BitBlade: Area and Energy-Efficient Precision-Scalable Neural Network Accelerator with Bitwise Summation

Sungju Ryu, Hyungjun Kim, Wooseok Yi, and Jae-Joon Kim
Pohang University of Science and Technology, Pohang, Korea
{sungju.ryu,hyungjun.kim,wooseok.yi,jaejoon}@postech.ac.kr

ABSTRACT

Deep Neural Networks (DNNs) have various performance requirements and power constraints depending on applications. To maximize the energy-efficiency of hardware accelerators for different applications, the accelerators need to support various bit-width configurations. When designing bit-reconfigurable accelerators, each PE must have variable shift-addition logic, which takes a large amount of area and power. This paper introduces an area and energy efficient precision-scalable neural network accelerator (*BitBlade*), which reduces the control overhead for variable shift-addition using bitwise summation method. The proposed *BitBlade*, when synthesized in a 28nm CMOS technology, showed reduction in area by 41% and in energy by 36-46% compared to the state-of-the-art precision-scalable architecture [14].

KEYWORDS

Neural network, hardware accelerator, precision scaling, bitwise summation

1 INTRODUCTION

Deep neural networks (DNNs) have been showing impressive performance on a variety of machine learning applications including classification, speech recognition, and language translation tasks. DNNs require a large number of computations, mostly consisting of multiply-accumulate (MAC) operations, and hence variety of custom hardware accelerators dedicated for computing MAC operations in a fast and energy-efficient manner have been proposed. However, even with these dedicated hardware accelerators, it still takes significant energy to perform such a large number of the MAC operations.

Many approaches were studied to reduce high computational cost of DNNs. Some of the well-known methods include reducing bit widths of weights and activations [5, 11], pruning some weights which are relatively less important [1–3], and performing approximate computing [15].

Meanwhile, DNNs usually have a trade-off relationship between power consumption and accuracy. For example, as bit widths of activations and weights are reduced, accuracy loss increases while energy-efficiency improves. Moreover, accuracy requirement and power constraint vary depending on applications ranging from small devices such as smartphone to large devices such as automobile. Considering the characteristics of DNNs and their applications, balancing the accuracy requirement and bit widths is an attractive approach to optimize the computational cost.

Recently, precision-scalable hardware accelerators have been introduced for a variety of neural networks enabling the optimization of power consumption and accuracy [7, 8, 10, 13, 14]. However, logic to support the bit-reconfigurability tends to cause large area and power overhead in these accelerators [14].

Based on the observation described above, this paper introduces *BitBlade*, area and energy-efficient precision-scalable neural network hardware accelerator. *BitBlade* minimized the logic related to supporting various bit widths through bitwise summation method.

The remainder of this paper is organized as follows. Section 2 reviews previous bit-width scalable neural network accelerators. In Section 3, *BitBlade* is introduced and the characteristics of *BitBlade* are compared with those of state-of-the-art precision-scalable architecture [14]. Section 4 evaluates *BitBlade*, and we finally conclude this paper in Section 5.

2 PREVIOUS WORK

Stripes [7] supports variable bit-width for activations only, and variable precision configurations are supported for weights only in *UNPU* [8]. Such constrained conditions, having bit-width flexibility only for activations or weights, lead to limitation to choose optimal bit widths. Selective choice of different bit-precision for both activation values and weights shows better energy-efficiency compared to the efficiency in the case of having flexibility for only one of them [13, 14].

Loom [13] adopts bit-serial multiplication to achieve the flexibility of bit widths. However, transpose computation, which converts input features to bit streams by packing bits with the same bit position into a word, must be performed before the bit-serial multiplication. The computation for transposing input features accounts for a significant amount of overhead in area and power consumption. To mitigate the overhead, *Loom* transposes output features instead of the input features, and then sends the transposed output features to inputs of following layer. It, however, still requires pre-processing of input features for the first layer before acceleration through *Loom* because *Loom* does not support transpose computation of input features.

ENVISION [10] proposes dynamic-voltage-accuracy-frequency-scalable (DVAFS) multiplier. DVAFS multiplier turns on or off sub-multipliers depending on precision configurations. A limitation in the scheme is that the utilization rate of the sub-multipliers is linearly decreased with the reduction in bit widths of activations and weights. For example, all the sub-multipliers are turned on in

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317784

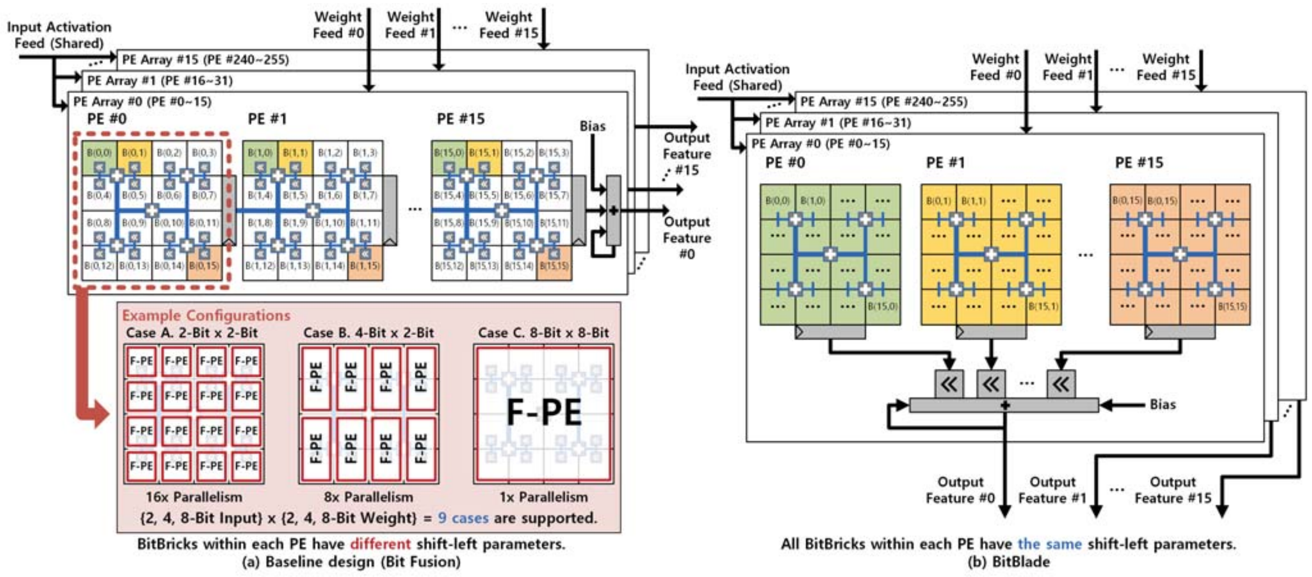


Figure 1: Comparison of architectures between (a) *Bit Fusion* (Baseline) [14] and (b) *BitBlade*. B(x,y) indicates a BitBrick.

the case of 16-bit × 16-bit precision configuration. However, only 25% of the sub-multipliers are turned on in the case of 4-bit × 4-bit precision configuration.

Bit Fusion [14] performs multiplication with variable bit-width by partitioning it into several 2-bit × 2-bit multiplications and then by performing shift-addition operations. By doing so, it can always fully utilize all the sub-multipliers unlike *ENVISION*. However, the shift-addition logic for the reconfigurable bit-width occupies large area (67%) and consumes a lot of power (79%) as shown in Table 1. Hence, there is a pressing need for reducing the overhead for variable shift-addition.

As our proposed architecture is based on *Bit Fusion* [14], let us first describe the structure of the *Bit Fusion* more in detail. Fig. 1a shows processing element (PE) array in *Bit Fusion*. Each PE in *Bit Fusion* consists of 16 BitBricks. A BitBrick is an 2-bit × 2-bit multiplier, and supports both signed and unsigned multiplications. After the multiplications are performed on the multiple BitBricks, shift-left operations are executed subsequently. The shifted multiplication results are added through the adder tree, and becomes partial sum. Partial sum from each PE is stored in the output buffer, and then sent to the following PE in the next clock cycle. Incoming partial sum from the PE in the previous stage is added to the partial sum calculated from the PE in the current stage, and then sent to the PE in the next stage in the following cycle. Each PE array, consisting of 16 PEs in the example, uses its own weights while all the PE Arrays share input activation values. In the convolutional layers, input activations are reused by multiple filters. Similarly, input activations are reused by multiple weights in the fully connected layers. Such a PE array organization in the *Bit Fusion* reduces the number of access to on-chip memory for loading the input activations to PEs.

PEs in *Bit Fusion* can calculate 1 to 16 2-bit × 2-bit multiplications in parallel depending on the bit widths of activations and weights. The set of BitBricks involved in a multiplication is called fused-PE (F-PE) (Fig. 1a). For example, in the case of both activations and weights having 2-bit precision, each F-PE consists of a BitBrick. A PE has 16 F-PEs, and 16 2-bit × 2-bit multiplications are performed in parallel within a PE. In addition, in the case of activations having 4-bit precision and weights having 2-bit precision, each F-PE consists of 2 BitBricks. A PE has 8 F-PEs, and 8 4-bit × 2-bit multiplications are performed in parallel within a PE. Finally, in the case of both activations and weights having 8-bit precision, each F-PE consists of 16 BitBricks. A PE has a F-PE, and an 8-bit × 8-bit multiplication is performed only in a PE.

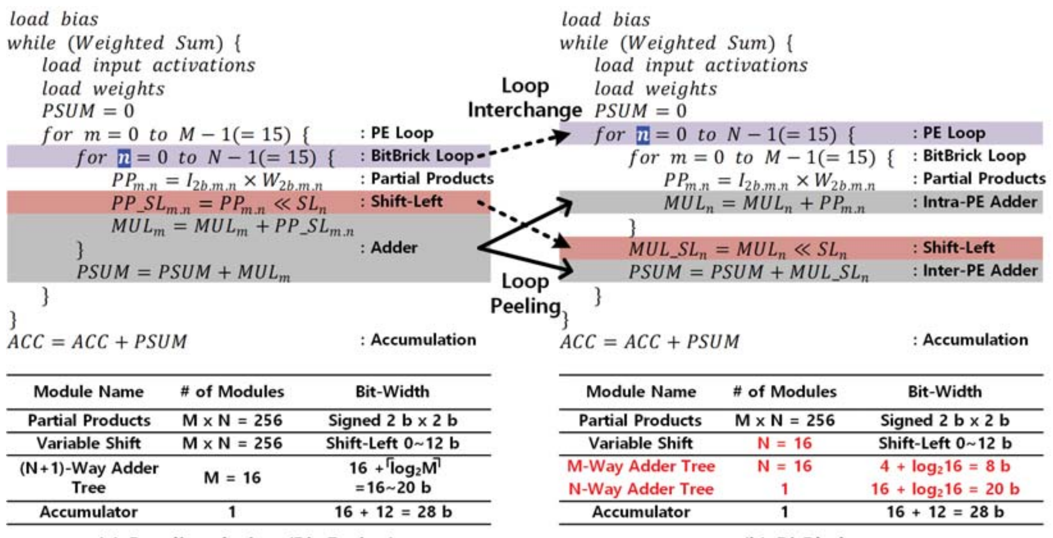
Regardless of the bit widths, the performance can be maximized considering that *Bit Fusion* architecture supports fusion capabilities which utilizes all multipliers within PEs. However, the bit-width reconfigurability leads to a significant amount of area and power overhead as each BitBrick needs to have a shift logic (Fig. 1a).

3 BITBLADE ARCHITECTURE

In this section, we introduce a precision-scalable architecture, *BitBlade*, which has much smaller overhead for shift-add logic compared to *Bit Fusion* (Fig. 1b) [14]. *BitBlade* reduces the shift-add logic using bitwise summation method. In the following subsections, we first exploit the loop nest optimization method to minimize shift operations for computation. And then, we introduce the bitwise summation method based on the loop nest optimization and apply it to *BitBlade* architecture. While each PE in *Bit Fusion* must have 16 variable shift logic, *BitBlade* requires only one variable shift logic for each PE.

3.1 Loop nest optimization

Loop nest optimization is a widely-used technique to increase system performance by achieving parallelization, maximizing locality, and minimizing overhead related to the loop nest. Among the various loop nest optimization techniques, loop interchange and loop



```
load bias
while (Weighted Sum) {
    load input activations
    load weights
    PSUM = 0
    for m = 0 to M − 1(= 15) {                        : PE Loop
        for n = 0 to N − 1(= 15) {                    : BitBrick Loop
            $PP_{m,n} = I_{2b,m,n} \times W_{2b,m,n}$    : Partial Products
            $PP\_SL_{m,n} = PP_{m,n} \ll SL_n$           : Shift-Left
            $MUL_m = MUL_m + PP\_SL_{m,n}$
        }                                             : Adder
        $PSUM = PSUM + MUL_m$
    }
}
ACC = ACC + PSUM                                      : Accumulation
```

Module Name	# of Modules	Bit-Width
Partial Products	M x N = 256	Signed 2 b x 2 b
Variable Shift	M x N = 256	Shift-Left 0~12 b
(N+1)-Way Adder Tree	M = 16	16 +⌈log₂M⌉ =16~20 b
Accumulator	1	16 + 12 = 28 b

(a) Baseline design (Bit Fusion).

```
load bias
while (Weighted Sum) {
    load input activations
    load weights
    PSUM = 0
    for n = 0 to N − 1(= 15) {                        : PE Loop
        for m = 0 to M − 1(= 15) {                    : BitBrick Loop
            $PP_{m,n} = I_{2b,m,n} \times W_{2b,m,n}$    : Partial Products
            $MUL_n = MUL_n + PP_{m,n}$                   : Intra-PE Adder
        }
        $MUL\_SL_n = MUL_n \ll SL_n$                     : Shift-Left
        $PSUM = PSUM + MUL\_SL_n$                        : Inter-PE Adder
    }
}
ACC = ACC + PSUM                                      : Accumulation
```

Module Name	# of Modules	Bit-Width
Partial Products	M x N = 256	Signed 2 b x 2 b
Variable Shift	N = 16	Shift-Left 0~12 b
M-Way Adder Tree	N = 16	$4 + \log_2 16 = 8$ b
N-Way Adder Tree	1	$16 + \log_2 16 = 20$ b
Accumulator	1	16 + 12 = 28 b

(b) BitBlade.

Figure 2: Loop nests of two accelerator architectures. (a) *Bit Fusion* and (b) *BitBlade*. In *BitBlade*, bitwise summation is performed using two loop nest optimization methods, loop interchange and loop peeling. Dotted arrows indicate loop interchange, and solid arrows indicate loop peeling.

Table 1: Area and power breakdown of PE in *Bit Fusion* [14].

	BitBricks	Shift-Add	Register	Total
Area [μm^2]	369	934	91	1394
Power [nW]	46	424	69	538

peeling help *BitBlade* to reduce the area and power consumption related to the logic for reprogrammability of bit-precision.

Fig. 2a shows the loop nest of *Bit Fusion*. The loop nests describe the process of performing weighted sum operations in the *Bit Fusion* and *BitBlade* architectures to compute neural network algorithm. At first, shift-left operations occur depending on the parameters which are determined by bit-width configurations and index of BitBricks. Such shift-left operations are performed in the same manner across all PEs. In this case, *Bit Fusion* requires 256(=M×N) variable shift modules where ‘M’ indicates the number of PEs, and ‘N’ indicates the number of BitBricks in each PE. To reduce the significant number of shift-left operations, PE and BitBrick loops are interchanged (loop interchange, dotted arrows in the Fig. 2) in the loop nest of *BitBlade*. As a result, the number of variable shift modules has been reduced to 16(=N) as shown in Fig. 2b.

Meanwhile, shifted partial products from BitBricks in the current stage and partial sum from PE in the previous stage, which have high bit precisions, are added in a 17(=N+1)-way adder tree in the *Bit Fusion* architecture. The adder tree is represented in bit-width of 16-20, and each PE has the adder tree. *Bit Fusion* has 16(=M) 17(=N+1)-way adder trees with wide bit-width, which leads to large area overhead. To reduce such a burden, the adder tree has been peeled (loop peeling, solid arrows in the Fig. 2) into intra-PE and inter-PE adder trees in *BitBlade*. The intra-PE adder tree only adds the multiplication results with the specified precision (2-bit × 2-bit), different from the baseline where variable shift-left operations are performed on each partial product. As the 16(=N) 16(=M)-way intra-PE adder tree can be represented in 8-bit precision, the area requirement becomes smaller than that of the adder tree in *Bit Fusion*. On the other hand, the inter-PE adder tree has almost the same bit-width (20-bit) as the width of adder tree in the *Bit Fusion*. However, the only 1 inter-PE adder is needed, and hence the area overhead of the inter-PE adder tree is negligible.

3.2 Bitwise Summation

In this Section, we elaborate details how to implement the loop optimization algorithm in our *BitBlade* architecture. Please refer to Fig. 1 for the explanation.

In the *Bit Fusion* architecture, BitBricks in the same position on different PEs (filled with same background color as illustrated in Fig. 1a) have the same parameters for variable shift operation. In other words, BitBricks filled in green color, B(i,0) = { B(0,0), B(1,0), · · · , B(15,0) }, have the same shift-left parameters. Additionally, BitBricks filled in yellow color, B(i,1) = { B(0,1), B(1,1), · · · , B(15,1) }, have the same shift-left parameters. Finally, BitBricks filled in orange color, B(i,15) = { B(0,15), B(1,15), · · · , B(15,15) }, have the same shift-left parameters. Base on the observation, *BitBlade* places BitBricks which have the same shift-left parameters on the same PE (BitBricks with the same background color in Fig. 1b). In the baseline architecture, variable shift-left operations occur at each BitBrick. On the other hand, in the *BitBlade* architecture, the variable shift-left operation is performed only once after calculation of a partial sum is finished in each PE (loop interchange).

In addition, PE in the *Bit Fusion* requires an adder tree with a wide bit-width to support 2b-to-8b precision configurations. Even worse, the bit-width of the adder tree is further increased to add partial sum from the previous PE to multiplication result of F-PE in the current PE. After performing the loop interchange, the shift-left operation occurs once for each PE, not for all BitBricks as in *Bit Fusion*. The adder tree is splitted into intra-PE and inter-PE

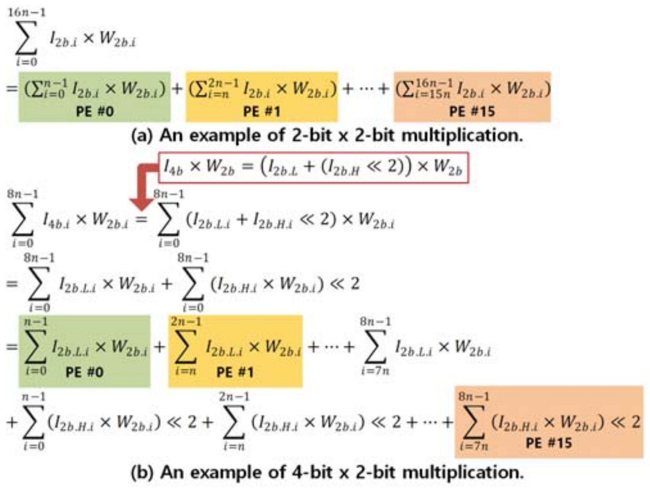

PE #0
PE #1
PE #15
(a) An example of 2-bit x 2-bit multiplication.
PE #0
PE #1
PE #15
(b) An example of 4-bit x 2-bit multiplication.

Figure 3: Supporting various bit widths in *BitBlade*.

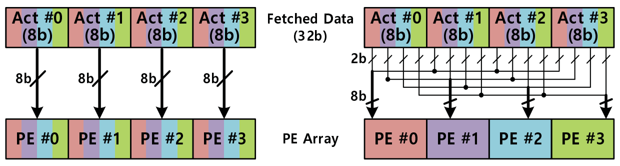


Figure 4: Fetching activations into PE array. Baseline (left). *BitBlade* with bitwise packing (right). Bitwise packing does not require additional logic.

adder trees based on the point of shift-left operation as shown in Fig. 1b (loop peeling). In this case, intra-PE adder tree, adding the partial products which did not experience the shift operations yet, can perform the addition operation with low bit-width precision and small area. Meanwhile, the partial sums calculated in each PE are incomplete, because multiplications are performed only with each two bits extracted from the input features and weights. However, neural network algorithm requires only final result of the weighted sum [12]. A complete partial sum is computed by adding the incomplete partial sums from each PE at the inter-PE adder tree. PE arrays in the *BitBlade* architecture require the same amount of the input activations and the weights as the PE arrays in the *Bit Fusion* architecture do. Therefore, the input activations are reused in the same manner as the case of *Bit Fusion*.

3.3 Supporting Various Bit Widths

BitBlade can multiply activations and weights with bit widths of { 2, 4, 8 }, thereby supports 9(=3×3) different bit-width configurations in total. Although the proposed bitwise summation architecture can support the precision of 16-bit or more in principle, recent studies have shown that 8-bit precision is good enough to execute inference operation of major DNN algorithms [6, 9]. Hence, we decided to support the bit-width up to 8-bit in our *BitBlade* design.

The partial sum from each PE experiences shift-left operation at the input of the inter-PE adder tree depending on the bit widths of activations and weights. For example, if both activations and weights have 2-bit precision, no shift-left operations are performed as described in Fig. 3a. On the other hand, if the activations have a 4-bit precision and the weights have 2-bit precision, no shift-left operations are performed for the results from PE ♯0 - ♯7 and 2-bit shift-left operations are applied tor the results from PE ♯8 - ♯15 as described in Fig. 3b.

BitBlade also supports XNOR-Net [11]. XNOR-Net uses 1-bit precision configuration for both the activations and the weights. Different from the usual multiplication methods of adding partial products after performing bitwise multiplications using AND gates, XNOR-Net performs bitwise multiplications using XNOR gates. *BitBlade* only supports AND gate based multiplication as most hardware accelerators do, because supporting both AND gate based multiplication and XNOR gate based multiplication increases area and power overhead. However, *BitBlade* can calculate XNOR-Net by mapping XNOR gate based multiplication to AND gate based multiplication. In the case of XNOR operation, binary number (0,1) is regarded as (-1,1). After the -1 or 1 is converted to 2-bit two's complement number (2'b11 or 2'b01), the number is fed into BitBrick. Afterwards, XNOR gate based multiplications are performed in the same way as AND gate based general multiplications. Although the number of BitBricks to support XNOR-Net is the same as the number of BitBricks to support 2-bit precision for both the activations and the weights, XNOR-Net still requires smaller memory footprint and fewer off-chip accesses thus increasing performance. We leave further optimizations related to XNOR-Net for future work.

3.4 Bitwise Packing

To perform bitwise summation in a PE array, bitwise packing operation is required to collect the bits located at the same bit position in the fetched data. Fig. 4 shows an example of bitwise packing with 4 PEs and 32-bit fetched data. The fetched data consist of 4 8-bit input activations. In the baseline, the 4 fetched 8-bit input activations are sent to 4 PEs in order from PE ♯0 to PE ♯3 as in Fig. 4. On the other hand, in the *BitBlade*, each fetched activation value is divided into 4 groups of 2 consecutive bits and then fed to different PEs. For example, the first two bits of Act ♯0 - ♯3 are assigned to PE ♯0, the 3rd and 4th bits of the activations are assigned to PE ♯1, the 5th and 6th bits of the activations are assigned to PE ♯2, and the 7th and 8th bits of the activations are assigned to PE ♯3. Bitwise packing only changes the order of the bit positions where each bit of data is located. In other words, baseline and *BitBlade* differ only in the bit order of the data received by the PE array, and the data transferred is the same. Therefore, bitwise packing does not require additional logic in *BitBlade*.

The bitwise packing in *BitBlade* may look similar to transpose operation in bit-serial computation. However, the transposer in bit-serial computation includes numerous flip-flops and multiplexers because the transposer needs to contain several activations first and then sequentially send them out a bit per clock cycle, which leads to area and power overhead.

4 EXPERIMENTAL RESULTS

4.1 Simulation Setup

In this section, we compare the area, performance, and energy consumption of *BitBlade* with 2 state-of-the-art precision-scalable

Table 2: Comparison of throughput among *ENVISION*, *Bit Fusion*, and *BitBlade* with various DNNs. 'EN', 'BF', 'BB' indicate *ENVISION*, *Bit Fusion*, *BitBlade*, respectively. 'M'×'N' expresses 'M'-bit activation and 'N'-bit weight precision case. Average values on the right two columns are relative throughput to *Bit Fusion*. The average values are calculated in the 2×2, 4×4, 8×8 cases, because *ENVISION* only supports the 3 different bit-width cases. LSTM layer for TIMIT dataset was configured by referring LSTM layer information used in ESE paper [4].

Throughput	2×2			2×4		2×8		4×4			4×8		8×8			(Avg.) Rel. to BF	
[TOPS]	EN	BF	BB	BF	BB	BF	BB	EN	BF	BB	BF	BB	EN	BF	BB	EN	BB
AlexNet	1.14	4.18	4.78	2.36	2.70	1.22	1.40	0.57	1.22	1.40	0.63	0.71	0.29	0.32	0.36	35%	114%
VGG-16	1.24	3.79	4.33	2.22	2.54	1.20	1.38	0.62	1.20	1.38	0.62	0.71	0.31	0.32	0.36	41%	114%
ResNet-152	1.13	3.22	3.67	2.02	2.31	1.14	1.30	0.56	1.14	1.30	0.60	0.69	0.28	0.31	0.35	42%	114%
LSTM-TIMIT	1.21	4.48	5.12	2.38	2.72	1.23	1.40	0.60	1.23	1.40	0.62	0.71	0.30	0.31	0.36	35%	114%

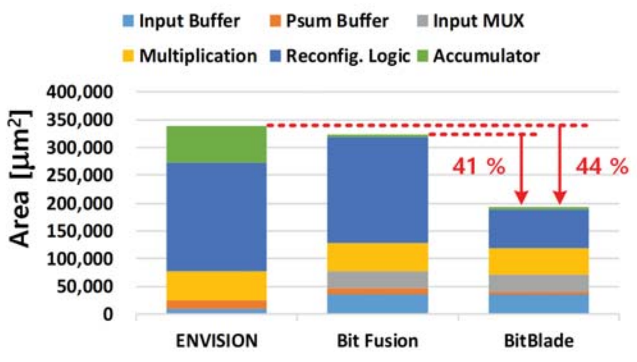


Figure 5: Comparison of area among *ENVISION*, *Bit Fusion*, and *BitBlade*.

neural network accelerators (*ENVISION* [10], *Bit Fusion* [14]). While *BitBlade* and *Bit Fusion* support the 2-bit to 8-bit, *ENVISION* supports the 4-bit to 16-bit. For fair comparison, each accelerator has 16 × 16 PEs and *ENVISION* was scaled to support 2-bit to 8-bit in our evaluation. The accelerators were synthesized in a 28nm CMOS technology using Synopsys Design Compiler. The estimation of area and power consumption was done using Synopsys PrimeTime PX.

4.2 Evaluation

Area: Fig. 5 shows that *BitBlade* requires smaller area than *ENVISION* by 44% and *Bit Fusion* by 41%. The area saving mostly comes from reduction of the logic gates for bit-width reconfigurability (blue portion in each bar in Fig. 5). In *ENVISION*, the DVAFS multiplier consists of multiple sub-multipliers, and turns on or off the sub-multipliers depending on precision configurations. Although the main concept of the DVAFS multiplier is simple, it requires complex circuit implementation for various sign-extension conditions to enable both signed multiplication operations and reconfiguration of bit-width. The variable sign-extension module and adder tree to collect and add partial products from the sub-multipliers accounts for 58% of area. Furthermore, each PE in *ENVISION* has multiple accumulators to support various single instruction multiple data (SIMD) configurations. The SIMD accumulator occupies 20% of area. *Bit Fusion* provides fusion capabilities to support bit widths of 2, 4, 8 for both activations and weights. To support various bit-width configurations, *Bit Fusion* has complex reconfigurable logic which consumes 59% of total area.

Throughput: Table 2 shows that *BitBlade* is faster than *ENVISION* and *Bit Fusion* in various neural network cases. The throughput improvement over the *ENVISION* is much larger than the improvement over the *Bit Fusion*. In the 8-bit × 8-bit case, throughput of *ENVISION* is almost same as that of *BitBlade*. However, the DVAFS multiplier does not use 50% and 75% of sub-multipliers in the 4-bit × 4-bit and 2-bit × 2-bit cases, and hence the throughput of *ENVISION* is significantly degraded in the lower bit-precision cases. The throughput of *ENVISION* is smaller than that of *BitBlade* by 14-76% depending on the network and bit widths.

It is worthwhile to note that *BitBlade* and *Bit Fusion* should have similar throughput in principle, because *BitBlade* is mostly based on loop interchange and loop peeling which do not affect the number of input features and weights fed to the PE array per clock cycle. Throughput improvement in *BitBlade* compared to *Bit Fusion* comes from the higher clock frequency. Different from PE with 16 variable shift modules in *Bit Fusion*, the variable shift operations do not occur inside a PE in *BitBlade*. As a result, the adder-tree in the PE of *BitBlade* has smaller bit-width than that of *Bit Fusion*, thereby decreasing critical path delay. In our experiment, *BitBlade*, *Bit Fusion*, *ENVISION* are synthesized with clock frequency of 714MHz, 625MHz, 667MHz, respectively. Therefore, *BitBlade* shows higher throughput than that of *Bit Fusion* by 14%.

Energy Consumption: Fig. 6a-d show detailed comparison of energy-efficiency of the *ENVISION*, *Bit Fusion*, and *BitBlade* for various neural networks. Overall, *BitBlade* showed improved energy consumption by 36-46% and 39-59% over that of *Bit Fusion* and *ENVISION*, respectively. It can be observed that the improvement in energy consumption of *BitBlade* over *Bit Fusion* is consistent regardless of bit-precision. It is because the difference in energy consumption mainly comes from the difference in the number of logic gates related to bit-width reconfigurability. As both of *BitBlade* and *Bit Fusion* utilize the same number of BitBricks for the same bit-precision computations, energy consumption in other parts of the PE is more or less same for both designs. On the other hand, the relative energy efficiency of *ENVISION* compared to *BitBlade* and *Bit Fusion* becomes worse as the bit-width decreases. For example, for 8-bit × 8-bit configuration, energy consumption of *ENVISION* is almost same as that of *Bit Fusion*, but *ENVISION* in 2-bit × 2-bit case has much higher energy consumption than that of *Bit Fusion*. The main reason is that *ENVISION* still consumes non-negligible energy for bit-width reconfiguration circuits even for the turned-off

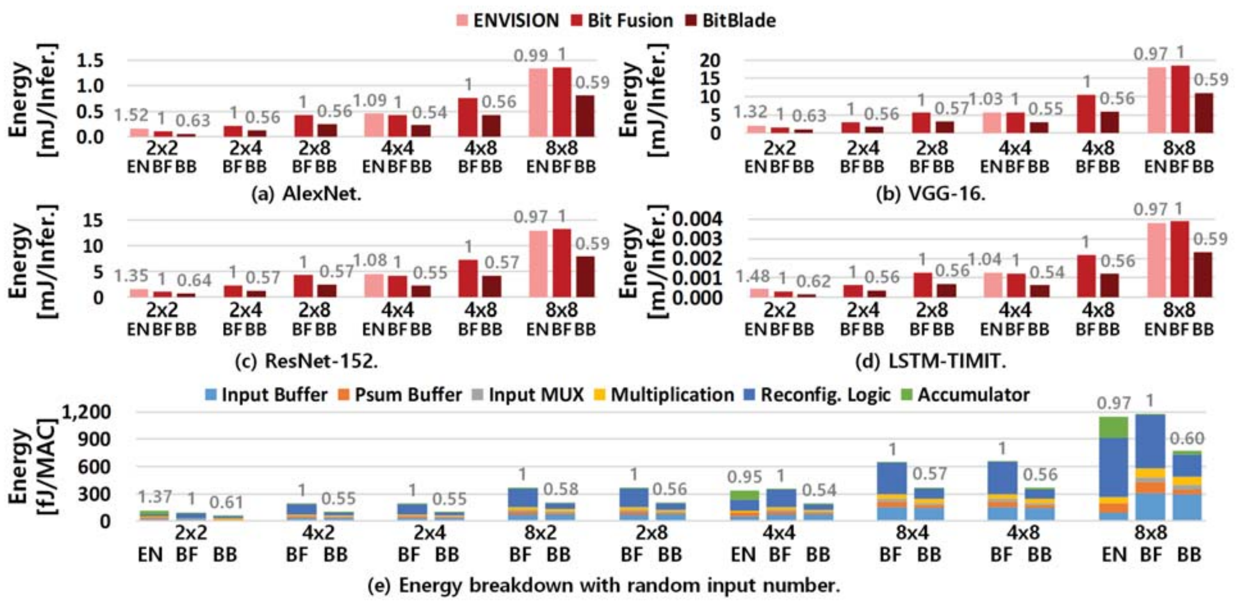


Figure 6: Comparison of energy consumption per inference among *ENVISION*, *Bit Fusion*, and *BitBlade* with various DNNs and bit widths. (a) AlexNet, (b) VGG-16, (c) ResNet-152, and (d) LSTM-TIMIT. (e) Energy breakdown with random input number. Numbers above bars in the figures indicate normalized energy to that of *Bit Fusion* architecture. 'EN', 'BF', 'BB' indicate *ENVISION*, *Bit Fusion*, *BitBlade*, respectively.

sub-multipliers in the lower bit-precision computations. In addition to the data in Fig. 6, we also analyzed the energy-efficiency for 4-bit × 2-bit, 8-bit × 2-bit, and 8-bit × 4-bit cases. As the energy consumption in 'm'-bit × 'n'-bit configurations show similar results to those of 'n'-bit × 'm'-bit configurations, the data is not shown in the figure.

Finally, we also experimented the random input case and observed that the results are similar to those with actual neural network simulations (Fig. 6e).

5 CONCLUSIONS

We introduced *BitBlade*, area and energy-efficient precison-scalable neural network accelerator. *BitBlade* minimized the logic gates related to the bit-width reconfiguration by using bitwise summation method, thereby minimizing area and maximizing throughput and energy-efficiency. When synthesized in a 28nm CMOS technology, *BitBlade* showed reduction in area by 41% and energy consumption by 36-46% compared with the state-of-the art precision-scalable accelerator (*Bit Fusion*). Moreover, *BitBlade* boosts performance 14% over *Bit Fusion*. This work aims to enable applications with various performance requirements and power constraints to perform neural network algorithms in a fast and energy-efficient manner.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ICT Consilience Creative program(IITP-2019-2011-1-00783) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation)

REFERENCES

[1] Song Han et al. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[2] Song Han et al. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[3] Song Han et al. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 243–254.

[4] Song Han et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 75–84.

[5] Itay Hubara et al. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[6] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 1–12.

[7] Patrick Judd et al. 2016. Stripes: Bit-serial deep neural network computing. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[8] Jinmook Lee et al. 2018. Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*. IEEE, 218–220.

[9] Asit Mishra et al. 2017. WRPN: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134* (2017).

[10] Bert Moons et al. 2017. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 246–247.

[11] Mohammad Rastegari et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[12] Sungju Ryu et al. 2019. Feedforward-Cutset-Free Pipelined Multiply-Accumulate Unit for the Machine Learning Accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2019), 138–146.

[13] Sayeh Sharify et al. 2018. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 20.

[14] Hardik Sharma et al. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 764–775.

[15] Hyeonuk Sim et al. 2017. A new stochastic computing multiplier with application to deep convolutional neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 29.