2023 International Conference on Field Programmable Technology (ICFPT) | 979-8-3503-5911-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICFPT59805.2023.00013

M4BRAM: Mixed-Precision Matrix-Matrix Multiplication in FPGA Block RAMs

Yuzong Chen, Jordan Dotzel, Mohamed S. Abdelfattah
Department of Electrical and Computer Engineering, Cornell University
{yc2367, dotzel, mohamed}@cornell.edu

Abstract—Mixed-precision quantization is a popular approach for compressing deep neural networks (DNNs). However, it is challenging to scale the performance efficiently with mixed-precision DNNs given the current FPGA architecture and conventional accelerator dataflows. In this work, we enhance the FPGA's capability for accelerating mixed-precision DNNs by proposing M4BRAM, a novel compute-in-block RAM (BRAM) architecture that can compute mixed-precision matrix-matrix multiplication. On the precision side, M4BRAM supports a wide range of mixed-precision DNN configurations – the weight precision can be 2/4/8 bits while the activation precision can vary from 2 to 8 bits. On the dataflow side, M4BRAM leverages a novel in-BRAM data duplication scheme to achieve high hardware utilization. Moreover, during M4BRAM computation, other FPGA resources can seamlessly access its data without the need for a separate buffer. Hence, unlike prior compute-in-BRAM proposals, M4BRAM can simultaneously perform mixed-precision computation and maintain full functionality as a memory unit to *truly* complement the existing compute resources on FPGAs. Experiments show that adding M4BRAM to a tiled DNN accelerator can achieve an average speedup of 2.16× across various DNNs on the ImageNet classification task while incurring a negligible accuracy loss of < 0.5%. Compared to the same tiled accelerator that employs a prior compute-in-BRAM architecture, M4BRAM delivers 1.43× higher performance on average across various DNNs.

I. Introduction

Deep neural networks (DNNs) have demonstrated remarkable accomplishments in various important fields such as computer vision and natural language processing. Unfortunately, the scaling of compute performance and storage density following Moore's law has fallen far behind the growth of DNN model size [1], which necessitates model compression to reduce the storage and computation cost of DNNs. As one of the key compression techniques, quantization has been widely explored at both algorithmic [2]–[9] and hardware levels [10]–[23]. Although uniform low-precision such as binary [2], ternary [3], and 4-bit [4] quantization can significantly reduce the model size and improve the computational throughput, it can lead to a noticeable reduction in model accuracy. To mitigate such accuracy loss, mixed-precision quantization [6]–[9], where both weights and activations can have different bit-widths, has emerged as a promising quantization approach.

In the meanwhile, FPGAs have become an increasingly popular accelerator platform for DNNs due to their bit-programmability that allows customized precision and data-path [24]. Indeed, many mixed-precision DNN accelerators based on FPGAs have been proposed [12]–[14], [16], [17]. These accelerators mainly rely on the digital signal processing (DSP) block to implement the multiply-accumulate (MAC) operation, the fundamental primitive in DNN computing. To further increase the peak MAC throughput of FPGAs for accelerating DNNs, recent works have suggested adding compute-in-memory (CIM) capability inside the FPGA block RAM (BRAM) [21]–[23]. But these proposals either restrict DNN weights and activations to have the same precision [23], or have limited flexibility in feeding DSPs as a normal BRAM [21]–[23]. For example, when the BRAM in [21] and [22] is configured to the CIM mode, it can no longer be accessed by DSPs. As a result, the available BRAM resources have to be partitioned into two groups – one that performs CIM operations, and the other that feeds data to DSPs. This raises questions about the effective performance enhancement in realistic DNN accelerators on FPGAs.

In this work, we address the above limitations to enhance the capability of FPGAs for accelerating mixed-precision DNNs by proposing a new CIM architecture called M4BRAM, which can compute Mixed-precision Matrix-Matrix Multiplication in BRAM. Compared to prior compute-in-BRAM proposals, M4BRAM has three novel contributions. First, it supports diverse mixed-precision DNNs whose weights can be 2/4/8 bits, and activations can vary from 2 to 8 bits. The MAC latency/throughput of M4BRAM can efficiently scale with lower activation/weight precision. Second, M4BRAM only occupies one BRAM port during in-memory computing, while another BRAM port is always available for the DSP to receive data. Therefore, M4BRAM can remain as a memory unit while performing computation to *truly* complement the DSP. Third, M4BRAM employs a novel in-BRAM data duplication scheme that exploits DNN computational parallelism through both weight-sharing and activation-sharing to achieve high hardware utilization.

To demonstrate the flexibility and efficiency of M4BRAM in accelerating mixed-precision DNNs, we propose a heterogeneous tiled accelerator with M4BRAM functioning as a bit-serial engine and the DSP operating as a bit-parallel engine. The heterogeneous tiled accelerator significantly outperforms the baseline tiled accelerator without M4BRAM. Through activation quantization, it can deliver an average speedup of 2.16× across various DNNs while incurring a negligible accuracy loss of < 0.5% compared to the floating-point model on ImageNet classification. The performance gains further increase by adopting intra-layer weight quantization. Finally, compared to the heterogeneous accelerator that employs an ex-

979-8-3503-5911-4/23/$31.00 ©2023 IEEE
DOI 10.1109/ICFPT59805.2023.00013

isting compute-in-BRAM architecture, M4BRAM can provide 1.43× higher performance on average across various DNNs.

II. Related Works

A. FPGA Acceleration of Mixed-Precision DNNs

The FPGA's bit-level programmability makes it a competitive acceleration platform for mixed-precision DNNs. Colangelo *et al.* [13] assigned two separate precisions to weights and activations, respectively, and quantified the effects on accuracy, throughput, and FPGA resource utilization. Wang *et al.* [14] and Sun *et al.* [16] investigated accelerating inter-layer and intra-layer mixed-precision DNNs on FPGA, respectively. Another recent study, MSD [17], proposed quantizing DNN weights using two number representations, which can take advantage of the heterogeneous computing resources on FPGA to perform bit-serial and bit-parallel computation simultaneously.

To efficiently utilize the high-precision DSP multiplier for mixed-precision DNNs, these FPGA accelerators commonly employ DSP-packing [25] to combine multiple low-precision multiplications into one DSP. Unlike these works based on the existing FPGA architecture, our proposed M4BRAM is a new BRAM architecture that can complement DSP-packing to further improve the performance of mixed-precision DNNs.

B. Compute-In-Memory for Mixed-Precision DNNs

CIM has emerged as a new computing paradigm that achieves higher performance and energy efficiency than conventional Von-Neumann architectures by performing computation closer to the data. Although many CIM variants for mixed-precision DNNs have been proposed as ASICs [18]–[20], they suffer from limited precision support, such as [20] that only supports 1-/2-bit DNN inference and [18] that allows arbitrary weight precision but fixed 16-bit activation precision.

On the other hand, two recent BRAM-based CIM architectures, CCB [21] and CoMeFa [22], can compute with any precision using bit-serial arithmetic. However, during the CIM mode, their BRAM cannot support double-buffering which is a widely used technique in FPGA accelerators for DNNs [12]–[14], [16], [17]. A later compute-in-BRAM work, BRAMAC [23], performs CIM operations in a small dummy BRAM array that is decoupled from the main BRAM array. It proposes two variants, BRAMAC-2SA with two synchronous dummy BRAM arrays and BRAMAC-1DA with one double-pumped dummy BRAM array. In addition, the dummy BRAM array is controlled by an embedded finite-state machine (eFSM) to enable double-buffering. Nevertheless, BRAMAC requires weights and activations to have the same precision in 2-/4-/8-bit, limiting its applicability to only uniform-precision DNNs. Our proposed M4BRAM extends some concepts from BRAMAC to support variable activation precision from 2 to 8 bits with linearly scaled MAC latency. Moreover, M4BRAM simplifies interoperability with DSPs and improves hardware utilization through a novel data duplication scheme that can extract DNN computation parallelism from both weight-sharing and activation-sharing.

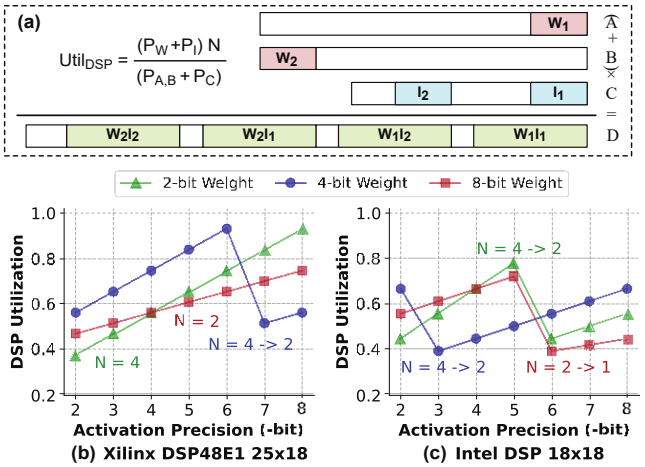

Fig. 1: (a) Packing 4 multiplications onto one DSP. The achievable packing factor (N) and resulting DSP utilization ($Util_{DSP}$) depend on weight precision (P_W), activation precision (P_I), and the DSP multiplier size ($P_{A,B}$ and P_C), which can vary across different FPGA vendors: (b) Xilinx 25×18-bit and (c) Intel 18×18-bit.

III. Motivation

A. DSP-Packing is Sub-Optimal for Mixed-Precision DNNs

As described in Section II-A, DSP-packing is widely adopted in FPGA accelerator design. Fig. 1(a) shows an example of packing four multiplications between two weights (W_1 and W_2) and two activations (I_1 and I_2) onto one DSP following the method in [25]. In general, the achievable packing factor and resulting DSP utilization depend on the operand precision and the DSP multiplier size, with the latter varying across different FPGA vendors.

Fig. 1(b) and (c) show the DSP utilization of packing mixed-precision multiplications on Xilinx and Intel DSPs, respectively. The decreasing part in a line reflects a 2× reduction in the achievable packing factor. First, given the same weight precision, reducing the activation precision cannot increase MAC throughput in most cases since the packing factor does not change, although one purpose of quantization is to scale the performance proportionally with reduced activation precision [10]. Second, given the same activation precision, reducing the weight precision by 2× also does not change the packing factor in many cases, although theoretically, a 2× throughput improvement is expected and can be achieved in some ASIC accelerators [11]. Based on these observations, it is necessary to design new FPGA blocks that can scale the performance with fine-grained quantization to get *measurable* hardware speedup for diverse mixed-precision DNNs.

B. CIM Should Complement DSPs

To fully exploit the potential of CIM on FPGA, BRAM should remain as a memory unit to feed the DSPs while performing CIM operations. Unfortunately, a pure bit-serial arithmetic approach (similar to CCB [21] and CoMeFa [22]) requires a transposed data layout, where each data word is stored along the BRAM column as shown in Fig. 2 (a) and (b), respectively. Hence, when a BRAM block is operating

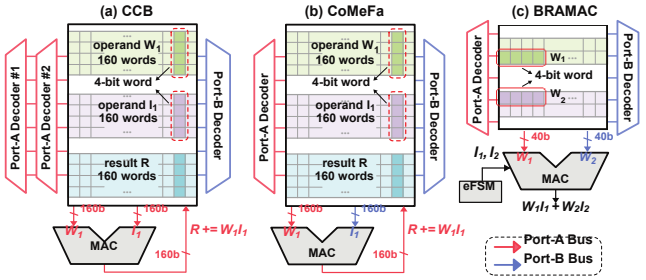

Fig. 2: Top-level dataflow of (a) CCB, (b) CoMeFa, (c) BRAMAC. The data word is assumed to be 4-bit.

in the CIM mode, a DSP can no longer access its data, therefore requiring a separate BRAM block configured as normal memory to feed the DSP. This can be wasteful and inefficient, especially when the data within the BRAM can be reused by more than one type of processing element as in heterogeneous DNN accelerators [17].

BRAMAC [23] solves the data layout issue by storing data words along the BRAM row, as shown in Fig. 2(c). However, when running in CIM mode, its two BRAM ports are both occupied by the in-BRAM MAC unit. Therefore, a DSP has to either receive data from a separate BRAM configured as normal memory or read the same operand being used by the in-BRAM MAC unit, resulting in constrained dataflow and limited parallelism. To overcome these challenges, a new CIM architecture that allows random access by the DSP while performing in-BRAM computation is desirable.

IV. M4BRAM Architecture and Dataflow

A. Overview

Fig. 3 shows the top-level architecture of M4BRAM based on the Intel M20K BRAM [26] with added circuit blocks colored in orange. The routing interface (i.e., input and output crossbar) of M4BRAM is the same as that of M20K. The configuration SRAM cell, mode-sram, allows M4BRAM to operate in the memory mode or the compute mode. During the memory mode, M4BRAM is identical to a conventional BRAM with configurable depth and data width. During the compute mode, M4BRAM can perform two MAC operations simultaneously, which is called a MAC2 operation [23] and defined as $P = (W_1 I_1 + W_2 I_2)$. The MAC2 operation is performed by 4 in-BRAM processing elements (BPEs) that are decoupled from the main BRAM array. This approach of decoupling computation from the main BRAM array is similar to that of BRAMAC. However, unlike BRAMAC, M4BRAM only occupies one port (port-A) of the main BRAM array to communicate with BPEs. A new duplication shuffler can reorder and replicate the weight vector read from the main BRAM array, which allows the 4 BPEs to compute 4 independent MAC2 operations, respectively. The rest of this section details M4BRAM's novel dataflow and new circuit blocks that make it much easier to be integrated within a DNN accelerator, surpassing prior works in terms of flexibility and hardware utilization.

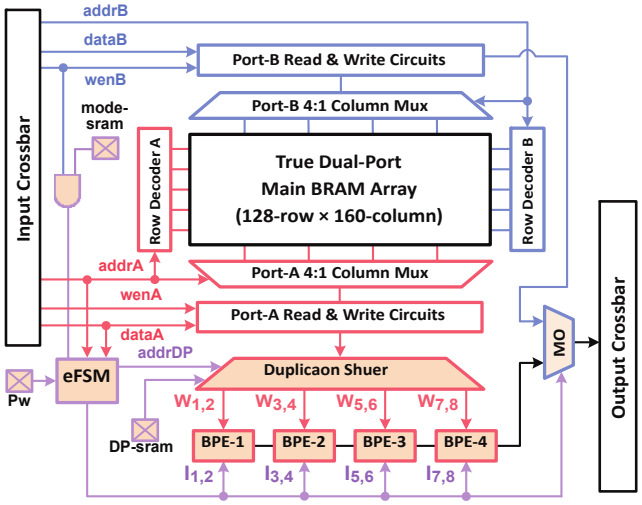

Fig. 3: Top-level architecture of M4BRAM modified from the Intel M20K BRAM. The added circuit blocks are highlighted in orange.

B. In-BRAM Compute Dataflow

In the compute mode, M4BRAM is automatically configured as a simple dual-port memory with a depth of 512 and a data width of 32 bits, which is the BRAM data width supported by both Intel [26] and Xilinx [27]. The main BRAM array's port-A and port-B are configured as write and read ports, respectively. Conventionally, the write-enable signal of the read port now becomes "don't care". This idle signal, however, is reused in M4BRAM to dynamically indicate a CIM instruction. If the write-enable signal wenB of the read port-B is asserted, then the write port-A's data dataA and address addrA will be treated as a CIM instruction and sent to the eFSM. This controls the 4 BPEs, each can compute an independent MAC2 operation between weights W and activations I. The weight precision Pw is pre-defined in configuration SRAM while the activation precision is run-time configurable through the CIM instruction as described later in Section IV-E.

To start a MAC2 operation, a 32-bit weight vector consisting of multiple low-precision weight elements is copied from the main BRAM array to 4 BPEs through port-A's read circuits. The duplication shuffler can select a subset of the weight vector and duplicate it multiple times depending on the duplication factor stored in configuration SRAM DP-sram. The activation vector is sent from outside (e.g., an activation buffer) to the eFSM through dataA, and then fed to the 4 BPEs. Since the read port-B is not occupied by BPE and eFSM during the entire MAC2 operation, it can continue to send the main BRAM array's data to other logic resources such as DSP. After M4BRAM finishes computing, port-B is used to read out the final result from BPE. The 2-to-1 mux MO selects the output data between BPE and the main BRAM array.

C. In-BRAM Processing Element (BPE)

The circuit design of BPE is based on the dummy BRAM array of BRAMAC [23] but with a different physical geom-

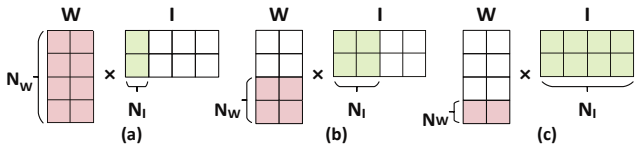

Fig. 4: Different parallelism configurations supported by M4BRAM for 8-bit weight precision: (a) N_W =4, N_I=1; (b) N_W =2, N_I=2; (c) N_W =1, N_I=4.

etry. In BRAMAC, the dummy BRAM array has a physical geometry of 7 rows × 160 columns. It can multiply the same activation with five 8-bit, ten 4-bit, or twenty 2-bit weights. In M4BRAM, however, each BPE contains a dummy array with 7 rows × 32 columns[1] that can multiply the same activation with one 8-bit, two 4-bit, or four 2-bit weights.

Another key difference between M4BRAM BPE and BRAMAC is the supported parallelism configuration defined by N_W and N_I, where N_W represents the number of weights multiplied by one activation and N_I represents the number of activations multiplied by one weight. BRAMAC only exploits the computation parallelism of DNNs through activation-sharing, i.e., it can only multiply the same activation with many weights by keeping N_I = 1 and scaling N_W. On the other hand, the 4 BPEs in M4BRAM can receive different activations and multiply them with the same weight to also exploit weight-sharing. Fig. 4 shows three different parallelism configurations supported by M4BRAM when the weight precision is 8 bits. The color-shaded area in each configuration represents 4 MAC2 that can be computed by 4 BPEs, respectively.

The reasoning behind supporting different parallelism configurations is that scaling only N_W can lead to low hardware utilization for DNNs that do not exhibit enough activation-sharing parallelism, as demonstrated in a recent study from Intel [28]. The study compares the hardware utilization of a tiled DNN accelerator across various parallelism configurations and DNNs, and shows that finding a balanced parallelism configuration based on the DNN's topology can significantly improve the hardware utilization and performance. Hence, compared to BRAMAC which only supports one parallelism configuration (i.e., N_I = 1), M4BRAM has the potential to achieve higher performance by flexibly adjusting its parallelism configuration for specific DNNs. Section V-E will quantify the benefits of M4BRAM over BRAMAC for accelerating various DNNs.

D. Duplication Shuffler

In conventional FPGA accelerators with the compute engine consisting of only DSP and soft logic, the same weight read from BRAM can be routed to different processing elements and multiplied by different activations to effectively increase N_I. In other words, the same weight must be *duplicated* in different processing elements to enable weight-sharing. To realize such weight duplication in M4BRAM, we propose

[1]As described later in Section IV-G, we also propose a second M4BRAM variant with a 7-row × 64-column dummy BRAM array in each BPE.

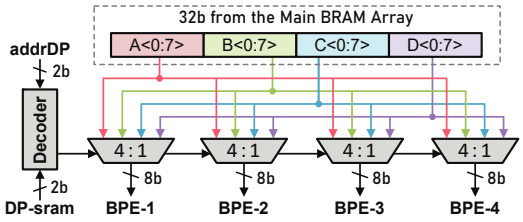

Fig. 5: Duplication shuffler.

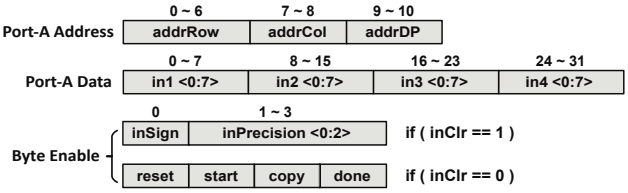

Fig. 6: M4BRAM's CIM instruction format.

a duplication shuffler as shown in Fig. 5. The duplication shuffler is composed of four 4-to-1 mux, all controlled by a decoder. The 32-bit weight vector read from the main BRAM array is divided into 4 slices (A, B, C, D) and sent to all 4-to-1 muxes. The decoder receives a 2-bit address addrDP to select a weight slice and duplicates it 1, 2, or 4 times depending on the duplication factor DP-sram (i.e., N_I). For example, if DP-sram = 1, then addrDP is ignored and the 4 BPEs will receive A, B, C, D, respectively, which gives the parallelism configuration shown in Fig. 4(a). On the other hand, if DP-sram = 4, then an 8-bit weight slice will be selected based on addrDP and broadcast to all BPEs, which gives the parallelism configuration shown in Fig. 4(c).

E. CIM Instruction Design

Fig. 6 shows M4BRAM's CIM instruction format. Before starting a MAC2 operation, the eFSM needs 2 cycles to receive 2 CIM instructions, which contain information for 2 weight/activation vectors, respectively. The port-A address contains row and column addresses (addrRow and addrCol) of a weight vector that needs to be copied from the main BRAM array to BPE. It also includes a 2-bit addrDP to control the duplication shuffler. The port-A data contains 4 input activations to be multiplied by the weight vector in 4 BPEs. Additional control information is encoded in each BRAM's 4-bit byte-enable signal which is supported by both Intel and Xilinx BRAMs. The inClr can be any M20K control signal that is unused by the main BRAM array during the compute mode. When inClr is activated, the input activation sign and precision can be changed to support layer-wise mixed activation precision. When inClr is deactivated, the byte-enable will contain flags to control the MAC2 operation.

F. Mixed-Precision Support

The proposed CIM instruction design allows M4BRAM's BPE to compute MAC2 with 2-/4-/8-bit weights and 2- to 8-bit activations chosen dynamically. Fig. 7(a) shows the diagram of a BPE consisting of a 7-row × 32-column dummy BRAM

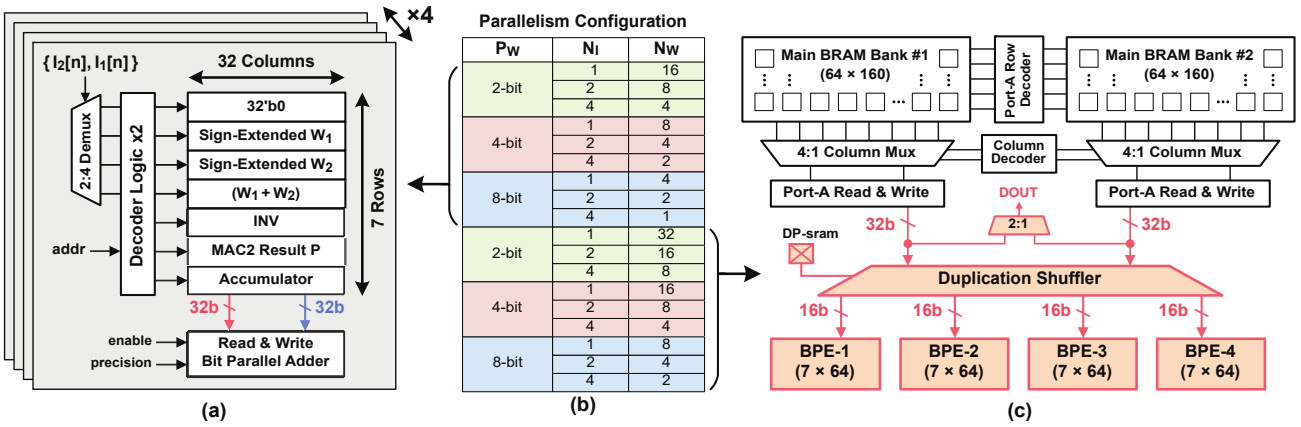

Fig. 7: M4BRAM's (a) BPE diagram; (b) Supported parallelism configurations (N_I and N_W) under different weight precision (P_W); (c) Architecture of M4BRAM-L. The DOUT control can come from the most significant bit of the read address.

array and peripheral circuits. The BPE performs computation on sign-extended weight vectors and uses a look-up table approach to generate the partial sum [19]. In every cycle, based on the received two activation bits $\{I_2[n], I_1[n]\}$, the partial sum is chosen from one of the first four BRAM rows and added to the MAC2 result P. The INV row is used to store a temporary inverted partial sum if the activation is a signed number. The final MAC2 result P is added to the last BRAM row for accumulation. More details on the above bit-serial MAC2 dataflow, as well as the circuit design of different components, are detailed in [23].

With bit-serial processing of activations, the MAC2 operation in M4BRAM takes $(n + 2)$ cycles for n-bit activation when the BPE is synchronous with the main BRAM array, and this latency can be further reduced to $(n/2 + 2)$ by double-pumping the BPE with a $2\times$ main BRAM clock frequency [23]. Furthermore, a lower weight precision improves MAC2 throughput because the 32-bit weight vector copied from the main BRAM array can contain more elements. As a result, M4BRAM can provide *measurable* hardware speedup for diverse mixed-precision DNNs through both activation quantization and weight quantization.

G. M4BRAM Variants

To explore the trade-off between the MAC throughput gain and the area overhead, we propose two M4BRAM variants, called M4BRAM-S and M4BRAM-L, whose BPEs contain small and large dummy BRAM arrays, respectively. The BPE of M4BRAM-S has the same structure as the one described in Section IV-F. Fig. 7(b) shows the parallelism configurations supported by M4BRAM-S for different weight precision. Reducing the weight precision allows multiplying the same activation with more weights, leading to higher N_W. This variant has a smaller area overhead and offers moderate MAC throughput gain.

Fig. 7(c) shows the architecture of M4BRAM-L whose BPE contains a large dummy BRAM array with 7 rows and 64 columns. Compared to M4BRAM-S, M4BRAM-L can achieve $2\times$ higher weight-sharing parallelism as shown in Fig. 7(b). But it also requires $2\times$ read bandwidth to copy more weights from the main BRAM array. To achieve this, we apply memory banking to divide the main BRAM array into two 64-row × 160-column banks so that each can provide a 32-bit weight vector. Note that the read bandwidth is increased only internal to the BRAM, an additional 2-to-1 mux selects between two banks during normal BRAM access.

H. Heterogeneous Accelerator with M4BRAM and DSP

During a MAC2 operation, the BPE and eFSM of M4BRAM only occupy the main BRAM array's write port (for 2 cycles) to receive operands, while the DSP can continue to randomly access the main BRAM array through the read port. Hence, the BPE and DSP can receive data from a shared M4BRAM cache and complement each other in a heterogeneous fashion, where the BPE is a bit-serial engine with its latency proportional to the activation precision, and the DSP is a bit-parallel engine with fixed 1-cycle latency.

To illustrate how the BPE and DSP can coordinate in a tiled accelerator, we use Intel's Deep Learning Accelerator (DLA) [28] as an example. Fig. 8(a) shows how a CNN can be partitioned into tiles along different dimensions in DLA, where C_{VEC}/K_{VEC} represents the tile size along the input/output channel, R_{VEC} represents the tile size along the filter height and width (assuming square filter), and P_{VEC}/Q_{VEC} represents the tile size along output height/width. In order for the BPE and DSP to work in parallel, the workload can be distributed along Q_{VEC} so that they will receive/compute different inputs/output features (indicated by blue and red cubes in Fig. 8(a)). It is also possible to partition the workload along other dimensions, which is left for future work.

Fig. 8(b) shows the proposed heterogeneous accelerator (Hetero-DLA) that enhances DLA by using M4BRAM to store data. The input buffer sends two sets of input features to the BPE and DSP, respectively. The filter data stored in M4BRAM's main BRAM array can be randomly accessed by both the BPE and DSP. For example, if the DSP computes faster, it can start to compute the next tile, C2, along the input

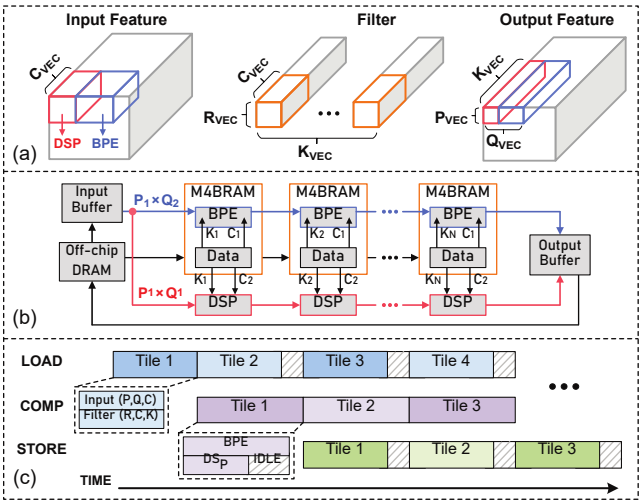

Fig. 8: (a) Partitioning a CNN into tiles along different dimensions in DLA. The tile is distributed to BPE and DSP along Q_{VEC}. (b) Proposed heterogeneous DLA. (c) Inter-tile pipelined dataflow.

channel while the BPE is still computing on C1. The final results from the two engines are sent to the output buffer.

Existing tiled CNN accelerators commonly apply double-buffering to reduce or hide off-chip DRAM access latency [12]–[14], [16], [17]. Double-buffering allows an accelerator to pipeline three stages: loading inputs and filters, computing convolution, and storing outputs, as illustrated in Fig. 8(c). Similar to BRAMAC, M4BRAM supports double-buffering during the compute mode – when the eFSM is not receiving a CIM instruction, the main BRAM array's write port is free and can be used to load the next tile. During convolution, the tile latency is determined by the slower latency between the BPE and DSP. In addition, when the BPE completes a dot product, the result needs to be read out through the main BRAM's read port, forcing the DSP to stall for 4 cycles and 8 cycles in M4BRAM-S and M4BRAM-L, respectively given a 32-bit BRAM data width. Fortunately, this stall overhead can be amortized over many MAC2 operations in the BPE since the dot product size is usually large in real-world CNNs. For example, our performance simulator (discussed in Section V-A) shows that such DSP stalls contribute to an average of only 4.8% of the total execution time for VGG-16 with 8-bit weights and the activation varying from 4 to 8 bits.

V. Evaluation

A. Experimental Setup

M4BRAM modeling: We use COFFE [29] with 22-nm predictive technology to model the area and delay of the baseline M20K as well as new M4BRAM circuits. For M4BRAM's eFSM, we write SystemVerilog and synthesize it using Synopsys Design Compiler with TSMC 28-nm technology and scale the reported area to 22-nm technology.

DNN benchmarks: We choose AlexNet, VGG-16, ResNet-18, ResNet-34, and one self-attention module from the base Vision Transformer (ViT-Base) [30]. For DNNs used in the mixed-precision training experiments, we train them on ImageNet classification. The baseline 32-bit floating-point (FP32) models are taken from PyTorchCV [31] and quantized to fixed-point using uniform symmetric quantization. The quantization clipping thresholds are determined by minimizing the mean absolute error on the original weights and activations, where activation statistics are estimated with a large random training batch. For experiments involving intra-layer weight quantization, the weights are partitioned into two slices along the output dimension and then quantized individually with the above method. The models are then fine-tuned using the default Adam optimizer with a learning rate of 1e-5 for 20 epochs following a cosine decay learning rate schedule.

Baseline and enhanced accelerator: We choose two baseline FPGAs from Intel's Stratix-10 family: GX400 and GX650 (Table I). We apply Hetero-DLA described in Section IV-H to evaluate the DNN benchmarks. We compare the performance of Hetero-DLA to that of DLA, which has normal BRAM instead of M4BRAM. The matrix multiplication operation in the self-attention module of ViT-Base is converted to 1D convolution [28]. To maximize each accelerator's performance, we develop a design space exploration tool to find the optimal tiling configuration for every DNN, following a similar approach used in prior FPGA accelerator works [16], [28]. The optimization target is set to $perf \times (perf/area)$ to balance the performance and area cost. We build a cycle-accurate simulator to obtain the latency of the DSP and BPE based on a given tiling configuration that splits the workload distribution between the M4BRAM and DSP along the Q_{VEC} dimension.

TABLE I: Properties of the baseline Stratix-10 FPGAs

	GX400		GX650	
Resources	Count	Area(%)[1]	Count	Area(%)[1]
Logic Block	12816	55.6	20736	54.7
DSP	648	15.7	1152	17.0
M20K BRAM	1537	28.7	2489	28.3

[1] Calculated based on the normalized area metrics in [32].

B. M4BRAM Area and Frequency

Compared to M20K, the area overhead of M4BRAM-S comes from the duplication shuffler, eFSM, and 4 BPEs, which are 42 μm^2, 238 μm^2, and 856 μm^2 respectively. This represents an area increase of 19.6% over M20K. For M4BRAM-L, the area of BPE and duplication shuffler double [2], leading to a 33.4% area increase compared to M20K. With M20K constituting ˜29% of the core area in the two baseline FPGAs, M4BRAM-S and M4BRAM-L will increase the FPGA core area by 5.6% and 9.5%, respectively, which are higher than those reported in the previous works [21]–[23] mainly because our baseline FPGAs have a higher M20K:DSP ratio. Furthermore, Section V-F will show that using the same area, M4BRAM will outperform DSP for accelerating mixed-precision DNNs, thus offering a better area vs. performance scaling.

[2]We do not consider the area overhead associated with memory banking as it is a feature already employed in some commercial BRAMs. The COFFE paper on BRAM [29] reports all results based on 2-bank architecture.

TABLE II: Comparison between M4BRAM and prior compute-in-BRAM architectures

Property	CCB	CoMeFa		BRAMAC		M4BRAM	
Variant	–	D	A	1DA	2SA	S	L
# of dummy arrays	–	–	–	1	2	4	4
Dummy array size	–	–	–	7×160	7×160	7×32	7×64
Weight precision (-bit)	Arbitrary	Arbitrary	Arbitrary	2, 4, 8	2, 4, 8	2, 4, 8	2, 4, 8
Activation precision (-bit)	Arbitrary	Arbitrary	Arbitrary	2, 4, 8	2, 4, 8	2 – 8	2 – 8
Weight-sharing factor N_I	1	1	1	1	2	1, 2, 4	1, 2, 4
Use transposed data layout	✓	✓	✓	×	×	×	×
Allow DSP access during CIM	×	×	×	×	×	✓	✓
Support mixed-precision	✓	✓	✓	×	×	✓	✓
Support multi-pumping	×	×	✓	✓	×	✓	✓
# of occupied main BRAM ports	Two	Two	Two	Two	Two	One	One
M20K area overhead	16.8%	25.4%	8.1%	16.9%	33.8%	19.6%	33.4%

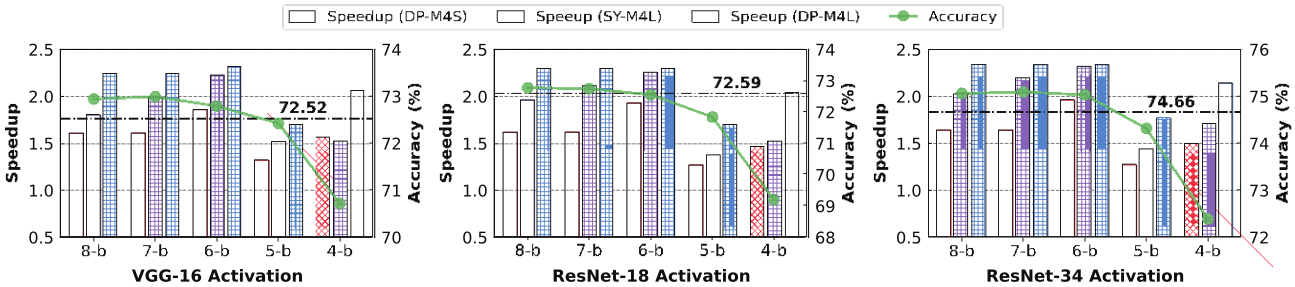

Fig. 9: Model accuracy and speedup over DLA for various DNNs. The weight precision is 8-bit. The dash-dotted black line represents a 0.5% Top-1 accuracy loss compared to the baseline FP32 model.

The baseline M20K has a maximum frequency of 730 MHz in 20-nm technology [33]. With COFFE's 22-nm technology, the BPE's critical path delays are 903 ps and 925 ps in M4BRAM-S and M4BRAM-L, respectively, which are lower compared to BRAMAC (958 ps) because of the smaller dummy BRAM array size. When the BPE is double-pumped, M4BRAM-S and M4BRAM-L will limit M20K's frequency to 553 MHz and 540 MHz, respectively, which are 1.26× lower than the baseline. A lower M20K frequency is not a concern because realistic FPGA delays are usually constrained by soft logic and routing, and it is unlikely that an FPGA accelerator will run more than 500 MHz even on Stratix-10 with a more advanced 14-nm technology [34].

C. Comparison with Prior CIM Architectures

Table II shows the architectural differences between M4BRAM and prior CIM architectures for FPGA, including CCB [21], CoMeFa [22] and BRAMAC [23]. Both CCB and CoMeFa require a transposed BRAM data layout that can not be accessed by DSP, leading to restricted dataflow and limited throughput improvement. Although M4BRAM shares a few common features with BRAMAC, such as using a dummy BRAM array (but with different sizes) to compute MAC2, it can perform mixed-precision computation as opposed to BRAMAC. Furthermore, unlike other works that only support a fixed N_I, M4BRAM can flexibly adjust its N_I based on a DNN's topology to improve the utilization efficiency. Notably, BRAMAC-2SA with $N_I = 2$ can cause significant under-utilization for DNNs that involve matrix-vector multiplication, e.g., unbatched long short-term memory (LSTM) networks or attention-based models that require $N_I = 1$. Finally, M4BRAM only occupies one main BRAM port when performing CIM operations, thus offering more efficient interoperability with DSP compared to other CIM architectures.

D. Mixed-Precision Accuracy vs. Performance Trade-off

Accuracy, performance vs. activation quantization: Since the computation latency of M4BRAM scales linearly with the activation precision, we first evaluate the model accuracy and performance by sweeping the activation precision from 4 to 8 bits while keeping the weight precision at 8 bits. We choose the GX650 FPGA in this experiment. For Hetero-DLA, we consider three M4BRAM configurations – M4BRAM-S with double-pumped BPE (DP-M4S), and M4BRAM-L with synchronous (SY-M4L) or double-pumped BPE (DP-M4L).

Fig. 9 presents the Top-1 model accuracy vs. performance trade-off for various DNNs. For each DNN, we observe a drop in speedup when the activation becomes 5 bits, which is due to a 2× increase in the DSP-packing factor as described in Sec-tion III-A. However, 5-bit activation causes a non-negligible accuracy loss of $>$ 0.5% for all three evaluated DNNs, with a maximum of 0.94% for ResNet-18. This highlights that solely relying on DSP-packing cannot provide an optimal accuracy vs. performance trade-off for mixed-precision DNNs.

When the activation precision is 6-bit, the three M4BRAM configurations deliver an average speedup of 2.16× across all evaluated DNNs, while incurring an accuracy loss of $<$ 0.5%. Specifically, the average speedups achieved by DP-

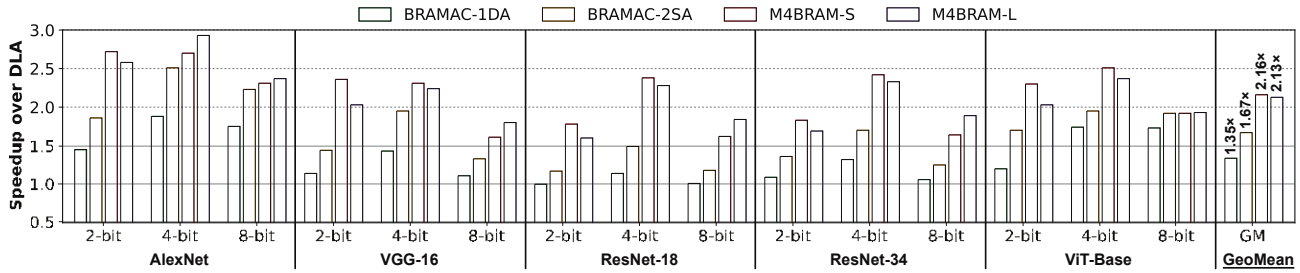

Fig. 10: Speedup of M4BRAM and BRAMAC over Intel's DLA for single-precision DNNs.

TABLE III: Top-1 accuracy of ResNet-34 under different % of 8-bit filters and the speedup of SY-M4L vs. all-4b DLA.

Model Config	Weight Precision	Activation Precision	Top-1 Accuracy	Speedup vs. 4b DLA
Baseline	32b	32b	75.16%	N.A.
Retrain	32b	32b	75.49%	N.A.
Mixed	95% 4b + 5% 8b	6b	75.22%	2.33× Mixed
85% 4b +	15% 8b	6b	75.26%	2.02× Mixed
75% 4b +	25% 8b	6b	75.37%	2.02×

M4S, SY-M4L, and DP-M4L are 1.92×, 2.26×, and 2.31×, respectively. When considering cases where the activation precision changes from 8 to 6 bits, the performance of SY-M4L linearly increases, while the double-pumping feature of DP-M4S leads to an incremental speedup for every 2-bit reduction in the activation precision. Interestingly, DP-M4L shows nearly identical speedup. We attribute this to the much higher MAC throughput of DP-M4L compared to DSP, which causes the tile latency to be bottlenecked by the latter.

Accuracy, performance vs. weight quantization: As described in Section IV-F, the MAC throughput of M4BRAM can be scaled by 2× with a 2× lower weight precision. This can lead to further performance improvement through intra-layer weight quantization. Recent work has demonstrated that an FPGA-friendly intra-layer quantization approach is to have two filter groups with 4-bit and 8-bit precision, respectively, in every layer [16]. By properly choosing a uniform activation precision and tuning the ratio R of the 8-bit filters, it is possible to achieve higher performance than the pure 8-bit model while maintaining accuracy comparable to the FP32 model.

To evaluate the potential of Hetero-DLA for intra-layer weight quantization, we choose the GX400 FPGA employing M4BRAM-L with synchronous BPE and follow the approach in [16] by dividing the FPGA resources into two groups. Each group utilizes M4BRAM-L to store either 4-bit or 8-bit filters and computes convolution using the available DSP and M4BRAM-L resources. We conduct intra-layer weight quantization experiments on ResNet-34 as a case study. Since we have shown that 6-bit activation precision has negligible accuracy loss compared to the FP32 model, we set all activation to 6-bit during mixed-precision training.

Table III shows the Top-1 accuracy of ResNet-34 under different mixtures of 4-bit and 8-bit filters, as well as the resulting speedup achieved by Hetero-DLA over the all 4-bit model on DLA. All mixed-precision configurations achieve < 0.3% accuracy loss compared to the retrained FP32 model. When the ratio of 8-bit filters R increases from 5% to 15%, the accuracy increases by only 0.04%, but the speedup reduces from 2.33× to 2.02×. This is because with R = 15% 8-bit filters, the number of DSPs on the GX400 FPGA cannot achieve the same tiling configuration as R = 5%. Specifically, the optimal tiling configuration for R = 5% consumes 816 M4BRAM and 612 DSP. To maintain the same tiling configuration when R = 15%, the resource utilization will increase to 912 M4BRAM and 666 DSP, which exceeds the number of DSP available on the GX400 FPGA. These results demonstrate that combin-ing M4BRAM and DSP on a heterogeneous accelerator can provide a good trade-off between accuracy and performance when quantizing both weights and activations

E. M4BRAM vs. BRAMAC

BRAMAC is the state-of-the-art compute-in-BRAM architecture and has outperformed other prior approaches. We therefore focus on comparing the performance of M4BRAM and BRAMAC for accelerating real-world DNNs. We follow the same evaluation methodology of BRAMAC [23] by using DLA as the baseline accelerator, and calculate the execution time of DLA that employs BRAMAC. We consider precision configurations where weights and activation have the same precision in 2-/4-/8-bit. For VGG-16, ResNet-18, and ResNet-34, the 8-bit configurations are evaluated using the GX650 FPGA due to the need for a larger buffer based on DLA's BRAM usage model [35]. Other DNNs and precision configurations are evaluated using the GX400 FPGA. In addition, we apply double-pumping to M4BRAM-S to ensure a fair comparison with BRAMAC-1DA since these two architectures have similar M20K area overhead.

Overall performance comparison: Fig. 10 shows the overall speedup over the baseline DLA after replacing the normal BRAM with BRAMAC or M4BRAM. On average, M4BRAM outperforms BRAMAC by 1.43× across various DNNs. Specifically, BRAMAC-1DA and BRAMAC-2SA achieve an average speedup of 1.35× and 1.67× over DLA, respectively. M4BRAM-S and M4BRAM-L provide even higher speedups over DLA, reaching 2.16× and 2.13× on average, respectively. These additional performance gains of M4BRAM are attributed to allowing DSP to freely access the main BRAM array and supporting different parallelism configurations. Among the evaluated DNNs, AlexNet and ViT-Base

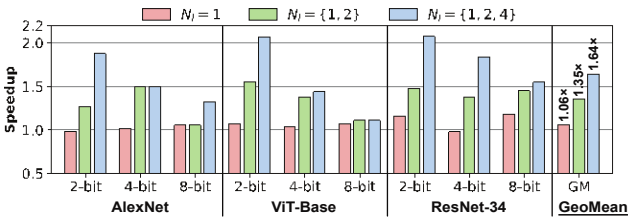

Fig. 11: Speedup over BRAMAC-1DA for when M4BRAM-S supports different sets of parallelism configurations.

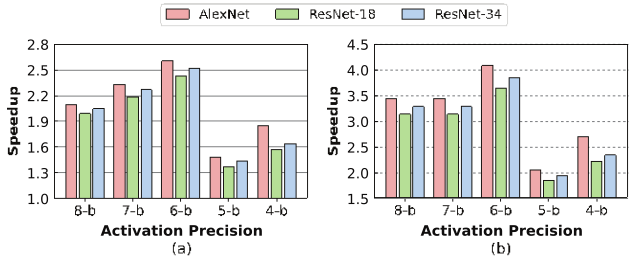

Fig. 12: Speedup over GX-DSP when GX-M4 employs M4BRAM-L with (a) synchronous and (b) double-pumped BPE. The weight precision is 8-bit.

show higher speedup since they contain more output channels in each layer, thus exhibiting higher N_W that is well exploited by both BRAMAC and M4BRAM. On the other hand, VGG-16, ResNet-18, and ResNet-34 contain relatively fewer output channels in their first several layers. As a result, continuing to increase N_W as in BRAMAC won't benefit performance due to low hardware utilization, while M4BRAM can exploit weight-sharing (by increasing N_I) to achieve higher hardware utilization and speedup.

Performance ablation study: The performance benefits of M4BRAM are twofold: better interoperability with DSP by using only one BRAM port for CIM, and flexible parallelism configurations through in-BRAM data duplication. To further investigate the effects of these two features, we conduct ablation studies to compare the performance of double-pumped M4BRAM-S and BRAMAC-1DA. We restrict the supported parallelism configurations of M4BRAM by changing the weight-sharing factor N_I in our design space exploration tool. To test the performance benefits of M4BRAM's better interoperability with DSP, we simply set $N_I = 1$, which effectively disables the in-BRAM data duplication scheme.

Fig. 11 shows the resulting speedup of M4BRAM-S over BRAMAC-1DA across three DNNs. When $N_I = 1$, i.e., without in-BRAM data duplication, M4BRAM-S achieves slightly better performance (1.06× on average) compared to BRAMAC-1DA. It is important to note that the total dummy array size of BRAMAC-1DA is 7×160, which provides $1.25\times$ higher peak performance than M4BRAM-S with a total dummy array size of 7×128. But the better interoperability between DSP and M4BRAM enables a more efficient dataflow, effectively compensating for the lower peak performance. On the other hand, as M4BRAM-S supports more parallelism configurations, the speedup over BRAMAC-1DA becomes higher. When supporting all three parallelism configurations shown in Fig. 4, M4BRAM-S achieves an average speedup of 1.64× over BRAMAC-1DA. These results demonstrate the additional performance that is enabled by the flexibility within M4BRAM, despite its lower peak performance compared to BRAMAC.

F. M4BRAM vs. DSP

This section shows that M4BRAM can outperform DSP in terms of performance per area. As described in Section V-B, the area overhead of M4BRAM-L translates to a 9.5% increase in the FPGA core area. On the GX650 FPGA, this is equivalent to 640 DSPs. To compare the performance of M4BRAM and DSP given the same area budget, we use two GX650-like FPGAs. The first is called GX-M4 with 2489 M4BRAM-L but no DSP, and another is called GX-DSP containing 2489 normal BRAM and 640 DSP. We analytically model the performance of these two FPGAs for accelerating AlexNet, ResNet-18, and ResNet-34 with 8-bit weight. Fig. 12 shows the speedup of GX-M4 over GX-DSP when sweeping the activation precision from 4 to 8 bits. On average GX-M4 offers 1.98× and 2.95× higher performance than GX-DSP across various DNNs. These results highlight that M4BRAM can offer a better performance vs. area scaling compared to DSP, whose high-precision multiplier suffers from significant under-utilization for low-precision multiplications.

VI. Conclusion

In this paper, we have proposed a new compute-in-BRAM architecture, called M4BRAM, to enhance the FPGA's capability for accelerating mixed-precision DNNs. M4BRAM supports diverse mixed-precision configurations with 2-/4-/8-bit weights and 2- to 8-bit activations to enable effective performance vs. accuracy trade-off. Evaluation results on various mixed-precision DNNs demonstrate that by employing M4BRAM to a tiled accelerator, an average speedup of 2.16× can be achieved with negligible accuracy loss of $<$ 0.5% with the mixed-precision DNN compared to the FP32 baseline. Furthermore, M4BRAM offers different parallelism configurations and better interoperability with DSPs, making it outperform a state-of-the-art compute-in-BRAM architecture by 1.43× on average across various DNNs. Finally, compared to DSPs, we show that M4BRAM can be more efficient for accelerating mixed-precision DNNs. With continuous advancements in mixed-precision quantization of DNNs, we believe that adding M4BRAM to future AI-optimized FPGAs can be highly valuable.

VII. Acknowledgement

We would like to thank the anonymous reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under Grant No. 2303626. We would also like to thank Ilya Ganusov for the valuable discussion about FPGA BRAM architectures.

REFERENCES

[1] S. Lie, "Thinking Outside the Die: Architecting the ML Accelerator of the Future," in *54th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 2016. [Online]. Available: https://arxiv.org/abs/1602.02830

[3] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, "Ternary Weight Networks," 2016. [Online]. Available: https://arxiv.org/abs/1605.04711

[4] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Neural Information Processing Systems (NIPS)*, 2018.

[5] Q. Jin, J. Ren, R. Zhuang, S. Hanumante, Z. Li, Z. Chen, Y. Wang, K.-M. Yang, and S. Tulyakov, "F8Net: Fixed-Point 8-bit Only Multiplication for Network Quantization," 2022. [Online]. Available: https://arxiv.org/abs/2202.05239

[6] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. Garcıa, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed Precision DNNs: All you need is a good parametrization," in *International Conference on Learning Representations (ICLR)*, 2019.

[7] Q. Lou, F. Guo, L. Liu, M. Kim, and L. Jiang, "AutoQ: Automated Kernel-Wise Neural Network Quantization," 2019. [Online]. Available: https://arxiv.org/abs/1902.05690

[8] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney, and K. Keutzer, "HAWQV3: Dyadic Neural Network Quantization," in *International Conference on Machine Learning (ICML)*, 2021.

[9] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[10] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *49th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[11] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *45th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2018.

[12] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K.-H. So, and K. Keutzer, "HAO: Hardware-aware Neural Architecture Optimization for Efficient Inference," in *29th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[13] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. K. Mishra, M. Margala, and K. Nealis, "Exploration of Low Numeric Precision Deep Learning Inference Using Intel® FPGAs," in *26th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.

[14] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, "Design Flow of Accelerating Hybrid Extremely Low Bit-Width Neural Network in Embedded FPGA," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[15] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 599–615.

[16] M. Sun, Z. Li, A. Lu, Y. Li, S.-E. Chang, X. Ma, X. Lin, and Z. Fang, "FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022.

[17] J. Wu, J. Zhou, Y. Gao, Y. Ding, N. Wong, and H. K.-H. So, "MSD: Mixing Signed Digit Representations for Hardware-efficient DNN Acceleration on FPGA with Heterogeneous Resources," in *31st IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[18] J. Kim, J. Lee, J. Lee, H.-J. Yoo, and J.-Y. Kim, "Z-PIM: An Energy-Efficient Sparsity Aware Processing-In-Memory Architecture with Fully-Variable Weight Precision," in *IEEE Symposium on VLSI Technology and Circuits*, 2020.

[19] C.-F. Lee, C. Lu, C.-E. Lee, H. Mori, H. Fujiwara, Y.-C. Shih, T.-L. Chou, Y. D. Chih, and T.-Y. J. Chang, "A 12nm 121-TOPS/W 41.6-TOPS/mm2 All Digital Full Precision SRAM-based Compute-in-Memory with Configurable Bit-width For AI Edge Applications," in *IEEE Symposium on VLSI Technology and Circuits*, 2022.

[20] B. Zhang, S. Yin, M. Kim, J. Saikia, S.-C. Kwon, S. Myung, H. Kim, S. J. Kim, J. sun Seo, and M. Seok, "PIMCA: A Programmable In-Memory Computing Accelerator for Energy-Efficient DNN Inference," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 5, pp. 1436–1449, 2023.

[21] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. R. Iyer, and R. Das, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," in *29th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[22] A. Arora, T. Anand, A. Borda, R. Sehgal, B. Hanindhito, J. Kulkarni, and L. K. John, "CoMeFa: Compute-in-Memory Blocks for FPGAs," in *30th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022.

[23] Y. Chen and M. S. Abdelfattah, "BRAMAC: Compute-in-BRAM Architectures for Multiply-Accumulate on FPGAs," in *31st IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[24] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in *24th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016.

[25] Xilinx, "Convolutional Neural Network with INT4 Optimization on Xilinx Devices," 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp521-4bit-optimization

[26] D. M. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, "Architectural Enhancements in Stratix V," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2013.

[27] Xilinx, "7 Series FPGAs Memory Resources User Guide (UG473)," 2019. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug473_7Series_Memory_Resources

[28] M. S. Abdelfattah, D. Han, A. Bitar, R. Dicecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling, and G. R. Chiu, "DLA: Compiler and FPGA Overlay for Neural Network Inference Acceleration," in *28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[29] S. Yazdanshenas, K. Tatsumura, and V. Betz, "Don't Forget the Memory: Automatic Block RAM Modelling, Optimization, and Architecture Exploration," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 115–124.

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[31] O. Sémery, "PyTorchCV," 2023. [Online]. Available: https://pypi.org/project/pytorchcv/

[32] A. Arora, S. Mehta, V. Betz, and L. K. John, "Tensor Slices to the Rescue: Supercharging ML Acceleration on FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.

[33] Intel, "Intel Arria 10 Core Fabric and General Purpose I/Os Handbook," 2022. [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683461.pdf

[34] M. Stan, M. Hall, M. A. Ibrahim, and V. Betz, "HPIPE NX: Boosting CNN Inference Acceleration Performance with AI-Optimized FPGAs," in *International Conference on Field-Programmable Technology (ICFPT)*, 2022, pp. 1–9.

[35] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ Deep Learning Accelerator on Arria 10," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.