Manual, 2018.
[28] N. Binkert *et al*. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.
[29] Y. Qureshi *et al*. Gem5-X: A Gem5-Based System Level Simulation Framework to Optimize Many-Core Platforms. In *Spring Simulation Conference (SpringSim'19)*. IEEE/ACM/SCS, 2019.
[30] A. Pahlevan *et al*. Energy proportionality in near-threshold computing servers and cloud data centers: Consolidating or Not? In *2018 Design, Automation Test in Europe Conference Exhibition*, pages 147–152, 2018.
[31] S. Li *et al*. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, pages 469–480. ACM, 2009.
[32] F. Abouzeid *et al*. 30% static power improvement on ARM Cortex®-A53 using static biasing-anticipation. In *ESSCIRC*, pages 37–40. IEEE, 2016.