Deeper Weight Pruning without Accuracy Loss in Deep Neural Networks

Byungmin Ahn
Dept. of Electrical and Computer Engineering
Seoul National University, Korea
bmahn@snucad.snu.ac.kr

Taewhan Kim
Dept. of Electrical and Computer Engineering
Seoul National University, Korea
tkim@snucad.snu.ac.kr

***Abstract*—This work overcomes the inherent limitation of the bit-level weight pruning, that is, the maximal computation speedup is bounded by the total number of non-zero bits of the weights and the bound is invariably considered "uncontrollable" (i.e., constant) for the neural network to be pruned. Precisely, this work, based on the canonical signed digit (CSD) encoding, (1) proposes *a transformation technique* which converts the two's complement representation of every weight into a set of CSD representations of *the minimal or near-minimal number of essential (i.e., non-zero) bits*, (2) formulates the problem of selecting CSD representations of weights that *maximize the parallelism of bit-level multiplication* on the weights into *a multi-objective shortest path problem* and solves it efficiently using an approximation algorithm, and (3) proposes *a supporting novel acceleration architecture* with no additional inclusion of non-trivial hardware. Through experiments, it is shown that our proposed approach reduces the number of essential bits by 69% on AlexNet and 74% on VGG-16, by which our accelerator reduces the inference computation time by 47% on AlexNet and 50% on VGG-16 over the conventional bit-level weight pruning.**

I. Introduction

Although deep neural networks (DNNs) have proved their excellent capabilities [1]–[5], the intensive computation requirement still makes it difficult to efficiently perform DNN inference on mobile devices with limited computing resources. In particular, in the convolutional DNNs, the convolutional operations between input activations and filter weights performed in successive layers occupy almost all of the DNN operations, and many studies have attempted to improve the computation efficiency.

The acceleration architectures (e.g., [6], [7]) introduced at the beginning research stage had focused mainly on parallelizing the multiply-and-accumulate (MAC) operations between input activations and weights, but they did not efficiently utilize the hardware resources in that they overlooked the resource waste on MAC operations with zero-weight. Then, a number of acceleration architectures have taken into account attenuating the computation waste, which is so-called *weight pruning*. The weight pruning in inference computation without accuracy loss is accomplished in two ways: one is *word-level* weight pruning, which skips the 0-value weights in the inference computation (e.g., [8], [9]), and the other is *bit-level* weight pruning, which skips the 0-bits in the value representation of weights in the inference computation (e.g., [10]). Recently, it is shown that the bit-level weight pruning in [10] achieves $1.3\times\sim1.5\times$ speedup of inference computation over the state-of-the-art baseline DaDianNao in [7].

This work addresses a new bit-level weight pruning. The inherent limitation of existing accelerators supporting bit-level pruning without accuracy loss is that the speedup is strictly limited. Specifically, for a DNN with M total number of bits in the representation of all weights, the maximal speedup cannot be more than $\frac{M}{n_e}$ where n_e indicates the total number of non-zero bits in the value representation of all weights. (We call all bits other than 0-bits *essential* bits.) So far, no work has attempted to push down the amount of n_e, though it is an important factor that critically determines the inference performance. This work overcomes this inherent limitation. The contribution of this work can be summarized as:

1. To increase the degree of freedom for representing weights, we propose *a transformation technique*, exploiting the canonical signed digit (CSD) encoding, which transforms the two's complement (fixed-point) representation of every weight into a set of multiple CSD representations of *the minimal or near-minimal number of essential bits*.
2. We formulate the problem of selecting CSD representations of weights that *maximize the parallelism of bit-level multiplication* on the weights into a *multi-objective shortest path problem* and solve it efficiently using an approximation algorithm.
3. We propose *a novel hardware architecture* that is able to exploit the bit-level weight pruning fully and effectively on the CSD representations of weights.

II. Background and Motivation

A. Background: Bit-level Weight Pruning

An operation F that performs MAC on N pairs of fixed-point weight W_i in two's complement representation of length B and activation A_i of the same length for $0 \leq i \leq N-1$ can be expressed as:

$$F = \sum_{i=0}^{N-1} A_i \times W_i = \sum_{b=0}^{B-1} 2^b \times \sum_{i=0}^{N-1} (A_i \times w_i^b) \qquad (1)$$

where w_i^b represents the b^{th} bit-value in W_i. (B is 8 or 16 in many DNN models.) Then, F can be evaluated in two steps:

978-3-9819263-4-7/DATE20/©2020 EDAA

Step 1. *Bit-level multiplication*:

$$X^b = \sum_{i=0}^{N-1} A_i \times w_i^b, \ b = 0, \cdots, B-1 \qquad (2)$$

Step 2. *Shift-and-accumulation*:

$$F = X^0 + X^1 \cdot 2^1 + X^2 \cdot 2^2 + \cdots + X^{B-1} \cdot 2^{B-1} \quad (3)$$

The *bit-level weight pruning* is applied to the bit-level multiplication in Step 1 by pruning the computation of $A_i \times w_i^b$ if $w_i^b = 0$. Since the weight values have already been known when performing inference, a bit column-wise condensed arrangement on the bit values in W_i, $i = 0, \cdots, N-1$ is possible by pulling out the 0-bits from the arrangement to accelerate the computation of X^b, $b = 0, \cdots, B-1$ in *Eq.2*.

A well-known conventional bit-level weight pruning called *weight kneading* [10] is illustrated in Fig. 1, in which the bit values of the six weights (i.e., $N = 6$), one row for each weight as shown in Fig. 1(a), are rearranged as shown in Fig. 1(b) by removing 0-bits and moving up 1-bits column-wise. Consequently, if a bit-level multiplication in *Eq. 2* for each row takes one clock cycle, the compressed arrangement of bit values enables to reduce the total number of execution cycles from 6 to 4.

We use notation k to refer the number of initial weights to be collectively considered for bit-level pruning, which we call *pruning stride* while we use notation k' $(< k)$ to refer the number of weights compressed via bit-level pruning (e.g., $k = 6$ and $k' = 4$ in Fig. 1).

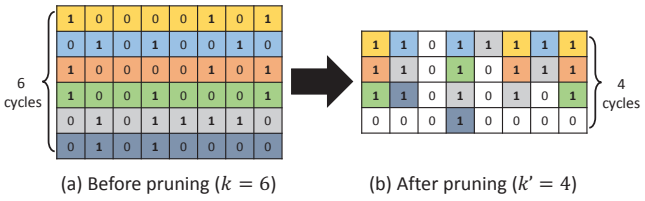

Fig. 1. Conventional bit-level weight pruning [10]. (a) Six weights of an 8-bit fixed-point representation before pruning. (b) Four weights after the application of bit-level pruning. Since the 4^{th} bit-column has four 1-bits, dummy 0-bits are added to the other columns to maintain four weight rows.

B. Motivation 1: Inefficiency in Two's Complement Representation

Two's complement number system is commonly used to represent signed fixed-point binary values. One biggest advantage of two's complement over the other number systems is that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those of unsigned binary number. This property makes the system simple to implement. However, it is not true in the domain of bit-level weight pruning.

For example, with 8 bits, the two's complement representation of -13 is $(11110011)_2$ which includes 6 non-zero bits out of the 8 bits. On the other hand, if we use sign-magnitude representation, -13 is expressed as $(10001101)_2$ which includes the leftmost sign-bit and 3 non-zero bits out of the remaining 7 bits. Thus, if we are able to handle the sign-bit efficiently[1] in sign-magnitude representation in performing the bit-level multiplication in *Eq. 2*, it is desirable to apply bit-level pruning to weights in sign-magnitude representation rather than to weights in two's complement representation.

Fig. 2 shows the comparison of the word/bit-level value distributions of the weights in AlexNet and VGG-16 when expressing the weights in two's complement and sign-magnitude representations. Fig. 2(a) shows the normalized weight value distributions, which form bell-shapes. Fig. 2(b) then shows non-zero bit (i.e, 1-bit) distribution over the 16 bit positions (LSB on the rightmost) when the weights are expressed in two's complement and sign-magnitude representation. Since the number of 1-bits in representing a near-zero negative weight in sign-magnitude is much smaller than that in two's complement (the bars on bit positions 11 to 14 in Fig. 2(b)), the sign-magnitude representation uses 31.3% fewer 1-bits on AlexNet and 35.9% fewer 1-bits on VGG-16 than the two's complement representation.

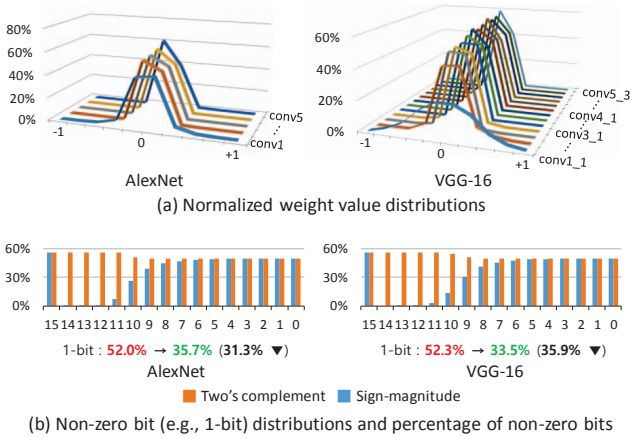

Fig. 2. Comparison of word/bit-level value distributions of the weights in AlexNet and VGG-16 when using two's complement and sign-magnitude representations.

C. Motivation 2: Inability to Exploit CSD Representation

By converting the magnitude of the sign-magnitude representation or the two's complement representation of every weight into the canonical signed digit (CSD) representation, it allows to use the same or fewer number of non-zero bits. Considering a ternary numeral system in CSD using three digits of -1, 0, and 1, for example, $(000011110)_2$ $(= 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 = 30)$ can be expressed as $(0001000\bar{1}0)_2$ $(= 1 \cdot 2^5 + (-1) \cdot 2^1 = 30)$ in which $\bar{1}$ denotes digit value of -1, thereby saving two non-zero digits. We call the non-zero digits other than 0 in a representation of number system *essential bits*.

Utilizing the CSD representation for a deeper bit-level weight pruning requires to solve the following four problems:

P1. (**Multiple CSD representations**) CSD representation for a number is not unique. For example, decimal number 103 (= $(001100111)_2$) can be expressed as any of

[1]Our idea of efficiently handling the sign-bit will be described in Sec. III.

the following CSD representations of a fewer or equal number of essential bits: $(00110100\bar{1})_2$, $(010\bar{1}00111)_2$, and $(010\bar{1}0100\bar{1})_2$. In Sec. III, we *propose a method of selecting CSD representations that leads to a maximal parallelism of bit-level multiplication* in *Eq. 2*.

P2. (**Memory space for representing ternary values**) Together with 0 and 1, CSD representation requires a memory space for encoding -1, which intuitively implies that for each digit, two bits are required, for example, 00, 01, and 10 respectively encode 0, 1, and $\bar{1}$ in a CSD representation. Accordingly, the double memory space for storing weights in CSD representation would be required. In Sec. IV, we *devise a novel technique to use nearly the same amount of memory space as required for storing binary numbers of 0 and 1* though we use CSD representation.

P3. (**Sign-bit in sign-magnitude representation**) The sign-bit which is the leftmost bit in a sign-magnitude representation indicates whether the magnitude is positive ($= 0$) or negative ($= 1$). The sign-bit is one major obstacle to complicate the process of arithmetic operation, i.e., (1) depending on the sign-bit, the operation type of addition or subtraction is determined, and (2) by comparing the magnitudes, the sign-bit of resulting magnitude of the operation is set. In Sec. III, we *propose a way of completely removing this burden by exploiting our extended notation of CSD representation.*

P4. (**Supporting subtraction for** $\bar{1}$) Two's complement representation offers the most economical hardware for supporting subtraction as well as addition, but our CSD based approach to minimizing the number of essential bits stems from sign-magnitude representation. In Sec. IV, we *propose an architectural technique which is able to seamlessly link the transformed CSD representation to efficient hardware supporting two's complement operations* in performing the bit-level multiplication in *Eq. 2*.

III. CSD Based Deeper Weight Pruning

The first step in Sec. III-A is to generate all CSD representations with the minimal or near-minimal number of essential bits for every weight. The next step in Sec. III-B is then to select CSD representations among the ones obtained in the first step that leads to a maximal parallelism on bit-level multiplication.

A. Generating CSD Representations

Since all the weight values are known, generating possible CSD representations for weight values is processed *off-line*, and the generation is performed in two steps.

Step 1 (*Recursively applying CSD conversion*): For a weight value in sign-magnitude representation, we recursively apply CSD conversion to the weight from the rightmost 1-bit string of size ≥ 2 to the left. Fig. 3(a) shows the *CSD enumeration tree* produced by recursively applying CSD conversion to $(\mathbf{1}0110111)_2$ $(= -55)$ where **1** is sign-bit and the $0/1/\bar{1}$-bit strings in red color are the CSDs resulting from the 1-bit strings in the dotted circles of their parent nodes.

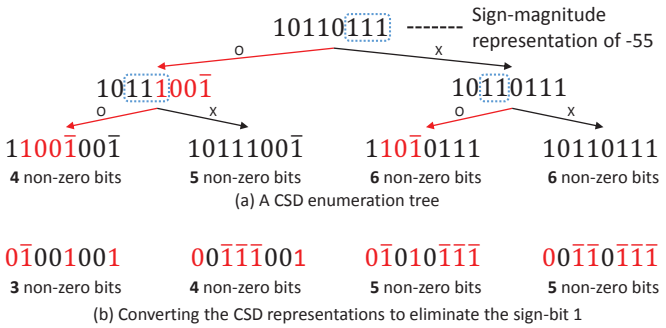

Fig. 3. A stepwise generation of all possible CSD representations for a weight value of -55 in sign-magnitude representation.

• **Step 2** (*Eliminating sign-bit for negative value*): Since CSD uses $\bar{1}$ as well as 0 and 1, it is possible to convert the leftmost 1-bit for a negative value into 0-bit by converting 1-bit in every remaining bit to $\bar{1}$ and $\bar{1}$-bit to 1. Fig. 3(b) shows the sign-bit (i.e., 1) free CSD representations produced by the application of such extended CSD conversion to every CSD representation obtained in Fig. 3(a).

Through Steps 1 and 2, we ensure that all possible CSD representations with a fewer or equal number of essential bits for every weight are available and there is no meaning on the leftmost sign-bit of every CSD representation. To boost up the opportunity of maximal parallelism of bit-level multiplication in Sec. III-B, rather than collecting the CSD representation of minimal essential bits only, we employ a user-defined *relaxing parameter* γ to additionally include the CSD representations of up to γ more essential bits over the minimal essential bits.[2]

B. Selecting CSD Representations for Maximal Parallelism

We formulate the problem of selecting CSD representations that maximizes bit-level parallelism into a variant problem of finding a shortest path in a graph. We describe our formulation using the example in Fig. 4, which illustrates a full flow of selecting CSD representations for maximal parallelism. The three columns at the top left in Fig. 4 illustrate the CSD generation process performed in Sec. III-A.

From the CSD representations, we construct a directed graph $G(V, A, C)$ as shown in Fig. 4 where every CSD representation has a distinct node in V arranged horizontally in G for CSD representations for each weight (labeled as gray or blue), and V has two additional nodes (labeled as red): src having no entering arc, and $dest$ having no leaving arc. There exists an arc between every pair of nodes $v^i_{(\cdot)}$ and $v^{i+1}_{(\cdot)}$ $(v^i_{(\cdot)} \rightarrow v^{i+1}_{(\cdot)})$ where $v^i_{(\cdot)}$ and $v^{i+1}_{(\cdot)}$ represent nodes arranged in the i^{th} and $(i+1)^{th}$ horizontal lines in G, respectively. (In such convention, $src = v_0^{-1}$ and $dest = v_0^k$ where k is pruning stride.) We set a vector cost $c^{i+1}_{j_1,j_2} \in C$ for arc $(v^i_{j_1}, v^{i+1}_{j_2}) \in A$ to a bit vector of the same bit-size of CSD representation such that for $0 \leq b \leq B-1$, $c^{i+1}_{j_1,j_2}[b] = 1$, if b^{th} bit-value of

[2]We set $\gamma = 2$ for $B = 8$ and $\gamma = 4$ for $B = 16$ in our experiments.

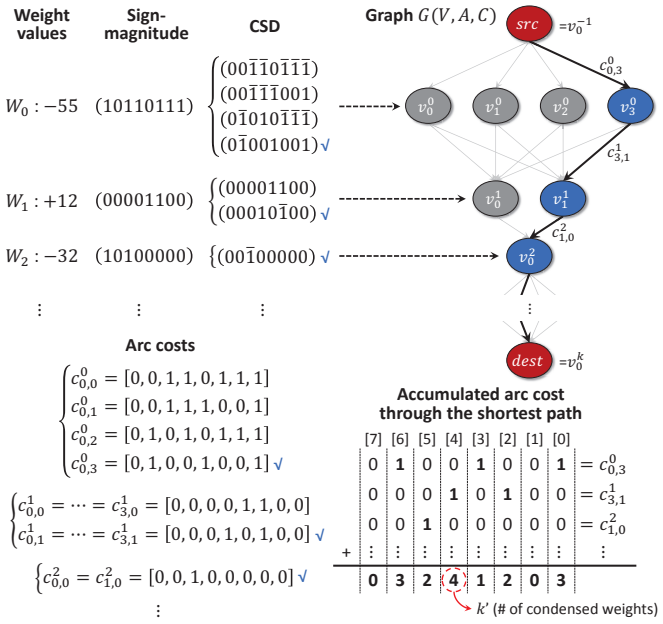

Fig. 4. An illustration of full flow (from top left to bottom right) of selecting CSD representations for maximal parallelism of bit-level multiplication. The top left illustrates the CSD generation process performed in Sec. III-A. The graph at the top right shows our graph-based formulation of selecting CSD representations, and the arc cost setting is shown at the bottom left. The alignment of essential bits for maximal parallelism taken from the selected CSD representations is placed at the bottom right.

$j_2{}^{th}$ CSD representation for W_{i+1} is 1 or $\bar{1}$, and $c^{i+1}_{j_1,j_2}[b] = 0$ otherwise, as illustrated in Fig. 4.[3]

Since we want to minimize the maximum among the numbers of essential bits in bit-columns when the selected CSD representations are vertically stacked, as shown at the bottom right in Fig. 4, we solve the CSD representation selection problem into a problem of finding a *multi-objective shortest path* (MOSP) from src to $dest$ in $G(V, A, C)$. Since the decision version of MOSP problem is known to be NP-complete even when $B = 2$ [11], we use Warburton's polynomial-time ϵ-approximation algorithm in [12], which guarantees the worst-case time and space bounds of $O(rn^3(n/\epsilon)^{2r})$ and $O(rn(n/\epsilon)^r)$ respectively, where $n = |V|$ and $r = B$. The heavy line in G in Fig. 4 shows the MOSP solution and the two figures at the bottom show the corresponding arc costs and the arrangement of the selected CSD representations for parallel bit-level multiplication.

IV. Supporting Hardware Architecture

The baseline of our CSD supporting architecture follows that in [10], but ours has a couple of novel features that are unique for efficiently handling CSD representations.

A. Technique for Using a Single Bit to Encode Ternary Value

Storing a ternary value (0, 1, and $\bar{1}$) in memory normally requires 2 bits. Thus, for k' ($< k$) weights compressed from a set of initial weights of pruning stride k, a memory space of

[3]One exception is that our full hardware utilization to be explained later in Sec. IV-C allows to set $c^{i+1}_{j_1,j_2}[0] = c^{i+1}_{j_1,j_2}[B-1] = 1/2$ if 0^{th} bit-value of $j_2{}^{th}$ CSD representation for W_{i+1} is 1 or $\bar{1}$.

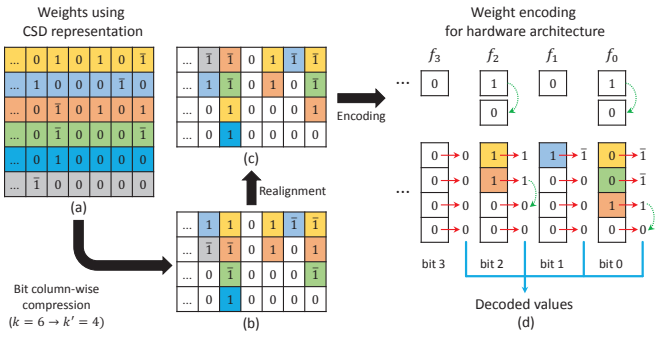

Fig. 5. (a) An initial weight alignment in CSD representation. (b) Bit column-wise compressed alignment for (a). (c) Bit column-wise realignment obeying the *ternary ordering rule* for (b). (d) Encoding ternary values of CSD representations in memory with the flag values, and the decoding implication.

$2k' \cdot B$ bits shall be needed. In the following, we propose a technique to reduce from the space requirement of $2k' \cdot B$ bits to $(k'+1) \cdot B$ bits, which is also much lower than $k \cdot B$ bits for storing initial k weights. (However, note that in processing parallel bit-level multiplication on weights produced by bit-level pruning, $k' \cdot B \cdot \log_2 k$ additional bits are commonly and essentially required to index the right ones among the k input activations for bit-level multiplication.) We describe our technique using the example in Fig. 5. An initial weight alignment in CSD representation produced by Sec. III-B with pruning stride $k = 6$ is shown in Fig. 5(a), which is then bit column-wise condensed as shown in Fig. 5(b). To achieve the memory constraint of $(k'+1) \cdot B$ bits, we introduce B *flags* $(f_0, f_1, \cdots, f_{B-1})$, and define *ternary ordering rule*:

Definition 1. (**Ternary ordering rule**): Let $\mathcal{L}_i$ be the list of all bit-values on the i^{th} bit-column in the condensed weight alignment of CSD representation. Thus, $|\mathcal{L}_i| = k'$. We then sort the elements in $\mathcal{L}_i$ such that $\bar{1}$ has the highest priority, and 1 has a higher priority over 0. We call such ordering of ternary values in $\mathcal{L}_i$ *ternary ordering rule*. Fig. 5(c) shows the bit column-wise ordered arrangement from Fig. 5(b). We denote $S_{\bar{1}}$, S_1, and S_0 to the sublists of all $\bar{1}$, all 1, and all 0 values in a list $\mathcal{L}_i$ satisfying $\mathcal{L}_i = S_{\bar{1}}||S_1||S_0$ where $||$ means concatenation.

Example 1. In Fig. 5(c), $\mathcal{L}_2 = [1, 1, 0, 0]$, thus $S_{\bar{1}} = []$, $S_1 = [1, 1]$, and $S_0 = [0, 0]$ while $\mathcal{L}_0 = [\bar{1}, \bar{1}, 1, 0]$, thus $S_{\bar{1}} = [\bar{1}, \bar{1}]$, $S_1 = [1]$, and $S_0 = [0]$.

Then, the rules for setting flag f_i and k' memory bits in the i^{th} bit-column are:

1. If $S_1 = []$, set $f_i = 0$, and all memory bits for $S_{\bar{1}}$ to 1 and the rest to 0.
2. If $S_1 \neq []$, set $f_i = 1$, and all memory bits for S_1 to 1 and the rest to 0.
3. When the ordered memory bits transit from 1 to 0, reset f_i to 0.

Example 2. Fig. 5(d) shows a set of cases in setting f_i and memory bits: For the 0^{th} bit-column, *Rule 2* is applied, which means memory bit 0 with flag 1 indicates the value is $\bar{1}$, thus subtraction (i.e., $A \times (-1) = -A$) will be performed. Then, during the subsequent accumulation process, when memory bit changes to 1, it means the value is 1, thus addition will

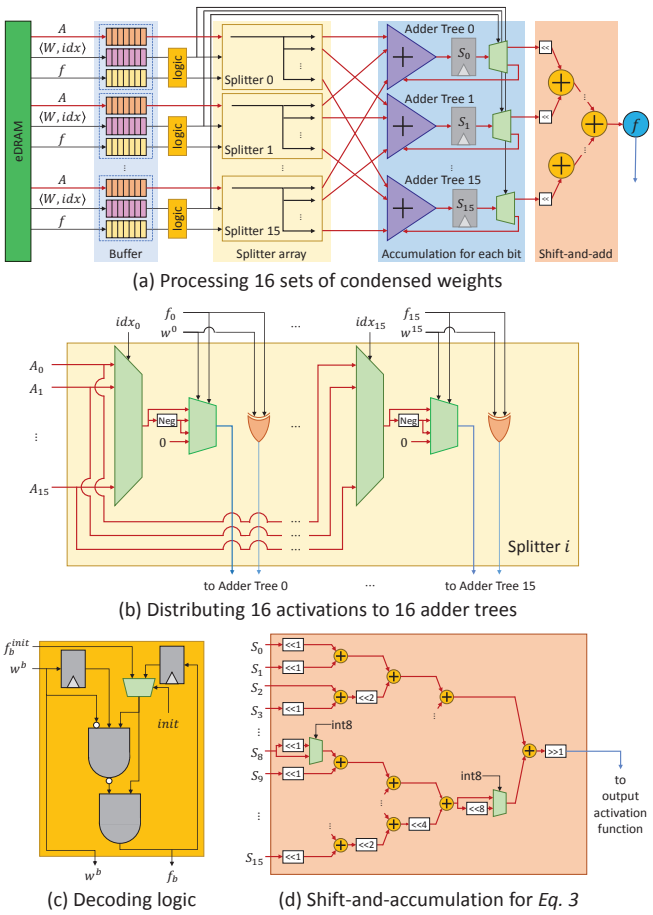

Fig. 6. (a) Our proposed hardware architecture with 16 parallel lanes, each lane processing k' condensed weights, one weight at a time. (b) A microarchitecture of splitter suited for processing CSD representation. (c) An implementation for decoding a flag and memory bit. (d) A final adder tree for suming all bit-level partial-sums.

be performed. Lastly, the flag is reset to 0 when memory bit changes to 0 according to *Rule 3*, as shown by the green dotted arrow in Fig. 5(d), which means no further operation will be performed. Likewise, *Rule 1*, *Rules 2 and 3*, and *Rule 1* are applied to the 1^{st}, 2^{nd}, and 3^{rd} bit-columns, respectively.

B. Structure of Supporting Architecture

Fig. 6(a) shows the overall hardware structure. Initially, all values of flags f, activations A, pruned weights W aligned by ternary order rule, and indexes idx for activation selection are loaded from on-chip eDRAM to internal buffers, which are then transmitted to the corresponding splitters for every time step. It should be noted that before feeding to splitters, the circuit in the yellow boxes in Figs. 6(a) and 6(c) decodes from f_i and i^{th} weight bit values to decipher if the encoding is addition ($+A$), subtraction ($-A$), or no ($0 \cdot A$) accumulation. The decoded two signal values are then transmitted to the splitters to prepare one's complement representations $\bar{A}$ for $-A$ in the splitters (shown in Fig. 6(b)) and to complete the two's complement accumulation by adding extra 1 for $-A$ (i.e., $-A = \bar{A} + 1$). The final adder tree as shown in Fig. 6(d) shifts and sums all bit-level partial accumulations produced by the adder tree on each lane and delivers the result to the output

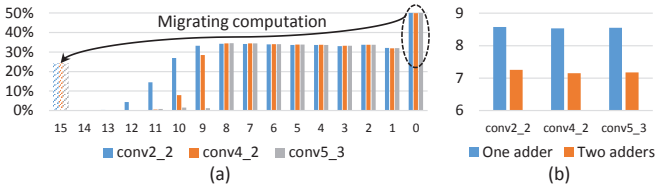

Fig. 7. Motivation and effectiveness of using two accumulation adders for the essential bits in the least significant bit (LSB) position. (a) Bit-wise comparison of the percentages of essential bits for a number of layers in VGG-16 with $B = 16$ and $k = 16$. (b) Reduction of the total computation cycles when one more accumulation adder is allocated in (a) for processing all essential bits in the LSB position.

activation function. Note that the architecture can speed up by $2\times$ if $B = 8$.

C. Full Utilization of Accumulation Adders

We have two observations regarding parallel bit-level multiplication: *Observation 1*: the sum of the sizes of $S_{\bar{1}}$ and S_1 in $\mathcal{L}_0$ (i.e., the number of the ternary values of $\bar{1}$ and 1 in the rightmost bit column in the CSD weight alignment) is the largest among all bits as shown in Fig. 7(a), which is a critical obstacle for boosting up maximal parallelism; *Observation 2*: the elimination of the 1-bit in the leftmost bit in sign-magnitude weight representation (e.g., Fig. 3(b)) means to save one accumulation adder. Thus, by equally partitioning the essential values in $S_{\bar{1}}$ and S_1 of $\mathcal{L}_0$ into two parts and performing them concurrently by the two accumulation adder trees responsible for the 0^{th} and $(B-1)^{th}$ bit-columns, the obstacle in *Observation 1* can be completely removed at no hardware cost as supported by *Observation 2*. As shown in Fig. 7(b), a considerable overall cycle reduction (16%) is possible. In the actual implementation as shown in Fig. 6(d), the $i^{th}(i \geq 1)$ bit-column is processed in the $(i+1)^{th}$ lane and the 0^{th} bit-column is divided into 0^{th} and 1^{st} lanes.

V. Experimental Results

We have used the pre-trained DNN models of AlexNet [1] and VGG-16 [2], trained using ImageNet dataset. Experiments are performed to assess (1) "how much our CSD representation is able to reduce the number of essential bits" and (2) "how much our accelerator is effective in performance and memory usage".

Table I shows the comparison of the total number of essential bits used by AlexNet and VGG-16 when representing all weights with two's complement, sign-magnitude, and our CSD. We set the pruning stride k to 8. In short, our CSD technique is able to represent the weights by using 52∼74% fewer number of essential bits over that of the two's complement.

Fig. 8 compares the memory sizes used by Tetris [10] and our accelerator. Since ours uses a much fewer number of essential bits, it directly affects the memory size reduction for both weight encoding and activation selection though flag bits are added. Overall, our architecture uses up to 52% less memory space than that of the conventional architecture supporting parallel bit-level multiplication.

Fig. 9 compares the performance of Tetris and ours in terms of clock cycles required to perform a set of weights with bit-size of 8 and 16, and pruning stride of 8, 16 and 32. We

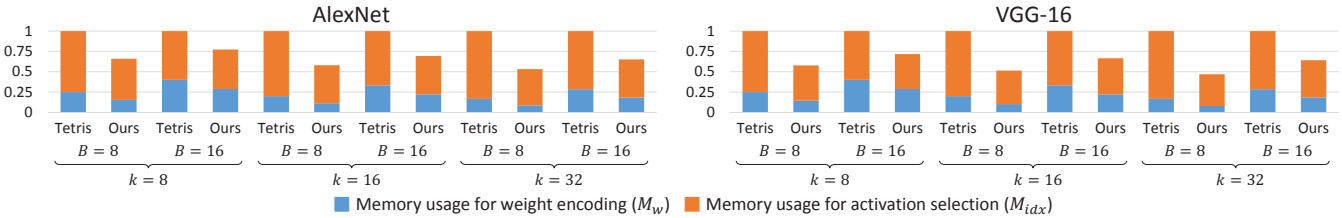

Fig. 8. Comparison of memory usage by Tetris [10] and ours when bit-size $B = 8$ and 16, and pruning stride $k = 8$, 16 and 32. M_w includes the bits for weight encoding (Tetris and ours) and flag (ours) and M_{idx} includes the bits for activation selection in the splitters (Tetris and ours).

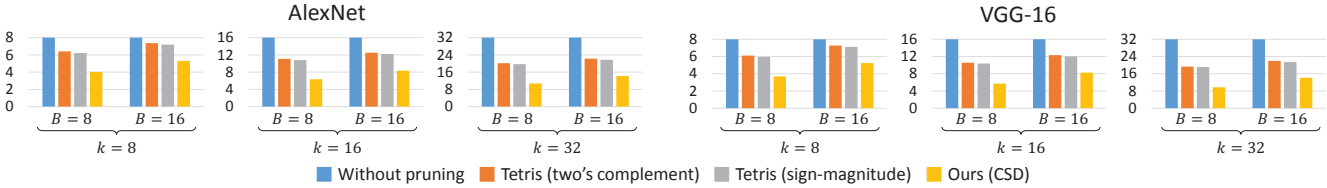

Fig. 9. Comparison of the average of clock cycles performing a set of weights with bit-size $B = 8$ and 16, and pruning stride $k = 8$, 16 and 32.

TABLE I
THE NORMALIZED NUMBER OF ESSENTIAL BITS FOR TWO'S COMPLEMENT, SIGN-MAGNITUDE, AND OUR CSD REPRESENTATIONS

Model	AlexNet		VGG-16	
Bit-size	8-bit	16-bit	8-bit	16-bit
Two's complement	1.000	1.000	1.000	1.000
Sign-magnitude	0.467	0.685	0.433	0.641
Our CSD	0.306	0.477	0.260	0.447

observed a considerable clock cycle reduction, saving up to 70% cycle time over the parallel bit-level multiplication without applying bit-column compression techniques, and 28∼50% cycle time over Tetris.

VI. CONCLUSION

In this work, we proposed an effective technique to resolve the inherent limitation of the bit-level weight pruning: the maximal computation speedup was bounded by the total number of non-zero bits of the weights, but the bound had consistently been considered as 'unoptimizable'. Specifically, based on the notion of canonical signed digit (CSD) encoding, we (1) proposed *a transformation technique* which converted the two's complement representation of every weight into a set of CSD representations of *the minimal or near-minimal number of essential bits*, (2) formulated the problem of selecting CSD representations of weights that *maximized the parallelism of bit-level multiplication* into *a multi-objective shortest path problem* and solved it efficiently, and (3) proposed *a supporting novel acceleration architecture* at no additional non-trivial hardware cost. Through experiments, it was shown that our proposed approach was able to reduce the number of essential bits by 52∼74% on AlexNet and VGG-16, and sped up the inference process by 28∼50% over the existing state-of-the-art accelerator supporting bit-level weight pruning without accuracy loss.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (No. NRF-2017R1E1A1A03070465), Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-TB1703-08, and Samsung Electronics.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in NeurIPS*, 2012.
[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.
[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, real-time object detection. In *CVPR*, 2016.
[6] 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.
[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. DaDianNao: A machine-learning supercomputer. In *MICRO*, 2014.
[8] 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.
[9] 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.
[10] H. Lu, X. Wei, N. Lin, G. Yan, and X. Li. Tetris: Re-architecting convolutional neural network computation for machine learning accelerators. In *ICCAD*, 2018.
[11] M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.
[12] A. Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79, 1987.