XOR-Net: An Efficient Computation Pipeline for Binary Neural Network Inference on Edge Devices

Shien Zhu, Luan H. K. Duong, Weichen Liu
School of Computer Science and Engineering, Nanyang Technological University, Singapore
Email: shien001@e.ntu.edu.sg, {lhkduong, liu}@ntu.edu.sg

2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS) | 978-1-7281-9074-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICPADS51040.2020.00026

***Abstract*—Accelerating the inference of Convolution Neural Networks (CNNs) on edge devices is essential due to the small memory size and poor computation capability of these devices. Network quantization methods such as XNOR-Net, Bi-Real-Net, and XNOR-Net++ reduce the memory usage of CNNs by binarizing the CNNs. They also simplify the multiplication operations to bit-wise operations and obtain good speedup on edge devices. However, there are hidden redundancies in the computation pipeline of these methods, constraining the speedup of those binarized CNNs.**

In this paper, we propose XOR-Net as an optimized computation pipeline for binary networks both without and with scaling factors. As XNOR is realized by two instructions XOR and NOT on CPU/GPU platforms, XOR-Net avoids NOT operations by using XOR instead of XNOR, thus reduces bit-wise operations in both aforementioned kinds of binary convolution layers. For the binary convolution with scaling factors, our XOR-Net further rearranges the computation sequence of calculating and multiplying the scaling factors to reduce full-precision operations. Theoretical analysis shows that XOR-Net reduces one-third of the bit-wise operations compared with traditional binary convolution, and up to 40% of the full-precision operations compared with XNOR-Net. Experimental results show that our XOR-Net binary convolution without scaling factors achieves up to 135× speedup and consumes no more than 0.8% energy compared with parallel full-precision convolution. For the binary convolution with scaling factors, XOR-Net is up to 17% faster and 19% more energy-efficient than XNOR-Net.

***Keywords*-CNN Acceleration; Neural Network Quantization; Binary Neural Networks; Edge Devices;**

I. Introduction

Deep Convolution Neural Networks (CNNs) are both computation and memory intensive. For example, state-of-the-art CNN EfficientNet-B7 [1] has 66 million parameters with 37 billion Float-Point Operations (FLOPS). Such large CNNs are hard to be deployed on edge devices with limited computation resources and small memory. Therefore, acceleration methods including transformation [2], pruning [3], and quantization [4] have been proposed to alleviate this execution problem.

Among the acceleration methods, quantization utilizes low-precision numbers instead of 32/64-bit floating-point numbers to represent the weights and activations, thus reduces the storage cost for CNN inference. Quantization also brings lower latency because low-precision computations are faster than full-precision ones and may reduce the hardware complexity of accelerators. For example, QNNPACK [5] utilizes 8-bit integers and achieves high-performance CNN inference on mobile platforms.

Among these quantization methods, Binary Neural Network [6] quantizes both the filters and the activations to 1-bit numbers and achieves extreme storage-saving and speedup. As both the activations and the filters are '0's and '1's, the binary convolution is now equivalent to the XNOR operations followed by pop-count (counting the number of "1"s in a binary integer) operations, resulting in much faster computation speed than the convolution using 32-bit float point Multiplication Accumulation (MAC) operations. However, binarization brings large accuracy loss. So XNOR-Net [7] and XNOR-Net++ [8] add 32-bit full-precision scaling factors to the quantized filters and activations to improve the accuracy of binarized networks.

However, not only these works but also latest works such as Bi-Real-Net [9] and CI-BCNN [10] take XNOR and pop-count operations for binary convolution without considering the hardware implementation. As there is no XNOR instruction on most CPU and GPU platforms [11]–[14], they have to conduct XNOR using two instructions (XOR and NOT) instead of one and degrade the speedup of binary neural networks. Besides, there exist many hidden redundancies in XNOR-Net when calculating the scaling factors and getting the final output, which are hardly noticed unless viewing the combined execution process in consecutive layers. We have observed that these redundant full-precision operations account for 25-40% FLOPS in convolution layers and seriously affect the computation efficiency.

In this paper, we propose XOR-Net as an efficient binary network inference method to solve these problems. First, as XOR is a universal instruction in off-the-shelf CPU and GPU platforms [11]–[14], XOR-Net uses XOR instead of XNOR for binary convolution and finishes the bit-wise operation within one cycle instead of two, reducing all the NOT operations compared with traditional methods. Second, for those binary networks with scaling factors, XOR-Net moves the multiplication of the scaling factor matrix to the next layer and moves constants to the scaling factor of the weights, so XOR-Net further reduces up to 40%

2690-5965/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPADS51040.2020.00026

full-precision operations compared with XNOR-Net. By carefully modifying the following computation after the bit-wise operations, XOR-Net produces the same convolution results and keeps the same neural network accuracy as traditional binary methods without bringing new overhead.

Our proposed XOR-Net can be implemented on all general-purpose platforms including CPU and GPU which provide XOR and pop-count instructions. We implement XOR-Net on a RISC-V based edge device GreenWaves GAP8, taking advantage of reduced branching operations, loop unrolling, and bit-level and filter-level parallelism. We evaluate the actual speedup and energy consumption of XOR-Net on the edge device at the layer level with different configurations on the input size, the input channel, and the filter number. Experimental results show that XOR-Net achieves 81-135$\times$ speedup and consumes no more than 0.8% energy compared with the parallel full-precision layers. For binary convolution with scaling factors, XOR-Net is 10-17% faster and 19% more energy-efficient than XNOR-Net. Please note that the speedup and energy efficiency are gained without any accuracy cost.

The rest of this paper is organized as follows. Section 2 introduces related works and Section 3 discusses existing binary convolution methods and our observations. Section 4 describes our proposed XOR-Net with a theoretical analysis provided, while Section 5 details our XOR-Net binary convolution implementation. Section 6 provides the experimental results and Section 7 concludes our works.

II. Related Works

Quantization is a popular CNN acceleration method. For example, 16/8-bit quantization has been adopted by deep learning frameworks TensorFlow Lite and PyTorch, and hardware accelerators Google TPUs and Nvidia GPUs.

Among different quantization methods, 1-bit quantization is an extreme case, which has been proposed and studied by many works. Binary Connect [15] quantizes the weights into {+1, -1} to replace the multiplication operations with additions and subtractions. BNN [6] quantizes both the input activations and weights into one bit to achieve the extreme speedup. XNOR-Net [7] and XNOR-Net++ [8] add scaling factors to the binarized weights and activations to improve the accuracy. Bi-Real-Net [9] adds real value activation short cuts to improve the information representation ability and CI-BCNN [10] mines channel-wise interactions to reduce the sign error, but these works focus on improving the accuracy without considering the inefficiency of XNOR operations. Meanwhile, BMXNet v1 and v2 [16], [17] implement binary convolution layers including XNOR-Net in MXNet and optimize the GEMM kernels for high speedups, but neither of the works has noticed the computation redundancy in XNOR-Net nor removed them.

III. Motivation

Existing methods such as BNN, Bi-Real-Net, and CI-BCNN use binary convolution without scaling factors, whose general steps include binarizing filter weights, binarizing the input activations, and performing bit-wise convolution. XNOR-Net and XNOR-Net++ use binary convolution with scaling factors, whose basic steps also include calculating the scaling factors of the filters and the activations as well as multiplying the scaling factors to the bit-wise convolution results.

A. BCNN: Binary Convolution without Scaling Factors

The binarization of input activations X and weights W is usually realized by getting the sign bits and then packing the single sign bits into 32/64-bit integers. So there will be high data parallelism when convoluting the quantized activations QX and weights QW to get the layer outputs O.

$$QX = sign(X) \tag{1}$$

$$QW = sign(W) \tag{2}$$

$$O = QX * QW \tag{3}$$

As the quantized activations and weights are all +1 and -1 represented by "0" and "1", the dot product inside the convolution can be finished using XNOR and pop-count. Suppose we need to calculate the dot product of two vectors V_{QX} and V_{QW} from the quantized activations and quantized weights respectively as equation (4)-(7) show. XNOR operations get "1" when the operands are both "0"s or "1"s, so the summation of the pop-count result sum stands for the number of +1 in the dot product result. N is the total bits of the XNOR result, so $N - sum$ is the number of -1 in the dot product result. Finally, the dot product result is obtained as equation (6)-(7) show.

$$sum = popcnt(V_{QX}\ XNOR\ V_{QW}) \tag{4}$$

$$= popcnt(NOT(V_{QX}\ XOR\ V_{QW})) \tag{5}$$

$$V_{QX} \cdot V_{QW} = sum - (N - sum) \tag{6}$$

$$= 2 \times sum - N \tag{7}$$

We notice that the XNOR is realized using XOR and NOT operations inside CPU and GPU platforms [11]–[14], so it is not efficient to use XNOR for the quantized convolution. We can reduce the number of bit-wise operations in the binary convolution by using XOR instead of XNOR. Thus we propose another convolution scheme called XOR-Net which uses XOR and pop-count to achieve the same functionality without the NOT operation.

B. XNOR-Net: Binary Convolution with Scaling Factors

1) Calculating the Scaling Factors: In XNOR-Net, the scaling factor α of a filter is the average absolute value of its weights. $W \in \mathbb{R}^{c \times k_h \times k_w}$ is the weight tensor with three dimensions: the channel, the kernel height, and the kernel

width. $n = c \times k_h \times k_w$ is the total number of weights in a filter. The scaling factors of filters are calculated during training, so they have no computation cost in inference.

$$\alpha = \frac{1}{n}\|W\|_{L1} \tag{8}$$

As equation (9)-(10) show, $X \in \mathbb{R}^{c\times h\times w}$ is the 32-bit full-precision input tensor with three dimensions: channel, height, and width. The original XNOR-Net performs the following steps to get the scaling factor matrix of the input activations. First, calculate the average absolute value matrix $A \in \mathbb{R}^{h\times w}$ of the input tensor X across the channel. Second, do a fake convolution between A and $I \in \{1\}^{k_h\times k_w}$, a matrix full of ones with the same size as kernels. The fake convolution adds up the corresponding elements of A and gets the scaling factor matrix $K \in \mathbb{R}^{o_h\times o_w}$ of the activations.

$$A = \frac{1}{c}\sum_{j=1}^{c}\|X_{j,:,:}\| \tag{9}$$

$$K = \frac{1}{k_h k_w} A * I \tag{10}$$

However, since the numbers of the input channel, the kernel height, and the kernel width are known before getting the input activations, there is no need to multiply constants $\frac{1}{c}$ and $\frac{1}{k_w k_h}$ when calculating each scaling factor matrix of the activations. We can move these two constants to the scaling factors of the weights that are calculated during training to reduce the computation cost of inference.

2) Multiplying the Scaling Factors: In the last step of XNOR-Net, the bit-wise convolution result needs to multiply the scaling factors mentioned in the previous subsection. The bit-wise convolution result $O \in \mathbb{N}^{o_h\times o_w}$ multiplies the scaling factor matrix K of the input activation in an element-wise manner $\odot$, then multiplies the scaling factor α of the filter weights. After all, we get $Y \in \mathbb{R}^{o_h\times o_w}$ the final output of convolution with one filter.

$$O = QX * QW \tag{11}$$

$$Y = O \odot K \cdot \alpha \tag{12}$$

The original XNOR-Net produces the final output by multiplying the same scaling factor matrix K on the bit-wise convolution result with every filter O. As Fig.1 (a) shows in the bracket, this leads to many repeating multiplications at the matrix level. Viewing the whole computation pipeline across layers in Fig.1 (a), we observe that there is an Avg. function (get the average absolute values across the channel) in the next layer. Therefore, we can move the element-wise multiplication K outside the average function to remove the unnecessary calculations as Fig.1 (b) shows.

IV. Proposed Binary Convolution

We propose XOR-Net in this paper to avoid using the NOT operations repeatedly. Our XOR-Net utilizes XOR, a universal instruction in off-the-shelf CPU and GPU platforms, and pop-count as the main operations in the bit-wise convolution. Because both the binary convolution layers with and without scaling factors need to conduct bit-wise convolution, XOR-Net is applicable to both of them. Especially, for the binary convolution with scaling factors like XNOR-Net, XOR-Net rearranges the computation pipeline based on the motivations mentioned in the previous section and further reduces the number of full-precision operations dealing with scaling factors.

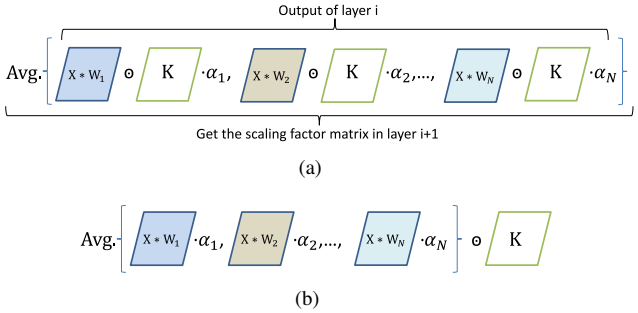

Figure 1. Illustration of the computation pipeline of binary convolution with scaling factors in consecutive layers. (a) The original XNOR-Net algorithm. (b) The proposed optimized XOR-Net algorithm.

A. XOR-Net: Binary Convolution without Scaling Factors

XOR-Net keeps the same computation sequence as equation (1)-(3) but changes the dot product inside the bit-wise convolution according to equation (13)-(15). As the vectors V_{QX} and V_{QW} of quantized activations and quantized weights are all +1 and -1 represented by "0" and "1", XOR operations get "1" when the operands include one "0" and one "1", so the summation of the pop-count result sum stands for the number of -1 in the dot product result. Similarly, N is the total bits of the XOR result, so $N - sum$ is the number of +1 in the dot product result. Therefore, we obtain the dot product result using only one XOR and one pop-count. Compared with existing methods, XOR-Net reduces one NOT operation for all the quantized activations.

$$sum = popcnt(V_{QX}\ XOR\ V_{QW}) \tag{13}$$

$$V_{QX}\cdot V_{QW} = (N - sum) - sum \tag{14}$$

$$= N - 2\times sum \tag{15}$$

B. XOR-Net-S: Binary Convolution with Scaling Factors

For binary convolution with scaling factors, our XOR-Net moves the multiplication of some constants to the scaling factor of the weights based on the motivation in Section III.B. XOR-Net also changes the way to produce the output in consecutive convolution layers as Fig.1 (b) shows.

$$\alpha' = \frac{1}{k_h k_w}\frac{1}{c}\alpha = \frac{1}{ck_h k_w n}\|W\|_{L1} \tag{16}$$

Equation (16) shows the calculation of the new scaling factor α' of the filter weights. Usually, α' is determined

during training, so we do not need to calculate it any more in the inference. Our method can be applied to pre-trained original XNOR-Net models by calculating α' from the original scaling factor α, which is only a one-time cost.

$$A' = \sum_{j=1}^{c} \|X_{:,:,j}\| \tag{17}$$

$$K^{(i)} = \begin{cases} A' * I, & \text{the first layer} \\ (A' \odot K^{(i-1)}) * I, & \text{otherwise} \end{cases} \tag{18}$$

$$O = sign(X)sign(W) \tag{19}$$

$$Y = X * W \approx \begin{cases} O \cdot \alpha', & \text{not the last layer} \\ O \odot K^{(i)} \cdot \alpha', & \text{otherwise} \end{cases} \tag{20}$$

Equations (17)-(20) show the calculating of scaling factor matrix of the activations and getting the layer output. Our method follows the same basic logic as the original XNOR-Net algorithm and optimizes the computation pipeline across layers in convolution blocks. For the first convolution layer using our method, the average absolute value matrix A' will do the fake convolution to get the scaling factor matrix K. Then the intermediate result O will not multiply the scaling factor matrix K, but convey it to the next layer. If this is a consecutive layer after the first one, A' will multiply the scaling factor matrix of the previous layer $K^{(i-1)}$ to get the scaling factor matrix of this layer $K^{(i)}$, and $K^{(i)}$ will be conveyed to the next layer as well. Only when it's the convolution layer before pooling layers or at the end of the convolution block, the intermediate result O multiplies the scaling factor matrix $K^{(i)}$ to get the layer output.

C. Theoretical Operation Reduction

XOR-Net uses XOR and pop-count instead of XOR, NOT and pop-count for bit-wise convolution to remove the computation redundancy introduced by XNOR. So XOR-Net reduces one-third of bit-wise operations in binary convolution layers both with and without scaling factors. As for the full-precision operation reduction in binary convolution with scaling factors, we take VGG-16 and YOLOv2 as example CNNs to compare with the original XNOR-Net. Our XOR-Net-S is also applicable to more complicated CNNs such as inception and residual blocks as long as we multiply the scaling factor matrix in convolution layers before pooling, concatenation and addition layers/nodes. Table I lists the convolution layers of the example CNNs in blocks according to the pooling layers in between.

Compared with the original XNOR-Net, XOR-Net achieves up to 40% full-precision operation reduction in convolution layers with 3×3 kernels and about 25% full-precision operation reduction in those with 1×1 kernels. The convolution layers before pooling layers (such as layer 4, 7, 10 and 13 in VGG-16 and layer 5, 8 and 13 in YOLOv2) have to multiply the scaling factor matrix to produce the exact final output, so they have almost the same number of full-precision MACs as the original XNOR-Net.

D. Accuracy and Limitation

Our method maintains the same accuracy as the original methods like BNN, Bi-Real-Net and XNOR-Net. XOR-Net is an efficient computation pipeline for binary network inference, and it provides the same output as shown in the mathematical equations, so XOR-Net keeps the same accuracy as the training schemes. If binary networks achieve higher accuracy with new training methods, doing inference using XOR-Net will achieve the same high accuracy.

The limitation of XOR-Net lies in the binary convolution with scaling factors. Changing the computation sequence in XNOR-Net involves conveying the scaling factor matrix to the next convolution layer, which means, the layer outputs in the first and consecutive layers are not the same final results as the original XNOR-Net in these layers. This will not affect activation layers such as ReLU and Leaky ReLU. Because the scaling factor matrix only consists of positive numbers, the ReLU output only relies on the convolution result which contains both negative and positive numbers. However, we have to multiply the scaling factor matrix and cannot get high speedup compared with XNOR-Net in those convolution layers before pooling layers including Max Pooling and Average Pooling (e.g. VGG-16 layer 4). Because the input shape is different after pooling, the scaling factor matrix will not fit into the next convolution layer.

Table I
FULL-PRECISION MAC OPERATION REDUCTION RATIO OF THE PROPOSED XOR-NET ALGORITHM

VGG-16	Input size	Filter size	Ratio
Layer 3	64,112,112	128,64,3,3	39.39%
Layer 4	128,112,112	128,128,3,3	0.25%
Layer 5	128,56,56	256,128,3,3	39.69%
Layer 6	256,56,56	256,256,3,3	33.03%
Layer 7	256,56,56	236,256,3,3	0.13%
Layer 8	256,28,28	512,256,3,3	39.84%
Layer 9	512,28,28	512,512,3,3	33.18%
Layer 10	512,28,28	512,512,3,3	0.06%
Layer 11	512,14,14	512,512,3,3	33.25%
Layer 12	512,14,14	512,512,3,3	33.18%
Layer 13	512,14,14	512,512,3,3	0.06%
YOLOv2	Input size	Filter size	Ratio
Layer 3	64,56,56	128,64,3,3	39.39%
Layer 4	128,56,56	64,128,1,1	25.19%
Layer 5	64,56,56	128,64,3,3	0.30%
Layer 6	128,28,28	256,128,3,3	39.69%
Layer 7	256,28,28	128,256,1,1	25.10%
Layer 8	128,28,28	256,128,3,3	0.15%
Layer 9	256,14,14	512,256,3,3	39.84%
Layer 10	512,14,14	256,512,1,1	25.05%
Layer 11	256,14,14	512,256,3,3	39.77%
Layer 12	512,14,14	256,512,1,1	25.05%
Layer 13	256,14,14	512,256,3,3	0.08%

V. Implementation on An Edge Device

We implement XOR-Net convolution layers on GreenWaves GAP8, a RISC-V based ultra-low power edge processor showed in Fig.2. We take the convolution benchmark suites [18] developed by GreenWaves Technologies as reference implementations to optimize our benchmark codes to ensure a fair comparison. We utilize the APIs in GAP_SDK [19] for bit-insert, XOR, and pop-count operations. Packing the sign bits of the activations during quantization needs bit-insert, while bit-wise convolution uses XOR and pop-count as main operations. Our proposed implementation can be deployed to other platforms by simply replacing the bit-insert and pop-count instructions.

As binary convolution with scaling factors XOR-Net-S has two more steps than XOR-Net without scaling factors, we only introduce XOR-Net-S for simplicity. We implement the data preparation part of the proposed XOR-Net-S following Algorithm 1 and implement the bit-wise convolution part following Algorithm 2. Excluding the statements concerning scaling factors, these algorithms will be binary convolution without scaling factors.

A. Data Preparation

Data preparation in XOR-Net-S includes packing the sign bits of the input tensor and calculating the scaling factor matrix of the input. First, we get the sign bits of the activations and pack the sign bits into 32/64-bit integers for bit-level parallelism. For example, one XOR operation on one packed integer equals to one operation on 64 original input elements after packing the sign bits into 64-bit integers. Therefore, packing the sign bits into integers brings high bit-level parallelism.

We pack the sign bits across the input channel following BitFlow [21], which keeps the logical input shape as CHW after the packing. The packing of sign bits is realized by the bit-insert instruction, i.e. inserting the first one bit of $X_{32pc+i,h,w}$ to the pack at offset i. In IEEE standard format, the sign bits of both int and float numbers are the first bit. We avoid if() statements or sign() functions in the loop by packing the first bit directly to the container to get good performance because branches degrade the packing speed.

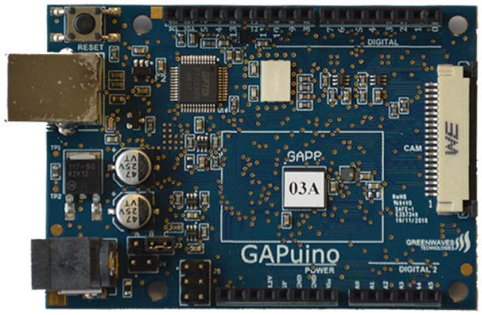

Figure 2. GreenWaves GAPuino development board. Source: [20]

Second, we decouple the calculation of scaling factors into two parts. The first part getting the average matrix of the input is combined with the packing sign bits, and the second part fake convolution is a standalone for() loop. Therefore, our implementation reduces a loop that goes through all the data, bringing better data locality and coding efficiency.

B. Bit-wise Convolution

We use XOR and pop-count operations to get the bit-wise convolution result O. In the previous section, we have proved that mathematically XOR-Net binary convolution results are the same as the binary convolution results that use XNOR and pop-count. For the reason that XOR is supported by the instruction sets of CPU and GPU platforms, and using XNOR has to do one more NOT operation on the XOR results, so our implementation reduces one NOT operation compared with traditional binary convolution.

Algorithm 1 Input binarization and calculating scaling factors

Input: input tensor X, scaling factor matrix from the previous layer $K^{(i-1)}$
Output: input sign tensor S, scaling factor matrix $K^{(i)}$

// Get the sign bits and the average matrix of X
packed channel = input channel / 32;
for each input height h **do**
  for each input width w **do**
    $sum = 0$
    for each packed channel pc **do**
      // Use a 32-bit integer to contain the sign bits
      pack=0
      for i from 0 to 31 **do**
        // Insert the sign bit to the pack
        $bitinsert(pack, X_{h,w,32pc+i}, 1, i)$
        // Sum up the absolute value of the input
        $sum+ = |X_{32pc+i,h,w}|$
      end for
      $S_{h,w,pc} = pack$
    end for
    $A_{h,w} = sum$, the first layer
    $A_{h,w} = sum \times K_{h,w}^{(i-1)}$, otherwise
  end for
end for
// Get the scaling factor matrix of the input tensor
for each input height h **do**
  for each input width w **do**
    $sum = 0$
    $sum+ = A_{h+0,w+0}$
    ...
    $sum+ = A_{h+kh,w+kw}$
    $K_{h,w}^{(i)} = sum$
  end for
end for
return $S, K^{(i)}$

Algorithm 2 XOR-Net bit-wise convolution

Input: input sign tensor S, weight sign tensor W and scaling factors α, input scaling factor matrix from the previous layer $K^{(i-1)}$

Output: convolution result Y, scaling factor matrix $K^{(i)}$

$N = c \times kh \times kw$
for each filter f **do**
  for each output height h **do**
    for each output width w **do**
      R=0
      for each packed channel c **do**
        $R+ = cnt(S_{c,h+0,w+0} \oplus W_{c,0,0})$
        ...
        $R+ = cnt(S_{c,h+kh,w+kw} \oplus W_{c,kh,kw})$
      end for
      $O_{f,h,w} = N - 2 \times R$
      $Y_{f,h,w} = O_{f,h,w} \times \alpha_f$, not the last layer
      $Y_{f,h,w} = O_{f,h,w} \times \alpha_f \times K^{(i)}_{h,w}$, otherwise
    end for
  end for
end for
return Y

Finally, we multiply the bit-wise convolution result O with the scaling factors of the activations and weights according to equation (20). What's more, we utilize filter level parallelism when implementing the bit-wise convolution. Since there are eight cores on GAP8, the filter number are usually multiples of 8 and there is no data dependency between the filters, it is an efficient approach to do parallel convolution across different filters. As for the binarization and scaling factor calculation, we perform the parallel processing across the input height/width/channel wherever possible.

VI. Evaluation

We evaluate full-precision convolution, XNOR-Net and XOR-Net-S (binary convolution with scaling factors), BCNN and XOR-Net (binary convolution without scaling factors) at layer level with different configurations on the input size, the input channel and the filter number. We employ a Ubuntu 16.04 based workstation as the host machine, then record the execution time by the hardware timer on GAP8, and record the power consumption through a USB power meter UM25C. Each test case is executed for at least 10 times to get the average execution time.

A. Increasing the Input Size

We increase the input size (C-H-W) of the convolution layers by two accordingly. There are 32 filters with 3×3 kernels in this experiment. Fig.3 shows the execution time of all the evaluated layers. The latency of binary convolution layers is much smaller than the latency of full-precision convolution. XOR-Net-S runs faster than XNOR-Net, and XOR-Net has less latency than BCNN as well.

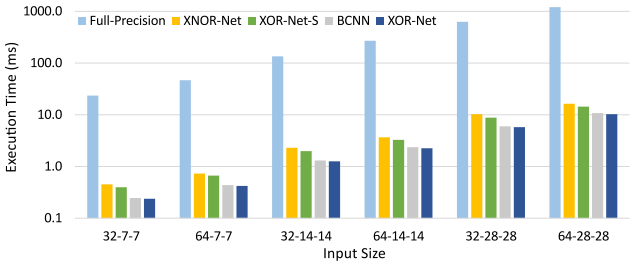

Figure 3. The execution time of different convolution layers when increasing the input size.

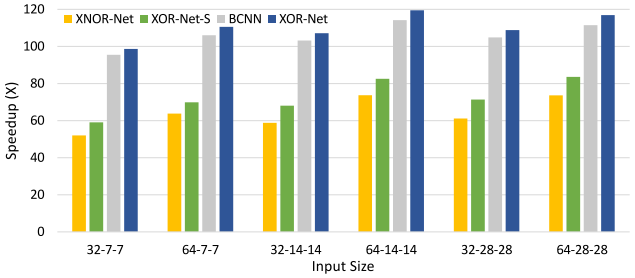

Figure 4. The speedup compared with parallel full-precision convolution when increasing the input size.

The speedup compared with parallel full-precision convolution is shown in Fig.4. The original XNOR-Net achieves 52-73× speedup while the optimized XOR-Net-S achieves 59-83× speedup. Compared with original XNOR-Net, our method XOR-Net-S has 10-17% speedup. Because there is no calculation dealing with scaling factors, BCNN achieves 95-114× speedup and XOR-Net has 98-119× speedup compared with parallel full-precision convolution. XOR-Net is 3-5% faster than BCNN because we have reduced a NOT operation in the binary convolution pipeline. As NOT is a simple bit-wise operation that finishes within one clock cycle, and the operands of the NOT operation are already at the register after the XOR operation, the overall performance gain of removing the NOT operation is moderate.

We notice that the speedup is relatively higher when the input channel is 64, so we explore the performance of XOR-Net by increasing the input channel in the next experiment.

B. Increasing the Input Channel

As observed in the previous section, we perform this experiment to find out the relationship between the speedup of XOR-Net and the input channel. We increase the input channel linearly from 32 to 192, with the input size set to be $C \times 14 \times 14$. There are still 32 filters with 3×3 kernels. The speedup compared with parallel full-precision convolution layers with the increasing input channel are shown in Fig.5.

Original XNOR-Net and XOR-Net-S have 58-89× and 68-95× speedup compared with full-precision convolution respectively, while BCNN and XOR-Net have 103-122× and 107-129× speedup respectively. When we increase the number of input channel linearly, the speedup first goes up

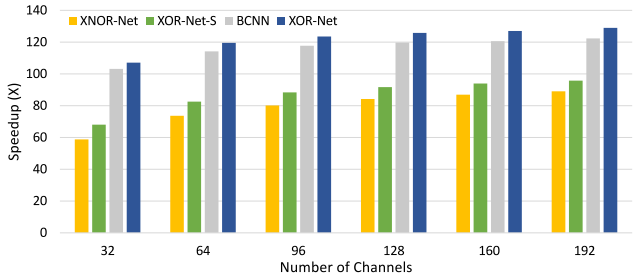

Figure 5. The speedup compared with parallel full-precision convolution when increasing the input channel.

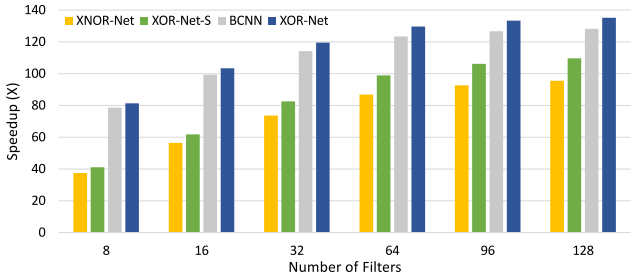

Figure 6. The speedup compared with parallel full-precision convolution when increasing the filter number.

then become steady. The reason behind such a phenomenon is that when increasing the input channel, we increase both the input depth and the quantization overhead. Increasing the input depth brings better data locality, but the data locality benefit will not grow much when the input depth is too large. What's more, increasing the input channel also increases the binarization and bit-packing workload. As the data preparation overhead increases with the input channel, the speedup cannot grow linearly with the input channel.

C. Increasing the Filter Number

The data preparation overhead in the binary convolution may affect the speedup. We keep the data preparation computation as constant by increasing the filter number only to verify this idea. The input size is $64 \times 14 \times 14$, and we still use the popular 3×3 kernels in this experiment. Though popular networks seldom use a small number of filters in a convolution layer, we include small filter numbers for experiment purpose. As Fig.6 shows, we obtain the highest speedup in all the experiments when the filter number is largest: XNOR-Net and XOR-Net-S have $95\times$ and $109\times$ speedup respectively, and BCNN and XOR-Net have $128\times$ and $135\times$ speedup respectively.

As Fig.6 shows, the speedup grows with the increment of the filter number. The bit-wise convolution accounts for more time as the filter number increases, and the speedup increases with the bit-wise convolution ratio. This observation is consistent with the basic logic of binary convolution: using much faster XOR/XNOR and pop-count operations instead of full-precision multiplication and accumulation operations to accelerate CNNs. Therefore, we can get higher speedup by reducing the data preparation overhead or increasing the bit-wise convolution ratio in the convolution layer.

D. Power Consumption Analysis

We report the energy efficiency of XOR-Net and other binary convolution methods in Table II. The input size for the convolution layer is $64 \times 14 \times 14$ and there are 128 filters with 3×3 kernels. The resolution of voltage of our power meter UM25C is 0.001V (error: 0.05%) and the current resolution is 0.0001A (error: 0.1%) [22], which means that we can measure the power to 0.0001 mW. We report the power to 0.01 mW for reliability. We execute the full-precision convolution layer for 20 times and XNOR-Net layers for 2500 times to make sure there are enough points of power measurement, then calculate the average value of a single inference. The speedups in Table II are slightly different from those reported in the previous subsection because this is another run.

Binary convolution layers consume no more than 1% energy compared with full-precision convolution. Our proposed method XOR-Net-S achieves $125\times$ energy efficiency compared with parallel full-precision convolution, and saves 16% energy compared with the original XNOR-Net. XOR-Net binary convolution without scaling factors achieves $159\times$ energy efficiency compared with full-precision convolution and is 10% more energy-efficient than BCNN. Though the speed benefits of XOR-Net compared with BCNN is moderate, the energy-efficiency of XOR-Net is good due to the lower arithmetic density.

VII. Conclusion

We proposed an optimized computation pipeline XOR-Net for binary network inference on edge devices. For the binary convolution without scaling factors, XOR-Net uses XOR instead of XNOR in the bit-wise convolution to save a NOT operation. For the binary convolution with scaling factors, XOR-Net-S further reduces the redundant full-precision operations. Viewing the whole computation pipeline across consecutive convolution layers, XOR-Net-S moves two constants to the scaling factor of the weights and multiplies the scaling factor matrix of the activations in the next layer wherever possible. Theoretical analysis shows that XOR-Net reduces one-third of the bit-wise operations compared with BCNN and XNOR-Net-S further reduces up to 40% full-precision operations compared with XNOR-Net while keeping the same accuracy.

We implemented XOR-Net on an edge device with bit-level and filter-level parallelism. The experiment results show that our optimized XOR-Net achieves 81-$135\times$ speedup and about $159\times$ energy efficiency compared with full-precision layers, and 3%-5% speedup and about 10% energy efficiency compared with traditional BCNN. The optimized binary convolution with scaling factors XOR-Net-

Table II
EXECUTION TIME AND ENERGY CONSUMPTION OF FULL-PRECISION AND BINARY CONVOLUTION LAYERS IN A GIVEN CONFIGURATION.

Conv Layer Type	Power(mW)	Time(ms)	Speedup	Energy(mJ)	Energy Ratio
Full-Precision	430.03	1074.83	1.0×	462.21	100.00%
XNOR-Net	395.76	11.14	96.5×	4.41	0.95%
XOR-Net-S	380.22	9.76	110.2×	3.71	0.80%
BCNN	384.93	8.28	129.7×	3.19	0.69%
XOR-Net	369.27	7.88	136.4×	2.91	0.63%

S is 10-17% faster improves 19% energy-efficiency compared to the original XNOR-Net. Exploring the performance by increasing the input channel and the filter number, we observe that XOR-Net can achieve higher speedup with more input channels and filters in the convolution layers.

ACKNOWLEDGMENT

This work is partially supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MOE2019-T2-1-071) and Tier 1 (MOE2019-T1-001-072), and partially supported by Nanyang Technological University, Singapore, under its NAP (M4082282) and SUG (M4082087).

REFERENCES

[1] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[2] A. Xygkis, D. Soudris, L. Papadopoulos, S. Yous, and D. Moloney, "Efficient winograd-based convolution kernel implementation on edge devices," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[3] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 2130–2141.

[4] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.

[5] H. L. Marat Dukhan, Yiming Wu and B. Maher, "QNNPACK." [Online]. Available: https://github.com/pytorch/QNNPACK

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 4107–4115.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[8] A. Bulat, G. Tzimiropoulos, and S. A. Center, "XNOR-Net++: Improved binary neural networks," *The British Machine Vision Conference (BMVC)*, 2019.

[9] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-Real Net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[10] Z. Wang, J. Lu, C. Tao, J. Zhou, and Q. Tian, "Learning channel-wise interactions for binary convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 568–577.

[11] Arm Limited, "Arm® Instruction Set Reference Guide." [Online]. Available: static.docs.arm.com/100076/0100/arm_instruction_set_reference_guide_100076_0100_00_en.pdf

[12] Intel Corporation, "Intel® 64 and IA-32 Architectures Software Developer's Manual." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf

[13] RISC-V Foundation, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2." [Online]. Available: https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[14] NVIDIA Corporation. (2020) Parallel Thread Execution ISA Version 7.0. [Online]. Available: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[16] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel, "Bmxnet: An open-source binary neural network implementation based on mxnet," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1209–1212.

[17] J. Bethge, M. Bornstein, A. Loy, H. Yang, and C. Meinel, "Training competitive binary neural networks from scratch," *ArXiv e-prints*, 2018.

[18] GreenWaves Technologies, "GAP8 BenchMark Test Suite," Dec. 2019. [Online]. Available: https://greenwaves-technologies.com/manuals/BUILD/BENCHMARKS/html/index.html

[19] ——. (2019) SDK Manuals. [Online]. Available: https://greenwaves-technologies.com/sdk-manuals/

[20] ——. (2020) Gapuino development board. [Online]. Available: https://greenwaves-technologies.com/product/gapuino/

[21] Y. Hu, J. Zhai, D. Li, Y. Gong, Y. Zhu, W. Liu, L. Su, and J. Jin, "Bitflow: Exploiting vector parallelism for binary neural networks on cpu," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 244–253.

[22] RuiDeng Technologies, "UM25C USB tester meter Instructions," Jul. 2020. [Online]. Available: https://phuketshopper.com/software/UM25C/UM25C%20USB%20tester%20meter%20Instructions.pdf