MSD: Mixing Signed Digit Representations for Hardware-efficient DNN Acceleration on FPGA with Heterogeneous Resources

Jiajun Wu*, Jiajun Zhou*, Yizhao Gao, Yuhao Ding, Ngai Wong, Hayden Kwok-Hay So
Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong
{jjwu, jjzhou, yzgao, yhding, nwong, hso}@eee.hku.hk

***Abstract*—By quantizing weights with different precision for different parts of a network, mixed-precision quantization promises to reduce the hardware cost and improve the speed of deep neural network (DNN) accelerators that typically operate with a fixed quantization scheme. However, the additional control needed, and the decreased hardware efficiency arising from multi-precision operations have made mixed-precision quantization schemes challenging to deploy in practice. In this paper, a practical mixed-precision quantization framework called MSD that leverages the heterogeneous computing resources on FPGA to perform bit-serial and bit-parallel operations simultaneously is presented. MSD combines the use of a custom restricted signed digit (RSD) representation, which utilizes a limited number of effectual bits, and the conventional 2's complement representation to quantize DNN weights. Depending on the availability of fine-grained and coarse-grained resources, MSD encodes a subset of weights with RSD to allow highly efficient bit-serial multiply-accumulate implementation using LUT resources. Furthermore, the number of effectual bits used in RSD is optimized to match the bit-serial hardware latency to the bit-parallel operation on the coarse-grained resources to ensure the highest run-time utilization of all on-chip resources. Experiments show that MSD achieved a 1.36× speedup on the ResNet-18 model over the state-of-the-art, and a remarkable 4.91% higher accuracy on MobileNet-V2.**

I. Introduction

Modern FPGAs are becoming increasingly heterogeneous with coarse-grained word-level processing resources running alongside traditional fine-grained bit-level configurable logic. While coarse-grained programmable resources such as the digital signal processing (DSP) blocks are very efficient in processing data encoded with standard 2's complement representation at relatively wide bitwidth (viz. 8 to 16 bits), it is the flexibility of the fine-grained reconfigurable logic resources (i.e., LUTs) that makes them uniquely promising to accelerate deeply quantized DNN that leverage 2 to 4 bits for custom encoding of weights. Leveraging the two types of resources available on an FPGA, researchers have explored using a mix of multiplication and accumulation (MAC) units that are optimized for different bitwidth to perform mixed-precision DNN inference [1]–[4]. Through extensive design space exploration (DSE) and dataflow scheduling, the DNN workloads can then be mapped onto both types of resources to maximize the accelerator's theoretical peak performance.

*Equal contribution.

The challenge of this approach, however, is twofold. First, it remains a significant challenge to maintain good accuracy with deeply quantized DNN, even when combined with advanced mixed-precision operations running on both types of resources [3]–[5]. For instance, the accuracy of MobileNet-V2 dropped from 71.88 % to 66.25 % when 3 to 8 bits mixed-precision quantization was employed as reported in [3]. Second, unlike software that can arbitrarily adjust the employed bitwidth as the application requirement changes, hardware compute units must be statically built with resources provisioned for all possible data types that they need to support during run time. As a result, the purported benefits of utilizing mixed-precision operation can easily be outweighed by the overhead of supporting them in hardware, especially in ultra-low bitwidth range of 1 to 4 bits.

One solution to support variable bitwidth operations naturally during run time is to employ a bit-serial computing architecture [3], [6]–[9]. Although many early works have demonstrated the hardware advantages of utilizing bit-serial computation for mixed-precision DNN inference on FPGAs, they have yet to exploit the bit-level sparsity that is needed to fully unleash the benefits of performing bit-serial computations at ultra-low bitwidth. Specifically, one can speed up the bit-serial multiplier by skipping the ineffectual zeros bits in the input. However, leveraging bit-level sparsity can also be challenging. If the input data has an arbitrary number of effectual bits (EB), it may also suffer from an unbalanced load of PEs as the one with the most EBs then dominates the performance. Works like PRA [10], Bit-Tactical [7], and Bitlet [8] have proposed sophisticated hardware dynamic scheduler or bit-interleaved PEs architecture to address this issue but they come with non-negligible overhead in control logic. On the other hand, BitCluster [11] and BitPruner [12] resort to hardware/software co-design by constraining identical effectual bits (EB) of weights within a model/layer to achieve load balance. Nevertheless, similar to the low-bitwidth quantization approaches, it can also downgrade a model's representation capability by limiting a small number of EB based on common representation (e.g., 2's complement). We still need to explore a more efficient representation for MAC deployed on LUTs so that there are different optimization methods for fine-grained resources (LUTs) and coarse-grained resources (DSPs).

In this paper, we introduce MSD, which utilizes a mix of

two signed digit representations for hardware-efficient DNN acceleration with heterogeneous resources on FPGA. A custom restricted signed digit (RSD) data representation, which uses ternary digit set $\{-1, 0, +1\}$ in the encoding scheme, is proposed to work in conjunction with conventional 2's complement operations for mixed-precision inference. Based on the RSD representation, a load-balanced bit-serial architecture that leverages bit-sparsity is implemented using fine-grained resources (LUTs). To achieve load balance in bit-serial computation, it also enforces the number of EB for weights in a kernel/layer to be identical. Under the constraint of the number of EB, RSD makes the fine-tuned weights closer to the original values, allowing bit-serial PEs to fully exploit bit-sparsity while having a more extensive numerical representation capability than the standard method. At the same time, conventional 8-bit fixed operations with 2's complement are implemented in DSP to complement the low bitwidth RSD operations to achieve high model accuracy and to fully exploit the computation capability of an FPGA. Finally, to balance the workload between DSPs and LUTs in the heterogeneous architecture, a hardware-aware fine-tuning algorithm based on a cycle-accurate hardware cost model is introduced. The key contributions of this work are:

- We propose to use a mixed signed digit (MSD) scheme for hardware-efficient DNN accelerations that can efficiently utilize the heterogeneous resources on FPGA. It is powered by the proposed RSD representation that supports bit-serial computation on LUTs and bit-parallel computations on DSPs.
- We propose a fine-tuning and encoding algorithm for the weights based on the RSD representation. The hardware can exploit bit-level sparsity and achieve workload balance by restricting the number of non-zero bits in the weights to be identical. The RSD-quantized weights typically cause smaller numerical errors and can be efficiently deployed on bit-serial architecture.
- We develop a hardware/software co-design DNN acceleration framework based on the proposed architecture and weight adjustment method. The hardware-aware framework receives a DNN model and automatically selects the optimal EB configuration, scheduling, and workload partitioning for the heterogeneous resources.
- The entire framework is publicly available. Artifacts associated with this work is available at https://doi.org/10.25442/hku.22182073. Latest version of the open source code can be found at https://github.com/CASR-HKU/MSD-FCCM23.

This paper is organized as follows. In the next section, background on mixing bit-serial and bit-parallel operations on FPGAs will be presented. The MSD framework including the encoding method, hardware architecture, and a quantization scheme will be discussed in Section III. We evaluate the performance of MSD in Section IV and finally conclude in Section V.

II. Background & Related Work

A. Heterogeneous Architecture in FPGA DNN Accelerators

Modern FPGAs usually have hardened arithmetic blocks like DSPs and soft programmable logic like LUTs. Previous research has extensively used DSPs as building blocks for DNN accelerators. In recent years, many research and industrial efforts have also been devoted to embedding more AI-optimized building blocks in FPGAs fabrics like Tensor slices [13] and AI Tensor Block [14] to improve the density of MAC units further. However, a given FPGA device comes with a fixed number of hardened arithmetic blocks. Higher peak performances can be achieved if soft programmable logic can also be used efficiently in a heterogeneous fashion. Some prior research has co-designed with a quantization scheme to improve the efficiency of the overall heterogeneous system. Mix and Match [1] applied different quantization schemes on different rows of weight and proposed a sum-of-power-2 quantization algorithm that allows simple shift-adders to be implemented on LUTs. HAO [4] designed a hardware/software co-search framework to find optimal mix-precision quantization configurations in an inter-layer dataflow architecture. N3H-Core [3] integrated BISMO [15], an area-efficient bit-serial overlay, in its LUT-based computation cores. However, they failed to leverage the acceleration opportunities within the sparsity of bit-serial architecture. In MSD scheme, we design a bit-sparsity-aware framework that uses RSD representation. Our framework is also co-optimized with quantization training and scheduling to fully exploit the potential of heterogeneous architecture.

B. Bit-Serial Computing with Bit-Sparsity

Bit-serial architecture has been widely used in many digital systems designs focusing on low power and area efficiency. One might explore serial computation on both multiplicand and multiplier in bit-serial multiplication to perform one-bit shift and accumulation, as shown BISMO [15]. This design typically takes n^2 cycles to compute for n bits input. Another type of bit-serial multiplier may exploit serialization on only one of the inputs and carry out parallel shifting and accumulation on the other, like BitCluster [11]. It reduces latency to n cycles but requires more area for parallel shifter&adder. In general, bit-serial architecture transformed multiplication into multiple shifting and accumulation operations, which only happens on effectual bits (non-zero bits). Thus, one can speed up bit-serial architecture by saving the cycles on zero bits.

Many previous works have explored using bit-serial architecture in DNN accelerators to generate energy-efficient designs or exploit bit-level sparsity. Strips [16] proposed a bit-serial DNN accelerator suitable for efficient acceleration with varying precision on different layers. PRA [10] further extended this architecture by eliminating the ineffectual computation on zero bits. However, leveraging sparsity in bit-serial architecture can also bring the problems of an unbalanced load of PEs as different inputs can have a different number of effectual bits. Bitlet [8] adopted a bit-interleaved design that

condenses effectual bits and achieves better load balance. Bit-Cluster [11] addressed this problem from a software/hardware co-design point of view by constraining the number of effectual bits to be identical within a layer/network. In this work, we extend this idea by using RSD representation that further reduces the numerical errors while fully exploiting the opportunities from the load-balanced bit-serial architecture.

C. Signed-Digit Representation

The signed-digit representation uses a ternary number system with the digit set $\{1, 0, -1\}$, which is often denoted as $\{1, 0, \bar{1}\}$. Among which, the Canonical Signed-Digit (CSD) is a unique representation that minimizes the number of non-zero digits (Hamming weight) of a number and is widely used in low-power, high-speed DSP applications [17]–[19]. Prior research has also explored using signed-digit representation in DNN compression or acceleration. CAxCNN [20] used CSD to approximate model weights and proposed a hardware accelerator with area-efficient multipliers. CoNLoCNN [21] proposed an Encoded Low-Precision Binary Signed Digit (ELP BSD) representation as well as a non-uniform quantization method to speed up network inference and maintain accuracy. DWP [22] used a multi-objective shortest path problem formulation to search for the signed-digit representation of weights that allows maximal digit-serial parallelism. It also incorporated other bit-condensing techniques and designed a customized hardware accelerator. Unlike those works, MSD framework adopts a hardware-aware quantization training method that constrains the number of effectual bits in a kernel/layer. It allows efficient load-balanced bit-serial to be deployed on LUTs as a heterogeneous core.

III. Methodology

This section introduces the MSD framework in terms of algorithm and software-hardware co-design. We first present the proposed weight fine-tuning algorithm based on the RSD representation under the restriction of identical EB of weights within each layer. Efficient hardware is designed to support bit-serial computing and heterogeneous DNN workloads by mixing the RSD and standard representation. Finally, we present a hardware-aware mixed-EB search framework to realize the speedup-accuracy trade-off on the proposed hardware design.

A. RSD-based Weight Fine-tuning & Encoding

As discussed in Section II, we need to restrict the number of EB in a workload of weights as the same to avoid unbalanced issues [11]. Nevertheless, directly removing/adding '1' bits based on standard 2's complement format downgrades the representation capability. Motivated by the signed digit number systems, a customized RSD data representation is proposed to keep the number precision while restricting the number of EB. Given an original number and restricted EB, we apply the binary search algorithm on the EB bases $S = (1, 2, 4, 8, 16, 32, 64, 128)$ (integer power of 2) to find the closest base and decide whether this base is added/subtracted to the previous bases. When the depth of the search tree

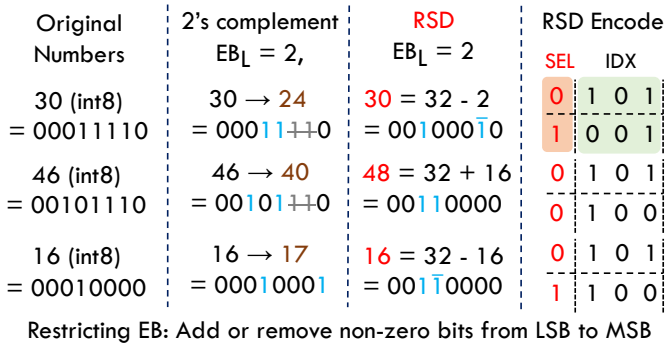

Fig. 1: Weight fine-tuning and encoding scheme based on the RSD representation. Under the same restriction of EB, RSD representation can make the fine-tuned values closer to the original values.

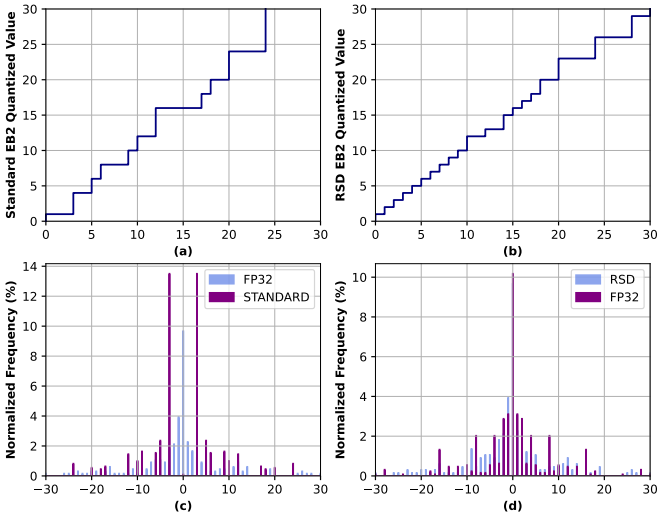

Fig. 2: Comparing numerical properties of standard 2's complement and RSD representation with $EB_{\mathrm{L}} = 2$. (a), (b): Visualizing the quantized values of 0 to 30 in the two representations. (c), (d): Normalized distribution of weights for a layer in MobileNet-V2 quantized using the two representations when compared with the original `fp32` values

reaches the restricted EB number, we get the final RSD-based fine-tuned value. Fig. 1 presents several examples of this fine-tuning process. It is worth noting that standard 2's complement representation is actually a particular case of signed-digit, with only the MSB can be the '-1' ($\bar{1}$) term in negative values.

By introducing subtraction into shift & add operations, RSD representation can make the value after fine-tuning closer to the original value compared with standard 2's complement. For instance, in Fig. 1, if we want to restrict `30(int8)` with the EB number of 2 based on 2's complement, the quantized value will be `8'b00011000(24)` with $30 - 24 = 6$ error. But for the RSD-based scheme, as shown in Fig. 1, the quantized value is still `30` without any error. Fig. 2 (a) and (b) present the quantized numbers based on the two representations under

the restriction of 2 EB, in which the RSD method curve is closer to linear mapping, thus it introduces less quantization error. Besides, Fig. 2 (c) and (d) further present the normalized weights distribution of two representations for one layer in the MobileNet-V2 model, in which the post-quantized weights distribution of RSD-based method remains closer to the original floating-point (FP32) one. It can be concluded that the RSD method adapts to the original numbers better than the standard 2's complement.

Since DNN models generally tend to have different tolerance to the numerical precision of each layer, we apply layer-wise fine-tuning for the weights, i.e., each layer has an identical number of EB (EB_{L}). All weights that need to be mapped for bit-serial computations will be fine-tuned according to this value based on the above algorithm. After obtaining the fine-tuned weights based on RSD representation, the framework will further encode them into the positions of '1' bits so that the hardware can skip ineffectual bits and achieve bit-sparsity. Fig. 1 shows the encoding process, and we define the encoded weights into two parts: bit-index of EB (IDX), which indicates the index of the non-zero bit starting from LSB, and an extra bit indicating '1' or '$\bar{1}$' (SEL). For instance, $8'b00100\bar{1}0\,(30)$ has two non-zero bits, and their indices are 1 ('1'-bit) and 5 ('$\bar{1}$'-bit), so the encoded results will be $(1, 001)$ and $(0, 101)$, as shown in Fig. 1. The EB_{L} is also set up as the hyper-parameter for the hardware control. Since our framework focuses on static quantization of weights, this process will be performed offline without introducing additional overhead to the hardware. It is worth noting that although RSD encoding based on the position of '1' bits reduces the bitwidth (from 8 to 4), it increases the amount of weights data. In our framework, we should carefully control the EB_{L} for each layer and the workload split to ensure that the increased data I/O will not affect the computing performance improvement. We will discuss this issue in Hardware Architecture.

B. Hardware Design

1) Bit-serial Multiplier: Fig. 3 presents the bit-serial multiplier based on the RSD representation of weights. The circuit consists of data registers, a negator for calculating the opposite of activations (NEG), a barrel shifter, and a partial product accumulator (PPA). According to the discussion in Section III-A, the fine-tuned weights are encoded as the bit-position of the effectual bits (IDX, 3-bit), with an additional bit indicating addition/subtraction (SEL, 1-bit). The combined 4-bit weights are serially input to the multiplier, while the activations are represented in the original standard binary. Firstly, the SEL bit selects the input activation or its opposite to the subsequent operations to realize the subtraction operation in RSD. After that, the barrel shifter shifts the activation according to the input IDX to get the partial product. The accumulator sums up the partial products based on the target EB for the current layer. Fig. 3 also gives an example (6×30 based on ($EB_{\mathrm{L}} = 3$) in which the multiplier needs three cycles for one multiplication. Although the encoded weight

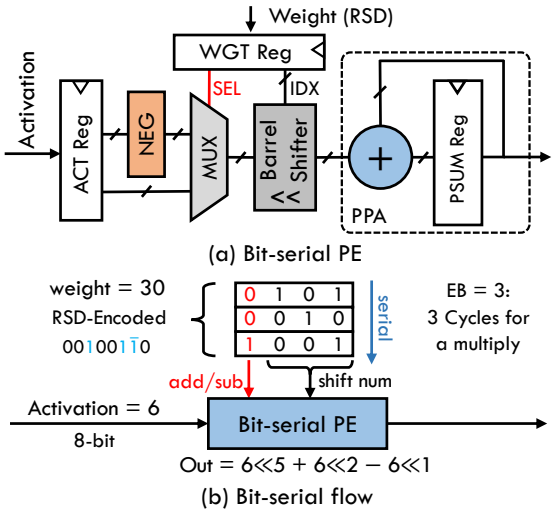

Fig. 3: Bit-serial PE and the computing dataflow. The bit-serial scheme needs EB cycles to compute a multiplication.

Fig. 4: Combined MAC operations on DSP48 [23]. Two `int8` MACs are combined in this work.

based on RSD increases by 1-bit compared to the standard binary method (3-bit), the total length of 4-bit makes it adapt to storage alignment requirements, which is suitable for hardware implementation. Since the bit-serial multiplier takes multiple cycles for one multiplication, we need to limit the EB_{L} to ensure the bit-serial multiplier with bit-sparsity can be faster than the conventional parallel one.

2) Combined MACs on DSP: As for the bit-parallel MACs based on hard blocks (DSPs on FPGA), the weights mapped to this part will not be fine-tuned and encoded (i.e., they will maintain the original 2's complement). We implement the combined operations by mapping multiple MACs into one DSP block to maximize the utilization rate, as Fig. 4 shows. The input bitwidth of DSPs on modern FPGAs is usually designed to be relatively large (e.g., 27-bit and 18-bit in Xilinx DSP48E2). Hence, it is efficient to implement combined MACs by separating different numbers with guard bits to improve the computation performance further [23], [24]. Specifically, if the scenario targets `int8` as the precision, this method can map two MACs in one DSP to reduce the computation latency by half. Note that we do not apply RSD method for the weights processed by DSP, hence the input activations and weights are all based on pre-trained `int8` with 2's complement representation. As claimed before, 2's complement is a particular case of signed-digit, so the run-time process of our design mixes signed-digit for different

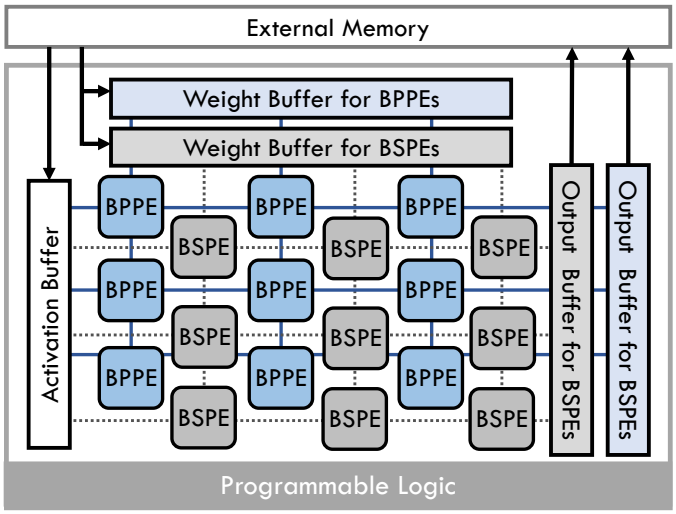

Fig. 5: Heterogenous architecture based on bit-serial PE (BSPE) and bit-parallel PE (BPPE). Both the BSPEs and BPPEs are connected as systolic arrays, and the activations are shared between the two types of PEs.

resources on FPGAs.

3) Heterogeneous Architecture: With the specific designs for both LUTs and DSPs, Fig. 5 presents the proposed heterogeneous architecture for higher peak performance. The accelerator consists of bit-serial processing elements (BSPE) and bit-parallel PE (BPPE) for different types of resources, targeting bit-serial and bit-parallel computation, respectively. We apply a systolic array as the connection between both BSPEs and BPPEs for homogeneous control. As discussed before, the weights mapped into the BSPEs will be fine-tuned and encoded before being transferred to the hardware, but the weights mapped to the DSP remain in the original format. Therefore, we deploy separate weight buffers in the two types of PEs to store different weight representations. Furthermore, since we only target bit-serial MACs for weights to achieve bit sparsity, neither the activations are quantized nor fine-tuned. Therefore, we implement a global activation buffer to reuse the activations between the two types of PEs and reduce data communication overhead between on-chip buffers and the external memory. The discussed heterogeneous architecture can be mapped to the FPGA layout since the BSPEs only utilizes LUTs. Hence, the proposed architecture can be implemented on FPGA with better timing even though the LUT logic is heavily utilized for computation.

To improve efficiency and simplify control logic, we select output stationary as the dataflow for the systolic arrays [25]. For bit-serial operations in the BSPE, it only consists of the bit-serial multiplier and scratchpads. Since in the output stationary dataflow, the output activations will not move until all the partial sums are accumulated, the accumulation of partial sums can be efficiently integrated into the PPA in the multiplier, to reduce the resource overhead. By applying this 1D bit-serial dataflow, we can exploit bit sparsity and achieve higher computation speedup. On the other part, each

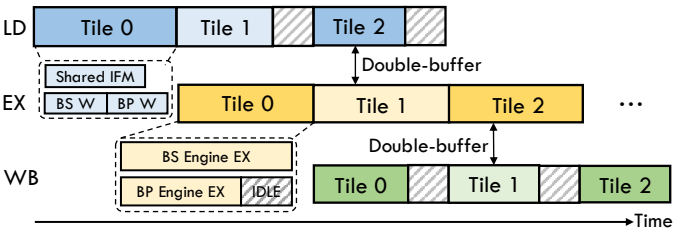

Fig. 6: Inter-tile dataflow in the proposed architecture. The dataflow is tile-pipelined between the three stages, with the double-buffer scheme.

BPPE takes a DSP for the combined MACs to maximize the computation performance. Similar to the bit-serial MACs, due to the characteristics of modern DSP design, the accumulator summing up the partial sums can still be integrated into each DSP. With the help of these optimizations, our hardware design reduces the LUT overhead other than computation logic, allowing a more extensive systolic array of BSPEs. Note that the proposed architecture is not tailored for one FPGA platform, since the accelerator is based on an RTL template with a header file and can be easily modified for various FPGAs.

The accelerator performs inference of the DNN model layer-wise, and the tile is the basic block running on it. When it starts processing a layer, the accelerator first loads (LD), executes (EX), writes back (WB) each tile, and then repeats for all tiles. The global controller handles the weight-loading process according to the current layer's EB_{L} and the ratio of workloads in the two types of PEs. Besides, since the memory hierarchy of the proposed architecture is based on the double-buffer strategy, the three stages can be pipelined through processing, greatly enhancing the throughput and reducing the latency, as Fig. 6 shows.

As the BSPE and BPPE target different weight workloads, how to allocate the number of weights processed by the two types of PEs will affect the computing performance of the accelerator. We define the weight-split ratio r as the proportion of bit-serial weights to the total weights, i.e., $r = weight_{\mathrm{BS}}/weight_{\mathrm{Total}}$, to partition the workloads. Since the RSD-based bit-serial multiplication has a larger computation latency than the DSP, the workload of the BSPEs has a more significant impact on the overall computation latency. In addition, the final hardware latency also relates to data I/O. When the weight of the BS part increases, the amount of weight data obtained by fine-tuning based on different EB_{L} may increase (e.g., $EB_{\mathrm{L}} = 3$) or decrease (e.g., $EB_{\mathrm{L}} = 1$). Therefore, to complete the hardware/software co-design framework, we still need a hardware analytical model to search the optimal schedule/dataflow and the weight-split ratio r, which will be introduced in the next part.

C. Cost Model and Scheduler

1) Hardware Abstraction Model: For the cost model, we need to abstract the hardware to build a hardware overhead model as the basis for the performance model. Given an

architecture based on the proposed design, we define a series of parameters to describe the architecture. The BSPE and BPPE generally have different systolic array scales limited by the FPGA resources. We use BS_r, BS_c, BP_r, BP_c as the numbers of rows & columns in BS cores and BP cores, respectively. Based on the array scales, we can formulate the utilization of LUTs and DSPs:

$$\begin{aligned} LUT_{\text{utl}} &= LUT_{\text{BSPE}} \times BS_r \times BS_c \\ DSP_{\text{utl}} &= DSP_{\text{BPPE}} \times BP_r \times BP_c \end{aligned} \tag{1}$$

in which the LUT_{BSPE} and DSP_{BPPE} indicate the LUT & DSP consumption in one BSPE and BPPE, respectively. As for the buffers, we assume each row/column of the two systolic arrays is connected with an identical number of BRAM36 ($BRAM_{\text{unit}}$) as the buffer/scratchpad. Hence, the total utilization of BRAM is:

$$\begin{aligned} BRAM_{\text{utl}} &= (BS_n + BP_n) \times BRAM_{\text{unit}} \\ BS_n &= BS_r + 2 \times BS_c \\ BP_n &= BP_r + 2 \times BP_c \end{aligned} \tag{2}$$

To simplify the framework and remain the discussion point in representation, this work does not discuss the hardware-schedule co-search based on the utilization model. For a specific FPGA device, we only set up a unique architecture for all DNN models. Still, given an FPGA device and a hardware architecture, it must be ensured that the total utilization does not exceed the limitation of the device.

2) Latency Model & Scheduler: We apply the widely used 6-dimension for-loop topology as the DNN model abstraction [26], [27]. Based on the for-loop model, we tile the output channel (K), output feature map height (H), width (W), and input channel (C) dimensions while keeping the tile size in the kernel height (I) & width (J) same with the DNN model to limit the design space. The fully-connected (FC) layers can be regarded as convolutional layers with $H, W, I, J = 1$. Define the DNN layer size $\mathbf{M} = (K, H, W, C, I, J)$ and the tile size of each dimension as $\mathbf{T} = (t_K, t_H, t_W, t_C, t_I, t_J)$ and the total number of tiles as N_T, we have:

$$N_{\text{T}} = \lceil \mathbf{M}/\mathbf{T} \rceil = \lceil K/t_K \rceil \times \lceil H/t_H \rceil \times \lceil W/t_W \rceil \times \lceil C/t_C \rceil \tag{3}$$

The scheduler is also responsible for searching the optimal workload partitioning ratio r. Based on the tile size of each dimension, EB_{L} and r, the tile sizes of the input feature map (S_i), weights (S_w), and output feature map (S_o) can be calculated as:

$$\begin{aligned} S_i &= t_C \times \{(t_H - 1) \times str + t_I\} \times \{(t_W - 1) \times str + t_J\} \\ S_w &= r \times \prod_{K,C,I,J} \mathbf{T} \times EB_L \times 0.5 + (1 - r) \times \prod_{K,C,I,J} \mathbf{T} \\ S_o &= \prod_{K,H,W} \mathbf{T} = t_K \times t_H \times t_W \end{aligned} \tag{4}$$

where the str is the convolution stride given by the DNN model. The 0.5 factor in S_w indicates that the weights for bit-serial computation have lower bitwidth (4-bit in the selected `int8` case). Introducing the memory bandwidth BW into our model, the latency of LD and WB stages can be formulated. Besides, according to the roofline model [28], we define the communication latency Lat_{comm} as the maximum of LD and WB latencies:

$$\begin{aligned} Lat_{\text{LD}} = \lceil (S_i + S_w)/BW \rceil,\ Lat_{\text{WB}} = \lceil S_o/BW \rceil \\ Lat_{comm} = \max(Lat_{\text{LD}}, Lat_{\text{WB}}) \end{aligned} \tag{5}$$

As for the computation latency of one tile, we adopt the cycle-accurate simulator in SCALE-Sim [25] and ANT [29], based on GEMM backend. When calculating as the output-stationary dataflow, the systolic depth is $t_C \times t_I \times t_J$, and the tile is divided into several sub-tiles based on the array size. Since the computation flows in BSPEs and BPPEs are different (bit-serial vs. bit-parallel), we define two computation latencies and use the maximum as the final one:

$$\begin{aligned} Lat_{\text{BS}} &= \prod_{C,I,J} \mathbf{T} \times EB_{\text{L}} \times \lceil t_H \times t_W/BS_r \rceil \times \lceil t_K/BS_c \rceil \\ Lat_{\text{BP}} &= \prod_{C,I,J} \mathbf{T} \times \lceil t_H \times t_W/BP_r \rceil \times \lceil t_K/(BP_c \times 2) \rceil \\ Lat_{comp} &= Lat_{\text{EX}} = \max(Lat_{\text{BS}}, Lat_{\text{BP}}) \end{aligned} \tag{6}$$

where the $BP_c \times 2$ factor in Lat_{BP} is the result of combined MACs optimization in `int8` case. Finally, considering the pipeline starting and ending stages in Fig. 6, the total latency for calculating this layer is:

$$Lat_{\text{L}} = N_T \times Lat_{\text{EX}} + Lat_{\text{LD}} + Lat_{\text{WB}} \tag{7}$$

With the latency model, we can set up the scheduler by searching all the possible tiling sizes and the weight-split ratios r of the current layer and find the optimal size & ratio combination. During brute-force searching, the tile size of feature maps and weights should not exceed the buffer size (BUF). Based on the given architecture, we can trivially calculate the buffer size and $BRAM_{\text{unit}}$ setup. Considering the DNN model has m layers, the problem can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{T},r} \quad & \sum_{j=1}^{m} Lat_{\text{L}}(j) \\ \text{s.t.} \quad & S_i \le BUF_i,\ S_w \le BUF_w,\ S_o \le BUF_o \end{aligned} \tag{8}$$

The scheduler is deployed in the host CPU to generate the optimal dataflow, partitioning, and run-time instructions for the FPGA devices, based on the input DNN model and EB_{L} configuration. Besides, the design space exploration can motivate the developer, using the metrics as a guide to developing the host scheduler in different platforms with faster search

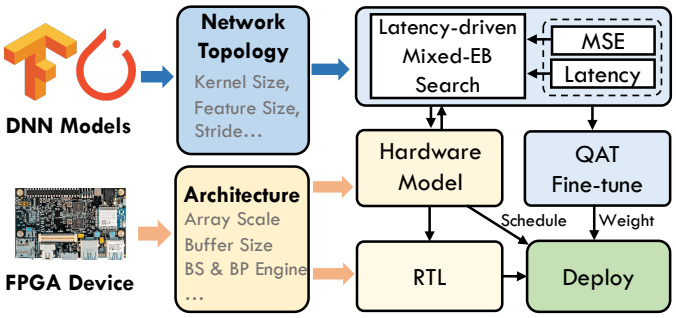

Fig. 7: Hardware-aware quantization framework. The hardware model generates latencies with different combinations of EB_{L} for the latency-driven mixed-EB search algorithm.

Algorithm 1 Latency-driven mixed-EB search strategy

Input: DNN model $\mathbf{M}$ with m layers $\{L_1, L_2, ...L_m\}$, latency constraint ω, top-k parameter k
Output: Layer-wise EB of weights $\mathbf{A} = (EB_1, EB_2, ..., EB_m)$

Initialize $\mathbf{A} = (3, 3, ..., 3)$
$Lat\_base \leftarrow$ TOTAL_LAT$(\mathbf{M}, \mathbf{A})$, $speedup \leftarrow 1$
while $speedup$ does not meet ω **do**
  lw_lat $\leftarrow$ LAYER_LAT$(\mathbf{M}, \mathbf{A})$ ▷ Layer-wise latency
  lat_top $\leftarrow$ LAYER_SEL(lw_lat, k) ▷ Select top-k layers
  layer_list $\leftarrow$ MSE_RANK(lat_top) ▷ MSE Re-rank
  REDUCE_EB(layer_list, $\mathbf{A}$) ▷ Reduce EB in order
end while

procedure REDUCE_EB(list, $\mathbf{A}$)
  for $l = 1 \rightarrow k$ **do**
    Degrade $\mathbf{A}_{list[l]}$: $EB_{\mathrm{L}} = 3 \rightarrow 2$ or $EB_{\mathrm{L}} = 2 \rightarrow 1$
    $speedup \leftarrow$ TOTAL_LAT$(\mathbf{M}, \mathbf{A})/Lat_base$
    break if $speedup \geq \omega$
  end for
end procedure

algorithms. Moreover, it can also be used for speedup-accuracy trade-off to find the optimal EB configurations of each layer, which will be discussed in the following subsection.

D. Hardware-aware Mixed-EB Quantization

As discussed before, our RSD-based fine-tuning algorithm changes the value of weights, slightly affecting DNN inference accuracy. To support the trade-off between accuracy and latency for different scenarios, we develop a hardware/software co-design DNN acceleration framework based on the proposed architecture and weight fine-tuning method to find the optimal number of EB (EB_{L}) for each layer. This part introduces our mixed-EB search methodology and the optimization based on the sub-gradient algorithm.

1) Quantization Metrics: Mean Squared Error (MSE) is a common metric to effectively evaluate the accuracy of the post-quantization DNN models [29]. The quantized model achieves higher accuracy with a lower MSE. Here we use MSE as a metric to measure the quantization error and facilitate the search process, defined as:

$$MSE = \sqrt{\Sigma_{i=1}^{n} \left(\frac{x - \hat{x}}{\sigma_i} \right)^2}, \tag{9}$$

where x and $\hat{x}$ are respectively the original `int8` and RSD-quantized values, and σ_i is the standard deviation of the tensor distribution. Another metric for the quantization scheme is latency (Lat_{L}). We introduce two functions from the scheduler to build the quantization scheme, TOTAL_LAT and LAYER_LAT, which are responsible for calculating the total latency and layer-wise latency under the current EB configuration, respectively. The latency formulation of these two functions has been presented in Section III-C.

2) Mixed-EB Search Strategy: Based on the quantization error MSE and latency metrics, we propose a speedup-based optimization strategy to suit different application scenarios with speedup constraint ω, as shown in Fig. 7. Our framework can limit the speedup to no less than ω for different real-time requirements to ensure hardware performance while minimizing the quantization error (MSE). In other words, the quantization selection of $\mathbf{A} = (EB_1, EB_2, ..., EB_m)$ for m layers with mixed-EB quantization can be searched by the strategy. Eqn. (10) formulates the optimization problem, in which the strategy aims to minimize the quantization error while meeting the speedup constraint ω. The baseline latency (Lat_{base}) here refers to the de facto design where all the computations are mapped into DSP only with `int8` representations, and we also can get the value by the scheduler.

$$\begin{aligned} \min_{\mathbf{A}} \quad & \sum_{j=1}^{m} MSE(j, \mathbf{A}[j]) \\ \text{s.t.} \quad & \omega \times \sum_{j=1}^{m} Lat_{\mathrm{L}}(j, \mathbf{A}[j]) \leq Lat_{\mathrm{base}}. \end{aligned} \tag{10}$$

The hierarchical mixed-EB quantization of weights leads to a vast design space. In our mixed-EB searching framework, we mainly support the selection of $EB_{\mathrm{L}} = 1, 2, 3$ for better hardware efficiency. Assuming the DNN model has m layers, the total possible solutions will be $(3 \times 3)^m$. A heuristic top-k search algorithm was devised to find a near-optimal and efficient solution. Algorithm 1 describes the proposed heuristic search algorithm. We first obtain the total baseline latency performance by TOTAL_LAT function. Then, the algorithm iteratively calculates layer-wise latency by LAYER_LAT and selects k layers with the largest latency as candidates since we intend to quantize the slowest layer first to obtain a better overall end-to-end speedup. In addition, to obtain the optimal solution with the smallest MSE, we also calculate the MSE of each candidate and reorder them in ascending order of MSE. The search algorithm reduces the EB of each candidate one by one to quantify the lowest MSE layer first. After that, the engine recalculates the latency and selects the next top-k candidate in the next iteration. The whole iteration stops when the end-to-end speedup constraint is satisfied. After the

TABLE I: Mixed-EB speedup results with different constraints ω. A larger ω means a higher speedup and a more aggressive quantization strategy.

Model	ω	Layer-wise Mixed-EB Result $\mathbf{A}$	Speedup
VGG-16	1.5	[3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3]	1.52
	1.6	[2 3 2 2 2 3 2 2 3 3 3 2 3 3 3 2]	1.60
	1.7	[2 2 2 2 2 3 2 2 3 2 3 2 2 2 2 2]	1.70
	2.0	[2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2]	2.00
	2.1	[1 1 2 1 1 1 1 2 2 1 2 1 2 1 2 1]	2.14
	2.2	[1 1 2 1 1 1 1 2 1 1 2 1 2 1 1 1]	2.24
ResNet-18	1.5	[3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 3 3 3 3 3]	1.51
	1.6	[2 3 2 3 2 3 3 3 2 2 2 3 3 2 2 2 2 3 2 3 2]	1.62
	1.65	[2 3 2 3 2 3 2 3 2 2 2 3 3 2 2 2 2 3 2 3 2]	1.65
	1.7	[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2]	1.71
	1.8	[2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 1 2 1 1]	1.84
	1.9	[2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1]	1.90

iteration stops, the framework fine-tunes the weights based on the RSD encoding scheme. It applies quantization-aware training (QAT) to maintain inference accuracy, which is a commonly used method in quantization works like [29], [30].

To illustrate, our framework can search different mixed-EB combinations in this optimization problem, Table I shows the search results based on different constraints ω on VGG-16 and ResNet18 model. It indicates that a larger ω represents a more aggressive speedup, so the searched mixed-EB result has more layers with a smaller number of EB_{L}, degrading from 3 to 1. Besides, the final speedup is close to the input constraint, proving that the search result is a near-optimal solution in the design space.

IV. Evaluation

A. Experimental Setup

The proposed accelerator is designed and implemented with Verilog HDL. We implemented the accelerator on the Pynq-Z2 (XC7Z020), Ultra96 (ZU3EG), and ZCU102 (ZU9EG) platforms. XC7Z020 is a lightweight FPGA device for testing scenarios with low memory bandwidth (in our setup, 64-bit data width in LD/WB channels) and limited resources. Since ZU3EG and ZU9EG are based on Xilinx Ultra-scale SoC, they have higher memory bandwidth (128-bit in LD/WB channels), while the ZU9EG device has more resources for computation. We apply the same hardware architecture for all DNN models in this experiment, as Table II shows. We do not fully utilize the fine-grained resources (i.e., LUTs) for BS cores to leave enough space for data pre- and pro-processing in real-world applications. For the DNN models, we conduct the experiments based on VGG16, ResNet18/50, MobileNetV2 and emerging models like Vision Transformer (ViT) [31] on ImageNet classification. We use the post-quantized INT8 weights/activations from PyTorch and a conventional hardware design implementing MACs only on DSPs as the baseline (i.e., BS_r and $BS_c = 0$). For each DNN model, we train $3 \sim 5$ fine-tuning epochs for QAT. To conduct a fair comparison, the training setup and the hyper-parameters are kept the same for all types under evaluation.

TABLE II: Hardware architecture parameters setup on FPGAs

FPGA Devices	Architecture Parameters				Memory Bitwidth BW
	BS_r	BS_c	BP_r	BP_c	
XC7Z020	40	40	14	15	64-bit (8 Bytes)
ZU3EG	48	48	16	16	128-bit (16 Bytes)
ZU9EG	80	80	48	48	128-bit (16 Bytes)

TABLE III: Accuracy comparison between the RSD encoding and the standard representation (2's complement) under the same constraint of EB number.

DNN Models	Post-quantized Top-1 Accuracy		Improv.
	RSD Encoding	2's complement (Reported in [12])	
ResNet-18	69.72 %	69.54 %	0.55 %
ResNet-50	76.05 %	76.19 %	0.27 %
MobileNet-V2	71.16 %	68.49 %	2.54 %

B. Encoding Scheme

As illustrated before, our RSD encoding can make the fine-tuned value closer to the original value than standard 2's complement representation. Hence it has the potential for higher accuracy with smaller quantization error. We compare the accuracy after fine-tuning with the previous work BitPruner [12], which applied the standard representation. Table III shows the accuracy improves from $0.55\,\%$ to $2.54\,\%$ in the selected DNN models based on the RSD encoding scheme. In terms of the hardware part, although the RSD method introduces the SEL bit for selecting addition or subtraction, the bit-serial PE does not have extra overhead compared with the conventional bit-serial design in [10]–[12] because the hardware design in these works still needs to handle negative activations in the shift&add operations. With higher post-quantized accuracy and negligible hardware overhead, it can be concluded that our RSD encoding approach is better than standard 2's complement for LUT-based synchronized bit-serial computation (i.e., with the same number of EB), in terms of maintaining accuracy of DNN inference.

C. Theoretical Analysis for Peak Performance

As discussed in hardware design, our accelerator can enhance the peak performance of the target device since the heterogeneous architecture maximizes the computation capability. Based on the roofline model [28], we calculate the peak performance of the three different devices based on the architecture parameters in Fig. 8. We consider each PE processes one MAC operation for the BS engine in EB_{L} cycles. To demonstrate that our framework can enhance the peak performance by the bit-sparsity-aware heterogeneous architecture, we present the improvement of the theoretical peak performance in Fig. 8, comparing the heterogeneous design and the DSP-only conventional design under the same clock frequency. We also apply the combined MAC operations (Fig. 4) in the DSP-only design. All the heterogeneous results

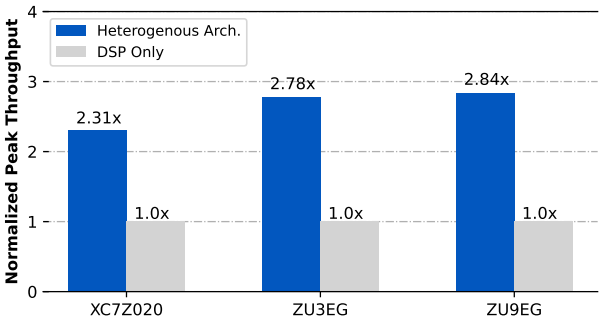

Fig. 8: Theoretical analysis of peak performance for three devices based on all the $EB_{\mathrm{L}} = 2$. The results are normalized based on DSP-only throughput individually for each device.

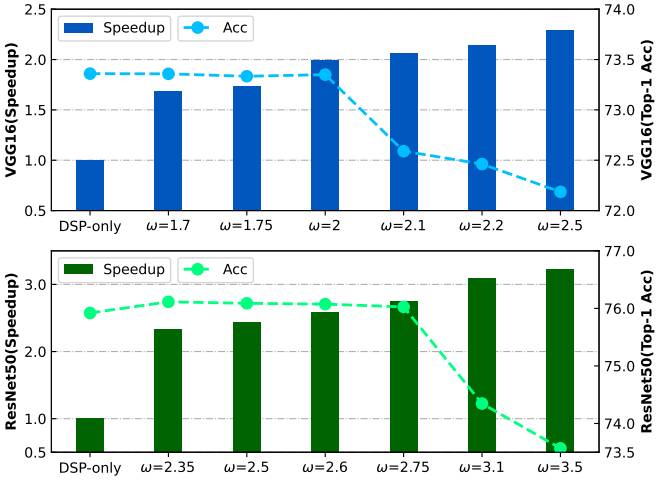

Fig. 9: Normalized speedup and post-quantization accuracy with different latency constraints ω on ZU3EG. Our framework can reduce the latency by $1.7\times$ to $3.5\times$ with negligible loss of accuracy on the VGG16 and ResNet50 models.

are normalized based on DSP-only throughputs. The theoretical results show that with the bit-sparsity-aware heterogeneous architecture, MSD framework can achieve $2.31\times$, $2.78\times$ and $2.84\times$ higher throughput on XC7Z020, ZU3EG, and ZU9EG devices. Therefore, the MSD framework has great potential to accelerate computation-intensive models by enhancing peak performance significantly.

D. Accuracy-Speedup Trade-off

To demonstrate the proposed hardware-aware mixed-EB quantization framework can balance between accuracy and speedup, we set up different constraints, quantize the ResNet50 and VGG16 models based on the search strategy, and use the DSP-only hardware as the baseline. According to Eqn. (10), the search framework obtains different combinations of EB_{L} for each layer. Fig. 9 presents the accuracy-speedup trade-off based on the mixed-EB framework on the Ultra-96 FPGA (ZU3EG). Generally, an increase in the latency constraint ω leads to higher speedup with accuracy loss because the framework will search for lower EB numbers to meet the demand. Besides, the quantized model can maintain a closer accuracy to the original model while still delivering a decent speedup (e.g., in VGG16, only $0.1\,\%$ accuracy drop with $2.0\times$ speedup). Moreover, our proposed framework with the heuristic search algorithm can quantize DNN models with trade-offs along the curves, which can serve the different requirements with various latency/accuracy constraints.

E. Comparison with the State-of-the-art

Table IV thoroughly compares the proposed accelerator design with the state-of-the-art FPGA accelerator works in terms of data precision, resource cost, post-quantization accuracy, latency, throughput, and compute efficiency. The throughput is calculated by the inference latency with the batch size set to 1. We set up all the layers to have $EB_{\mathrm{L}} = 2$ in our design since the accuracy-latency trade-off results show that the accuracy drop is negligible in this scenario. Besides, the measurement does not include the `im2col` pre-processing latency. Compared with Mix-and-Match [1] and N3H-Core [3], although they apply mixed-precision quantization that leads to lower computation latency and fewer data I/O, our work is still comparable with them due to the bit-sparsity optimization. For the compute-intensive model Resnet-18 on the XC7Z020 device, the latency result overpasses Mix-and-Match [1] and N3H-Core [3] as $1.79\times$ and $1.36\times$, respectively, even though we do not fully utilize the memory bandwidth. For the lightweight MobileNet-V2 model with small computation workloads, most layers are bounded in communication since we only apply 64-bit bitwidth per channel (i.e., we did not fully utilize the memory bandwidth compared with the prior works). Therefore, the previous works perform better than ours due to the mixed-precision optimization with smaller data I/O. Nevertheless, since the RSD-based bit-serial strategy keeps the data precision of weights and activations, our post-quantization accuracy is $4.91\,\%$ higher than the N3H-Core.

Our work still shows inspiring results compared to other general FPGA accelerators in DNN inference. Our work is faster than the Angel-eye [33] at $1.27\times$ regarding latency and throughput on VGG-16 due to the bit-sparsity. Besides, compared with the Vitis-AI [34], a vendor-specific framework in the industry for DNN accelerators on FPGA, we reduce the latency of ResNet-18 and ResNet-50 on ZU3EG as $44\,\%$ and $5\,\%$. For the models on ZU9EG, our results are still comparable with the Vitis-AI, even though they set up $1.34\times$ higher clock frequency. Finally, we also test our framework on one of the emerging transformer models, ViT-base, to demonstrate our universal design. The throughput result is comparable with the state-of-the-art FPGA accelerator, Auto-ViT-Acc [2] targeting a more hardware-friendly variant, DeiT-base. It is worth emphasizing that our work does not perform the best on lightweight models since they are not compute-intensive (e.g., MobileNet-V2), and the bit-sparsity properties cannot achieve higher performance when DNNs are bounded in communication. To conclude, by targeting a specific device, the proposed MSD scheme reaches promising improvement in

TABLE IV: Board-level comparison with state-of-the-art FPGA accelerators

Works	DP*	FPGA Devices	Frequency (MHz)	Resource Cost			DNN Models	Top-1 Acc.§	Latency (ms)	Throughput (GOPS)	Compute Efficiency	
				kLUT	DSP	BRAM					GOPS/kLUT	GOPS/DSP
DNNExplorer [32]	16	KU115	200	-	4686	-	VGG-16	-	18.05	1702.3	-	0.36
Angel-eye [33]	8	XC7Z020	125	29.87	190	85.5	VGG-16	-	364.00	84.3	2.83	0.44
Auto-ViT-Acc [2]	MP[1]	ZU9EG	150	179.0	1555	-	DeiT-base	81.14%	-	1970.3	11.01	1.27
Vitis-AI [34]	8	ZU3EG	287	-	326	126	ResNet-18	-	13.80	270.9	-	0.83
							ResNet-50	74.50%	30.80	250.0	-	0.77
		ZU9EG	287	-	2130	765	ResNet-18	-	5.10	713.2	-	0.33
							ResNet-50	74.50%	12.85	599.1	-	0.28
Mix&Match‡[1]	MP[1]	XC7Z020	100	28.29	220	56	ResNet-18	70.27%	47.10	77.0	2.72	0.35
							MobileNet-V2	65.64%	8.29	71.8	2.54	0.32
N3H-Core‡[3]	MP[1]	XC7Z020	100	39.62	220	137	ResNet-18	70.45%	35.79	101.3	2.56	0.46
				45.76	220	137	MobileNet-V2	66.25%	7.51	80.1	1.75	0.36
MSD-Ours	8†	XC7Z020	100	38.09	214	139	VGG-16	73.37%	287.18	107.9	2.83	0.50
							ResNet-18	69.72%	26.31	138.3	3.63	0.65
							MobileNet-V2	71.16%	16.40	38.9	1.02	0.18
		ZU3EG	214	55.71	264	194	VGG-16	73.37%	74.22	417.6	7.93	1.58
							ResNet-18	69.72%	7.72	471.7	8.96	1.79
							ResNet-50	76.05%	29.06	283.6	5.39	1.07
							MobileNet-V2	71.16%	7.41	86.1	1.64	0.33
		ZU9EG	214	151.69	2312	771	VGG-16	73.37%	52.70	588.2	3.88	0.25
							ResNet-18	69.72%	5.69	639.8	4.22	0.28
							ResNet-50	76.05%	15.94	516.9	3.41	0.22
							ViT-base¶	79.28%	22.30	1481.4	9.77	0.64

* Data precision in the hardware setup. [1] Mixed-precision based on bitwidths or formats. † We set all the $EB_{\mathrm{L}} = 2$ based on `int8` in this comparison.
§ Accuracy results in our work are based on QAT. ‡ For a fair comparison, we only selected results with batch size = 1. ¶ We only apply RSD in the MLP blocks.

the computation-bound models [28] due to the enhancement of peak performance.

V. Conclusions

This work proposed MSD framework, an FPGA-tailored and heterogeneous DNN acceleration framework that utilizes both LUTs and DSPs as computation resources and to exploit bit-sparsity. The RSD data representation enables MSD framework to fine-tune and encode the DNN weights into a bit-sparsity-aware format, making the bit-serial computation on LUTs more efficient. Furthermore, MSD framework uses a latency-driven algorithm to search for the optimal schedule, the number of EB, and the workload partitioning ratio for each layer. Evaluation results on various DNN models and edge FPGA devices demonstrate that MSD framework achieves $1.36\times$ speedup when compared with the state-of-the-art on ResNet-18 model, and $4.91\,\%$ higher accuracy on MobileNet-V2. In the future, we will explore more efficient scheduling methods for workload splitting in the heterogeneous architecture and EB selection in the bit-serial computation and exploit FPGA-layout-tailored hardware design to enhance the hardware clock frequency further.

VI. Acknowledgements

This work was supported in part by the Research Grants Council (RGC) of Hong Kong under the Research Impact Fund project R7003-21 and the Theme-based Research Scheme (TRS) Project T45-701-22-R. This work was also supported by AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Hong Kong SAR.

References

[1] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, "Mix and match: A novel FPGA-centric deep neural network quantization framework," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 208–220.

[2] M. Sun, Z. Li, A. Lu, H. Ma, G. Yuan, Y. Xie, H. Tang, Y. Li, M. Leeser, Z. Wang, X. Lin, and Z. Fang, "FPGA-aware automatic acceleration framework for vision transformer with mixed-scheme quantization: Late breaking results," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1394–1395. [Online]. Available: https://doi.org/10.1145/3489517.3530618

[3] Y. Gong, Z. Xu, Z. He, W. Zhang, X. Tu, X. Liang, and L. Jiang, "N3H-Core: Neuron-designed neural network accelerator via FPGA-based heterogeneous computing cores," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 112–122. [Online]. Available: https://doi.org/10.1145/3490422.3502367

[4] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. So, and K. Keutzer, "HAO: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 50–59.

[5] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[6] S. Li and P. Gupta, "Bit-serial Weight Pools: Compression and arbitrary precision execution of neural networks on resource constrained processors," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 238–250, 2022.

[7] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-Tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA:

Association for Computing Machinery, 2019, p. 749–763. [Online]. Available: https://doi.org/10.1145/3297858.3304041

[8] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 963–976.

[9] Y. Hao, Y. Zhao, C. Liu, Z. Du, S. Cheng, X. Li, X. Hu, Q. Guo, Z. Xu, and T. Chen, "Cambricon-P: A bitflow architecture for arbitrary precision computing," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 57–72.

[10] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-Pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 382–394.

[11] A. Li, H. Mo, W. Zhu, Q. Li, S. Yin, S. Wei, and L. Liu, "BitCluster: Fine-grained weight quantization for load-balanced bit-serial neural network accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.

[12] X. Zhao, Y. Wang, C. Liu, C. Shi, K. Tu, and L. Zhang, "BitPruner: Network pruning for bit-serial accelerators," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[13] A. Arora, S. Mehta, V. Betz, and L. K. John, "Tensor slices to the rescue: Supercharging ML acceleration on FPGAs," ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 23–33. [Online]. Available: https://doi.org/10.1145/3431920.3439282

[14] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 NX architecture and applications," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 57–67. [Online]. Available: https://doi.org/10.1145/3431920.3439293

[15] Y. Umuroglu, L. Rasnayake, and M. Själander, "BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 307–3077.

[16] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[17] A. Avizienis, "Signed-Digit numbe representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, 1961.

[18] R. M. Hewlitt and E. Swartzlantler, "Canonical signed digit representation for FIR digital filters," in *2000 IEEE Workshop on SiGNAL PROCESSING SYSTEMS. SiPS 2000. Design and Implementation (Cat. No. 00TH8528)*. IEEE, 2000, pp. 416–426.

[19] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 7, pp. 1044–1047, 1989.

[20] M. Riaz, R. Hafiz, S. A. Khaliq, M. Faisal, H. T. Iqbal, M. Ali, and M. Shafique, "CAxCNN: Towards the use of canonic sign digit based approximation for hardware-friendly convolutional neural networks," *IEEE Access*, vol. 8, pp. 127 014–127 021, 2020.

[21] M. A. Hanif, G. M. Sarda, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "CoNLoCNN: Exploiting correlation and non-uniform quantization for energy-efficient low-precision deep convolutional neural networks," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[22] B. Ahn and T. Kim, "Deeper weight pruning without accuracy loss in deep neural networks: Signed-Digit representation-based approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 656–668, 2022.

[23] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on Xilinx devices," *Xilinx White Paper*, 2016.

[24] X. Liu, Y. Chen, P. Ganesh, J. Pan, J. Xiong, and D. Chen, "HiKonv: High throughput quantized convolution with novel bit-wise management and computation," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 140–146.

[25] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 58–68.

[26] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to DNN accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[27] R. Shi, Y. Ding, X. Wei, H. Li, H. Liu, H. K.-H. So, and C. Ding, "FTDL: a tailored FPGA-overlay for deep learning with high scalability," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[28] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[29] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "ANT: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[30] F. Liu, W. Zhao, Z. He, Y. Wang, Z. Wang, C. Dai, X. Liang, and L. Jiang, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5281–5290.

[31] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[32] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "DNNExplorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[33] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 1, pp. 35–47, 2017.

[34] "Vitis-ai/models/ai-model-zoo at master · xilinx/vitis-ai," https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo, accessed: 2021-11-20.