XNOR-POP: A Processing-in-Memory Architecture for Binary Convolutional Neural Networks in Wide-IO2 DRAMs

Lei Jiang Minje Kim
Indiana University Bloomington
{jiang60,minje}@indiana.edu

Wujie Wen
Florida International University
wwen@fiu.edu

Danghui Wang
Northwestern Polytechnical University
wangdh@nwpu.edu.cn

***Abstract*—It is challenging to adopt computing-intensive and parameter-rich Convolutional Neural Networks (CNNs) in mobile devices due to limited hardware resources and low power budgets. To support multiple concurrently running applications, one mobile device needs to perform multiple CNN tests simultaneously in real-time. Previous solutions cannot guarantee a high enough frame rate when serving multiple applications with reasonable hardware and power cost. In this paper, we present a novel process-in-memory architecture to process emerging binary CNN tests in Wide-IO2 DRAMs. Compared to state-of-the-art accelerators, our design improves CNN test performance by $4\times \sim 11\times$ with small hardware and power overhead.**

I. Introduction

The deep CNNs have been the dominant approach to solving a wide variety of problems such as natural language processing, recommender systems and computer vision [19], e.g., face, object and scene labeling. They have demonstrated excellent recognition accuracy, which is extremely relevant and important to mobile devices such as virtual reality (VR) devices (Oculus Rift, Microsoft HoloLens and Google Glass) and self-driving cars.

However, CNN tests require not only huge volumes of parameters generating intensive off-chip memory accesses but also a large number of computing-intensive convolutions. For instance, an AlexNet test [19] involves 61 million parameters and 1.5 billion floating point (FP) operations. Although 16-bit fixed point approximation reduces only <1% accuracy [10], the essential computing effort of CNNs makes CPUs and GPUs hardly guarantee a high enough frame rate with tight power budgets in mobile devices [1], [11]. Various accelerators, such as FPGAs [32], [22], ASICs [7], [1], [2], hybrid memory cube (HMC) [17] and ReRAM [3], [28] process-in-memories (PIMs), are proposed to improve CNN test performance and energy efficiency.

Unfortunately, it is still difficult for mobile devices to adopt existing solutions to process CNNs, because of their tight power budget, high frames-per-second (FPS) requirement and urgent need to support multiple CNN tests concurrently. The battery of Google Glass stands for only 45 minutes when tracking consecutive human and object actions [9]. Intensive off-chip memory accesses seriously limit computing throughput of FPGA [32], [22] and ASIC [7], [1], [2] accelerators, and further shorten the battery lifetime of mobile devices. Though the memory wall issue is alleviated by HMC [17] and ReRAM [3], [28] PIMs, power hungry HMC interfaces, analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) hinder the practical deployment of these PIMs in mobile devices. With low power budgets, most accelerators merely meet the basic motion picture frame rate requirement [25], 30 FPS, when processing complex CNNs, e.g., ResNet-18 [12]. Some accelerators [7], [2] even fail to achieve 30 FPS when running ResNet-18 tests. Moreover, in most real-life cases, multiple applications conducting CNN tests run simultaneously in a single mobile device [11]. However, it is difficult for existing accelerators to support multiple concurrent CNN tests and maintain a high enough frame rate for all with reasonable hardware and power overhead in mobile devices.

The emerging binary CNN, XNOR-Net [24], reduces computing overhead and memory footprint of CNN tests by binarizing their weights, activations and inputs. Instead of FP multiply accumulate (MAC) operations, it performs XNORs and population counts (popcounts) on binary weights and inputs during CNN tests, so it is a promising solution to support multiple CNN tests concurrently and deliver high frame rates in mobile settings. In this paper, we propose a novel Wide-IO2 PIM architecture to support XNOR-Net in mobile devices. Our contributions include:

- We propose XNOR-POP to process CNN tests with competitive accuracy for mobile devices by XNORing and popcounting in Wide-IO2 DRAMs. XNOR-POP conducts XNORs inside DRAM arrays, transfers XNOR results by through-silicon-vias (TSVs) and completes popcounts on the logic die.
- We present XNOR Loop Unrolling to fully take advantage of large DRAM row buffer to increase XNOR operation parallelism in DRAM arrays. XNOR Loop Unrolling supports a board range of convolutional layer parameters, e.g., kernel size.
- We evaluate XNOR-POP and compare it against state-of-the-art designs. Our experimental results show that it improves CNN test performance by $4\times \sim 11\times$ and reduces test energy consumption by $> 90\%$ on average.

II. Background, Prior Art and Motivation

A. Wide-IO2 Mobile DRAM

DRAM increases density by adopting 3D integration technology to stack multiple DRAM layers vertically [31]. 3D stacking technology also improves pin/wire routing and reduces wire length, so that the latency and power along DRAM data path can be minimized. Particularly, for high-end mobile devices, e.g., VR game consoles, Wide-IO2 DRAM [15] has

To appear in ISLPED2017

L. Jiang, M. Kim, W. Wen and D. Wang, "XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs," 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Taipei, 2017, pp. 1-6. doi: 10.1109/ISLPED.2017.8009163, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8009163&isnumber=8009138

been standardized by JEDEC as a high bandwidth and low power main memory candidate. The wide and low frequency interface of Wide-IO2 DRAM does not require the delay lock loop (DLL) or the on-die termination (ODT) to maintain signal integrity under a low supply voltage.

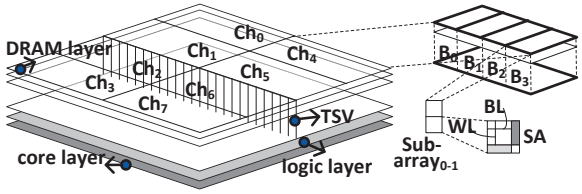

Fig. 1. Wide-IO2 DRAM architecture.

Figure 1 shows the Wide-IO2 DRAM architecture [15], which has eight independent 64-bit channels with double data rate. A channel lies across multiple DRAM layers and contains 4 banks, each of which has 2KB local sense amplifiers (SAs) and local word-lines. A dedicated peripheral area hosting global SAs is shared by all banks in a channel. One Wide-IO2 DRAM layer achieves 8Gb capacity [31]. All DRAM layers are connected to the logic layer containing a memory controller [31] by TSVs.

B. CNN Basics and Binary CNN

The state-of-the-art CNNs [20], [19], [12] exhibit excellent effectiveness in computer vision applications. CNN is a supervised learning algorithm deploying a feedforward function during **test**s and a backward process for **training**s. In mobile devices, object recognitions in images or videos are performed in real-time, and hence CNN test speed matters. On the contrary, CNN trainings can be done offline and in powerful servers with multiple GPUs. In this paper, we focus on accelerating CNN tests in mobile devices.

```
//convolution kernel
for(row=0; row<R; row++)
    for(col=0; col<C; col++)
        for(outn=0; outn<M; outn++){
            for(inn=0; inn<N; inn++)
                for(i=0; i<K; i++)
                    for(j=0; j<K;j++)
                        // floating point MAC
                        output[outn][row][col]=
                        weight[outn][inn][i][j]*
                        input[inn][S*row+i][S*col+j];
        }
```

```
// XNOR, popcount and scaling
O[outn][row][col]=
popcount(W[outn]==I[row,col]);
output[outn][row][col]=
      O[outn][row][col]
     *Z[outn][row][col]
     *a[outn][row][col];
```

Fig. 2. Pseudo code of a convolutional layer.

A CNN usually consists of multiple layers including convolutional, batch normalization, activation and pooling layers. Among all layers, convolutional layers consume $> 90\%$ of total test time [32]. Figure 2 describes the pseudo code of solving a convolutional layer receiving N feature matrices as inputs, each of which is convolved by a moving window with a $K \times K$ weight kernel to produce an element in one of output feature matrices. The stride of the moving window is represented by S often $< K$. The output consisting of M feature matrices is the input for the next convolutional layer.

To reduce computing complexity, recent works [6], [24], [18], [5] propose binary neural networks through binarizing weights, so that tests can replace expensive FP MACs by cheap arithmetic or logic operations. BinaryConnect [6] and Binary-Weight-Networks [24] binarize weights to -1 and 1, and test results can be computed by only additions and subtractions. XNOR-Net [24] and BinaryNet [5] further adopt binary activations and binarize each element in input matrices to 1 or -1. In hardware implementations, all -1s are mapped to 0s. Therefore, tests can use XNOR-popcounts as a low power alternative to convolutions [24], [18], [5]. Compared to BinaryNet, XNOR-Net supports more complex CNN topologies, e.g., AlexNet and ResNet-18. So, in this work, we focus on designing a mobile PIM to support XNOR-Net. The pseudo code of solving an XNOR-Net convolutional layer is also shown in Figure 2.

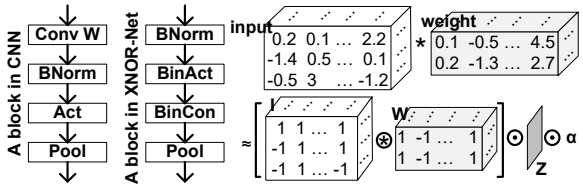

Fig. 3. Comparison between conventional CNN and XNOR-Net.

As Figure 3 shows, unlike FP CNNs, the pooling layer is put after the binary convolutional layer in XNOR-Net [24], since doing pooling on binary inputs significantly increases information loss. To further decrease information loss, batch normalization is applied before binarization. The FP input and weight convoluation approximately equals to the result of XNOR-popcounts between binary inputs and weights element-wise multiply input (Z) and weight (α) scaling factor matrices. *All variable names* in Figure 2 are used and consistent in following sections.

The downside of XNOR-Net is the decreased CNN test accuracy. For simple CNNs with a small number of parameters, e.g., LeNet-5 and MCDNN [4], test accuracy decreases by $1\% \sim 3\%$. For complex CNNs with millions of parameters, like AlexNet and ResNet-18, test accuracy (top5%) degrades to $\sim 70\%$ from $80\% \sim 89\%$. Such test accuracy is still competitive for real-time object recognition in mobile settings [25], [11]. Normally, the CNN test average accuracy of real-time object recognition is $50\% \sim 70\%$ [25], [11] in mobile devices.

C. Prior Art and Motivation

Because of power hungry devices such as ADCs, DACs or HMC interfaces, high performance CNN accelerators, the ReRAM PIM ISAAC [28] or Neurocube [17], usually require $> 65W$ to perform CNN tests. So it is infeasible for mobile devices to adopt them. Mobile ASIC accelerators, ShiDianNao [7], YodaNN [1] and Eyeriss [2], dissipate only $250 \sim 320mW$ power. However, only the BinaryConnect-based accelerator, YodaNN, is able to provide > 30 FPS during complex CNN (AlexNet and ResNet-18) tests to one single application. When running multiple concurrent applications, even YodaNN cannot support the minimal real-time requirement, 30 FPS, for all applications.

Test accuracy is not the only concern in mobile environment [11]. To achieve high energy efficiency, mobile devices have a large volume of real-time jobs with tiny power budget

and realistic accuracy requirement, e.g., real-time action or object tracking [25]. The binary CNN achieving reasonable accuracy has become a promising solution for mobile devices [5], [24]. A recent FPGA design (XNOR-FPGA) [22] implements BinaryNet [5] to test CNNs and achieves 6218 FPS [22] when working on a simple image database, CIFAR-10 [5], for an application. However, its $8W$ power consumption discourages its practical deployment in mobile devices. Moreover, BinaryNet is tested only on simple databases, but has never been proved to work with complex CNN (AlexNet) tests on the large ImageNet database [19]. Considering XNOR-Net can perform complex CNN tests [24], in this paper, we propose a mobile PIM design to support XNOR-Net, which can run multiple real-time complex CNN tests with low hardware and power overhead.

Previous DRAM PIM [27] supports simple row logic operations, i.e., ANDs and ORs. Although it can do XNORs by dividing each into two ANDs and one OR, our design reduces XNOR latency by 70%. A recent work [21] conducts row XORs, ANDs and ORs in phase change memories. However, large current mode SAs reduce memory row size and limit logic operation parallelism.

III. XNOR-POP

A. BNorm, BinAct and Scaling Factor Matrix

1) Normalization and Activation: Normalization layers stabilize and accelerate CNN trainings [24], so we adopt shift-based batch normalizations [5] in both trainings and tests. References are calculated across the entire training set as $\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\gamma+\beta$, where μ is the mean, γ is the scale, β is the shift, and σ^2 is the variance with a tiny constant ϵ to void zero denominator. An Activation is conducted by an element-wise nonlinear function after normalization in convolutional and fully-connected layers. We combine the normalization and activation into a sign function defined as $NormBin(x,c)=1$, if $x \geq c$; $NormBin(x,c)=0$, otherwise; where c is a constant and $c=\mu-\beta/\gamma\sqrt{\sigma^2+\epsilon}$. To perform $NormBin$, we only need a 16-bit comparator, since c can be computed in the previous pooling layer.

2) Scaling Factor Matrix: Weights are binarized (W) and their scaling matrices α are calculated during trainings [24]. In this paper, we use 16-bit fixed point numbers to represent original image inputs. The binarization of inputs (I) is done by $NormBin$. And the input scaling factor matrix Z is derived by performing a $K:1$ average pooling on the matrix of $(\sum input[0..N])/N$ [24]. We calculate Z in the previous pooling layer.

B. Binary Convolution: XNOR-Popcount

1) XNOR-DRAM: Modern DRAMs adopt hierarchical bit-lines (BLs) and hierarchical differential sense amplifiers (SAs) [26] to achieve high array density and reduce the parasitic capacitance on long BLs. The detailed layout of BLs and SAs can be found in [26], [14]. As Figure 4 shows, each cell has a true local BL (LBL) and a neighboring complement LBL ($\overline{\text{LBL}}$) connected to a local SA (LSA) in a sub-array. Initially, all of them are pre-charged to $0.5V_{dd}$. After a word-line (WL) is enabled, the LSA detects voltage fluctuation triggered by charges from the cell connected to the LBL. After the LSA amplifies voltage difference between the LBL and the $\overline{\text{LBL}}$, it restores original cell content through the LBL and leaves the negation of cell content on the $\overline{\text{LBL}}$. LBL pairs are connected by switches to true and complement global BLs (GBL and $\overline{\text{GBL}}$) attached to a global SA (GSA). Multiple sub-arrays form a bank. GSAs serve as a row buffer and latch the data from one of sub-arrays in a bank.

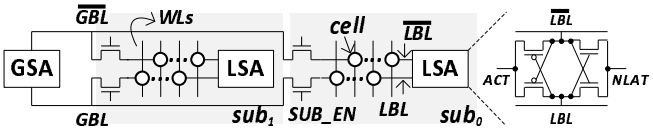

Fig. 4. Hierarchical differential sensing in a bank.

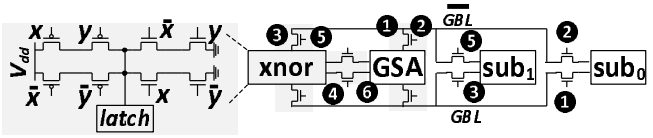

Fig. 5. XNOR-DRAM.

We propose XNOR-DRAM to perform XNOR operations at row level in each DRAM bank in Figure 5. On each GBL pair, we add an XNOR engine. Since both cell content and its negation are available on GBLs, we use only 8 transistors to implement an XNOR gate. The output latch costs another 8 transistors, while 6 transistors are added to control the XNOR logic. To compute x XNOR y, ❶ x and $\overline{x}$ are first sensed out of sub-array 0 (sub_0) and latched in the GSA. ❷ Then, the GSA and sub_0 are detached from GBLs. And the LSA performs a pre-charge operation in sub_0. ❸ The XNOR engine and sub-array 1 (sub_1) are connected to GBLs, so that y and $\overline{y}$ can be read from sub_1 and sent to the XNOR engine by GBLs. ❹ The GSA is connected to the XNOR engine to produce the result. ❺ After the XNOR result is latched, the XNOR engine and sub_1 are detached from GBLs. And then, the LSA pre-charges sub_1. ❻ At last, the GSA pre-charges itself and GBLs. One XNOR-DRAM operation involves two row activations, three pre-charges, and an XNOR. We simulated and synthesized the layout of our XNOR engine by Cadence Virtuoso. The actual XNOR operation finishes within 8ns. Hence, an XNOR-DRAM operation costs $37.5ns\times2+15ns\times3+8ns=128ns$ ($tRAS\times2+tRP\times3+Delay_{xnor}$). The XNOR engine can be easily disabled by detaching from the GBL pair and the GSA without affecting normal DRAM reads and writes.

2) XNOR Loop Unrolling: Since a DRAM row usually has 2KB (16384-bit), we present XNOR Loop Unrolling (XLU) to improve the XNOR-DRAM computation efficiency. In a convolutional layer of a state-of-the-art CNN, such as AlexNet, an $O[outn][row][col]$ usually involves hundreds of XNORs. And hence, there are enough resources to compute multiple $O[outn][row][col]$s in a DRAM row. The data layout required by XLU for a convolutional layer can be viewed in Figure 6. For an $O[outn][row][col]$, we use $W[outn]$ to represent $K\times K\times N$ (K^2N) $W[outn][inn][i][j]$s and $I[row][col]$ to denote

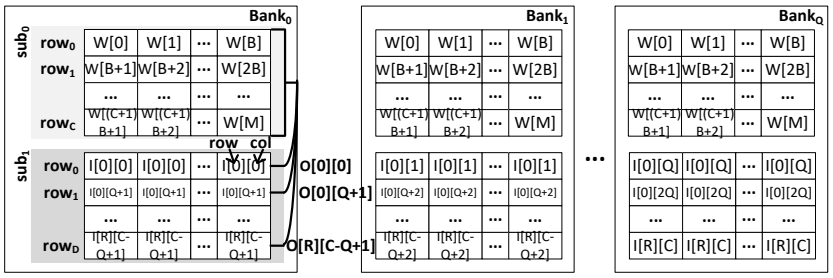

Fig. 6. XNOR Loop Unrolling (We define $W[outn]$, $I[row][col]$ & $O[row][col]$ in Sec III-B2).

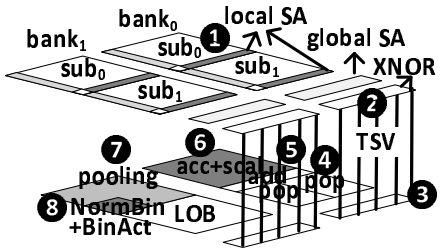

Fig. 7. XNOR-POP flow.

K^2N $I[inn][S*row+i][S*col+j]$s. One row contains $\lfloor \frac{16384}{K^2N} \rfloor$ (B) $W[outn]$s, so M $W[outn]$s are stored in $\lceil \frac{M}{B} \rceil$ (C) rows of a sub-array (sub_w) in a bank. B copies of $I[row][col]$ are consecutively recorded in one row of another sub-array (sub_i) in the same bank, so that this $I[row][col]$ row can perform C XNOR-DRAM operations with C $W[outn]$ rows respectively to produce M $O[outn][row][col]$s, which are represented by $O[row][col]$. For these C XNOR-DRAM operations, we only need to read the $I[row][col]$ row into GSAs once since for the rest we have all GSA row buffer hit. All (Q) banks have a copy of the sub-array sub_w containing all M $W[outn]$s to compute Q $O[row][col]$s simultaneously. $R \times C$ $O[row][col]$s are evenly distributed into Q banks, so each bank has $\lceil \frac{R \times C}{Q} \rceil$ (D) rows in the sub-array sub_i. XLU increases GSA row buffer hit rate to eliminate the first read operation in most cases. If x and $\overline{x}$ are hit in a GSA, we need to do only ❸∼❺ in Figure 5 and pre-charge GBLs. This entire process costs $37.5ns + 15ns \times 2 + 8ns = 75.5ns$.

3) TSVs and Logic Die: **TSV**: Due to the limitation of process technology, circuits fabricated on DRAM dies can be operated at only very low frequency, i.e., DRAM dies run at only $533MHz$ [31]. Similar to Neurocube [17], we fabricated computing components except XNOR engines on the logic die in Wide-IO2 DRAMs. All XNOR results need to be fetched from DRAM dies and sent to popcount engines on the logic die by TSVs. Two Wide-IO2 channels share 1752 TSVs supporting 1024-bit data path [31]. On average, on the data path of each bank, there are 128 TSVs running at $1GHz$ with double data rate (DDR) [31]. Even if XNOR results are ready in XNOR engine latches, memory dies require $14ns$ (tCL) to put the first 128-bit on TSVs. It takes $64ns$ for a bank to move 2KB data to the logic die from memory dies. So the transfer of 2KB XNOR results costs $78ns$. **Population Count and Scaling**: We adopt the popcount engine design from [23]. It counts the number of 1s in 64 bits every cycle and can be operated at 2.1GHz at $65nm$ node. We fixed its frequency to 2GHz. Because one bank has 128 TSVs running at 1GHz with DDR style, we need two popcount engines to take care of each bank. We also implemented a 2GHz 6-bit adder to sum the results from two popcount engines. A 11-bit adder per bank running at 2GHz accumulates results of the 6-bit adder for an $O[outn][row][col]$. We use a 16-bit multiplier to calculate $O[outn][row][col] * Z[outn][row][col] * \alpha[outn][row][col]$. We keep a bank on the DRAM die to store weight and input scaling matrices - Z and α.

C. Pooling

We use a 16-bit comparator to implement a max pooling unit by linearly scanning the data stream and always keeping the latest maximum value. For a n : 1 max pooling, the latest maximum value is outputted and reset every n cycles. A average pooling unit consists of a 16-bit adder, and shifter registers. For a n : 1 average pooling, the average result is available every n cycles. Max pooling units are mainly used for pooling layers, but average pooling units can be also used to compute input scaling factor matrix Z for the next $BinCon$ layer. We have shift logic units to calculate the constant c in Section III-A1 for the next $BNorm$ and $BinAc$ layers.

D. Pipelined Design and Write-back

1) Pipelined Design: Figure 7 shows the pipelined design of XNOR-POP to perform XNOR-Net tests. On DRAM dies, when outputs of XNOR engines from the current XNOR-DRAM operation are latched, ❶ The DRAM bank can continue to start the next XNOR-DRAM operation. Meanwhile, ❷ TSVs initialize transfers of the latched results of the current XNOR-DRAM operation to the logic die. In each cycle, ❸ a popcount engine receives data from 64 TSVs of a bank and ❹ does a popcount on the 64-bit data it received in the last cycle. ❺ In next cycle, the 6-bit adder merges results from two popcount engines. ❻ The 11-bit adder accumulates each result of the 6-bit adder to produce an $O[outn][row][col]$. A 16-bit multiplier scales it with Z and α to $output[outn][row][col]$ every $\lceil \frac{K^2N}{64} \rceil + 4$ cycles. ❼ The max pooling engine processes the $output$ stream to produce one x for every n $output$s (n : 1 max pooling). The pooling unit also produces Z and c for the next convoluational layer and writes them to the 512KB SRAM Layer Output Buffer (LOB). ❽ For a $NormBin$ and $BinAct$ operation, each x is compared to c to output a 1 or 0 to the LOB. Z, α and c are read from a dedicated bank and cached in the LOB. In a bank, a 2KB row level XNOR-DRAM operation takes $75.5ns$, when input data hit in GSAs. Otherwise, it costs $128ns$. The latency of transferring 2KB data on TSVs and processing them on the logic die can be prolonged to $142+6 = 147ns$ from $78+6 = 83ns$ by disabling TSV DDR mode and a popcount engine per bank. For a copy of the layers in Figure 3, XNOR-DRAM operations, data transfers on TSVs, popcounts, accumulations, scales, poolings, $NormBin$s and $BinAct$s form a pipeline.

2) Write-back: When the computation of a convolutional layer is finished or the LOB is full, the XNOR-POP pipeline has to stop. At this time, data in the LOB should be written

back to DRAM dies. We need to maintain the data layout of $I[row][col]$s shown in Figure 6 for the next convolutional layer. One $I[row][col]$ is propagated by B times in one row, but it has to wait $15 + 11 = 26ns$ ($tRCD + tCWL$) to start writes. TSVs of a bank spend $64ns$ in transferring the 2KB data containing B copies of the same $I[row][col]$ back to DRAM dies. After each row write, it takes $15ns$ (tRP) to close the row. The write-back of $I[row][col]$s requires $D \times 105ns$. Here, B and D are parameters of the next convolutional layer (Section III-B2). Z and c are written to the dedicated bank. At the beginning of each write-back, it takes $7.5ns$ ($tWTR$) to turn the bus around first. Each write-back obeys $tFAW$.

E. Design Overhead

1) Memory: Before the first test happens, XNOR-POP needs to load and unroll all weights of a CNN into main memory forming the data layout shown in Figure 6. Compared to millions of tests, this weight initialization overhead is trivial. Because of XLU, we increases the memory capacity occupied by weights by Q and inputs by K^2M for each convolutional layer. The unrolled weights of all convolutional layers of a complex CNN test, e.g., ResNet-18, may occupy $> 5GB$ for 32 banks. Considering there is only 1GB Wide-IO2 DRAM in our baseline configuration, original weights are stored in a sub-array. Between the computations of two convolutional layers, weights are read and unrolled by XLU controller on logic die by TSVs.

2) Hardware: For DRAM dies, we estimated the design overhead of XNOR-DRAM by CACTI [13]. XNOR-DRAM increases memory die area by 3.84% and static power by 4.4%. For the logic die, we implemented and synthesized all components by Cadence Virtuoso with the $32nm$ PTM technology [30]. The hardware overhead on the logic die is shown in Table I. Totally, XNOR-POP components on logic die cost $2.24mm^2$ (2.9% of memory die area [31]) and consume $237mW$.

TABLE I
HARDWARE OVERHEAD.

Component	Area	P_{leak}	P_{dyn}
2 popcnts + 6-bit adder	$0.0134mm^2$	$0.49mW$	$2.18mW$
11-bit adder + 16-bit multiplier	$0.0101mm^2$	$0.38mW$	$0.83mW$
max+average pooling	$0.0162mm^2$	$0.65mW$	$0.71mW$
normalization(shifter+comparator)	$0.0042mm^2$	$0.15mW$	$0.23mW$
sub-total (above$\times$31)	$1.36mm^2$	$52.2mW$	$122.1mW$
XLU controller	$0.0053mm^2$	$0.33mW$	$0.73mW$
512KB SRAM LOB	$0.87mm^2$	$60.8mW$	$0.8mW$

IV. EXPERIMENT METHODOLOGY

Workload: The CNNs we simulated and their topologies are listed in Table II. LeNet-5, Multilayer Perceptron (MLP) [20], CNP [8], SCNN [29] and MCDNN were trained with MNIST to recognize simple hand-written digits, while AlexNet and ResNet-18 were trained with ImageNet to classify complex objects. We trained all CNNs by Caffe [16]. Fully connected layers in CNNs can be converted to convolutional layers. The numbers of convolutional layers, pooling layers and fully connected layers are also listed in Table II. We also compare full precision CNNs and XNOR-Net in terms of test accuracy. XNOR-Net degrades the test accuracy of simple CNNs by $1\% \sim 3\%$ and decreases the test accuracy (top5%) of complex CNNs to $\sim 70\%$.

TABLE II
CNN BENCHMARKS (ACC: ACCURACY; C: CONVOLUTIONAL LAYER; S: POOLING LAYER; F: FULLY CONNECTED LAYER).

Name	DataBase	Topology	Acc_{orig}(%)	Acc_{xnor}(%)
LeNet-5	MNIST	3C,2S,1F	99.1	97.2
MLP	MNIST	5F	98.5	96.9
CNP	MNIST	3C,2S,1F	97.0	96.1
SCNN	MNIST	2C,2F	99.0	97.8
MCDNN	MNIST	3C,3S,3F	96.8	95.7
AlexNet	ImageNet	5C,3S,2F	80.2	69.2
ResNet-18	ImageNet	18C,2S,1F	89.2	73.2

Scheme: We compare XNOR-POP against several counterparts shown in Table III. Our baseline is ShiDianNao, a mobile accelerator for processing simple CNNs. Eyeriss is another mobile CNN accelerator for solving complex CNNs, e.g., AlexNet. YodaNN takes advantage of BinaryConnect to test CNNs with only additions and subtractions. XNOR-FPGA is a FPGA implementation for BinaryNet [5]. We assume it can run both simple and complex CNN tests. All mobile accelerators frequently access main memory to fetch weights, especially for complex CNNs with large volumes of parameters. We customized a mobile version of ReRAM PIM ISAAC by keeping 3 out of original 168 tiles. Due to less tiles, ISAAC needs to write weights into ReRAM arrays during testings. We assume writing $128\times12\times3$ cells costs $100ns$. We also consider the HMC PIM Neurocube. Except Neurocube, we assume *1GB Wide-IO2 DRAM for all schemes*, since they need memory space to store weights and inputs during CNN tests. We wrote an in-house functional simulator to estimate the performance and energy of these schemes by assuming they can always achieve their peak computing throughput.

TABLE III
SCHEME COMPARISON ($Power_{acc}$ IS SCALED TO $32nm$).

Name	Description	$Power_{acc}$	$Power_{mem}$
ShiDianNao [7]	simple CNNs	$320mW$ [7]	$1.91W$ [31]
Eyeriss [2]	complex CNNs	$278mW$ [2]	$1.91W$ [31]
YodaNN [1]	BinaryConnect	$248mW$ [1]	$1.91W$ [31]
XNOR-FPGA [22]	BinaryNet FPGA	$8.2W$ [22]	$1.91W$ [31]
Neurocube [17]	HMC PIM	$21.2W$ [17]	$57.8W$ [17]
ISAAC [28]	ReRAM PIM	$990mW$ [28]	$1.91W$ [31]
XNOR-POP	XNOR-Net PIM	$237mW$	$1.99W$

V. EVALUATION

Frame-per-second: The FPS comparison of all accelerators is shown in Figure 8. For **simple** CNNs such as LeNet-5, MLP, CNP, SCNN and MCNN, all accelerators achieve > 200 FPS. Among all previous designs, YodaNN obtains the third best performance, since it processes CNNs by only additions and subtractions. Its binary weights also decrease off-chip DRAM traffic. On average, compared to YodaNN, XNOR-FPGA improves CNN test performance by $2.88\times$, because it performs only XNORs and popcounts on the FPGA. Both YodaNN and XNOR-FPGA are able to attain > 10000 FPS for simple CNNs. Averagely, compared to XNOR-FPGA, XNOR-POP improves CNN test performance by $4.03\times$, since 2KB

DRAM row buffers and XLU significantly enhance XNOR operation parallelism. Only for SCNN, XNOR-POP has longer test latency, due to the SCNN topology thrashes DRAM row buffers. For **complex** CNNs, XNOR-FPGA delivers only 247 FPS during AlexNet tests and 196 FPS during ResNet-18 tests. On the contrary, XNOR-POP reaches 3390 FPS for AlexNet and 1391 FPS for ResNet-18.

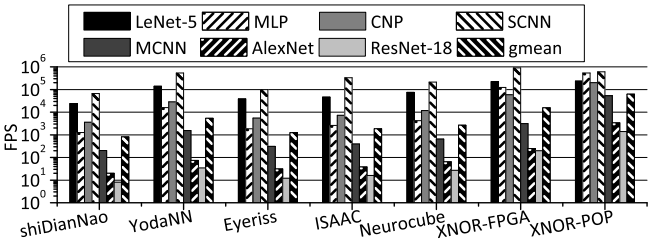

Fig. 8. FPS comparison.

Energy consumption: The energy consumption of all accelerators is exhibited in Figure 9, where all bars are normalized to ShiDianNao. YodaNN decreases CNN test energy over ShiDianNao averagely by 83.4%, mainly because of its shorter test latency. Although, compared to Eyeriss, ISAAC boosts CNN test performance by 47% on average, it also increases power consumption by 33% due to its power hungry ADCs and DACs, so they have similar energy consumptions. Compared to ShiDianNao, Eyeriss and ISAAC reduce test energy by 36% and 42% respectively. Averagely, Neurocube increases CNN test energy by $10.94\times$ due to its huge power HMC interface. Therefore, Neurocube is not the right CNN accelerator option for mobile devices. Although XNOR-FPGA achieves the best performance among all previous designs, it consumes larger energy than YodaNN, and reduces test energy by 76% over ShiDianNao averagely. On average, XNOR-POP reduces CNN test energy substantially by 98.7%.

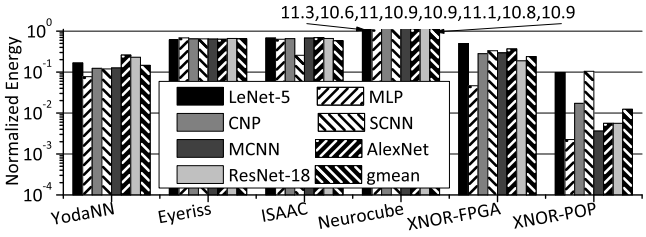

Fig. 9. Energy comparison (Normalized to ShiDianNao).

Multiple concurrent tests: With 30 FPS, XNOR-POP can support $>$ 46 concurrent applications even when running complex CNN tests. A larger capacity DRAM stores more weights and reduces weight updates due to spatial conflicts. With 30 FPS, compared to 1GB DRAM, 2GB (4GB) increases the concurrent test number by 11% (30%) averagely.

VI. Conclusion

In this paper, we present a Wide-IO2 DRAM PIM, XNOR-POP, to accelerate CNN tests for low power mobile devices by massive XNORs and popcounts. Compared to state-of-the-art accelerators, on average XNOR-POP improves CNN test performance by $4\times \sim 11\times$ and reduces CNN test energy by $> 90\%$. It can guarantee 30 FPS when running complex CNN (AlexNet and ResNet-18) tests for $>$ 46 simultaneous applications in mobile devices.

References

[1] R. Andri, *et al.*, "YodaNN: An Ultra-Low Power CNN Accelerator Based on Binary Weights," in *ISVLSI*, pages 236–241, July 2016.

[2] Y. H. Chen, *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *ISSCC*, 2016.

[3] P. Chi, *et al.*, "PRIME: A Novel PIM Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *ISCA*, 2016.

[4] D. Ciregan, *et al.*, "Multi-column deep neural networks for image classification," in *CVPR*, 2012.

[5] M. Courbariaux, "BinaryNet: Training DNNs with Weights and Activations Constrained to +1 or -1," *CoRR*, abs/1602.02830, 2016.

[6] M. Courbariaux, *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.

[7] Z. Du, *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *ISCA*, 2015.

[8] C. Farabet, *et al.*, "CNP: An FPGA-based processor for Convolutional Networks," in *FPL*, 2009.

[9] W. Glauser, "Doctors among early adopters of Google Glass," *Canadian Medical Association Journal*, 185(16):1385, 2013.

[10] S. Gupta, *et al.*, "Deep Learning with Limited Numerical Precision," *CoRR*, abs/1502.02551, 2015.

[11] S. Han, *et al.*, "MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints," in *MobiSys*, 2016.

[12] K. He, *et al.*, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[13] HP, "CACTI," `http://www.hpl.hp.com/research/cacti/`.

[14] B. Jacob, *et al.*, *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann, San Francisco, CA, 2007.

[15] JEDEC, "Wide I/O 2 Mobile DRAM Standard," `http://www.jedec.org/standards-documents/results/jesd229-2`.

[16] Y. Jia, *et al.*, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[17] D. Kim, *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *ISCA*, 2016.

[18] M. Kim and P. Smaragdis, "Bitwise neural networks," in *ICML Workshop on Resource-Efficient Machine Learning*, 2016.

[19] A. Krizhevsky, *et al.*, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[20] Y. Lecun, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86(11), Nov 1998.

[21] S. Li, *et al.*, "Pinatubo: A PIM Architecture for Bulk Bitwise Operations in Emerging NVMs," in *DAC*, 2016.

[22] Y. Li, *et al.*, "A 7.663-TOPS 8.2-W Energy-efficient FPGA Accelerator for Binary Convolutional Neural Networks," in *FPGA*, 2017.

[23] R. Ramanarayanan, *et al.*, "A 2.1GHz 6.5mW 64-bit Unified PopCount/BitScan Datapath Unit for 65nm High-Performance Microprocessor Execution Cores," in *VLSID*, 2008.

[24] M. Rastegari, *et al.*, "XNOR-Net: Imagenet Classification Using Binary Convolutional Neural Networks," in *ECCV*, 2016.

[25] J. Redmon, *et al.*, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[26] S. E. Schuster and R. E. Matick, "Fast Low Power eDRAM Hierarchical Differential Sense Amplifier," *JSSC*, 44(2):631–641, Feb 2009.

[27] V. Seshadri, *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *IEEE CAL*, 14(2):127–131, July 2015.

[28] A. Shafiee, *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.

[29] P. Y. Simard, *et al.*, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," in *ICDAR*, 2003.

[30] S. Sinha, *et al.*, "Exploring Sub-20Nm FinFET Design with Predictive Technology Models," in *DAC*, 2012.

[31] Y. J. Yoon, *et al.*, "An 1.1V 68.2GB/s 8Gb Wide-IO2 DRAM with non-contact microbump I/O test scheme," in *ISSCC*, 2016.

[32] C. Zhang, *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.