Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks

Hardik Sharma* Jongse Park* Naveen Suda† Liangzhen Lai†
Benson Chau* Vikas Chandra† Hadi Esmaeilzadeh‡

Alternative **C**omputing **T**echnologies (**ACT**) Lab

*Georgia Institute of Technology †Arm, Inc. ‡University of California, San Diego
{hsharma,jspark,ben.chau}@gatech.edu {naveen.suda,liangzhen.lai,vikas.chandra}@arm.com hadi@eng.ucsd.edu

***Abstract*—Hardware acceleration of Deep Neural Networks (DNNs) aims to tame their enormous compute intensity. Fully realizing the potential of acceleration in this domain requires understanding and leveraging algorithmic properties of DNNs. This paper builds upon the algorithmic insight that bitwidth of operations in DNNs can be reduced without compromising their classification accuracy. However, to prevent loss of accuracy, the bitwidth varies significantly across DNNs and it may even be adjusted for each layer individually. Thus, a fixed-bitwidth accelerator would either offer limited benefits to accommodate the worst-case bitwidth requirements, or inevitably lead to a degradation in final accuracy. To alleviate these deficiencies, this work introduces dynamic bit-level fusion/decomposition as a new dimension in the design of DNN accelerators. We explore this dimension by designing Bit Fusion, a bit-flexible accelerator, that constitutes an array of bit-level processing elements that dynamically fuse to match the bitwidth of individual DNN layers. This flexibility in the architecture enables minimizing the computation and the communication at the finest granularity possible with no loss in accuracy. We evaluate the benefits of Bit Fusion using eight real-world feed-forward and recurrent DNNs. The proposed microarchitecture is implemented in Verilog and synthesized in 45 nm technology. Using the synthesis results and cycle accurate simulation, we compare the benefits of Bit Fusion to two state-of-the-art DNN accelerators, Eyeriss [1] and Stripes [2]. In the same area, frequency, and process technology, Bit Fusion offers 3.9× speedup and 5.1× energy savings over Eyeriss. Compared to Stripes, Bit Fusion provides 2.6× speedup and 3.9× energy reduction at 45 nm node when Bit Fusion area and frequency are set to those of Stripes. Scaling to GPU technology node of 16 nm, Bit Fusion almost matches the performance of a 250-Watt Titan Xp, which uses 8-bit vector instructions, while Bit Fusion merely consumes 895 milliwatts of power.**

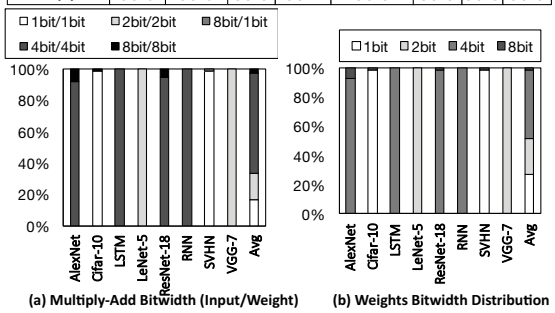
***Keywords*-Bit-Level Composability; Dynamic Composability; Deep Neural Networks; Accelerators; DNN; Convolutional Neural Networks; CNN; Long Short-Term Memory; LSTM; Recurrent Neural Networks; RNN; Quantization; Bit Fusion; Bit Brick**

I. Introduction

Advances in high-performance computer architecture design has been a major driver for the rapid evolution of Deep Neural Networks (DNN). Due to their insatiable demand for compute power, naturally, both the research community [1–28] as well the industry [29–31] have turned to accelerators to accommodate modern DNN computation. However, the algorithmic properties of DNNs have not fully been utilized to push the envelope on their acceleration efficiency and performance.

To that end, we leverage the following three algorithmic properties of DNNs to introduce a novel acceleration architecture, called Bit Fusion. (1) DNNs are mostly a collection of massively parallel multiply-adds. (2) The bitwidth of these operations can be reduced with no loss in accuracy [32–36]. (3) However, to preserve accuracy, the bitwidth varies significantly across DNNs and may even be adjusted for each layer individually. Thus, a fixed-bitwidth accelerator design would either yield limited benefits to accommodate the worst-case bitwidth requirements, or inevitably lead to a degradation in final accuracy. To alleviate these deficiencies, Bit Fusion introduces the concept of runtime bit-level fusion/decomposition as a new dimension in the design of DNN accelerators. We explore this dimension by designing a bit-flexible accelerator, which comprises an array of processing engines that fuse at the bit-level granularity to match the bitwidth of the individual DNN layers.

DNN	AlexNet	Cifar-10	LSTM	LeNet-5	ResNet-18	RNN	SVHN	VGG-7
% Multiply-Add	99.8	99.8	99.9	99.4	99.9	99.9	99.8	99.5



(a) Multiply-Add Bitwidth (Input/Weight) (b) Weights Bitwidth Distribution

Fig. 1: Bitwidth variation across real-world DNNs.

The bit-level flexibility in the architecture enables minimizing the computation and the communication at the finest granularity possible with no loss in accuracy. As such, the following three insights both motivate and guide Bit Fusion.

First, the number of bit-level operations required for the multiply operator is proportional to the product of the operands' bitwidths and scales linearly for the addition operator. Therefore, matching the bitwidth of the multiply-add units to the reduced bitwidth of the DNN layers, almost quadratically reduces the bit-level computations. This strategy will significantly affect the acceleration since the large majority of DNN operations (> 99%) are multiply-adds as shown in the table included in Figure 1. For instance, each single image classification with AlexNet [36] requires a total of 2682 million operations, of which 99.86% (2678 million) are multiply-adds. To this end, the compute units of Bit Fusion can dynamically fuse or decompose to match the bitwidth of each individual multiply-add operand without requiring the operands to be encoded in the same bitwidth.

Second, energy consumption for DNN acceleration is usually dominated by data accesses to on-chip storage and off-chip memory [1, 3, 4]. Therefore, Bit Fusion comes with encoding and memory access logic that stores and retrieves the values in the lowest required bitwidth. This logic reduces the overall number of bits read or written to on-chip and off-chip memory, proportionally reducing the energy dissipation of memory accesses. Furthermore, this strategy increases the effective on-chip storage capacity.

Third, Bit Fusion builds upon the extensive prior work that shows

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00069

DNNs can operate with reduced bitwidth without degradation in classification accuracy [2, 32–35, 37]. This opportunity exists across different classes of real-world DNNs, as shown in Figure 1. One category is Convolutional Neural Networks (CNNs) that usually use convolution and pooling layers followed by a stack of fully-connected layers. AlexNet, Cifar-10, LeNet-5, ResNet-18, SVHN, and VGG-7 in Figure 1 belong to this category. Recurrent Neural Networks (RNN) are another sub-class of DNNs that use recurrent layers including Long Short Term Memory (LSTM) and vanilla RNN layers to extract *temporal* features from time-varying data. The RNN and LSTM benchmark DNNs in Figure 1 represent these categories. Furthermore, as the table in Figure 1 shows, most operations in DNNs ($> 99\%$), regardless of their categories, are multiply-adds. As Figure 1(a) illustrates, on average, 97.3% of multiply-adds require four or fewer bits and even in some DNNs a large fraction of the operations can be done with bitwidth equal to one. More interestingly, the bitwidths vary within and across DNNs to guarantee no loss of accuracy. Such a variation is not limited to the intermediate operands and exists in trained weights as illustrated in Figure 1(b). To exploit this property, a programmable accelerator needs to offer bit-level flexibility at runtime, which leads us to Bit Fusion.

To harvest the aforementioned opportunities, this paper makes the following contributions and realizes a new dimension in the design of DNN accelerators.

1) **Dynamic bit-level fusion and decomposition.** The paper introduces and explores the dimension of bit-level flexible DNN accelerator architectures, Bit Fusion, that dynamically matches bit-level composable processing engines to the varying bitwidths required by DNN layers. By offering this flexibility, Bit Fusion aims to minimize the computation and communication required by a DNN at the bit granularity on a per layer basis.
2) **Microarchitecture design for bit-level composability.** To explore Bit Fusion, we design and implement a DNN accelerator using a novel bit-flexible computation unit, called BitBricks. The accelerator supports both feed-forward (CNN) and recurrent (LSTM and RNN) layers. A 2D array of BitBricks constructs a fusible processing engine that can perform the DNN computation at various bitwidths. The microarchitecture also comes with a storage logic that allows feeding the BitBricks with different bitwidth operands.
3) **Hardware-software abstractions for bit-flexible acceleration.** To enable DNN applications to take advantage of these unique bit-level fusion capabilities, we propose a block-structured instruction set architecture, called Fusion-ISA. To amortize the cost of programmability, Fusion-ISA expresses operations of DNN layers as bit-flexible instruction blocks with iterative semantics.

These three contributions define the novel architecture of Bit Fusion, a possible microarchitecture implementation, and the hardware-software abstractions to offer bit-level flexibility. Other complementary and inspiring works have explored bit serial computation [2, 6] without exploring the fusion dimension. In contrast, Bit Fusion *spatially* fuses a group of BitBricks together, to collectively execute operations at different bitwidths. Using eight real-world feed-forward and recurrent real-world DNNs, we evaluate the benefits of Bit Fusion. We implemented the proposed microarchitecture in Verilog and synthesized in 45 nm technology. Using the synthesis results and cycle accurate simulation, we compare the benefits of Bit Fusion to two state-of-the-art DNN accelerators, Eyeriss [1] and Stripes [2]. The latter is an optimized bit-serial architecture. In the same area, frequency, and technology node, Bit Fusion offers 3.9× speedup and 5.1× energy savings over Eyeriss. Compared to Stripes [2], Bit Fusion provides 2.6× speedup and 3.9× energy reduction at 45 nm node when Bit Fusion area and frequency are set to those of Stripes. Scaling to GPU technology node of 16 nm, Bit Fusion provides a 16× speedup over the Jetson TX2 mobile GPU. Further, Bit Fusion almost matches the performance of a 250-Watt Titan Xp, which uses 8-bit vector instructions, while Bit Fusion merely consumes 895 milliwatts of power.

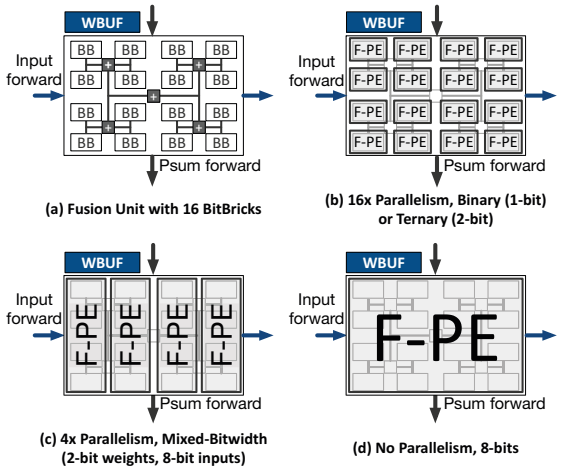


Fig. 2: Dynamic composition of BitBricks (BBs) in a Fusion Unit to construct Fused Processing Engines (Fused-PE), shown as F-PE.

II. Bit Fusion Architecture

To minimize the computation and communication at the finest granularity, Bit Fusion dynamically matches the architecture of the accelerator to the bitwidth required for the DNN, which may vary layer by layer, without any loss in accuracy. As such, Bit Fusion is a collection of bit-level computational elements, called BitBricks, that dynamically compose to *logically* construct Fused Processing Engines (Fused-PE) that execute DNN operations with the required bitwidth. Specifically, Fused-PEs provide bit-level flexibility for multiply-adds, which are the dominant operations across all types of DNNs. Below, we discuss how BitBricks can be dynamically fused together to support a range of bitwidths, yet provide a significant increase in parallelism when operating at lower bitwidths.

A. Bit-Level Flexibility via Dynamic Fusion

As depicted in Figure 2, Bit Fusion arranges the BitBricks in a 2-dimensional *physical* grouping, called Fusion Unit. Each BitBrick in a Fusion Unit can perform individual binary (0, +1) and ternary (-1, 0, +1) multiply-add operations. As Figure 2 shows, the BitBricks *logically* fuse together at run-time to form Fused Processing Engines (Fused-PEs) that match the bitwidths required by the multiply-add operations of a DNN layer. The BitBricks in a Fusion Unit multiply an incoming variable-bitwidth input (input forward) to a variable-bitwidth weight (from WBUF) to generate the product. The Fusion Unit then adds the product to an incoming partial sum to generate an outgoing partial sum (`Psum forward` in Figure 2(a)).

Figures 2(b), 2(c), and 2(d) show three different ways of logically fusing BitBricks to form (b) 16 Fused-PEs that support ternary (binary); (c) four Fused-PEs that support mixed-bitwidths (2-bits for weights and 8-bits for inputs), (d) one Fused-PE that supports 8-bit

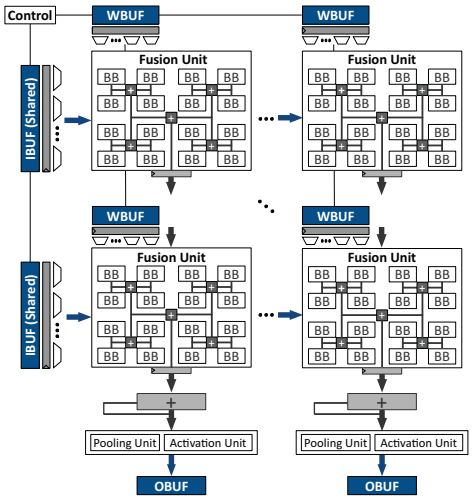


Fig. 3: Bit Fusion systolic architecture comprising a collection of BitBricks (BBs) that can fuse to form Fused-PEs.

operands, respectively. For binary or ternary operations (Figures 2(b)), each Fused-PE contains a single BitBrick, offering the highest parallelism. The Fusion Unit then adds the results from all Fused-PEs and the incoming partial sum to generate a single outgoing partial sum. Figure 2(c) shows four BitBricks fused together in a column to form a Fused-PE that can multiply 2-bit weights with 8-bit inputs. The bitwidths of operands supported by a Fused-PE depend on the spatial arrangement of BitBricks fused together. Alternatively, by varying the spatial arrangement of the four fused BitBricks, the Fused-PE can support 8-bit/2-bit, 4-bit/4-bit, and 2-bit/8-bit configurations for inputs/weights. Finally, up to 16 BitBricks can fuse together to construct a single Fused-PE that can operate on 8-bit operands for the multiply-add operations (Figure 2(d)). The BitBricks fuse together in powers of 2. That is, a single Fusion Unit with 16 BitBricks can offer 1, 2, 4, 8, and 16 Fused-PEs with varying operand bitwidths. Dynamic composability of the Fusion Units at the bit level enables the architecture to expose the maximum possible level of parallelism with the finest granularity that matches the bitwidth of the DNN operands.

B. Accelerator Organization

Two insights guide the architecture design of Bit Fusion. First, DNNs offer high degrees of parallelism and benefit significantly from increasing the number of Fusion Units available within the accelerator's area budget. Therefore, it is essential to minimize the overhead of control in the accelerator by not only maximizing the number of Fusion Units but also minimizing the overhead of dynamically constructing Fused-PEs, thereby integrating the maximum number of BitBricks in the area budget. Second, on-chip SRAM and register-file accesses dominate the energy consumption when accelerating DNNs [1, 3, 4]. Therefore, it is essential to reduce the number of bits exchanged with on-chip and off-chip memory while maximizing data reuse.

Bit Fusion Systolic array. With these insights, we employ a 2-dimensional systolic array of Fusion Units as the architecture for Bit Fusion, as shown in Figure 3. The systolic organization reduces the overhead of control by sharing the control logic across the entire systolic array. More importantly, systolic execution alleviates the need for provisioning control for each Fused-PE as a dataflow architecture would have required. As such, the systolic architectures fit the most

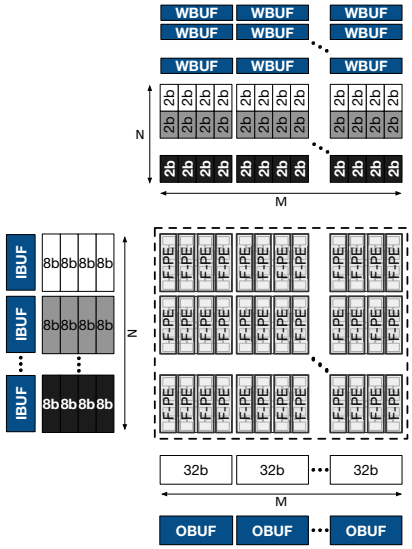


Fig. 4: Bit-Flexible matrix-vector multiplication.

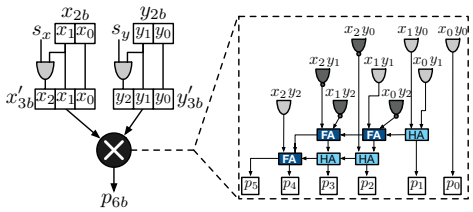


Fig. 5: A single BitBrick. (HA: Half Adder, FA: Full Adder.)

number of BitBricks in a given area budget. Thus, the entire systolic array composed of Fused-PEs acts as a single compute unit that can execute, for example, a single matrix-vector multiplication operation with various bitwidths, which also sets the level of parallelism. In addition, the systolic organization of Fusion Units enforces sharing of input data across columns of the array and accumulates partial results across rows of the array to minimize access to on-chip memory. As depicted in Figure 3, the input buffers (`IBUFs`) only located at the borders and feed the rows simultaneously. Similarly, the output buffers (`OBUFs`) reside on the bottom and collect the flowing results, which is accumulated by each column's accumulator. As shown in Figure 3, each column harbors a pooling and an activation unit before its output buffer. Finally, the systolic organization also eliminates the need for local buffers for input, output, or partial results within Fusion Units. As such, each Fusion Unit is accompanied by only a weight buffer (`WBUF`). Using Fused-PEs as the building blocks, the performance of the systolic array maximally matches the bitwidths, with the highest performance at binary and ternary settings.

Memory organization. Depending on the number of Fused-PEs and their organization, the buffers must supply different number of operands with various bitwidths. As such, we augment the input and the weight buffers with a register that holds a row of data that is gradually fed to the Fused-PEs according to their bitwidth. As illustrated in Figure 3, a series of multiplexers after the register make this data infusion possible. The benefit of this design is avoiding multiple accesses to the data array of the buffer which conserves energy. With this design, at each cycle, the systolic array consumes a vector of inputs and matrix of weights to produce a vector of outputs with the

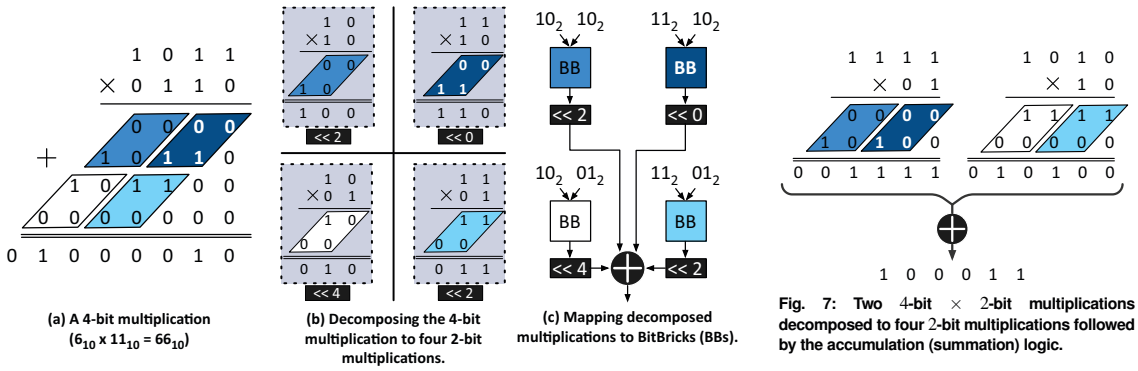


Fig. 6: Using BitBricks to execute 4-bit multiplications.

Fig. 7: Two 4-bit × 2-bit multiplications decomposed to four 2-bit multiplications followed by the accumulation (summation) logic.

fewest accesses to the buffers and the minimal bitwidth possible.

C. Bit Fusion Execution Model

Figure 4 illustrates the Bit Fusion systolic execution in the mixed-bitwidth mode using when an input vector is multiplied to a weight matrix. The input vector has $4 \times N$ 8-bit elements that are being multiplied to a matrix with $4 \times N \times M$ 2-bit elements. As such, the 16-BitBricks in a Fusion Unit logically compose to form four 8×2 Fused-PEs. Both input and weight buffers provide 32 bits per access. The read values are split into 8-bit input values and 2-bit weight values in the output register of each buffer using its accompanying multiplexers as mentioned before. The input values are shared across the Fusion Units of each row and weight values are specific to each Fused-PE. As such, all of the $4 \times N \times M$ Fused-PEs work in parallel while only a single 32-bit value is read from the input and weight buffers. Exploiting the lower bitwidth of weights, Bit Fusion increases the level of parallelism by $4\times$ while reducing the number of accesses to the weight buffer data arrays by the same factor of four. As discussed above, each Fusion Unit adds the results of its Fused-PEs with its incoming partials results and forwards the partial output to the Fusion Unit underneath it. As shown in Figure 4, we support 32-bit bitwidth for the partial and final results to avoid any inaccuracies.

III. Bit Fusion MicroArchitecture

Given the overall organization of Bit Fusion and its bit-flexible systolic execution model, this section delves into the details of BitBricks and Fusion Units. The key insight that enables bit-level dynamic composability in Bit Fusion is the mathematical property that a multiply operation between operands with power-of-2 bitwidths (4-bit, 8-bit, 16-bit, and so on) can be decomposed to 2-bit multiplications. The products from the decomposed multiplications can then be put together by shift-add operations to generate the results of the original multiplication. The bitwidths of the operands dictates the number of decomposed multiplications required and the shift amounts that are applied to the decomposed products before addition. Using this insight, we design BitBrick, the basic compute unit of the Bit Fusion architecture, to support multiply operations for the smallest bitwidth of 2-bits. The 2-bit operands for a BitBrick can be both signed or unsigned. Below, we describe the design of a single BitBrick.

A. BitBrick Microarchitecture

Figure 5 shows the microarchitecture of a single BitBrick. As shown, a BitBrick takes as input two 2-bit operands– x_{2b} and y_{2b} and two corresponding sign-bits–s_x and s_y. The sign-bits s_x and s_y define if the 2-bit operands are signed (between -2 to 1) or unsigned (between 0 to 3). According to the sign-bit, the BitBricks first extend the 2-bit operands x_{2b} or y_{2b} to respectively create 3-bit sign extended operands x'_{3b} or y'_{3b}. Finally, the BitBricks employ a 3-bit signed multiplier (shown with an encircled $\times$ in Figure 5) to generate a 6-bit product p_{6b}. Thus, a BitBrick supports both signed and unsigned numbers as its inputs. The following subsection discusses how Bit Fusion maps multiply-add operations with varying bitwidths to BitBricks.

B. Mapping Variable Bitwidth Operations to BitBricks

To explain how BitBricks compose to multiply operands with variable bitwidths, the discussion below uses a 4-bit multiplication as an example. As mentioned, a multiply operation with power-of-2 bitwidths can be decomposed to 2-bit multiplies that can execute using BitBricks. Figure 6(a) illustrates this mathematical property for a multiplication between 4-bit operands 1011_2 (11_{10}) and 0110_2 (6_{10}) to produce 01000010_2 (66_{10}). The 4-bit multiplication in Figure 6(a) decomposes to four 2-bit multiplications, shown in Figure 6(b). The decomposed multiplications execute using BitBricks to generate decomposed products, as shown in Figure 6(c). The decomposed products require shifting before being put together. For a 4-bit multiplication using BitBricks, the results from the decomposed 2-bit multiplications are left-shifted by 0, 2, 2, and 4, as shown in Figure 6(c).

Dynamic bitwidth flexibility. The bitwidths for the operands dictate how the results from the decomposed multiplications are left-shifted (multiplied with power of 2) before being added together. By adding flexibility in the shifting logic, the BitBricks can support 2-bit and even mixed-bitwidth (4-bit $\times$ 2-bit) multiplications. Figure 7 shows the summation of two 4-bit $\times$ 2-bit multiplications ($15_{10} \times 1_{10} + 10_{10} \times 2_{10} = 35_{10}$). The operation in Figure 7 breaks down to four 2-bit decomposed multiplications that map to four BitBricks. Both the single 4-bit $\times$ 4-bit operation in Figure 6(a) and the two 4-bit $\times$ 2-bit operations in Figure 7 require the same number of BitBricks. Therefore, the performance at 4-bit $\times$ 2-bit is twice that of 4-bit $\times$ 4-bit. The only difference between the operations in Figure 6(a) and Figure 7 is the shift amount required by the decomposed products. Similarly, when operating at 2-bit $\times$ 2-bit, each BitBrick can perform a single multiplication by setting the all the shift amounts to zero.

Supporting arbitrary bitwidths. The discussion so far shows how multiply operations between 4-bit and 2-bit operands map to BitBricks. The same mathematical property can be recursively applied to support higher than 4-bit for the operands. Bit Fusion supports up to 16-bit operands by first recursively breaking down the 16-bit multiplication to 8-bit, 4-bit and then 2-bit multiplications which can execute using

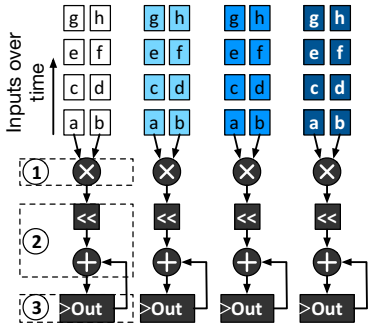


Fig. 8: Temporal design. Operands $a-h$ are 2-bit.

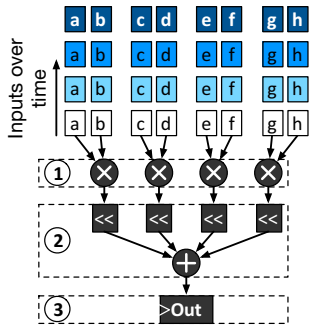


Fig. 9: Spatial fusion. Operands $a-h$ are 2-bit.

Area (μm^2)	BitBricks	Shift-Add	Register	Total Area
Temporal	463	2989	1454	4905
Fusion Unit	369	934	91	1394
Area reduction over Temporal	1.3x	3.2x	16.0x	3.5x

Power (nW)	BitBricks	Shift-Add	Register	Total Power
Temporal	60	550	1103	1712
Fusion Unit	46	424	69	538
Power reduction over Temporal	1.3x	1.3x	16.0x	3.2x

Synthesized using a commercial 45 nm technology

Fig. 10: Area and Power comparison of the Fusion Unit. Temporal design provided as reference.

BitBricks. For a multiplication between $2n$-bit operands A_{2n} and B_{2n}, the recursion can be expressed mathematically as follows.

$$A_{2n} = 2^n \times (A_{2n})_{hi} + 2^0 \times (A_{2n})_{lo}$$
$$B_{2n} = 2^n \times (B_{2n})_{hi} + 2^0 \times (B_{2n})_{lo} \quad (1)$$
$$A_{2n} \times B_{2n} = 2^{2n} \times (A_{2n})_{hi} \times (B_{2n})_{hi} + 2^n \times (A_{2n})_{hi} \times (B_{2n})_{lo} + 2^n \times (A_{2n})_{lo} \times (B_{2n})_{hi} + 2^0 \times (A_{2n})_{lo} \times (B_{2n})_{lo} \quad (2)$$

$(A_{2n})_{hi}$ and $(A_{2n})_{lo}$ refer to the n most significant and n least significant bits of A, respectively. By applying the above equation recursively, Bit Fusion supports up to 16-bit operands. When one of the operand's bitwidths is larger, we use the formulation below.

$$A_{2n} \times B_n = 2^n \times (A_{2n})_{hi} \times B_n + 2^0 \times (A_{2n})_{lo} \times B_n \quad (3)$$

Each level of recursion, from 16-bits to 8-bits, 8-bits to 4-bits, and 4-bits to 2-bits, requires additional shift-add logic. The overhead from the shift-add logic represents the hardware cost of bit-level flexibility. The next subsection details the design of a Fusion Unit that uses BitBricks to execute multiply-adds with variable bitwidths, up to 16-bit.

C. Fusion Unit Micro-Architecture

To enable bit-level composability, Bit Fusion introduces *spatial fusion*, a paradigm that spatially combines the decomposed products generated by multiple BitBricks over a single cycle. Prior works [2, 38], on the other hand, devise a temporal design that use single-bit multiply-add units independently over the span of multiple cycles. The following elaborates on these two approaches. To offer a fair comparison, we assume that even the temporal design uses 2-bit multipliers, a configuration that provides a better area, delay, and power as opposed to a fully bit-serial design.

Temporal design. Figure 8 shows a temporal design that can support variable bitwidths. The variable-bitwidth multiply operation for the temporal design consists of three steps: (1) 2-bit multiplication to generate a partial product, (2) shift operation to multiply with the appropriate power of 2, and (3) accumulation in a register. The temporal design requires 4 cycles to execute a 4-bit $\times$ 4-bit multiplication. The shift operation is simply a 4-input multiplexer (mux). Compared to a fixed 4-bit multiplier, the temporal design uses much smaller multiply units for 2-bit operands, which require significantly less area. However, the number of gates required for the shifter and the accumulator depend on the highest supported bitwidth (16-bit for Bit Fusion). For instance, to support up to 16-bits using a temporal design, the shifter and the accumulator use up around 90% of the area, which limits the benefits provided by this approach. Nevertheless, the temporal design reduces area consumption over a fixed-bitwidth multiplier for the highest required bitwidth.

Spatial fusion. In contrast, our spatial multiplier spatially combines (or fuses) the results from four BitBricks over a single cycle to execute either one 4-bit $\times$ 4-bit multiplication, two 4-bit $\times$ 2-bit multiplications, or four 2-bit $\times$ 2-bit multiplications. Figure 9 illustrates the design of a spatial multiplier that supports up to 4 bits for either of the two operands using BitBricks. Similar to the temporal design, the spatial multiplier requires three steps: (1) multiplication using BitBricks, (2) shift-add using the shift-add tree, and (3) accumulation of results in a register. The spatial multiplier improves upon the temporal design by using a shift-add tree and a single shared accumulator to reduce the number of gates required. Each level of the shift-add tree consists of three shift-units and a four-input adder that represent the multiplication with power of 2 in Equations (2) and (3). Compared to a 4-bit fixed bitwidth multiplier the spatial multiplier requires more area but delivers 4$\times$ higher performance for 2-bit operations. Overall, spatial fusion provides higher $\frac{performance}{area}$ compared to temporal design by packing more BitBricks in the same area.

Fusion Unit using spatio-temporal fusion. As discussed, a Fusion Unit can execute variable-bitwidth multiply-add operations and supports 2-bit to 16-bit operands. Using Equations (2) and (3) recursively, we can realize a Fusion Unit using either the temporal design, spatial fusion, or a combination of both. For a fixed area budget, using spatial fusion with 64 BitBrick would pack the highest number of BitBricks. At the same time, feeding the 64 BitBricks for spatial fusion would require 128-bit wide accesses to the SRAM buffers (IBUF and WBUF in Figure 3) per Fusion Unit. Increasing the width of SRAMs increases the area required by the IBUF and WBUF. Therefore, we make a tradeoff wherein we use spatial fusion to combine 16 BitBricks spatially to realize support up to 8-bit operands, and then combine it with temporal design to support up to 16-bit operands over four cycles. This hybrid approach balances both bit-level flexibility and the corresponding area overhead due to increased SRAM sizes. Figure 10 compares the area and the power requirements for a Fusion Unit with 16 BitBricks that uses the hybrid approach with a temporal design using 16 BitBricks. As shown, for 16 BitBricks, the hybrid Fusion Unit has 3.5$\times$ less area and 3.2$\times$ less power compared to temporal design with the same number of 2-bit multipliers.

Comparison to bit-serial temporal execution. Prior works in Stripes [2], UNPU [39], and Loom [38] devise bit-serial computation as a means to support flexible bitwidths for DNN operations. Of the three, Loom is a fully-temporal architecture, similar to the temporal design discussed above (Figure 8). Stripes and UNPU are hybrid designs that fix the bitwidth of one operand and support variable

TABLE I: Bit Fusion Instruction Set.

OpCode	Operand Specification		Loop Identifier	Immediate
5-bits	6-bits		5-bits	16-bits
setup	op0.bitwidth	op1.bitwidth	X	X
ld-mem / *st-mem*	scratchpad-type	mem.bitwidth	loop-id	num-words
rd-buf / *wr-buf*		X		X
gen-addr		ld/st		stride
compute	fn			X
loop	X	loop-level		num-iterations
block-end	Address of next instruction			

bitwidths for the other. We provide a head-to-head comparison to Stripes in Section V-B4 and provide a qualitative comparison to Loom below. As the results from Figure 10 indicate, for the same throughput, a fully-temporal design, such as the one used in Loom, would consume significantly larger area and power compared to our spatially composable Fusion Unit. Furthermore, a fully-temporal design iterates in the form of a nested loop over the bits the two operands; hence, requiring more number of accesses to the SRAM.

The next section discusses the Bit Fusion-ISA, that exposes the bit-level flexibility of Bit Fusion to software.

IV. Instruction Set Architecture

To leverage the unique bit-level flexibility of Bit Fusion, we need to design a new hardware-software interface that exposes those capabilities in an abstract manner. Furthermore, the abstraction must be flexible to enable a wide range of DNN models so as to exploit bit-level fusion. The following lists the requirements for an ISA that provides this abstraction and enables efficient use of Bit Fusion for various categories of DNNs.

1) **Amortize the cost of bit-level fusion by grouping operations.** The operations in a DNN are organized into groups, called layers, wherein the same mathematical operation repeats a large number of times (often hundreds of thousands). To avoid the overhead of *fine-grained* control over the operations at such a scale, the abstraction needs to amortize the cost of bit-level fusion across blocks of instruction that implement the layers.
2) **Enable a flexible data-path for Bit Fusion.** Both the number of words and the bitwidth of each word that feeds the Fused-PEs varies depending on how the BitBricks are composed as discussed in Section II. Thus, the semantics of instructions for data accesses must vary according to the fusion configuration to enable a flexible data-path.
3) **Provide a concise expression for a wide range of DNN layers**. As research in DNNs is still volatile, it is necessary to devise an ISA that is general enough to express a wide range of DNN operations/layers. Yet, minimizes the von Neumann overhead of instruction handling and require a small footprint.

A. Fusion-ISA for Bit-Flexible Acceleration

Table I summarizes the Bit Fusion-ISA that aims to satisfy these requirements. The rest of this section discusses the instruction formats and provides the insight that drives them.

Block-structured ISA for DNN layers. To leverage the commonalities in the operations of a layer, the Bit Fusion ISA is *block structured*. As such, the fusion configuration of the BitBricks is fixed across each block of instructions that implement a specific layer. In this work, we did not explore within layer bitwidth variations. Nevertheless, the Bit Fusion ISA and this incarnation of its microarchitecture can readily support it by using multiple instruction blocks for an individual layer. The `setup` instruction marks the beginning of an instruction block and configures the Fusion Units and its data delivery logic to the specified bitwidth for the operands. This instruction effectively defines the *logical* fusion of the BitBricks into Fused-PEs for all the instructions in the block. The `block-end` instruction signifies the end of a block and provides the address to the next instruction in the `next-inst` field.

Concise expression of DNN layers. DNNs consist of a large number of simple operations like multiply-accumulate and max, repeated over a large number of neurons (over 2600 million multiply-adds in AlexNet. See Table II). Thus, the von Neumann overhead of instruction fetch and decode can limit performance due to the large number of operations required by a DNN. To minimize the number of instruction fetches/decodes required, we leverage the following insight. Each layer in a DNN is a series of simple mathematical operations over hyper-dimensional arrays. How the operations walk through the array elements and the type of mathematical operation (multiply-add/max) uniquely defines a layer. As such, the ISA provides `loop` instructions that enable a concise way of defining the walks and operations in a DNN layer. Each `loop` instruction has a unique ID in the block. As shown in Table I, the `num-iterations` field in the `loop` instruction defines iteration count. The `compute` instruction specifies the type of operation, while the `gen-addr` instruction dictates how to walk through the elements of the input/output hyper-dimensional arrays. The `stride` field in the `gen-addr` instruction specifies how to walk through the array elements in the `loop`, which is identified by the `loop-id` field. The words after the `setup` instruction define the memory base address for the data that fills the three buffers of input, output, and weights. The `gen-addr` instruction generates the addresses that walk through the memory data and fill the buffers.

$$Address = base + \sum_{id} (loop\_iterator[id] \times stride[id]) \qquad (4)$$

In Equation (4), *id* is the `loop-id` field of all the `gen-addr` instruction in the block and the *loop_iterator* is the current iteration of the corresponding loops and their *stride*s. The fundamental assumption is that multiple `gen-addr` instructions repeated by corresponding `loop` instructions define the complex multi-dimensional walks that expresses various kinds of DNN layers from LSTM to CNN. In the evaluated benchmarks, blocks with 30-86 instructions are enough to cover LSTM, CNN, pooling, and fully connected. These blocks use a combination of `loop`, `compute`, and `gen-addr` instructions to define these DNN layers nested loops. These statistics show that our ISA can concisely express various DNN layers while providing bit-level fusion capabilities. Note that these instructions are fetched and decoded once at the beginning of an instruction block, amortizing the von Neumann overhead over the entire execution of the block.

Managing memory accesses for Fused-PEs. The `ld-mem`/`st-mem` instructions exchange data between the on-chip buffers (`IBUF`, `OBUF`, and `WBUF`) in Figure 3–and the off-chip memory. Similarly, the `rd-buf`/`wr-buf` instructions read/write data from the on-chip buffers specified by the `scratchpad-type` field as shown in Table I. In these four instructions, the size of the operands, which are variable-bitwidth arrays, depends on the number of array elements and their bitwidths. These parameters, which control the logic that feeds the Fused-PEs, are dependent on the bit-level fusion

configuration (number of Fused-PEs in each Fusion Unit) and the type of data (input/weights). To capture this variation in the size of data, the semantics of `rd-buf/wr-buf` and `ld-mem/st-mem` instructions for accessing on-chip and off-chip memory vary according to the fusion configuration of their instruction block, set apriori. In particular, the sizes of memory accesses by `ld-mem/st-mem` instructions depend on both its `num-words` field and the fusion configuration defined by the corresponding `setup` instruction.

Decoupling on-chip and off-chip memory accesses. The data required by DNNs, and subsequently, the number of memory accesses are large. Hence, the latency due to off-chip memory accesses can be a performance bottleneck. To hide the latency of off-chip accesses, the ISA decouples the on-chip memory accesses with off-chip. Furthermore, decoupling the two types of memory accesses allows the accelerator to reuse on-chip data using simple scratchpad buffers, instead of hardware-managed caches.

B. Code Optimizations

As discussed in Section IV, the Fusion-ISA uses simple instructions combined with explicit loop instructions to express neural networks. The use of simpler instructions makes the ISA flexible to express a large range of DNNs. Nonetheless, the flexibility in the ISA enables incorporating layer-specific optimizations to improve the performance and energy gains. For brevity, we use an example fully-connected layer to discuss the code optimizations. Figure 11 shows the matrix-matrix multiplication associated with this example. We perform the following three optimizations as depicted in Figure 12.

Loop ordering. Loop-ordering optimizes the order of the outer loops and memory instructions to further reduce off-chip accesses. Recall that Bit Fusion-ISA uses loop indices to generate memory addresses (Section IV). When the address for a memory instruction does not depend on the index of the previous loop instruction, their order can be exchanged. The optimized code in Figure 12(b) uses `Output-Stationary` for executing the fully-connected layer, to reduce read/write accesses to the output buffer. Changing the order allows Bit Fusion to switch between `Input-Stationary`, `Output-Stationary`, and `Weight-Stationary` to minimize off-chip and on-chip accesses.

Loop tiling. Loop-tiling partitions a loop instruction in the Bit Fusion-ISA into smaller *tiles* such that the data required by a loop operation fits inside the on-chip scratchpads. The smaller tiles are accessed using a single LD/ST instruction and are reused in the inner-loop to reduce off-chip accesses. Compared to the original code in Figure 12(a), the tiled version in Figure 12(b) reduces off-chip accesses for output buffer by a factor of `IC`×, and on-chip accesses for output buffer by a factor of $tile_{ic}$. Note that `IC` is a dimension in the matrix multiplication operation as depicted in Figure 11. Convolution layers typically require six loop instructions, which increases to 12 after tiling optimizations. The overhead of increasing the number of instructions on performance is negligible since the cost of fetch and decode is amortized throughout the execution of the layer.

Layer fusion. As discussed, the Bit Fusion architecture consists of a 2-D systolic array of multipliers, along with a 1-D array of pooling/activation units. When two or more consecutive layers use mutually exclusive on-chip resources, the instructions for the two layers are combined such that the data produced by the first layer is directly fed into the subsequent layer, avoiding costly off-chip accesses. For example, the fully-connected layer in Figure 11 uses the 2-D systolic

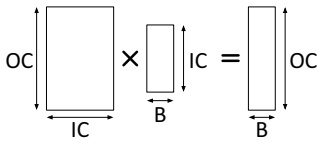


Fig. 11: A single Fully-Connected Layer. The × represents matrix multiplication.

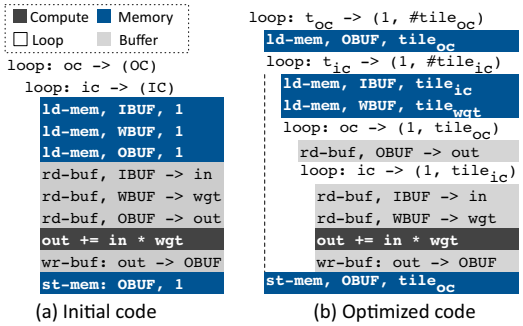


Fig. 12: (a) Code for the Fully-Connected Layer. (b) Optimized code using loop tiling and ordering. `setup` and `gen-addr` instructions omitted for clarity.

array. If the next layer is activation, then we can fuse the layers and create one block of instruction for computing both the layers.

V. Evaluation

A. Methodology

Benchmarks. Table II shows the list of 8 CNN and RNN benchmarks from diverse domains including image classification, object and optical character recognition, and language modeling. The selected DNN benchmarks use a diverse size of input data, which allows us to evaluate the effect of input data size on the Bit Fusion architecture. AlexNet [36, 40], SVHN [35, 41], CIFAR10 [35, 42], LeNet-5 [34, 43], VGG-7 [34, 44], ResNet-18 [36, 45] are popular and widely-used CNN models. Among them, AlexNet and ResNet-18 benchmarks are image classification applications that have different network topologies that use the ImageNet dataset. The SVHN and LeNet-5 benchmarks are optical character recognition applications that recognize the house numbers from the house view photos and handwritten/machine-printed characters, respectively. CIFAR10 and VGG-7 are object recognition applications based on the CIFAR-10 and ImageNet dataset, respectively. The RNN [35] and LSTM [35, 46] are recurrent networks that perform language modeling on the Penn TreeBank dataset [47]. In Table II, the "Multiply-Add Operations" column shows the required number of Multiply-Add operations for each model and the "Model Weights" column shows the size of model parameter. Note that the multiply-add operations and model weights have variable bitwidths as presented in Figure 1.

Reduced bitwidth DNN models. Bit Fusion aims to accelerate the inference of a wide range of DNN models with varying bitwidth requirements, with *no loss in classification accuracy*. The benchmarks, listed in Table II, employ the model topologies proposed in prior work [32, 34–36] that train low bitwidth DNNs and achieve the same accuracy as the 32-bit floating-point models. We did not engineer these quantized DNNs and merely took them from the existing deep learning literature [32, 34–36]. Benchmarks Cifar-10, SVHN, LSTM, and RNN use the quantized models presented in [35]. Benchmarks LeNet-5 and VGG-7 use ternary (+1,0,-1) networks [34]. AlexNet and ResNet-18 use the 4-bit 2× wide models presented in [36] that double the number of channels for

TABLE II: Evaluated CNN/RNN benchmarks.

DNN	Type	Domain	Dataset	Multiply-Add Operations	Model Weights
AlexNet	CNN	Image Classification	ImageNet	2,678 Mops	116.3 Mbytes
Cifar-10	CNN	Object Recognition	CIFAR-10	617 Mops	3.3 MBytes
LSTM	RNN	Language Modeling	Penn TreeBank	13 Mops	6.2 MBytes
LeNet-5	CNN	Optical Character Recognition	MNIST	16 Mops	0.5 MBytes
ResNet-18	CNN	Image Classification	ImageNet	4,269 Mops	13.0 Mbytes
RNN	RNN	Language Modeling	Penn TreeBank	17 Mops	8.0 MBytes
SVHN	CNN	Optical Character Recognition	SVHN	158 Mops	0.8 MBytes
VGG-7	CNN	Object Recognition	CIFAR-10	317 Mops	2.7 MBytes

TABLE III: Evaluated ASIC and GPU platforms. *Stripes entries per-tile.

	ASIC			GPU	
Chip	Eyeriss	Stripes*	Chip	Titan X	Tegra X2
Cores (1.1 mm^2)	168 PEs	4096 SIPs	Cores	3,584	256
On-chip Memory	181.5 KB	2 MB eDRAM 16 KB SRAM	Memory	12 GB	8 GB
Chip Area (mm^2)	5.87	3.62	ChipArea (mm^2)	471	-
			TDP	250 W	7.5 W
Frequency	500 MHz	980 MHz	Frequency	1,531 MHz	875 MHz
Technology	45 nm	45 nm	Technology	16 nm	16 nm

convolution and fully-connected layers. We use the regular AlexNet and ResNet-18 models for Eyeriss and the GPU baselines, and use their 2× wide quantized models for Bit Fusion and Stripes.

Accelerator development and synthesis. We use RTL-Verilog to implement the configuration of the Bit Fusion architecture and verify the design through extensive RTL-simulations. We synthesize Bit Fusion at 45nm technology node using Synopsys Design Compiler (L-2016.03-SP5) and a commercial standard-cell library. Design Compiler provides the chip area, achievable frequency, and dynamic/static power, which we use to estimate the performance and energy-efficiency of the Bit Fusion accelerator.

Simulation infrastructure for Bit Fusion. We compile each DNN benchmark to the instructions of the Fusion-ISA (Section IV). We develop a cycle-accurate simulator that takes the Fusion-ISA instructions for the given DNN and simulates the execution to calculate the cycle counts as well as the number of accesses to on-chip buffers (IBUF, OBUF, and WBUF in Figure 3) and off-chip memory. We verify the cycle counts of the simulator against our Verilog implementation of the Bit Fusion architecture. Using the frequency defined in Table III and the cycle counts, the simulator measures the execution time of the Bit Fusion architecture. To evaluate the energy efficiency, we model the energy consumption for on-chip buffers for the Bit Fusion accelerator using the results from CACTI-P [48].

Comparison with Eyeriss. To measure the performance and energy dissipation of our comparison point, Eyeriss, we use their open-source simulation infrastructure [4]. The resulting area and energy metrics are shown in Table III. As mentioned, we use the same area budgets as Eyeriss, which is 1.1 mm^2 for compute units and 5.87 mm^2 for chip to synthesize Bit Fusion, shown in Table III. We use a total 112 KB SRAM for on-chip buffers (IBUF, OBUF, and WBUF in Figure 3). Eyeriss operates on the 16-bit operands and Bit Fusion supports flexible bitwidths from 2, 4, 8, to 16 bits.

Comparison with Stripes. The authors of Stripes graciously shared their simulator [2]. Their power estimation tools were in 65 nm node, which we scaled to 45 nm. Stripes operates on 16-bit inputs and variable-bitwidth weights (1 through 16), using Serial Inner-Product units (SIPs). Stripes is organized into 16 tiles each of which has 4096 SIPs. For a fair comparison, we replace the 4096 SIPs in each tile of Stripes with our proposed Bit Fusion systolic array with 512 Fusion Units, each with 16 BitBricks to match the same budget of 1.1mm^2 for compute, which is the area after scaling to 45 nm and use the same total on-chip memory.

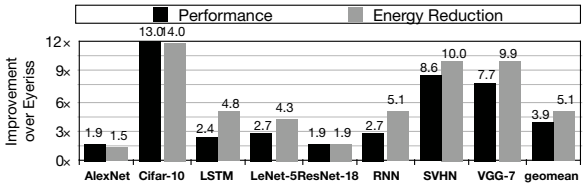


Fig. 13: Bit Fusion performance and energy improvements over Eyeriss.

Comparison with GPUs. We use two GPUs (Titan Xp and Tegra X2) based on Nvidia's Pascal architecture to compare with Bit Fusion. Table III shows the details of the two GPUs. We use Nvidia's custom TensorRT 4.0 [49] library compiled with the latest CUDA 9.0 and cuDNN 7.1 which support 8-bit quantized calculations, the smallest possible in the architecture. Across GPU platforms, we use 1,000 warm-up batches, followed by 10,000 batches to measure performance and use the average. For a head-to-head comparison, we conservatively scale Bit Fusion to 16 nm technology node assuming a 0.86× voltage scaling and 0.42× capacitance scaling according to the methodology presented in [50]. However, we assume the same frequency of 500 MHz as Eyeriss and do not increase the Bit Fusion frequency. The scaled Bit Fusion architecture has 4096 Fusion Units with 896 KB SRAM and has a total chip area of 5.93 mm^2 and consumes 895 milliwatts of power. As a point of reference, Titan Xp in the same 16 nm node, has a chip area of 471 mm^2 and has a TDP of 250 Watts, as summarized in Table III.

B. Experimental Results

1) Comparison to Eyeriss:

Performance and energy improvement. To evaluate the performance and efficiency benefits from the Bit Fusion architecture, we compare with a state-of-the-art accelerator Eyeriss [1] that proposes an optimized dataflow architecture for DNNs. We match the same area budget of 1.1mm^2 for computational logic across both architectures: systolic array in Bit Fusion and PEs in Eyeriss, and match the total SRAM capacity. We scale the area and energy consumption of the PEs, register-files, on-chip network, and DRAM in Eyeriss to 45nm technology according to the methodology proposed in [4]. For a fair comparison between the two architectures, we use the same frequency of 500MHz reported in the paper [4] for both Eyeriss and Bit Fusion. Figure 13 presents the performance and energy benefits of Bit Fusion in comparison with Eyeriss. On average, Bit Fusion delivers 3.9× speedup since the Bit Fusion architecture can perform more DNN operations with lower bitwidth in a given area compared to Eyeriss. Depending on the types of DNN operations and the required bitwidths, the benchmarks see different performance gains. The CNN benchmarks (AlexNet, SVHN, Cifar-10, LeNet-5, VGG-7, and ResNet-18) see higher performance gains than the recurrent networks (RNN and LSTM) since the convolution operations are more amenable for data reuse in systolic architecture of Bit Fusion. Cifar-10 sees the highest benefits of 13× speedup since most of its operations can be computed with the smallest bitwidth (1-bit input and 1-bit weight) and its operations provide a large degree of parallelism that can exploit the increased number of Fused-PEs. In contrast, ResNet-18 and AlexNet achieve the lowest speedup of 1.9×, because these two benchmarks use twice the number of channels (2× wide) for convolution and fully-connected layers [36] for quantized execution on Bit Fusion. We use the original AlexNet and ResNet-18 models on Eyeriss, which effectively requires 4× less multiply-add operations. Overall, using variable bitwidth improves performance

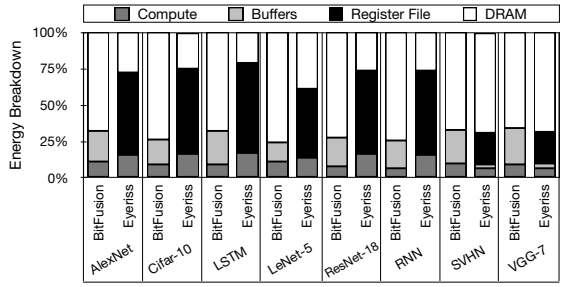


Fig. 14: Energy breakdown of Bit Fusion and Eyeriss.

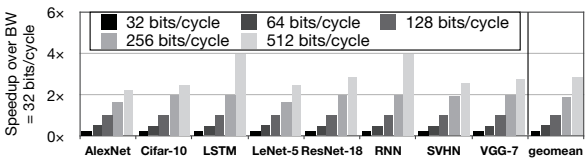


Fig. 15: Bit Fusion performance as the bandwidth changes.

and energy efficiency, since it increases compute capacity and reduces active hardware components. Figure 13 also shows the energy reduction. The average improvement is 5.1×, with the largest of 14× from Cifar-10 and the smallest of 1.5× from AlexNet. The significant energy reduction attributes to both Fusion Unit organizations and memory access reductions, which we discuss below in more detail.

Energy breakdown. To understand the sources of the energy reduction, we break down the energy consumptions for each hardware component (compute units, on-chip SRAM buffers, register file, and off-chip DRAM memory). Figure 14 shows the per-component energy dissipation for Bit Fusion and Eyeriss. This figure should be considered with the energy reduction results from Figure 13. Both accelerators consume more than 80% of energy for on-chip and off-chip memory accesses. The bit-level flexibility for memory accesses in Bit Fusion significantly reduces energy consumption for both on-chip buffers (IBUF, OBUF, and WBUF in Figure 3) and off-chip DRAM. Furthermore, with bit-level flexibility, our buffers can hold more data at lower-bitwidths, effectively giving Bit Fusion more on-chip storage capacity, which leads to fewer off-chip memory accesses. Eyeriss employs local register files within each PE, which constitutes a significant portion of the energy consumption. Bit Fusion's systolic architecture avoids the need for register files and enforces explicit data sharing for inputs and partial results, as shown in Figure 3. Therefore, Bit Fusion saves on Register File energy, but requires more SRAM accesses. The combined effect of bit-level flexibility and the systolic organization of BitBricks in the Bit Fusion architecture provides an average energy savings of 5.1×. Off-chip DRAM accesses, however, are still a significant portion of Bit Fusion's energy consumption and its share grows due to the significant reduction of compute and on-chip storage energy.

2) Sensitivity Study:

Sensitivity to memory bandwidth. Depending on the DNN topology, the impact of off-chip bandwidth on performance varies. To understand the correlation between bandwidth and performance, we perform a sensitivity study for bandwidth. Figure 15 shows the performance improvements with Bit Fusion as we change the bandwidth from 0.25× to 4× of the default value. The baseline in this study the Bit Fusion with the default bandwidth of 128 bits per cycle. On average, when we scale the bandwidth up to 4×, Bit Fusion provides 1.6× speedup compared to the default setting, while

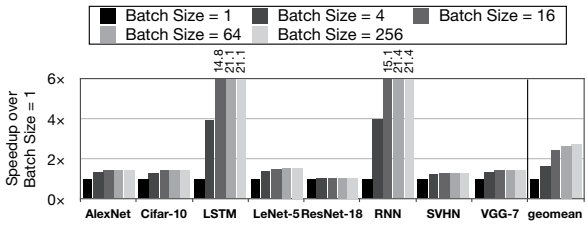


Fig. 16: Bit Fusion performance as the batch size increases.

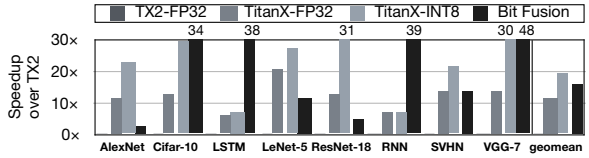


Fig. 17: Performance comparison to GPUs.

with 0.25× bandwidth, the performance degrades 60%. Since CNN benchmarks see more opportunities for data reuse, they have less sensitivity to the bandwidth compared to the RNN benchmarks. The two RNN benchmarks, LSTM and RNN, provide almost linearly-scaling speedup as they are bottlenecked by the bandwidth.

Sensitivity to batch size. Batching amortizes the cost of weight reads by sharing weights across a batch of inputs. Figure 16 shows how performance changes as we increase batch size from 1 through 256 with the batch size 1 as the baseline (no batching). Our default batch size is 16. On average, Bit Fusion with the batch size of 256 engenders 2.7× speedup with the highest speedup of 21.4× from RNN. Since batching is effective when the bandwidth is limited and the performance is bandwidth-bound, the trends are similar to the bandwidth sensitivity results presented in Figure 15. However, there is a marginal gain across all the benchmarks when the batch is increased from 64 to 256, since beyond a batch size of 64, the bandwidth is sufficient to keep all the Fusion Units occupied.

3) Comparison to GPUs:

Performance comparison to GPUs. GPUs are the most widely-used general-purpose processors for DNNs. We compare the performance of Bit Fusion accelerators with two GPUs: (1) Tegra X2 (TX2), and (2) Titan X based on the Pascal architecture (Titan Xp), details of which are presented in Table III. As mentioned in the methodology section V-A, we scale Bit Fusion to match the 16 nm technology node of the GPUs, and use a total of 4096 Fusion Units. Figure 17 shows the speedup of TitanX and Bit Fusion using the TX2 as the baseline. TX2 does not support 8-bit mode natively. Due to this lack of support, empirical results show slow down when the 8-bit instruction are used in TX2. As Figure 17 depicts, TitanX in single-precision floating point (FP32), is, on average, 12× faster than TX2. The speedup grows to 19× when 8-bit mode is used. While GPUs can benefit from using as low as 8-bits, Bit Fusion can extract performance benefits for as low as 2-bit operations. Using bit-level composability, Bit Fusion provides a 16× speedup over TX2. The VGG-7 benchmark sees the maximum gains of 30× and 48× performance from Titan Xp and Bit Fusion, respectively. The high degrees of parallelism in VGG-7 enables both Titan Xp and Bit Fusion to utilize all the available on-chip compute resources. Bit Fusion, while consuming 895 milliwatts of power, is only 16% slower than the 250-Watt Titan Xp that uses 8-bit computations, almost matching its performance.

4) Comparison to Stripes:

Performance compared to Stripes. Figure 18 presents the

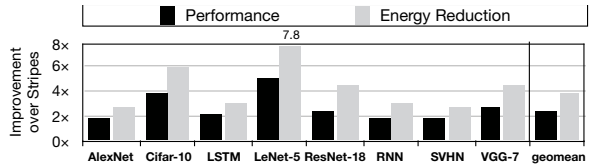


Fig. 18: Bit Fusion performance and energy improvements over Stripes.

performance and energy benefits of Bit Fusion in comparison with Stripes. On average, Bit Fusion provides 2.6× speedup over Stripes. Stripes uses bit-serial computations to support variable bitwidths just for DNN weights. As opposed to Stripes, the Bit Fusion architecture offers dynamically composable BitBricks to support flexible bitwidths for both inputs and weights in DNNs. Bit Fusion achieves the highest speedup of 5.2× and lowest speedup of 1.8× over Stripes for benchmarks LeNet-5 and AlexNet, respectively. ResNet-18 which is the most recent and the biggest of the benchmarks sees 2.6× performance benefits as it can use low bitwidth on both operands. AlexNet uses 8-bit inputs/weights for the first convolution layer and the last fully-connected layer. The two 8-bit layers limit the benefits of Bit Fusion over Stripes. Benchmark LeNet-5, on the other hand, uses low bitwidths for both inputs and weights, resulting in the highest performance benefits with Bit Fusion.

Energy reduction compared to Stripes. Figure 18 also depicts the improvement in energy when Bit Fusion is compared to Stripes. As mentioned, Bit Fusion benefits from reduction in both computation and memory access at lower bitwidths for *both* inputs and weights. On average, Bit Fusion reduces energy consumption by 3.9× over Stripes. LeNet-5 sees the highest energy reduction of 7.8×, while benchmark AlexNet sees the least energy reduction of 2.7× over Stripes. For ResNet-18, the energy is reduced by a factor of 4×.

Bit Fusion offers a fundamentally different approach from Stripes and explores the dimension of bit-level dynamic composabililty, which significantly improves performance and energy.

VI. Related Work

A growing body of related works develop DNN accelerators. Bit Fusion fundamentally differs from prior work as it introduces and explores a new dimension of bit-level composable architectures that can dynamically match the bitwidth required by DNN operations. Bit Fusion aims to minimize both computations and communications in the finest granularity possible without compromising on the DNN accuracy. Below, we discuss the most related work.

Precision flexibility in DNNs. Stripes [2] and Tartan [6] use bit-serial compute units to provide precision flexibility for inputs at the cost of additional area overhead. Both works provide performance and efficiency benefits that are proportional to the precision reduction for inputs. We directly compare the benefits of Bit Fusion to Stripes in Section V. UNPU [39] fabricates a bit-serial DNN accelerator at 65 nm, similar to Stripes [2]. Loom [38] uses bit-serial computation for precision flexibility. DeepRecon [51] skips stages of a fully-pipelined floating-point-multiplier to perform either one 16-bit, two 12-bit, or four 8-bit multiplications. In contrast, the Fusion Units are spatial designs that use combinational logic to dynamically compose and decompose 2-bit multipliers (BitBricks) to construct variable bitwidth multiply-add units. Moons et al. propose aggressive voltage scaling techniques at low precision for increased energy efficiency at constant throughput by turning off parts of the multiplier [37, 52]. As such, they do not offer fusion capabilities. TPU [30] proposes a systolic architecture for DNNs and supports 8-bit and 16-bit precision. This work, on the other hand, proposes an architecture that dynamically composes low-bitwidth compute units (BitBricks) to match the bitwidth requirements of DNN layers.

Binary DNN accelerators. Several inspiring works have explored ASIC and FPGA accelerators optimized for Binary DNNs. FINN [53] uses FPGAs for accelerating Binary DNNs, while YodaNN [54] and BRein [55] propose an ASIC accelerator for binary DNNs. Kim, et al. [56] decompose the convolution weights for binary CNNs to improve performance and energy efficiency. The above works focus solely on binary DNNs to achieve high performance at the cost of classification accuracy. Bit Fusion, on the other hand, flexibly matches the bitwidths of DNN operations for performance/energy benefits without losing accuracy.

Sparse Accelerators for DNNs. EIE [5], Cambricon-X [15], Cnvlutin [13], and SCNN [57] explore the sparsity in the DNN layers and use zero-skipping to provide performance and energy-efficiency benefits. Orthogonal to the works above, Bit Fusion explores the dimension of bit-flexible accelerators for DNNs.

Other ASIC accelerators for DNNs. DaDianNao [7] uses eDRAM to eliminate off-chip accesses and provide high performance and efficiency for DNNs. PuDianNao [9] is an accelerator designed for machine learning, but does not support CNNs. Minerva [12] proposes operation pruning and data quantization techniques to reduce power consumption for ASIC acceleration. Eyeriss [1, 3] presents an optimized row-stationary dataflow for DNNs to improve efficiency. Tetris [4] and Neurocube [11] propose 3-D stacked DNN accelerators to provide high bandwidth for DNN operations. ISAAC [26], PipeLayer [28], and Prime [27] use resistive RAM (ReRAM) for accelerating DNNs. Ganax [58] uses a SIMD-MIMD architecture to support DNNs and generative models. Snapea [59] employs early termination to skip computations.

Instruction Sets for DNNs. Cambricon [14] provides an ISA to express the different computations in a DNN using vector and matrix operations without significant loss in efficiency over DaDianNao. DnnWeaver [22] proposes a coarse grained ISA to express layers of DNNs, which are first translated to micro-codes for FPGA acceleration. Unlike prior work, the Fusion-ISA proposed in the work is designed to enable bit-level flexibility for accelerating DNNs. Further, the Fusion-ISA uses `loop` instructions with iterative semantics to significantly reduce instruction footprint.

Code optimization techniques. Alwani, et. al [60] propose layer-fusion, that combines multiple convolutional layers to save off-chip accesses for FPGA acceleration of CNNs. Escher [61] proposes a CNN FPGA accelerator using flexible buffering that balances the off-chip accesses for inputs and weights in CNNs. The above works have inspired the code-optimizations explored in this paper, however, the key contribution of this work is a bit-level flexible DNN accelerator.

Software techniques for Binary/XNOR DNNs. QNN [35] shows that efficient GPU kernels for XNOR-based binary DNNs can provide up to 3.4× improvement in performance. XNOR-Net [62] shows that specialized libraries for Binary/XNOR-nets can achieve 58× performance on CPUs. In contrast, Bit Fusion is an ASIC accelerator architecture that supports a wide range of bitwidths (binary to 16-bits) for DNNs with no accuracy loss.

Core Fusion and CLPs. Core Fusion [63] and CLPs [64] are dynamically configurable chip multiprocessors that a group of

independent processors can fuse and form a more capable CPU. In contrast to these inspiring works, Bit Fusion performs the composition in the bit level rather than at the level of full-fledged cores.

VII. Conclusion

Deep neural networks use abundant computation, but can withstand very low bitwidth operations without any loss in accuracy. Leveraging this property of DNNs, we develop Bit Fusion, a bit-level dynamically composable architecture, for their efficient acceleration. The architecture comes with an ISA that enables the software to utilize this bit-level fusion capability to maximize the parallelism in computations and minimize the data transfer in the finest granularity possible. We evaluate the benefits of Bit Fusion by synthesizing the Verilog implementation of the proposed microarchitecture in 45 nm technology node and using cycle accurate simulations with eight real-world DNNs that require different bitwidths in their layers. Bit Fusion achieves significant speedup and energy benefits compared to state-of-the-art accelerators.

VIII. Acknowledgments

We thank Amir Yazdanbaksh, Divya Mahajan, Jacob Sacks, and Payal Preet Bagga for insightful discussions and comments. This work was in part supported by NSF awards CNS#1703812, ECCS#1609823, Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) award #FA9550-17-1-0274, and gifts from Google, Microsoft, Xilinx, and Qualcomm.

References

[1] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA*, 2016.

[2] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *MICRO*, 2016.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *JSSC*, 2017.

[4] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *ASPLOS*, 2017.

[5] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.

[6] A. Delmas, S. Sharify, P. Judd, and A. Moshovos, "Tartan: Accelerating fully-connected and convolutional layers in deep learning networks by exploiting numerical precision variability," *arXiv*, 2017.

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[8] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, 2014.

[9] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ASPLOS*, 2015.

[10] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: shifting vision processing closer to the sensor," in *ISCA*, 2015.

[11] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *ISCA*, 2016.

[12] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016.

[13] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: ineffectual-neuron-free deep neural network computing," in *ISCA*, 2016.

[14] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *ISCA*, 2016.

[15] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *MICRO*, 2016.

[16] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPRW*, 2014.

[17] J. Sim, J. S. Park, M. Kim, D. Bae, Y. Choi, and L. S. Kim, "14.6 a 1.42tops/w deep convolutional neural network recognition processor for intelligent ioe systems," in *ISSCC*, 2016.

[18] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *DATE*, 2015.

[19] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *DAC*, 2016.

[20] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *DAC*, 2016.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

[22] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Misra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *MICRO*, 2016.

[23] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *MICRO*, 2016.

[24] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *FPGA*, 2016.

[25] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016.

[26] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.

[27] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.

[28] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*, 2017.

[29] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengil, M. Liu, D. Lo, S. Alkalay,

M. Haselman, C. Boehn, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, T. Juhasz, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, S. Reinhardt, A. Sapek, R. Seera, B. Sridharan, L. Woods, P. Yi-Xiao, R. Zhao, and D. Burger, "Accelerating persistent neural networks at datacenter scale," in *HotChips*, 2017.

[30] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.

[31] "Apple a11-bionic." https://en.wikipedia.org/wiki/Apple_A11.

[32] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv*, 2016.

[33] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv*, 2016.

[34] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv*, 2016.

[35] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv*, 2016.

[36] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: wide reduced-precision networks," *arXiv*, 2017.

[37] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling," in *DATE*, 2017.

[38] S. Sharify, A. D. Lascorz, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," *arXiv*, 2017.

[39] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *ISSCC*, 2018.

[40] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv*, 2014.

[41] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

[42] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[47] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, 1993.

[48] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques," in *ICCAD*, 2011.

[49] "Nvidia tensor rt 4.0." https://developer.nvidia.com/tensorrt.

[50] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[51] T. Rzayev, S. Moradi, D. H. Albonesi, and R. Manohar, "Deeprecon: Dynamically reconfigurable architecture for accelerating deep neural networks," *IJCNN*, 2017.

[52] B. Moons and M. Verhelst, "A 0.3–2.6 tops/w precision-scalable processor for real-time large-scale convnets," in *VLSI-Circuits*, 2016.

[53] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2017.

[54] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," *arXiv*, 2016.

[55] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, T. Kuroda, *et al.*, "Brein memory: A 13-layer 4.2 k neuron/0.8 m synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos," in *VLSI*, 2017.

[56] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "A kernel decomposition architecture for binary-weight convolutional neural networks," in *DAC*, 2017.

[57] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," in *ISCA*, 2017.

[58] A. Yazdanbakhsh, H. Falahati, P. J. Wolfe, K. Samadi, H. Esmaeilzadeh, and N. S. Kim, "GANAX: A Unified SIMD-MIMD Acceleration for Generative Adversarial Network," in *ISCA*, 2018.

[59] V. Aklaghi, A. Yazdanbakhsh, K. Samadi, H. Esmaeilzadeh, and R. K. Gupta, "Snapea: Predictive early activation for reducing computation in deep convolutional neural networks," in *ISCA*, 2018.

[60] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerator," in *MICRO*, 2016.

[61] Y. Shen, M. Ferdman, and P. Milder, "Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer," in *FCCM*, 2017.

[62] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *arXiv*, 2016.

[63] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "Core fusion: accommodating software diversity in chip multiprocessors," in *ISCA*, 2007.

[64] C. Kim, S. Sethumadhavan, M. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler, "Composable lightweight processors," in *MICRO*, 2007.