2019 IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)

LP-BNN: Ultra-low-Latency BNN Inference with Layer Parallelism

Tong Geng[1,2], Tianqi Wang[1], Chunshu Wu[1], Chen Yang[1], Shuaiwen Leon Song[2], Ang Li[2], and Martin Herbordt[1]

[1]Boston University, Boston, MA - {tgeng, tianqi, happycwu, cyang90, herbordt}@bu.edu
[2]Pacific Northwest National Laboratory, Richland, WA - {shuaiwen.song, ang.li}@pnnl.gov

***Abstract*—High inference latency seriously limits the deployment of DNNs in real-time domains such as autonomous driving, robotic control, and many others. To address this emerging challenge, researchers have proposed approximate DNNs with reduced precision, e.g., Binarized Neural Networks (BNNs). While BNNs can be built to have little loss in accuracy, latency reduction still has much room for improvement. In this paper, we propose a single-FPGA-based BNN accelerator that achieves microsecond-level ultra-low-latency inference of ImageNet, LP-BNN. We obtain this performance via several design optimizations. First, we optimize the network structure by removing Batch Normalization (BN) functions which leads to significant latency in BNNs without any loss on accuracy. Second, we propose a parameterized architecture which is based on layer parallelism and supports nearly perfect load balancing for various types of BNNs. Third, we fuse all the convolution layers and the first fully connected layer. We process them in parallel through fine-grained inter-layer pipelining. With our proposed accelerator, the inference of binarized AlexNet, VGGNet, and ResNet are completed within 21.5μs, 335μs, and 67.8μs respectively, with no loss in accuracy as compared with other BNN implementations.**

***Index Terms*—BNN, FPGA, Deep Learning, Ultra Low-latency**

I. Introduction

The past decade has witnessed the emergence and widespread adoption of Deep Neural Networks (DNNs), not only in image processing and speech recognition but also in High Performance Computing domains such as extreme big-data processing for in-situ analysis [6]. To continue to improve prediction accuracy, DNNs are becoming ever deeper and more complex, leading to increasingly long processing latency and high resource demands. A large number of accelerator designs have been proposed for DNN inference [9]. However, these designs have achieved only limited benefit in real-time domains with strict latency constraints such as autonomous driving and robotic control.

Since DNNs can often tolerate some inaccuracy, researchers have begun to explore reduced bit-width for DNN training and inference [2] [3] [24]; Binarized-Neural-Networks (BNNs) [4] in particular have gained much attention. BNNs use a single bit to encode each neuron and parameter, thus significantly reducing computational complexity and memory demand, and potentially reducing inference latency by orders-of-magnitude. BNNs map particularly well to FPGAs whose configurability enables millions of one-bit ALUs to be implemented on a single device [16] [21]. In contrast, a recent report from Intel [5] indicates that only 10% and 7% peak performance can be achieved when running a BNN on a Xeon CPU and a Titan X GPU, respectively (with a batch size of 10).

Despite the attractiveness of mapping BNNs to FPGAs, creating an efficient design that minimizes the inference latency of large networks with big data inputs still poses a great challenge. To the best of our knowledge, the shortest reported inference latency of AlexNet inference is around 1.16ms [7], which is still far from ideal for real-time scenarios. This delay is mainly due to three factors:

1) The critical Normalization Layer (NL) [4] [8] uses full-precision floating point operations, i.e., two FP MUL/DIV and three FP ADD/SUB. These operations and their parameters incur significant latency along with large storage demands compared with the other sub-layers. Although some researchers tried to simplify NL function by using threshold-based comparison, they did not apply their approach to large scale datasets such as ImageNet and their approach is not useful for the networks with more complex structure, e.g., ResNet [10].

2) Previous designs accelerate BNNs by exploiting data parallelism with layers processed sequentially. Hence, the overall latency is the accumulated latency of each layer, plus the communication and reconfiguration overheads. As a result, latency expands with network depth. And since a layer cannot start processing until the previous layer has finished, large on-chip storage is required to buffer all of the intermediate data between layers.

3) To efficiently process a large BNN in a single FPGA, each layer must be (1) optimally designed and (2) avoid reconfiguration. To the best of our knowledge, no existing design does both.

Our main contribution is to address these three challenges and demonstrate a single-FPGA design that guarantees μs-level inference latency for ImageNet. We achieve this performance via the following innovations. First, since BNNs remain computationally intensive (because of the floating point in the BNs), we optimize the structure of the BNN by fusing the BNs and several other sub-functions (Activation and Binarization). Our optimization can be applied not only for conventional CNNs, e.g., AlexNet and VggNet but also CNNs with shortcuts, e.g., ResNet. Second, we fuse all of the convolution layers and the first fully-connected (FC) layer. We process them in parallel through fine-grained inter-layer

2160-052X/19/$31.00 ©2019 IEEE
DOI 10.1109/ASAP.2019.00-43

pipelining. Therefore, the latencies of different layers are overlapped to the point that the overall latency of the inference is just the latency of a single convolution layer plus the two FC layers. Note that the dependency pattern of the last two FC layers prevents them from being fused with the others. Third, unlike the majority of previous FPGA-based designs, which use data parallelism, we propose a parameterized architecture based on layer parallelism. This design supports nearly perfect load balancing for various types of BNNs, leading to nearly 100% pipeline utilization. We also propose a Design Space Exploration (DSE) method to determine the values of the parameters used in the proposed architecture. The single-FPGA design achieves 21.5, 335 and 67.8μs inference latency for Binarized AlexNet, VggNet, and ResNet-18 of ImageNet.

II. Preliminary and Related Work

A. Preliminary of BNN

BNNs evolve from conventional DNNs through Binarized Weight Networks (BWN) [4]. It has been observed that if the weights can be binarized into 1 and -1, expensive floating-point multiplication can be eliminated since multiplication by 1 is equivalent to addition while multiplication by -1 is identical to subtraction. It has been further observed that if both the weights and inputs are binarized, even 32-bit additions and subtractions can be degraded to logical bit operations. Based on this observation, XNOR-Net is proposed and has become one of the most researched BNNs. In XNOR-Net, both the weights and the inputs of the convolutional and fully connected layers (except the first layer) are approximated with binary values. Binary weights and binary inputs allow an efficient way of implementing convolutional operations. If all of the operands of the convolutions are binary, then the convolutions can be estimated by XNOR and bit-counting operations [4]. In this paper, we focus on XNOR-Net. In the following sections, BNN refers to XNOR-Net.

The basic structure of BNNs has four essential functions in each CONV/FC layer: XNOR, POPCOUNT, Batch Normalization (BN), and Binarization (BIN) (illustrated in Figure 1). The weights, inputs, and outputs are binary, so multiply-accumulate in traditional DNNs becomes XNOR and Population Count (POPCOUNT) in BNNs. The output of POPCOUNT is normalized in BN which is compulsory for high accuracy in BNNs. Batch Normalization (BN) incorporates full-precision floating-point (FP) operations, i.e., two FP MUL/DIV and three FP ADD/SUB:

$$y_{i,j} = \left(\frac{x_{i,j} - \mathbb{E}[x_{*,j}]}{\sqrt{Var[x_{*,j}] + \epsilon}} \right) \cdot \gamma_j + \beta_j \qquad (1)$$

The normalized outputs from BN (i.e., $y_{i,j}$), which are floating point, are binarized in BIN by comparing with 0:

$$x^b = sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \qquad (2)$$

Here, BIN acts as the non-linear activation function. Max pooling can be required. Traditionally, pooling is between BN and BIN. However, it is equivalent to placing pooling after BIN so that the FP operations in pooling become bit-OR operations, significantly reducing computation complexity.

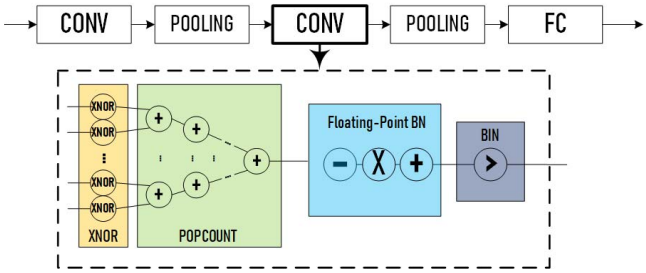

Fig. 1. The structures of BNNs with original BN in [4]

B. Related Work

There is much previous work in building high-performance BNNs to meet the requirements of real-time inference for delay-critical applications [5], [7], [10], [12], [13]. Because of FPGAs' flexibility and powerful bit-manipulation capability, the majority have been FPGA-based [5], [7], [10], [13]. Recently, a CPU-based BNN implementation was proposed [12] which uses bit-packing and AVX/SSE vector instructions to achieve good bit computation performance. However, in this design, Binarized CONV (BConv) is not supported directly; rather, BConv is converted to Binarized Matrix Multiplication (BMM) through the conventional flatten or unfold approach with expensive pre/post-processing for padding.

We now expand on the FPGA/BNN prior art. FINN [10] presents a framework for building fast and flexible FPGA accelerators using a flexible heterogeneous streaming architecture. This design can perform millions of classifications per second with sub-microsecond latency, thereby making them ideal for supporting real-time embedded applications. However, FINN only demonstrates μs-level-latency inference for small-scale datasets and networks, such as MNIST and CIFAR-10 and only support networks without shortcuts, i.e., ResNet is not supported. In this work, our accelerator provides μs-level-latency inference on large datasets such as ImageNet which are widely deployed in the real-life applications. Also, our design supports networks with branches, e.g., ResNet. Authors of FINN also observe the significant precision gap between BN (FP32) and Binarization (0/1) and eliminate this gap by simplifying BN without hurting the accuracy of the network. In FINN, the original expensive BN functions are replaced by threshold-based comparisons. By doing so, data from the POPCOUNT layer is binarized directly by comparison with a threshold, leading to potentially much lower latency and less hardware demand. However, this optimization proposed in FINN can only be applied to the old-fashioned networks which topology are quite straightforward and have no branches, bypasses or shortcuts. For modern and particularly important networks, such as ResNet, where intermediate results of different layers need to be added up before further batch normalization, FINN's optimization does

not fit. ReBNet is another well-known BNN implementation [11]. It provides an end-to-end framework for training reconfigurable binary neural networks in software and developing an efficient accelerator for execution on FPGAs. ReBNet improves classification accuracy by representing features with multiple levels of residual binarization. Their design supports ResNet and large-scale dataset, however, ReBNet keeps the expensive BN function in the original ResNet which leads to significant overhead on latency. In our design, we get rid of all BN functions in ResNet, therefore, providing significantly low-latency inference.

Furthermore, in most previous work, the network is processed layer by layer, which is similar to how conventional CNNs are processed and leads to large inference latency. The small model size of BNN creates the potential to process all layers simultaneously on a single FPGA, i.e. layer parallelism. However, there is one catch: low pipeline utilization which can only be solved by careful workload balancing, which has not been quantitatively utilized in current designs.

Although the majority of recent works focus mainly on the optimization of individual layers, we believe that inter-layer and whole-network optimization are even more important. For example, TensorFlow XLA [14] and Tensor Comprehensions [15] recently compiled entire neural network graphs at once, performing various transformations and achieving 4x speedup over the manually tuned individual layers. Our design follows this emerging trend by merging all the layers, except the last two FC layers, into a fused layer.

III. Network Structure Optimization

This section introduces the optimization approach applied in our work that removes all BN functions in BNNs. ResNet is used as a motivational example, as its topology includes not only shortcuts widely used in modern networks but also the classical topology of old-fashioned BNNs, such as AlexNet.

In ResNet, 2 neighbor CONV layers have heterogeneous structures, while every 2 adjacent layers share the same topology. Among 2 adjacent layers, one is the same as the CONV layer used in AlexNet ($layer2$ in Figure 2), and the other one is equipped with shortcuts ($layer1/layer3$ in Figure 2). Figure 2 (A) illustrates 3 adjacent CONV layers in ResNet. The BN outputs of the 1^{st} CONV layer, $NORM1$, are (1) binarized and then fed to the 2^{nd} CONV layer; (2) bypassed to the 3^{rd} layer. The 2^{nd} layer is the same as the CONV layer in binarized AlexNet. At the 3^{rd} layer, the outputs of POPCOUNT, $POP3$, are not directly normalized, instead, they are first summed with the bypassed $NORM1$.

As mentioned in Section II, using FINN's optimization, the BN function of $layer2$ can be easily replaced by a threshold-based comparison, while the ones in $layer1/layer3$ cannot be removed. In this work, we propose an optimization that replaces the expensive and complex BNs along with the summation in front of it at $layer1/layer3$ with threshold-based comparison and Threshold Look-up (TL). As shown in Figure 2 (B), the comparison is operated sequentially after $POP3$ is calculated, while TL can be operated in parallel with the inference of layer 2 and 3. By doing so, the serious overhead on latency incurred by the FP-based summation and the operations of BN functions is significantly reduced.

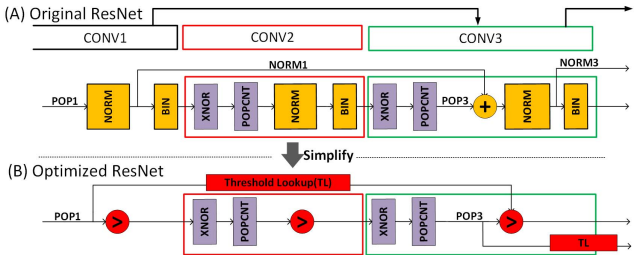

Fig. 2. The original and simplified structures of Binary-ResNet.

For layers without shortcuts (e.g. $layer2$), each output channel has its own constant threshold. The thresholds can be calculated based on Equation 3. L refers to the length of the vector counted in POPCOUNT, i.e. K× K× IC (K: filter size; IC: number of input channels). The POPCOUNT results are binarized by comparing to the corresponding threshold.

$$Threshold_{i,j,k} = \frac{\mathbb{E}_{*,j,k} + \mathbb{L}_{j,k}}{2} - \frac{\beta_{j,k} \cdot \sqrt{Var[x_{*,j,k}] + \epsilon}}{2 \cdot \gamma_{j,k}} \tag{3}$$

For layers with shortcuts (e.g. $layer3$), the calculation of threshold is more complex, as the value of threshold of each channel depends on not only the layer under calculation but also the layer feeding the bypassed data. In our optimization, when POP3 is calculated, it is binarized by comparing to a threshold which is calculated based on Equation 4, 5, 6 by TL. Here, $x_{i,j,k-2}$ are the popcount results which are the input of TL. Each output channel has its own constant $\alpha_{j,k}$ and $\theta_{j,k}$, which values are decided during the training phase.

$$Threshold\_s_{i,j,k} = \alpha_{j,k} \cdot x_{i,j,k-2} + \theta_{j,k} \tag{4}$$

$$\alpha_{j,k} = \frac{\gamma_{j,k-2}}{\sqrt{Var[x_{*,j,k-2}] + \epsilon}} \tag{5}$$

$$\theta_{j,k} = \frac{\mathbb{E}_{*,j,k} + \mathbb{L}_{j,k}}{2} - \frac{\beta_{j,k} \cdot \sqrt{Var[x_{*,j,k}] + \epsilon}}{2 \cdot \gamma_{j,k}} + \frac{\beta_{j,k-2}}{2} + \frac{(\mathbb{E}_{*,j,k-2} + \mathbb{L}_{j,k-2}) \cdot \gamma_{j,k-2}}{2 \cdot \sqrt{Var[x_{*,j,k-2}] + \epsilon}} \tag{6}$$

Using our optimization, every CONV/FC layer in BNNs have the following 3 functions: XNOR, POPCOUNT, and Threshold-based Comparison. For layers with shortcuts, one extra function, TL, is equipped and operated in parallel with other 3 functions. The proposed optimization of network structure provides significant potential to reduce the inference latency of BNN.

To evaluate the benefits of the proposed optimization, we use ImageNet ResNet-18, CIFAR-10 VGG-like, and ImageNet VGGNet as motivational examples to show the latency reductions of CONV and FC layers in these 3 networks by applying the proposed optimization. We also measure the percentage attributed to each function. Here we assume all functions in

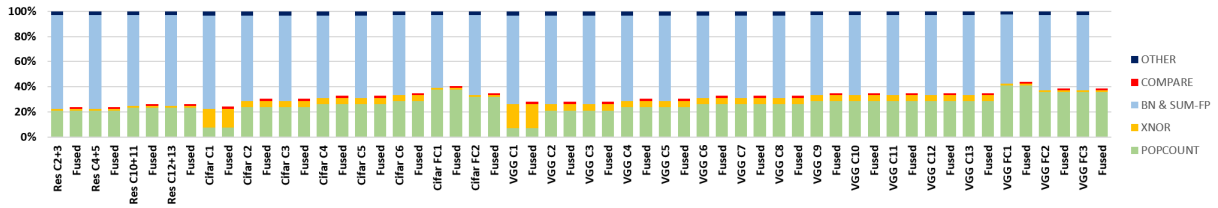

Fig. 3. Portion of execution time for different functions. 4 pairs of adjacent CONV layers in ImageNet ResNet-18, 6 CONV layers and 2 FC layers in Cirfar VGG, 13 CONV layers and 3 FC layers in ImageNet VGG are measured. For each layer, a pair of bars is given in this chart. The left and right ones are for original and optimized networks. The execution time of each function is divided by the execution time of the whole layer with the original network.

a single layer are processed in a pipelined manner and the data parallelisms are fully utilized. Figure 3 demonstrates that, without the proposed optimization, on average, 60% of the execution time is spent on processing the workload of BN, while only 40% time is for XNOR, POPCOUNT, and the others. With the proposed optimization, the latency of single layer processing is reduced, on average, to only 30%. For ResNet, the proposed optimization brings even more latency reduction because it gets rid of the floating-point operations in both BN and SUM (e.g. NORM1+POP3 in Figure 2).

IV. Design Methodology

A. Layer Fusion

We construct fine-grained intra- & inter-layer pipelined execution by fusing all the CONV layers and the first FC layer, leading to effective overlapping among layers for latency reduction. As a result, all layers except the last two FC layers are processed in parallel and the overall latency is equal to that of a single layer plus time waiting for dependent inputs.

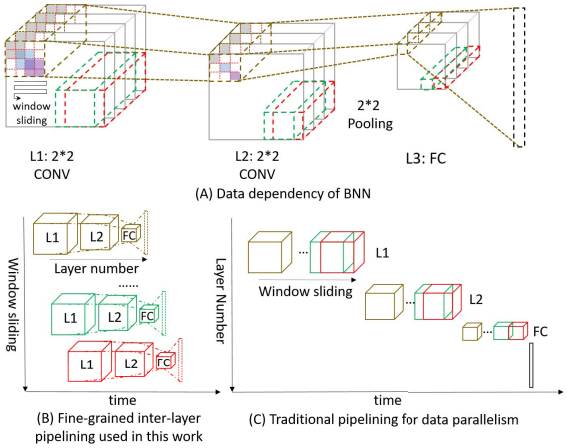

Fig. 4. Data dependency of BNN and the proposed fine-grained inter-layer pipelining for layer fusion

Figure 4(A) depicts the data dependency of the BNN using a 3-layer network consisting of two 2×2 CONV layers and one FC layer as an example. To calculate each gray pixel in layer 2, all gray pixels from layer 1 are required. All the gray pixels at layer 2 share the same data dependency. To calculate each pixel covered by the brown cuboid in layer 3, all pixels covered by brown cuboids from layers 2 and 1 are required. To calculate a pixel of the FC output, every pixel in the third layer is weighted and then accumulated; thus, any pixel in layer 3 can be consumed immediately after it is produced. The latency of calculating pixels in layer 3 determines the overall latency of this 3-layer BNN. To calculate pixels in layer 3 with the shortest latency, and to make data propagation faster with fewer intermediate results buffered between layers, pixels that share the same data dependency (i.e., the pixels covered by the cuboids with the same color at layer 1 and 2) are expected to be calculated immediately when all dependent data are ready, instead of waiting for the completion of the previous layer.

The symbols used in the following sections are defined as follows: IC/OC are the number of input/output channels; K is the filter kernel size; P is the pooling kernel size and W is the dimension of a feature map.

Intra-layer Pipeline: In order to calculate all the dependent data of an output pixel, for CONV layers, pixels with the same coordinate and different output channels are ideally calculated in parallel; pixels with the succeeding coordinates are processed sequentially in pipeline in the left-right-down direction. To calculate OC pixels with a certain coordinate at all output channels, $IC \times OC \times K \times K$ XNOR operations are needed. However, in the case that there are not enough hardware resources to process all these operations in parallel, maximum possible parallelism should still be obtained. We do this as follows:

(1) All input channels are partitioned into SIC segments, each with PIC input channels. The input channels in each segment are processed in parallel, while different segments are processed sequentially. In each iteration, partial results of all output channels are calculated. After SIC iterations, complete output features are computed.

(2) All output channels are partitioned into SOC segments, each with POC output channels. At each iteration, complete outputs of POC output channels are produced. After SOC iterations, all the output channels are completely processed.

(3) At each iteration, outputs of POC channels are partially calculated using inputs from PIC input channels. All output channels are completely processed in $SIC \times SOC$ iterations.

By tuning these 4 parameters, parallelism can be adjusted not only for decreased hardware resources but also for balancing the workload. The workload balancing scheme is discussed in Section III-C. Pseudo code is given in Listing 1.

Inter-layer Pipeline: Using the proposed intra-layer pipeline, input data which are dependent on the same output

```
1  for ho in Image.Height do
2   for wo in Image.width do
3     for scin in SIC do
4       for scout in SOC do
5         for pcout in POC do
6           for pcin in PIC do
7             for kh in kernel.height do
8               for kw in kernel.width do
9               out.channel[scout*pcout+pcout][ro][co] +=
10              in.channel[pcin*scin+pcin][ho+kh][wo+kw] *
11              weight[scout*pcout+pcout][pcin*scin+pcin][kh][kw]
```

Listing 1. Pseudo code for intra-layer pipelining

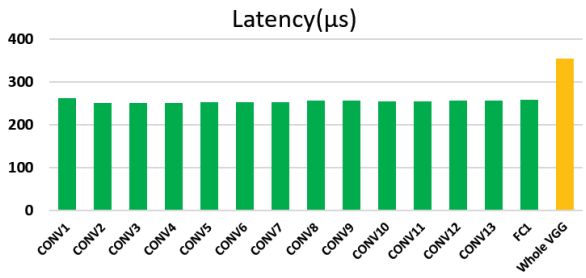

Fig. 5. Latencies of executing each layer of VGGNet inference and the overall latency of the whole VGGNet when using inter-layer fusion

features are produced quickly. To propagate these output features faster and so reduce overall latency, operations must begin processing as soon as all their dependent data are ready. A fine-grained inter-layer pipelining is proposed. Using this pipelining, all the CONV layers and the first FC layer are fused and processed in parallel. Taking the 3-layer BNN illustrated in Figure 4(A) as an example, when all the gray pixels at the first layer are ready, the gray pixels in the second layer are computed. Afterward, the kernel window shifts to the right and then down at the first layer. Immediately after the last data covered by the brown rectangles are calculated at the first layer, the last data covered by the brown cuboids at layer 2 is calculated. Then, the pixels covered by the brown cuboids at layer 3 are calculated. At this point, using the data at layer 3, partial results of the FC layer are computed. The set of cuboids in the same color indicate the data dependency from the first CONV layer to the first FC layer. When the data at layer 1 covered by the green cuboid set has been processed completely, the window slides to the right and reaches the red one. At the same time, the bottom-right features of the green cuboid at layer 2 is calculated. Then, data covered by the red cuboid at layer 2 starts processing and the pixels in green cuboid at layer 3 are calculated. In other words, different layers are fused and processed in parallel and their latencies are overlapped.

Figures 4(B) and (C) show, respectively, inter-layer pipelining and the traditional pipelining used in data parallelism. Our pipelining can significantly reduce the processing latency and the storage demand because the activations are propagated and consumed quickly between layers. Table 5 lists the latencies of each layer and the whole network of VGGNet. The resulting overall latency is only 1.42x that of a single layer.

Using this pipelining approach, in order to get high utilization, it is critical for the data production rate at a layer match the data consumption rate at the following layer. In the next subsection, we describe our hardware-based workload balancing strategy.

B. FPGA Architecture

We use layer parallelism to accelerate the inference of BNNs. That is, all layers are configured simultaneously on one FPGA and processed in a pipeline as described in the previous section. For each layer, the optimal architecture is deployed with no reconfiguration.

The proposed architecture is parameterized. Four main parameters control the architecture configuration: SIC, PIC, SOC and POC. Two goals are achieved by tuning the parallelism of each layer. First, modules of all layers are allocated with balanced workloads: data production and consumption rates between consecutive layers are matched. Second, the hardware resources can be adjusted according to the available on-chip resources.

*1) **Design of the CONV Layer**:* Figure 6 illustrates the design of a BCONV layer. Data of all input channels from the previous layer are buffered in the Shared Input Data Shift Register (SIDSR). SIDSR is composed of K sets of BRAM-based FIFOs. Each set has FIFOs providing data access capability of PIC bits/cycle. To gain higher concurrency for the BRAM-based FIFOs, data from 32 different channels are packed and stored as a word. Thus, each set needs $PIC/32$ BRAM blocks. The data layout of each FIFO is shown in the upper left corner of Figure 6. Data from SIC segments of input channels that are processed sequentially are stored in an interleaved order. Each FIFO buffers an entire row of the feature map. Therefore, $W * SIC * 32$ input channels are stored in a FIFO. K sets of FIFOs are linked in a head-to-tail manner to support a sliding kernel window and input data reuse. When enough data from the output channels of a certain layer are ready in SIDSR, the next layer starts being processed and $PIC * K$ data are broadcast to POC PEs.

Each PE has PIC XNOR engines. Each engine consists K^2 XNOR gates, a POPCOUNT engine, an accumulator, and a comparator. The comparator is coupled with a local Threshold Buffer (TB) built with distributed RAMs. All PEs work in lockstep under a control unit. PEs gather weights from the shared weight memory. To saturate the POC PEs, $POC * PIC * K * K$ weights must be accessed in parallel. We use a hybrid of BRAMs and Distributed-RAMs to build the Shared Weight Memory (SWM) for buffering weights.

After weights are read from the head of the SWM FIFOs, they are fed into the PEs and restored in SWM at the tail of FIFOs. Similar to SIDSR, in order to achieve sufficient concurrency, in case BRAM is adopted to build SWM, weights for 32 channels are packed and stored as a word in SWM.

If there is a pooling layer, the outputs of PEs are directed to two data paths depending on whether or not this output is the first element in a row. If so, it is buffered in the Horizontal

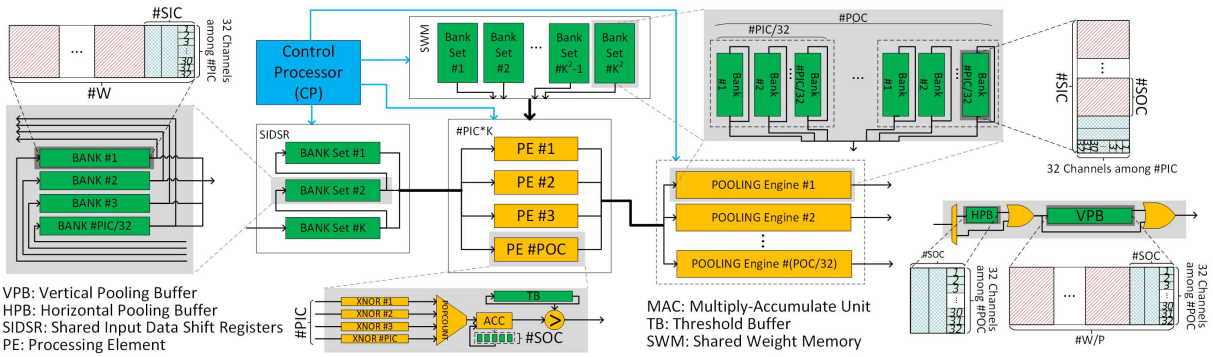

Fig. 6. Parameterized architecture of single CONV layer: SOC, SIC, POC, PIC can be tuned for workload balancing and to degrade the parallelism

Pooling Buffer (HPB); otherwise, it is compared with data that are already cached in the HPB. The comparison result is buffered in the Vertical Pooling Buffer (VPB) if it falls into the first row of the pooling kernel. Otherwise, it is compared with the data already cached in the VPB. The comparison outcome is used as the output of the layer. The data layout of the pooling buffer is similar to SIDSR; the output data from 32 output channels are packed before being buffered in the HPB. Again, the outputs are stored in an interleaved order.

The Pooling Buffers, SWM, and SIDSR are implemented with a combination of BRAMs and distributed RAMs. The choice is determined by comparing the required depth of each memory bank to a threshold. We propose a DSE strategy to decide the threshold and other parameters, i.e., $POC\&PIC$, for the optimal resource utilization. This DSE strategy is discussed in Section 3.C.(4).

*2) **Threshold Lookup (TL) in CONV layers**:* For CONV layers of ResNets (layers connected by shortcuts), TL modules are used instead of TB buffer. As mentioned in Section III, to lookup each threshold, 2 operations (multiply+addition) are required. In TL module, each lookup engine has a multiply-adder. For a certain layer, the number of lookup engines is the same as the number of PEs, i.e. POC.

*3) **Design of the FC layer**:* The design of the FC layer is similar to that of CONV, except that: (1) There is no data reuse since each input channel has its own weight rather than sharing the same filter window. (2) No Pooling Engine is required. (3) Weights are accessed from DDR instead of SWM.

*4) **Support of Workload Balancing**:* As all the layers are fully configured on the FPGA chip and layers are processed in a fine-grained pipeline manner, the workload of different layers must be balanced to achieve maximum hardware utilization. In other words, the output production rate of a layer must be the same as the input consumption rate for the next layer. We achieve this goal by adjusting the parameters PIC and POC (or SIC and PIC) of each layer. The parallelism of a layer is defined in Equations (7) and (4). In BNNs, the same padding strategy is adopted for high accuracy; thus $\frac{W[l-1]^2}{W[l]^2}$ is equal to $P[l-1]^2$.

$$Parallelism = PIC * POC \tag{7}$$

$$Parallelism[l] = \frac{Parallelism[l-1] * IC[l] * OC[l] * W[l]^2}{IC[l-1] * OC[l-1] * W[l-1]^2} \tag{8}$$

Compared with the hardware resource utilization without workload balancing, the proposed strategy requires only 3.8% of the LUTs, 6.0% of the FFs, and 0.19% of the BRAMs to complete VGGNet inference with the same latency. Table I lists the utilizations with and without workload balancing for VGGNet inference using the KCU1500 as the platform. KCU1500 is a Xilinx acceleration kit with 20nm Kintex UltraScale family XCKU115 FPGA. This kit has 16 GB DDR4 DRAM and can communicate to host with PCIe Gen3 x16.

TABLE I
RESOURCE USAGE OF LUTS, FFS, BRAMS, AND DSPS FOR VGG INFERENCE WITH/WITHOUT WORKLOAD BALANCING (SAME LATENCY)

LUT	FF	BRAM	DSP
With Workload Balancing			
509K/663K (76.8%)	513K/1327K (38.7%)	446/2160 (20.6%)	1728/5520 (31.3%)
Without Workload Balancing			
13401K/663K (2020%)	8553K/1327K (645%)	459951/2160 (10647%)	1728/5520 (31.3%)

*5) **Design Optimization**:* For large BNN networks, without orchestrated resource utilization, it can hardly fit into a single FPGA. Figure 7 shows the different hardware utilization options resulting from combining various POC and PIC with fixed parallelism for a CONV layer.

The utilization of the entire system can be described by the vector: $\vec{U} = (LUT, FF, BRAM)$, where

$$\vec{U} = f(K, W, P, SIC, PIC, SOC, POC), \tag{9}$$

For a certain BNN layer, some of these parameters are fixed, including K, W, P, IC, and OC. We set PIC, POC as independent variables, so that:

$$\vec{U} = f(x, y), x = PIC \in [1, IC], y = POC \in [1, OC] \tag{10}$$

As a result, the overall utilization, including conv-engine, popcount, comparators, pooling, line buffer, and weight buffer, becomes:

$$\vec{U} = \vec{U_{conv}} + \vec{U_{pop\&comp}} + \vec{U_{pool}} + \vec{U_{lb}} + \vec{U_{win}} \tag{11}$$

Since the binary conv-engine and comparator modules are built with distributed LUTs/FFs, we have

$$\vec{U_{conv}} + \vec{U_{pop\&comp}} = x \cdot y \cdot \vec{v_1} + y \cdot \vec{v_2} \tag{12}$$

The pooling, line buffer, and weight buffer modules all have two implementation choices: distributed LUTs/FFs or hard Block RAMs (BRAMs), depending on whether the depth is less than a threshold θ:

$$\vec{U_{pool}} = \begin{cases} g_{lut}(y), & \text{if } \frac{W \cdot OC}{P} \cdot \frac{1}{y} < \theta \\ g_{bram}(y), & \text{otherwise} \end{cases} \tag{13}$$

$$\vec{U_{lb}} = \begin{cases} h_{lut}(x), & \text{if } W \cdot IC \cdot \frac{1}{x} < \theta \\ h_{bram}(x), & \text{otherwise} \end{cases} \tag{14}$$

$$\vec{U_{win}} = \begin{cases} p_{lut}(x,y), & \text{if } OC \cdot IC \cdot \frac{1}{x \cdot y} < \theta \\ p_{bram}(x,y), & \text{otherwise} \end{cases} \tag{15}$$

As a result, we only need to obtain optimal x, y and θ. Figure 7(A) shows the workflow. θ is initialized as θ_0.

Step 1: Based on the FPGA resource constraints, calculate the system's overall parallelism (i.e., $x\dot{y}$).

Step 2: Compute the input/output channel parallelism for optimal x and y. Figure 7(B) shows the utilization change with respect to x under a fixed $x \cdot y$ for a BConv layer. Note that the optimal point lies in the middle of the curve.

Step 3: Adjust the ratio between distributed LUTs/FFs and BRAMs for the optimal θ.

Step 4: Check the utilization estimate. If there are unused resources, go back to Step 1 and repeat.

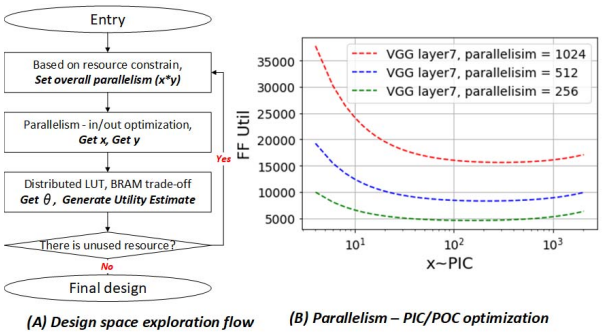

Fig. 7. DSE flow and FF utilizations with different combinations of PIC and POC when fixed parallelisms are used at the 7^{th} CONV layer of VGGNet

V. Experimental Results

The benefits from the proposed network structure optimization approach, layer-fusion and workload balancing are evaluated and shown in Figure 3, Figure 5 and Table I respectively. In this section, we are not going to discuss their benefits separately. Instead, we evaluate their overall contributions in accelerating BNN inference.

We use a Kintex KCU1500 FPGA to evaluate performance, energy efficiency, and latency; the networks used are Cifar-10 VGG-like [17], ImageNet AlexNet [18], VGGNet-16 [19] and ImageNet ResNet-18 [20], [22], [23].

Our results are compared with existing work on BNN acceleration using GPUs, CPUs, and FPGAs. As GPUs and CPUs are extremely underutilized when executing BNNs in no-batch mode. We mainly compare our results with a recently published high-end FPGA-based BNN accelerator. As shown in Table II, our latencies of Cifar-10 VGG-like and AlexNet are 15.9x and 54.0x less than the state-of-the-art FPGA-based BNN design. We also measure the inference latency of VGGNet of ImageNet. It takes 335μs to complete the inference of a 224×224 image. Using our design, 2817 224×224 images can be inferred per second with the accuracy of 76.8%. The latency of VGGNet is not compared to the existing FPGA accelerators, as our work is the first work to accelerate Binarized VGGNet of ImageNet on single FPGA. There is no existing FPGA-based work reporting the inference latency of binarized VGGNet. Thus, our latency, performance, and energy efficiency results are compared with results of CPUs and GPUs. Our latency and performance are at least 33.2x better than the state-of-the-art CPU results. For ResNet-18, the latency is 67.8μs. Comparing to the existing ASIC work, YodaNN, LP-BNN is 118$\times$ faster, with 128$\times$ better energy efficiency. Here, we do not compare our result to ReBNet [11], as the structure of the ResNet used in their paper is self-customized.

Performance is evaluated with respect to Image/s. Here, the number of images which can be inferred in one second is used to show the processing capability of our design. Considering the I/O ports on FPGA board can be customized and each FPGA chip provides hundreds of high-bandwidth I/O resources, we do not take I/O constraints into consideration in the performance evaluation.

Energy efficiency is evaluated with respect to Images/J. Our result is, on average, 47.3$\times$ better than previous work. The pipeline utilization of our design for VGG-16 is 99.7%, i.e., the percentage of idle stages in the pipeline is only 0.3%. The resource usages for the implementation of VGGNet are listed in Table I. The highest operating frequency which can be deployed is 310MHz. In order to provide a fair comparison to the existing works and highlight the benefits of our design instead of the delicate implementation, we use 200MHz as the operating frequency.

VI. Conclusions

In this paper, a single-FPGA-based accelerator for ultra-low-latency inference of BNNs, LP-BNN, is proposed. Our design can complete the inference of Binarized ImageNet, such as AlexNet, VGG-16 and ResNet-18, within 21.5μs, 335μs and 67.8μs respectively, with no accuracy loss compared with other BNN works. For small-scale networks, such as VGG-like in Cifar-10, the latency is only 8.2μs. The proposed accelerator and the resulting ultra-low latency inference make it possible to deploy deep neural networks in real-time applications, such as autonomous cars and robotic control.

Acknowledgement

This work was supported, in part, by the NSF through Awards CNS-1405695 and CCF-1618303/7960; by the NIH through Award 1R41GM128533; by grants from Microsoft and

TABLE II
LATENCY, PERFORMANCE, ENERGY EFFICIENCY COMPARISON USING DIFFERENT TEMPLATES, GPUS, FPGAS, CPUS TO EXECUTE INFERENCE OF 4 NETWORKS: CIFAR-10 VGG-LIKE [17], IMAGENET ALEXNET [18], VGGNET-16 [19] AND RESNET-18 [22]

	CPU				GPU			
Platform	Xeon E5-2640 [7]		Phi 7210 [12]	i7-7700 [12]	Tesla K40 [7]	V100 (self-implemented)		GTX 1080 [12]
Frequency	2.4GHz		1.3GHz	3.6GHz	745MHz	1.37GHz		1.61GHz
Dataset	Cifar	ImageNet			ImageNet	Cifar	ImageNet	
Network	VGG-Like	AlexNet	VGG-16		AlexNet	VGG-Like	AlexNet	VGG-16
Latency	1.36s	10.8s	11.8ms	16.1ms	1.26s	994μs	2.23ms	12.9ms
Performance (Img/s)	0.74	0.09	85	62	0.79	1006	448	78
Energy (Img/KJ)	7.79	0.95	395	954	3.36	5543	2475	433
Accuracy (%)	86.31	66.8	76.8	76.8	66.8	89.9	71.2	76.8
	FPGA			ASIC	FPGA			
Platform	Stratix V [7]		VCU108 [11]	UMC 65-nm [22]	This work: KCU1500			
Frequency	150MHz		200MHz	450MHz	200MHz			
Dataset	Cifar	ImageNet			Cifar	ImageNet		
Network	VGG-Like	AlexNet	AlexNet	ResNet-18	VGG-Like	AlexNet	VGG-16	ResNet-18
Latency	130μs	1.16ms	1.92ms	8ms	8.2μs	21.5μs	335μs	67.8μs
Performance (Img/s)	7692	862	521	125	1.2E5	4.7E4	2817	1.47E4
Energy (Img/KJ)	2.9E5	3.3E4	2.7E4	2.9E3	3.6E6	1.4E6	8.3E4	3.7E5
Accuracy (%)	86.31	66.8	N/A	N/A	88.5	72.7	74.3	65.6

Red Hat; and by Intel through donated FPGAs, tools, and IP. This research was also partially funded by the Deep Learning for Scientific Discovery Investment under Pacific Northwest National Laboratory's Laboratory Directed Research and Development Program. The evaluation was supported by U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

REFERENCES

[1] Das, Anup, and Akash Kumar. "Dataflow-Based Mapping of Spiking Neural Networks on Neuromorphic Hardware." In Proceedings of the 2018 on Great Lakes Symposium on VLSI, pp. 419-422. ACM, 2018.

[2] Tang, Wei, Gang Hua, and Liang Wang. "How to train a compact binary neural network with high accuracy?" In AAAI, pp. 2625-2631, 2017.

[3] Zhou, Shuchang, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." arXiv preprint, 2016.

[4] Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In European Conference on Computer Vision, 2016.

[5] Nurvitadhi, Eriko, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. "Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC." In Int Conf Field-Programmable Technology, pp. 77-84, 2016.

[6] Geng, Tong, Tianqi Wang, Ahmed Sanaullah, Chen Yang, R. Xuy, Rushi Patel, and Martin Herbordt. "FPDeep: Acceleration and load balancing of CNN training on FPGA clusters." In Proc. IEEE Symp. on Field Programmable Custom Computing Machines. 2018.

[7] Liang, Shuang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. "FP-BNN: Binarized neural network on FPGA." Neurocomputing, 2018.

[8] Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1." arXiv preprint, 2016.

[9] Geng, Tong, Tianqi Wang, Ahmed Sanaullah, Chen Yang, Rushi Patel, and Martin Herbordt. "A Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Work and Weight Load Balancing." In 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 394-3944. IEEE, 2018.

[10] Umuroglu, Yaman, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. "Finn: A framework for fast, scalable binarized neural network inference." In International Symposium on Field-Programmable Gate Arrays, 2017.

[11] Ghasemzadeh, Mohammad, Mohammad Samragh, and Farinaz Koushanfar. "ReBNet: Residual Binarized Neural Network."

[12] Hu, Yuwei, Jidong Zhai, Dinghua Li, Yifan Gong, Yuhao Zhu, Wei Liu, Lei Su, and Jiangming Jin. "BitFlow: Exploiting Vector Parallelism for Binary Neural Networks on CPU." In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 244-253, 2018.

[13] Zhao, Ritchie, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. "Accelerating binarized convolutional neural networks with software-programmable FPGAs." In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 15-24. ACM, 2017.

[14] Leary, Chris, and Todd Wang. "XLA: TensorFlow, compiled." TensorFlow Dev Summit (2017).

[15] Vasilache, Nicolas, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. "Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions." arXiv preprint arXiv:1802.04730 (2018).

[16] Geng, Tong, Tianqi Wang, Ang Li, Xi Jin, and Martin Herbordt. "A Scalable Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Weight and Workload Balancing." arXiv preprint arXiv:1901.01007 (2019).

[17] Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." In NIPS, 2015.

[18] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

[19] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[20] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." arXiv preprint arXiv:1605.07678 (2016).

[21] George, Alan D., Martin C. Herbordt, Herman Lam, Abhijeet G. Lawande, Jiayi Sheng, and Chen Yang. "Novo-G: Large-scale reconfigurable computing with direct and programmable interconnects." In 2016 IEEE High Performance Extreme Computing Conference, 2016.

[22] Andri, Renzo, Lukas Cavigelli, Davide Rossi, and Luca Benini. "YodaNN: An architecture for ultralow power binary-weight CNN acceleration." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37, no. 1 (2018): 48-60.

[23] Lin, Xiaofan, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network." In NIPS, 2017.

[24] Geng, Tong, Tianqi Wang, Chunshu Wu, Chen Yang, Wei Wu, Ang Li, Martin C. Herbordt. "O3BNN: An Out-Of-Order Architecture for High-Performance Binarized Neural Network Inference with Fine-Grained Pruning." In Proceedings of the 2018 International Conference on Supercomputing, pp.461-472. ACM, 2019.