TALIPOT: Energy Efficient DNN Booster Employing Hybrid Bit Parallel-Serial Processing in MSB-First Fashion

M.Burak Karadeniz and Mustafa Altun

***Abstract*—We propose a novel hybrid bit parallel-serial processing technique called *TALIPOT* in order to reduce energy consumption of Deep Neural Networks (DNNs). *TALIPOT* works as computation booster and has inborn ability of quality adjusting trade-off between accuracy and energy consumption of DNNs. The core principal of *TALIPOT* is keeping the serial number of bits same in the entire computing process, so the energy is consumed effectively without extra waiting times between inputs and outputs of the computing blocks (adders, multipliers and activation blocks). To achieve this, we implement activation rounding which scales down the accumulation of parallel bits at the output of the hidden layers of DNN. *TALIPOT* utilizes most significant bit first (MSB-first) fashion, so we obtain the most valuable bit information first, between the layers of DNN, which ensures the activation rounding process done accurately and efficiently. Thanks to this method, we optimize operating accuracy/energy point by cutting off bits at the output whenever we obtain the desired accuracy. Simulations using the MNIST and CIFAR-10 datasets show that *TALIPOT* outperforms the state-of-the-art computation techniques in terms of energy consumption. *TALIPOT* performs the MNIST classification with the energy efficiency of 25.3 TOPS/W and the accuracy of 98.2% in ASIC environment using 40 nm CMOS process.**

***Index Terms*—Deep Neural Network (DNN), Hybrid Number Representation (HNR), activation rounding, hardware accelerator, ASIC.**

I. Introduction

RECENT improvements in accuracy and functionality of Deep Neural Networks (DNNs) have gained them increasing popularity in autonomous systems and diagnosis tools. However, these improvements come with a price. Energy consumption of DNNs increases dramatically that requires new techniques to improve their energy efficiency. For this purpose a good number of inference accelerators have been proposed. *DaDianNao* [1] proposes multi-chip framework where each chip is tiled and each tile has its own memory and arithmetic function processor. *Stripe* [2] suggests bit-serial computation to improve energy efficiency over [1]. *UNPU* [3] also proposes bit-serial computation with a modified computation architecture based on lookup tables. Bit-serial computation also motivates the studies [4]–[6] aiming to reduce energy consumption. However, since the increase in latency is more than or almost equal to the decrease in power for these studies, energy consumption is not satisfactorily improved.

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) 1001 projects #116E250 and #119E507 as well as by the Istanbul Technical University BAP project #42435.

M.B. Karadeniz and M. Altun are with Department of Electronics and Communications Engineering, Istanbul Technical University, 34469 Maslak-Istanbul, Turkey. E-mails: karadeniz17@itu.edu.tr; altunmus@itu.edu.tr.

Considering that multiplication operations take an important part of the computation workload, accelerators with energy-efficient multipliers are introduced [7]–[9]. *Vesti* [10] designs a hardware accelerator without conventional multipliers by exploiting in-memory computing. However, its usage is very limited considering that it uses 1-bit (binarized) weight precision. Along with the hardware modification, different number representations or mix precision for multiplicands are used to reduce energy [11]–[14].

Additionally, to reduce energy *Eyeriss v2* [15] proposes a hierarchical mesh network (HM-NoC) consisting of processing elements each of which executes its own computation task by the use of multiply and accumulate (MAC) blocks. Thanks to sparse or compact mapping of processing elements, data movement and energy consumption can be reduced. This strategy is also followed by [16]–[19]. However these approaches necessitate complex control blocks that restrict energy efficiency.

Examining the above studies on energy efficient DNNs in the literature, we see that they mainly use the conventional computing paradigm with an integration of different performance optimization techniques. These techniques include memory partitioning, spatial mapping, energy efficient multiplication, optimization of weight and input precision, bit-serial computation, and MAC-based processing element management. Different from these studies, we use a new computing paradigm with a new number representation.

We propose *TALIPOT*: a novel hybrid bit parallel-serial processing technique to reduce energy consumption of DNNs. There are different architectures which are related to DNNs such as convolutional neural networks (CNNs) and multilayer perceptrons (MLPs) based networks [20], [21]. We develop *TALIPOT* applicable to both CNNs and MLPs in feed-forward process. So, *TALIPOT* can be embedded in a general purpose neural processor. Note that although CNNs have more complex processing flows, they commonly use the computation footprint of the MLPs.

Each layer that comprises MLP has computation workload which is composed of relatively basic operations, i.e., multiplication, addition and activation. As the network is layered, arithmetic operations run in a pipeline (feed-forward) where inputs of one layer wait for the outputs of the previous layer.

Our energy optimization strategy has two main principles. The first one is to optimize latency and the second one is to use energy efficient and simple computing blocks. Examining the literature for energy efficient arithmetic blocks, we see that bit-serial multipliers and adders come forward with their simple and energy efficient structures [2], [3]. However, in terms of

latency and time management, bit-serial systems have burdens.

For example, consider a 2-hidden layer MLP architecture where every layer has 4-bit precision as shown in Fig. 1. Fig. 1(a) depicts the timing diagram of network inputs and outputs which are processed by conventional bit-serial computing in the best case scenario. To process next inputs, every layer needs to wait for the time duration of 4 serial bits which is a waste of time. The latency which has the factor of 12 clock cycles in this example, is dramatically increased by the factor of the number of the hidden layers. In order to relieve this burden, *TALIPOT* uses activation rounding which will be detailed in the following sections. Timing diagram of the implementation of the same network with *TALIPOT* is displayed in Fig. 1(b). Whenever the MSBs of the input of the layer are fetched, they are processed and activated to be used for the next layer. Thanks to activation rounding technique, *TALIPOT* is able to decrease the latency factor from 12 clock cycles to 7 clock cycles in this specific example. Ignoring the fact that the every sequential block in a layer has its own one clock cycle delay, *TALIPOT* keeps the serial number of bits same, which is 4 in this example, in the entire computing process, so the energy is consumed effectively without extra waiting times between inputs and outputs of computing blocks (adders, multipliers, and activation blocks).

Additionally, to optimize delay which is the important part of the latency, we aim to use simple computation blocks especially for the matrix multiplication and the activation rounding with MSB-first processing. So, we obtain the most valuable bit information first to ensure that the activation rounding process is done accurately and efficiently. As opposed to the MSB-first processing, in least significant bit first (LSB-first) processing, activation operation can not be done accurately until receiving the MSB of the input that results in controlling and latency overheads as seen in [3]. As a result, in this study, we contribute to implement energy efficient DNNs. Our main contributions include:

- We have introduced Hybrid Number Representation over Conventional Binary Representation that allows to keep the serial number of bits same for energy efficiency.
- We have implemented new arithmetic blocks as well as a new activation technique that we call activation rounding instead of regular activation.
- We have developed energy-efficient hardware implementation of DNNs with our proposed technique over conventional bit-serial or bit-parallel processing methods.

The rest of the paper is organized as follows. Section II introduces *TALIPOT*. Section III evaluates *TALIPOT* in comparison with the state-of-the-art architectures. Section IV gives experimental results and Section V concludes the paper.

II. *TALIPOT*: Hybrid Bit Processor

A. Hybrid Number Representation

TALIPOT uses Hybrid Number Representation (HNR) as opposed to Conventional Binary Representation (CBR). Consider a decimal number D represented by a CBR using N binary digits where the most significant Nth bit is reserved for the sign operation either in signed or in sign-magnitude

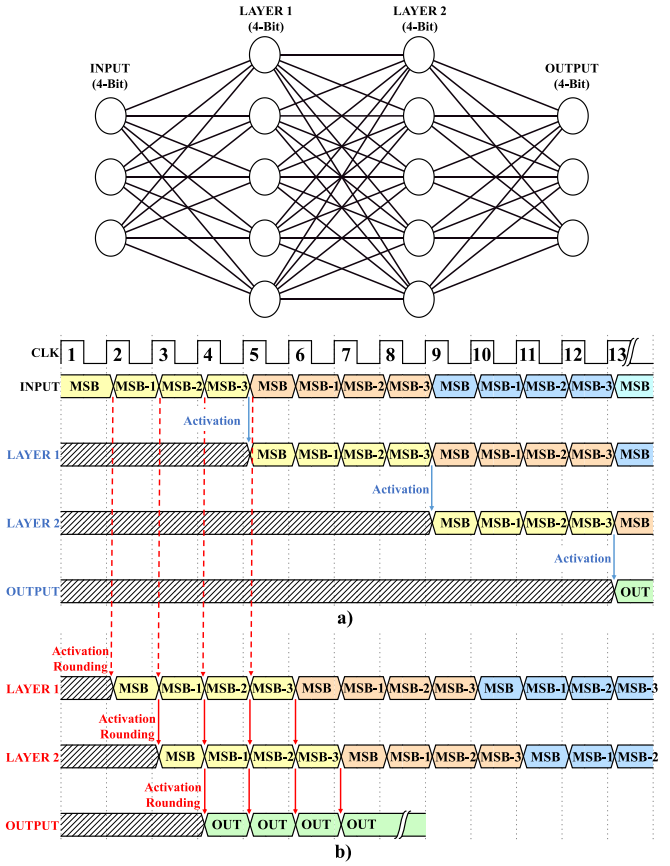

Fig. 1. MLP timing diagram: a) with conventional bit-serial computing, b) with *TALIPOT*.

fashion. Excluding the sign bit, the first $N-1$ binary digits calculate

$$\sum_{k=1}^{N-1} b_k W_k \tag{1}$$

where b_k and W_k are the kth digit and its weight, respectively, and $W_k = 2^{k-1}$.

To represent the same decimal number D, HNR uses a two dimensional matrix $M \times N$ consisting of binary digits, where M and N are the number of the rows and columns, respectively. The most significant Mth row which has N bits is reserved for the sign operation either in signed or in sign-magnitude fashion. Excluding the sign bits, the first $M-1$ rows calculate

$$\sum_{k=1}^{M-1} \sum_{l=1}^{N} b_{(k,l)} W_{(k,l)} \tag{2}$$

where $b_{(k,l)}$ and $W_{(k,l)}$ are the digit in the kth row and lth column and its weight, respectively, and $W_{k,l} = 2^{k+l-1}$.

An example for $D = -42$ is shown in Fig. 2. While Fig. 2(a) and (b) show signed CBR(-42) and HNR(-42), respectively, Fig. 2(c) and (d) show their sign-magnitude versions. To elucidate, lets analyse Fig. 2(d):

$$\begin{aligned} -42 \;=\; & +(8\times 1+4\times 0+2\times 1+1\times 0) \\ & -(16\times 1+8\times 1+4\times 0+2\times 0) \\ & +(32\times 0+16\times 0+8\times 1+4\times 1) \\ & -(64\times 0+32\times 1+16\times 0+8\times 1). \end{aligned}$$

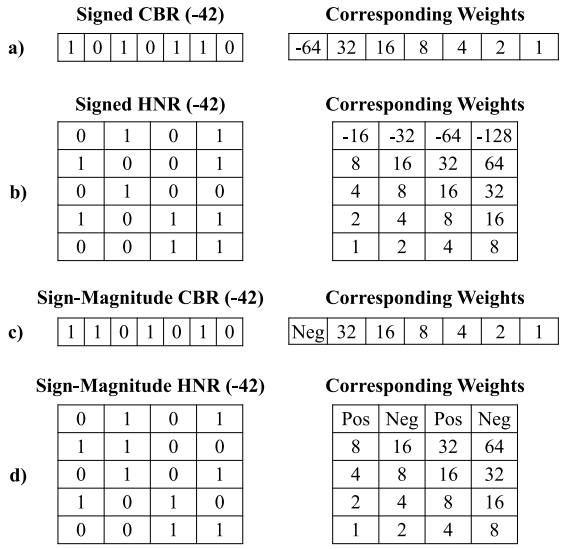

Fig. 2. HNR versus CBR: a) signed CBR, b) signed HNR, c) sign-magnitude CBR, d) sign-magnitude HNR.

Here $+$ and $-$ signs before parenthesis represent *Pos* and *Neg*, respectively, determining that the corresponding column is positive or negative after it is multiplied and summed with its corresponding weights. In our implementations we use binary 1 for *Neg* and binary 0 for *Pos*. One can also calculate D which is represented by HNR with column by column calculation:

$$HNR(D) = \sum_{k=1}^{N} D_{N-k} \times 2^{N-k} \tag{3}$$

where D_k is the decimal value of the kth column as if the column has a weight set of the first column. For Fig. 2(d), starting from the right column, D_k values are -5, 3, -12, and 10, so $-42 = -5 \times 2^3 + 3 \times 2^2 - 12 \times 2^1 + 10 \times 2^0$.

Note that a decimal number can be represented by HNR in different ways, so HNR is a redundant representation while CBR is a unique. This allows us to use different M and N values to represent a decimal number D by satisfying $M_{HNR(D)} + N_{HNR(D)} - 1 \geq N_{CBR(D)}$.

B. TALIPOT - Multiplication

TALIPOT uses constant multiplication in the implementation of a DNN. Inputs are fed in serial bit by bit while weights (synapses) are fed in parallel as constant numbers. To perform constant multiplication, either inputs or constant numbers must be represented by sign-magnitude representations. If the inputs are represented by sign-magnitude digits and the constant numbers are represented by signed ones, then the output of the constant multiplication is in signed HNR.

As an example, consider the constant multiplication of decimal numbers of 11 and -12 as given in Fig. 3. First input, in1, is represented by sign-magnitude CBR and is fed in serial bit by bit. Second input, in2, is represented by signed CBR and is fed in parallel and stationary until all of the digits of in1 are processed. If the MSB and a serial digit of in1 are both binary 1, in2 is inverted and 1 added. These operations are done by the multiplexer. The output is in signed HNR. Fig. 4 displays the same example in which both in1 and in2

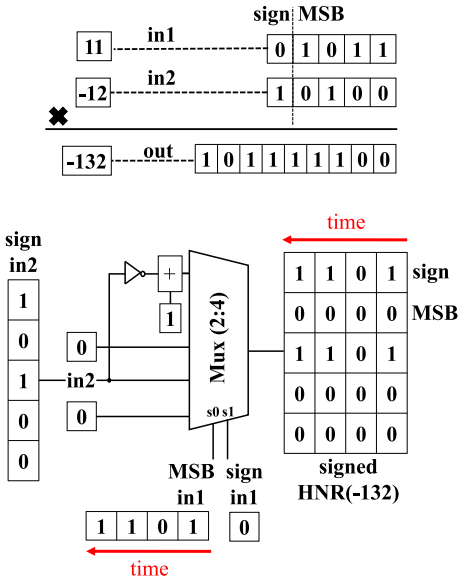

Fig. 3. Constant multiplication of (11) $\times$ (-12) in signed-HNR.

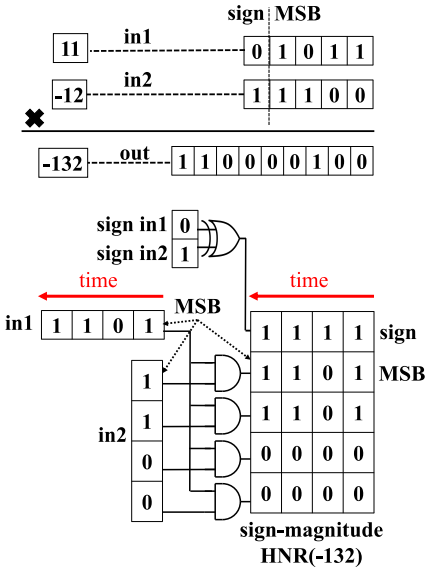

Fig. 4. Constant multiplication of (11) $\times$ (-12) in sign-magnitude HNR.

are represented by sign-magnitude CBR. In order to perform constant multiplication, MSBs of in1 and in2 are bitwise XOR'ed while bits of in2 and a serial bit of in1 are bitwise AND'ed. In this case, the output is in sign-magnitude HNR. Considering both of these examples, output is composed of D_k's (-12, 0, -12, -12) in the direction of time which outputs HNR(-132) according to Equation (3).

C. TALIPOT - Summation

TALIPOT adds two outputs of the multipliers, each of which is represented in HNR format with an $M \times N$ matrix, in parallel column by column. The column size of the inputs and the output are same. Summation of two signed HNR is done with M-bit binary adder. Fig. 5 depicts addition of signed HNR(100) which is composed of D_k's (7, 7, 6, 4) and signed HNR(-30) which is composed of D_k's (-5, 2, 3, -4). Summation of them is done with a 4-bit binary adder with its

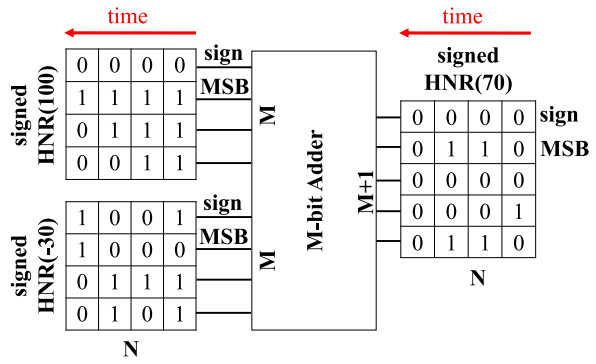

Fig. 5. Summation of signed HNR(100) and signed HNR(-30).

output having D_k's (2, 9, 9, 0) in the direction of time that results in HNR(70) according to Equation (3).

D. TALIPOT - Activation

The activation operation takes place after multiplication and summation. Until activation, N is same as the number of input activation. However, M grows further, having positive correlation with the number of summations or weights in a layer. To be able to directly use activation outputs as inputs for the next layer, M should be 2 (with the sign row) after the activation operation. This is illustrated in Fig. 6 with weight sets. HNR(D) with a weight matrix of $M \times N$ is rounded to HNR(D_r) with a weight matrix of $2 \times N$ after activation. We call this operation as activation rounding. Each column of HNR(D_r), denoted by $(D_{r_k}, k \in 0, 1, 2, ..., N-1)$, has $2\times$ bigger weight than the MSB weight of the corresponding column of the HNR(D), denoted by $(D_k, k \in 0, 1, 2, ..., N-1)$.

It is clear that there is an information lost in activation rounding. The accuracy loss should be minimized for the activation block. Additionally, the block should be energy efficient with relatively small circuit size and latency overhead. By considering these parameters, we develop two different types of activation rounding techniques, called unfolded activation rounding and folded activation rounding. We also categorize folded activation rounding into 1-bit folded activation and n-bit folded activation rounding. Unfolded activation rounding is done column by column, so there is no latency overhead. 1-bit folded activation rounding is done using column pairs with a latency overhead of 1 clock cycle in order to increase accuracy. To achieve further accuracy improvement, n-bit folded activation rounding is used in which n+1 columns are evaluated at the cost of n clock cycles.

We focus on *Tanh* and *ReLU* activation functions specifically, as models of any nonlinear or linear activation functions that are existed in the literature. The proposed activation rounding techniques for these two functions are explained in the following two subsections.

1) Tanh Activation: *Tanh* classifies weighted sums (v) regarding to strengths of their magnitude as

$$Tanh(v) = \frac{2}{1+e^{-2v}} - 1. \tag{4}$$

a) Tanh Unfolded Activation Rounding: Consider an integer number (D) in HNR with $M \times N$ matrix. *TALIPOT* applies *Tanh* activation to D column by column (*Tanha*) as

HNR(D) WEIGHTS ($M \times N$) BEFORE ACTIVATION

D_0	...	...	D_{N-4}	D_{N-3}	D_{N-2}	D_{N-1}
sign	sign	sign	sign	sign	sign	sign
2^{M-1}	2^{M}	2^{M+1}	...	2^{M+N-4}	2^{M+N-3}	2^{M+N-2}
...	...	...	...	...	2^{M+N-4}	2^{M+N-3}
...	...	...	...	...	...	2^{M+N-4}
...	...	...	...	...	...	...
2^{2}	2^{3}	2^{4}	...	2^{N-1}	2^{N}	2^{N+1}
2^{1}	2^{2}	2^{3}	...	2^{N-2}	2^{N-1}	2^{N}
2^{0}	2^{1}	2^{2}	...	2^{N-3}	2^{N-2}	2^{N-1}

ROUNDED HNR (D_r) WEIGHTS ($2 \times N$) AFTER ACTIVATION

D_{r_0}	...	...	$D_{r_{N-4}}$	$D_{r_{N-3}}$	$D_{r_{N-2}}$	$D_{r_{N-1}}$
sign	sign	sign	sign	sign	sign	sign
2^{M}	2^{M+1}	2^{M+2}	...	2^{M+N-3}	2^{M+N-2}	2^{M+N-1}

Fig. 6. Weight sets of the input, HNR(D), before activation and the weight sets of the output, HNR(D_r), after activation.

$$Tanha(D_k) = \begin{cases} 1 & \text{if } D_k \geq TH \\ -1 & \text{if } D_k \leq -TH \\ 0 & \text{if } -TH < D_k < TH \end{cases} \tag{5}$$

where D_k is the decimal value of the kth column of HNR(D), and TH stands for threshold. At the output of the activation rounding, D_r, each column has 3 options $+1$, -1, and 0 due to rounding. Thus, 3 quantization levels are applied to each column while the quantization level (Q) of 15 ($Q = 2^N - 1$) is applied to both sides of HNR(D) (Fig. 7). Here, our main goal is finding the best choice of TH for minimizing the rounding error.

Normalized D_r (D_{r_n}) is defined as

$$D_{r_n} = \sum_{k=1}^{N} 2^{-k} Tanha(D_{N-k}). \tag{6}$$

If Equation (3) is rewritten with *Tanh* activation of normalized decimal value of $HNR(D)$ ($HNR(D)_n$), it is converted to

$$Tanh(HNR(D)_n) = Tanh\left(\frac{\sum_{k=1}^{N} D_{N-k} 2^{N-k}}{2^{2N}}\right) \tag{7}$$

where inputs and weights are quantized by N bits. Equation (7) is approximated by Equation (6) as

$$Tanh\left(\frac{2^{-1}D_{N-1} + 2^{-2}D_{N-2} + ... + \frac{D_0}{2^N}}{2^N}\right) \approx 2^{-1}Tanha(D_{N-1}) + 2^{-2}Tanha(D_{N-2}) + ... + \frac{Tanha(D_0)}{2^N}. \tag{8}$$

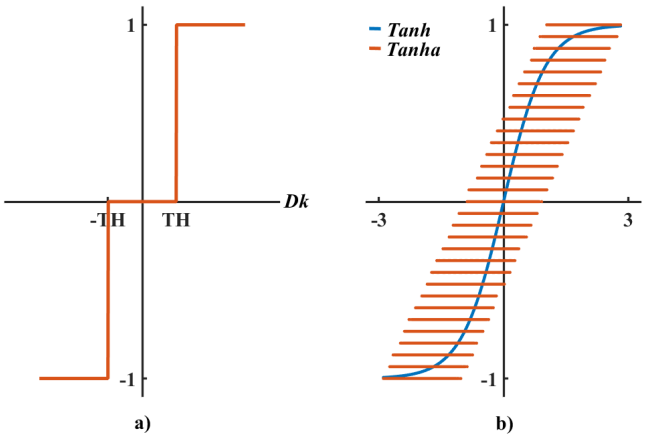

Fig. 7. Unfolded activation rounding (*Tanh* vs *Tanha*): a) column (D_k) activation, b) overall HNR(D) activation ($N = 4$).

The sum of inner products of *Tanh* is defined as

$$Tanh(a+b) = \frac{Tanh(a) + Tanh(b)}{1 + Tanh(a)Tanh(b)}. \quad (9)$$

After implementing Equation (9) into the first and second terms of Equation (8) we have

$$\frac{Tanh(KD_{N-1}) + Tanh(\frac{K}{2}D_{N-2})}{1 + Tanh(KD_{N-1})Tanh(\frac{K}{2}D_{N-2})} \approx 2^{-1}Tanha(D_{N-1}) + 2^{-2}Tanha(D_{N-2}) \quad (10)$$

where $K = 1/2^{N+1} \simeq 1/2Q$. In this situation, Table I is used as activation rounding scheme. Here UB and LB stand for upper bound and lower bound of *TH*, respectively. In order to approximate right-hand side (RHS) of Equation (10) to the left-hand side (LHS), following conditions are considered for rounding:

If $KD_{N-1} \simeq 0$ and $KD_{N-2} \simeq 0$, then $D_{N-1} \lesssim Q/N$ and $D_{N-2} \lesssim Q/N$. LHS of Equation (10) is 0. In order not to activate both terms of RHS, *TH* should be set at $\geq Q/N$.

If $KD_{N-1} \simeq 0.5$ and $KD_{N-2} \simeq 0$, then $D_{N-1} \simeq Q$ and $D_{N-2} \lesssim Q/N$. LHS of Equation (10) is 0.46. In order to activate first term but not the second term of the RHS, *TH* should be set at $\geq Q/N$.

If $KD_{N-1} \simeq 1$ and $KD_{N-2} \simeq 0$, then $D_{N-1} \simeq 2Q$ and $D_{N-2} \lesssim Q/N$. LHS of Equation (10) is 0.76. In order to activate both terms of RHS, *TH* should be set at $\leq Q/N$.

The other conditions are handled in the same as seen in Table I. *TALIPOT*'s overall strategy is to minimize mean square error (MSE) between *Tanha* and *Tanh*. To do so, TH is needed to be calibrated. Calibration process is done right after training. Deducing from the above-mentioned cases *TALIPOT* starts with TH as intersection of UB and LB as Q/N. Once calibration process is done, TH parameter is fixed in the feed-forward DNN.

b) Tanh 1-Bit Folded Activation Rounding: Consider a signed decimal (D) represented by HNR with $M \times N$ matrix. Revisiting Fig. 6, each column pairs of HNR(D) are rounded to each column pairs of HNR(D_r). We call this operation as 1-bit folded activation rounding which means that the data

TABLE I
UNFOLDED ACTIVATION ROUNDING SCHEME

KD_{N-1}	KD_{N-2}	D_{N-1}	D_{N-2}	TH UB	TH LB	LHS	RHS
0	0	<Q/N	<Q/N	-	Q/N	0	0
0	0.5	<Q/N	Q	-	Q/N	0.24	0.25
0	1	<Q/N	2Q	-	Q/N	0.46	0.25
0.5	0	Q	<Q/N	-	Q/N	0.46	0.5
0.5	0.5	Q	Q	Q	-	0.64	0.75
0.5	1	Q	2Q	Q	-	0.76	0.75
1	0	2Q	<Q/N	Q/N	-	0.76	0.75
1	0.5	2Q	Q	Q	-	0.85	0.75
1	1	2Q	2Q	2Q	-	0.91	0.75
						MSE	0.01

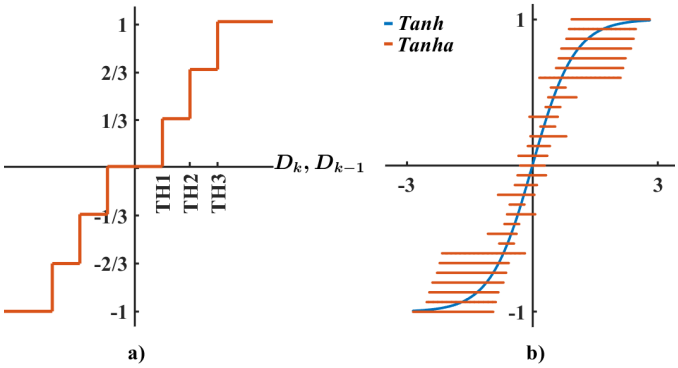

Fig. 8. 1-Bit folded activation rounding (*Tanh* vs *Tanha*): a) column pairs (D_k and D_{k-1}) activation, b) overall HNR(D) activation ($N = 4$).

of high order column, e.g., D_{N-1}, of HNR(D) are shifted and summed with low order adjacent column, e.g., D_{N-2}. So, Equation (3) is rewritten as

$$\begin{aligned} HNR(D) &= 2^{N-2}(2 \times D_{N-1} + D_{N-2}) \\ &\quad + 2^{N-4}(2 \times D_{N-3} + D_{N-4}) \\ &\quad + ... + D_0. \end{aligned} \quad (11)$$

Seven quantization levels $(1, \frac{2}{3}, \frac{1}{3}, 0, -\frac{1}{3}, -\frac{2}{3}, -1)$ are applied to each column pair. Fig. 8 shows that when column-wise quantization level increases, *Tanha* gets sharp in *Tanh*. Truth table of 1-bit folded activation rounding is detailed in Table II. When 1-bit folding operation is implemented for each double columns of HNR(D), two times magnitude of high-order D_{N-1} term is summed with low-order D_{N-2} term of Equation (11), so does Q. Thus, 3Q is set as UB for TH while 0 is set as LB.

c) Tanh n-Bit Folded Activation Rounding: HNR(D) represented by $M \times N$ matrix can be folded $N - 1$ times. While number of folding bits $(n) \rightarrow N-1$; $Tanha \rightarrow Tanh$. UB and LB of *TH* are set to

$$UB(TH_k) = kQ \quad (12)$$
$$LB(TH_k) = (k-1)Q \quad (13)$$

in an n-bit folded HNR where $k \in \mathbb{Z}^+ \rightarrow 1, 2, 3, ..., 2^{n+1} - 1$. TH_1 has special LB as Q/N which is detailed before.

2) ReLU Activation: *ReLU* filters positive weighted sums while cancelling out negative ones:

$$Relu(v) = \begin{cases} v & if \quad v \geq 0 \\ 0 & else. \end{cases} \quad (14)$$

TALIPOT applies *ReLU* activation function in two stages. In the first stage, the weighted sum which is constructed in

TABLE II
1-BIT FOLDED ACTIVATION ROUNDING TRUTH TABLE

D_{rN-1}	D_{rN-2}	*Tanha*	TH	UB	LB
1	1	1	TH3	3Q	2Q
1	0	2/3	TH2	2Q	1Q
0	1	1/3	TH1	1Q	Q/N
0	0	0	0	Q/N	0
-1	-1	-1	-TH3	-2Q	-3Q
-1	0	-2/3	-TH2	-1Q	-2Q
0	-1	-1/3	-TH1	-Q/N	-1Q
0	0	0	0	0	-Q/N

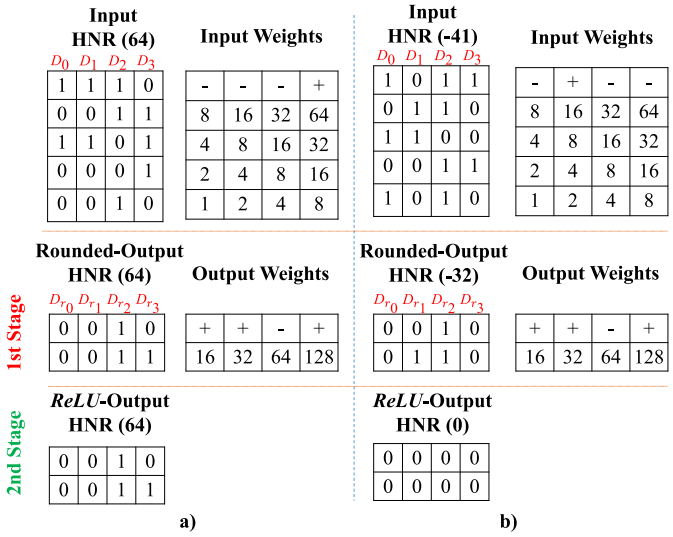

Fig. 9. *TALIPOT ReLU* activation rounding: a) input HNR(64), b) input HNR(-41).

HNR with $M \times N$ format is structured to HNR with $2 \times N$ format by rounding. In the second stage, *TALIPOT* applies *ReLU* activation bit by bit. Both of the stages take place simultaneously.

Consider an example of inputs of sign-magnitude HNR(64) and sign-magnitude HNR(-41) in Fig. 9(a) and (b) respectively. Both are represented with $(M = 5) \times (N = 4)$ matrix where first row is reserved for sign bits. In the first stage, both of inputs are rounded such that

$$D_{r_k} = \begin{cases} 1 & \text{if } D_k \geq TH \\ -1 & \text{if } D_k \leq -TH \\ 0 & \text{if } -TH < D_k < TH \end{cases} \quad (15)$$

where D_k and TH are decimal value of column k and rounding threshold, respectively. D_{r_k} is the rounded-output. In this example TH is set as the magnitude of MSBs of each D_k which is $2^{M-2} = 8$. HNR(64) is composed of D_k's (+14, -9, -4, -4) and rounded to HNR(64) in the first stage according to Equation (15). On the other hand, HNR(-41) in Fig. 9(b), is rounded to HNR(-32) in the same manner. In order to apply *ReLU* in the second stage Algorithm I is addressed. Since *TALIPOT* uses MSB-first format, the input can be defined whether it is positive or negative from the first clock cycle or whenever the positive(pos) or negative(neg) flag is raised.

a) ReLU Unfolded Activation Rounding: Recall from Equation (8), $HNR(D)_n$ is approximated by D_{r_n} as

Algorithm 1 *TALIPOT ReLU* Activation

```
procedure RELU out(in(D_{r_k}))
    for k = N → 1 do
        if (in(k) = 1)&(pos, neg = 0) then
            out(k) ← 1
            pos ← 1                    ▷ positive flag is raised
            neg ← 0
        else if (in(k) = −1)&(neg, pos = 0) then
            out(k) ← 0
            pos ← 0
            neg ← 1                    ▷ negative flag is raised
        else if (pos = 1) then
            out(k) ← in(k)
        else if (neg = 1) then
            out(k) ← 0
        else
            out(k) ← 0
            pos ← 0
            neg ← 0
        end if
    end for
end procedure
```

$$HNR(D)_n = \frac{\sum_{k=1}^{N} D_{N-k} 2^{N-k}}{2^{2N}} \simeq \sum_{k=1}^{N} D_{r_{N-k}} 2^{-k}. \quad (16)$$

Equation (16) is rewritten as

$$\frac{D_{N-k}}{Q} \simeq D_{r_{N-k}}. \quad (17)$$

Considering Equation (15) and Equation (17) together, *TALIPOT* applies activation rounding in a DNN such that

$$D_{r_{N-k}} = \begin{cases} 1 & \text{if } \frac{D_{N-k}}{Q} \geq mean(abs(D_x)) \\ -1 & \text{if } \frac{D_{N-k}}{Q} \leq -mean(abs(D_x)) \\ 0 & \text{if } \frac{D_{N-k}}{Q} > -mean(abs(D_x)) \\ & \text{and } \frac{D_{N-k}}{Q} < mean(abs(D_x)) \end{cases} \quad (18)$$

where $mean(abs(D_x))$ is the mean absolute of D_k's in the xth layer of DNN. To find mean absolute, D_x is written as

$$D_x = \sum_{m=1}^{L_{x+1}} \sum_{k=1}^{L_x} IN_{x_k} W_{x_{k+L_x(m-1)}} \quad (19)$$

where L_x and L_{x+1} are the number of neurons in the xth and the adjacent layer of the DNN respectively. IN_x is the input bits of xth layer which are consist of logic $[1, 0]$ while W_x is the weights of the xth layer. Equation (19) is approximated to

$$D_x \simeq mean\,(IN_x) \times \sum W_x. \quad (20)$$

If we implement Equation (20) into Equation (18), we get

$$D_{r_{N-k}} = \begin{cases} 1 & \text{if } D_{N-k} \geq TH_x \\ -1 & \text{if } D_{N-k} \leq -TH_x \\ 0 & \text{if } -TH_x < D_{N-k} < TH_x \end{cases} \quad (21)$$

where $TH_x = mean(IN_x)mean(abs(W_x))Q$. *TALIPOT* applies different TH for different layer of DNN since every

layer has different first term ($mean(IN_x)$). Second term ($mean(abs(W_x))Q$) is calculated for each layer of the DNN right after the training is done. First term ($mean(IN_x)$) needs to be adjusted between layers relatively. Firstly, *TALIPOT* assumes the mean inputs of the first layer of DNN ($mean(IN_1)$) as 1. Then, it correlates the mean inputs of xth layer ($mean(IN_x)$) to the mean inputs of the first layer accordingly to their sizes, literally the number of neurons that they have. *TALIPOT* states TH of xth layer (TH_x) of DNN as

$$TH_x = \frac{L_x}{L_1} mean(abs(W_x))Q \tag{22}$$

where L_x is the number of neurons in the xth layer.

b) ReLU 1-Bit or n-Bit Folded Activation Rounding: Aforementioned folding strategy is also considered for *ReLU* activation. TALIPOT states TH of the xth layer of the DNN in folded activation as

$$TH_{x_k} = k\frac{L_x}{L_1} mean(abs(W_x))Q \tag{23}$$

where $k \in \mathbb{Z}^+ \rightarrow 1, 2, ..., 2^{n+1} - 1$.

III. Design and Evaluation of TALIPOT

We develop tile-based MLP architectures both for *TALIPOT* and other compared techniques as follows. Additionally, in the last subsection, we discuss how to apply *TALIPOT* on a CNN architecture.

A. Driving Example

Consider a DNN chip having tile-based organization in Fig. 10(a). Fig. 10(b) shows tile structure where input buffer (IB) and weight buffer (WB) are processed by Arithmetic Process Unit (APU) in order to evaluate output buffer (OB). Consider that, IB and WB have 16 neurons and 256 weights, respectively, each of which has 8-bit precisions. In every processing cycle, APU is dedicated to multiply input neurons by corresponding weights, producing 256 (16 group of 16) products as shown in Fig. 10(c). Each group of products is summed within the group, producing 16 weighted sum products. Then, each weighted sum product is activated and evaluated as output neuron.

B. Baseline DNN Accelerator Tiles

a) DaDianNao Tile: DaDianNao [1] is a custom multi-chip architecture (supercomputer) which is designed for reducing energy and accelerating DNN comparing to GPU. Block diagram of *DaDianNao* tile APU is given in Fig. 11. Referring the aforementioned driving example, *DaDianNao* fetches 16 input neurons in parallel from the IB and multiplies them by corresponding weights of WB, producing 256 (16 group of 16) 16-bit products. Each group of products is summed within the group, producing 16 20-bit weighted sum products. Each weighted sum product is truncated to an 8-bit data followed by the activation; 16 8-bit output neurons are then buffered to the OB.

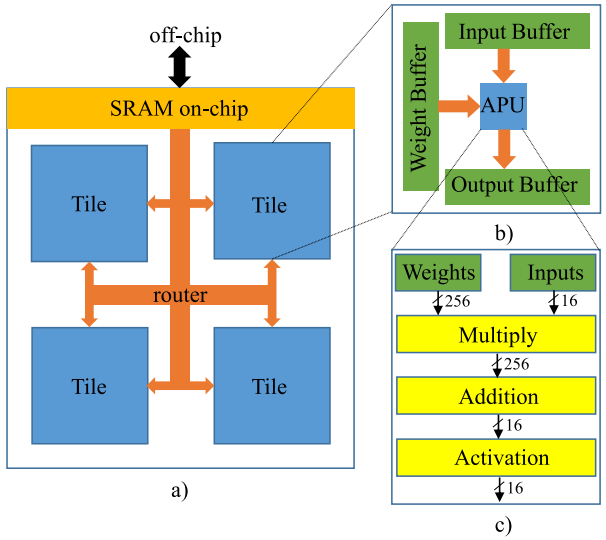

Fig. 10. DNN chip: a) chip architecture, b) tile architecture, c) APU architecture.

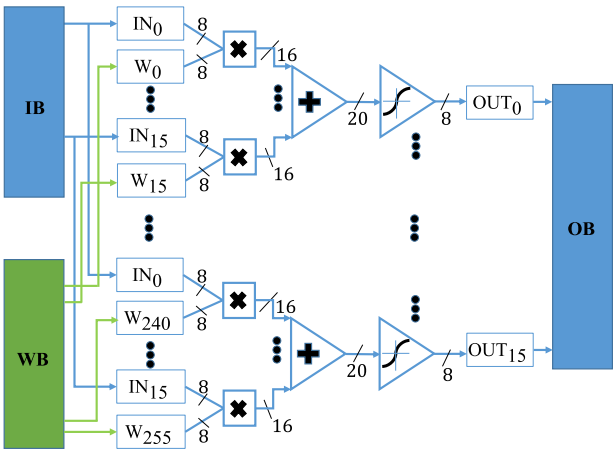

Fig. 11. *DaDianNao* APU.

b) STRIPE Tile: STRIPE [2] is a bit-serial DNN accelerator to enhance performance of *DaDianNao*. Block diagram of the tile APU is given in Fig. 12. Inputs are fed in serial in MSB-first format and weights are fetched in parallel. In each clock cycle 16 bits from IB and 256 8-bit weights from the WB perform constant multiplication, producing 256 8-bit partial products. While constant multiplication takes places, shift-add operations are performed to accumulate partial products. After 8 clock cycles, total of 2048 8-bit partial products are reduced to 16 20-bit weighted sum products. Each weighted sum product is truncated to an 8-bit data followed by the activation; 16 8-bit output neurons are then buffered to the OB.

c) MAC Tile: There are quite a bit DNN accelerators that use MAC tile [8], [11], [20]. Block diagram of tile APU is given in Fig. 13. *MAC* APU is a state machine that orders arithmetic operations. In the first cycle, an input from the IB is multiplied by the corresponding weight. The produced 16-bit product is saved to be accumulated. After 16 clock cycles, 16 16-bit products are summed in a 20-bit accumulator. The weighted sum is truncated to 8-bit data followed by the activation. Then, it becomes the first output of the OB. APU

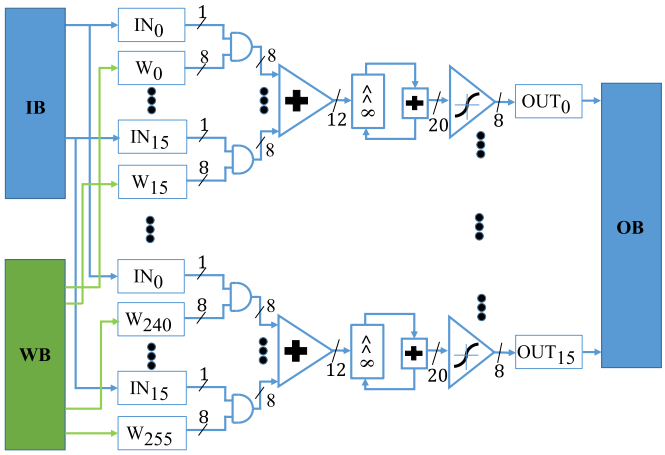

Fig. 12. *STRIPE* APU.

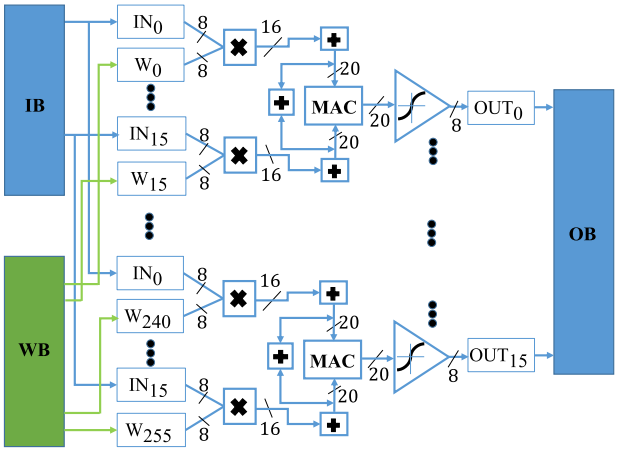

Fig. 13. *MAC* APU.

resets operations after each output neuron is evaluated until all of the outputs are produced.

C. TALIPOT Tile

TALIPOT system architecture is shown in Fig. 14. Fig. 14(a) shows its data and time flow. Fig. 14(b) shows the tile APU in the unfolded activation rounding mode. Referring the driving example, APU is dedicated to perform constant multiplication of the MSB buffer with the corresponding weights in WB, producing in total of 256 8-bit terms. Right after the summation, all of the terms are reduced to 16 12-bit products. *TALIPOT* implements activation rounding in real-time which means that whenever the inputs from the input MSB buffer are fetched at time t_k, it immediately activates and buffers them to the output MSB buffer in the next clock cycle at time t_{k+1}. After the activation rounding, 16 12-bit products are reduced to their rounded versions of 16 1-bit form and buffered directly to the next layer to be processed.

Fig. 14(c) shows the tile APU in the 1-bit folded activation rounding mode. In this mode, the MSB and the second MSB (MSB-1) buffers are processed sequentially. Whenever the inputs from the MSB buffer are fetched at time t_k, it activates and buffers them to the output MSB buffer after the next clock cycle at time t_{k+2}. If we reiterate the same example, after the MSB buffer is processed which produces 16 12-bit products, *TALIPOT* APU waits one clock cycle to process the second MSB buffer. Outputs of these two operations are combined by a 1-bit shift adder which produces 16 13-bit products. By doing so, *TALIPOT* performs 1-bit folded activation rounding. At the end, 16 13-bit products are reduced to their rounded versions of 16 2-bit form (MSB and MSB-1) and buffered to next layer to be processed. After 8 clock cycles, all of the output bits are produced and buffered.

D. MLP Architecture and Design Evaluation

Consider a fully connected feed-forward MLP having k layers, each of which has L neurons with P trainable parameters as illustrated in Fig. 14(a). Neuron inputs and weights in each layer have different precision of N and M, respectively. *TALIPOT* starts with buffering input data to the input buffer. Here the data is registered in CBR. The blocks are colored according to whether they operate in CBR or HNR format.

After buffering MSBs of the input layer, *TALIPOT* performs tile operations (constant multiplication, summation, and activation) in HNR format. The data, stored in MSB buffers and main memory block, is in CBR format. Here there are two strategies available for *TALIPOT*. The first strategy is to maximize the tile size (size of batching) so that the MSBs can be fetched and processed in one clock cycle and get activated in the next clock cycle, or after next clock cycle if *TALIPOT* operates in 1-bit folded mode. This strategy brings about the maximum energy efficiency since the latency is minimized and also there are no partial products need to be saved in the memory, thus cutting down the memory power budget. This strategy is also reasonable as a result of relaxed computation bandwidth, thanks to $TALIPOT$'s activation rounding method, by $\frac{1}{N}$ where N is defined as the input precision in each layer. Referring the aforementioned driving example, Table III summarizes tile operations of designs until the output buffer is evaluated. The operations are counted regarding to parallel multiplications, parallel summations, accumulation(acc) and activation(act). *TALIPOT-U* and *TALIPOT-F1* stand for *TALIPOT*'s unfolded mode and 1-bit folded mode, respectively. While state-of-the-art designs need to fetch 128 (16×8) bits to be done with the operation, *TALIPOT* needs to fetch 16 bits or 32 bits accordingly to operating in unfolding or 1-bit folding mode, respectively, in order to begin processing next layer. In this strategy, *TALIPOT* fully supports single instruction for multiple data (SIMD) flow until the output bits are generated at full precision. However, since the output bits are produced in real-time in MSB format, they can be cut-off to satisfy a desired accuracy.

The second strategy is to batch input MSBs as much as the tile structure allows. So, a number of MSBs are fetched and processed. The produced partial products are saved to memory buffer until all of the MSBs are fetched. Once the MSB-fetching process is done, $TALIPOT$ applies rounded activation in the next clock cycle. Consider a two hidden layer MLP where every layer has 32 neurons (32-32-32-32). Using the aforementioned tile structure (16×16 with 8-bit precision) in every layer, $TALIPOT$ spends 4 clock cycles ($\frac{32\times32}{16\times16}$) to be done with the layer and starts to process next layer, whereas

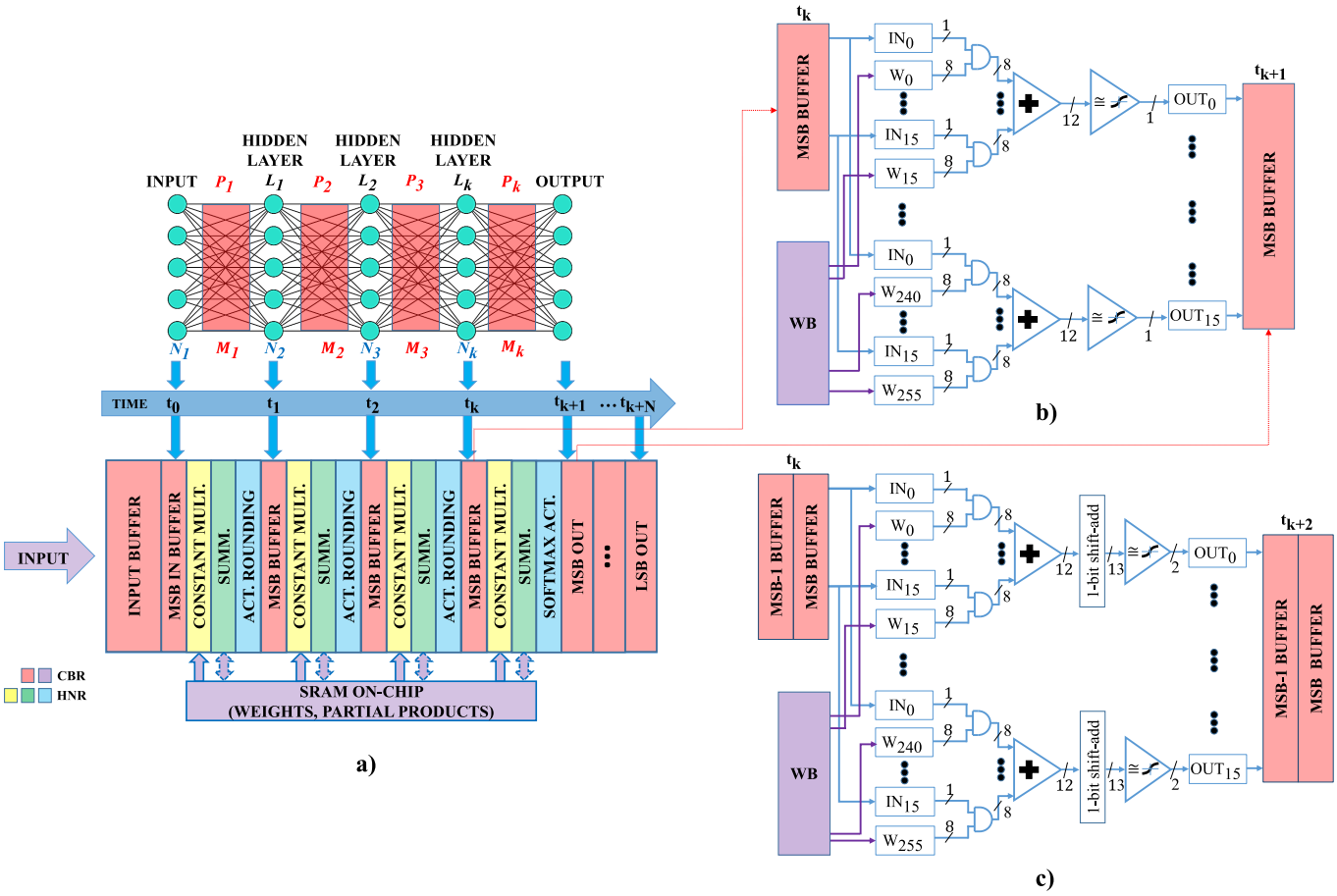

Fig. 14. *TALIPOT* system architecture: a) *TALIPOT* MLP data flow and time diagram, b) *TALIPOT* APU in unfolded activation rounding mode, c) *TALIPOT* APU in 1-bit folded activation rounding mode.

TABLE III
NUMBER OF TILE OPERATIONS UNTIL THE OUTPUT IS GENERATED

TILE 16 × 16 N=8, M=8	**Parallel Mult.**	**Parallel Summ.**	**Acc.**	**Act.**
MAC	256 8-bit	None	16 20-bit	16 8-bit
DaDianNao	256 8-bit	256 16-bit	None	16 8-bit
STRIPE	None	2048 8-bit	16 20-bit	16 8-bit
TALIPOT-U	None	256 8-bit	None	16 1-bit
TALIPOT-F1	None	512 8-bit	16 13-bit	16 2-bit

conventional bit-serial methods spend 32 (4×8) clock cycles in each layer. In this topology, $TALIPOT$ has a latency factor of 40 ($4\times8+4+4$) clock cycles. On the other hand, conventional bit-serial methods have a latency factor of 96 (32×3) clock cycles. So, about 2.4× ($\frac{96}{40}$) energy efficiency likely to be achieved over conventional methods.

E. CNN Architecture and Design Evaluation

TALIPOT's booster option is valid for the single or multi-cascaded convolutional layers of the CNNs. Since the size of the inputs and the filters is massive, the acceleration hardware needs to scale down the matrix multiplication. Consider a single cascaded convolutional layers (CONV1 and CONV2) of the CNN architecture in Fig. 15. The inputs of CONV1 have the dimension of the I_x and I_y in the direction of x and y, respectively, with the number of I_z channels. The kernel of F_x and F_y is applied to extract O_z output features each of which has the dimension of O_x and O_y. The output of CONV1 is followed by a pooling layer to be processed as inputs for CONV2. The kernel of FF_x and FF_y is applied to extract OO_z output features each of which has the dimension of OO_x and OO_y. For CONV1, the input vector with $F_x \times F_y \times I_z$ nodes is used to generate the output vector with $O_x \times O_y \times O_z$ nodes. There are total number of $F_x \times F_y \times I_z \times O_x \times O_y \times O_z$ matrix multiplications in CONV1. The number of matrix multiplications of CONV2 is calculated in the same way. In order to meet the maximum applicable bandwidth requirement of the hardware (BW_{max}), input and output vectors of CONV1 are scaled down by T_A and T_B, respectively, such that $BW_1 \geq \frac{F_x \times F_y \times I_z \times O_x \times O_y \times O_z}{T_A \times T_B} \times N$ where T_A and T_B are scale factors and BW_1 is the memory bandwidth of CONV1 meaning that number of BW_1 bits are cached per cycle in layer CONV1 of which weights have N-bit precision. Input and output vectors of CONV2 are scaled down by T_{AA} and T_{BB}, respectively. *TALIPOT*'s boosting strategy on cascaded convolutional layers of the CNN is to find the best scaling factors of T_A, T_B, T_{AA} and T_{BB} where $T_A \times T_B \cong T_{AA} \times T_{BB}$ to ensure $BW_{max} \geq BW_1 + BW_2$. For example, if $T_A \times T_B \gg T_{AA} \times T_{BB}$ or $T_A \times T_B \ll T_{AA} \times T_{BB}$ there is small boosting option for *TALIPOT* since the overall computation latency will be dominated by the greater side of the equation

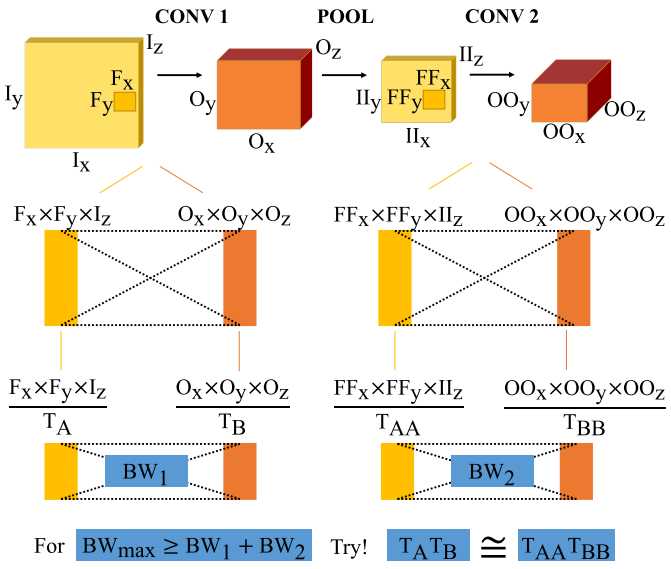

Fig. 15. *TALIPOT*'s boosting strategy for cascaded convolutional layers of the CNN.

(layer). Consider that each of CONV1 and CONV2 has N-bit precision. The latency of the CONV1 and CONV2 has the factor of $(N \times T_A \times T_B) + (N \times T_{AA} \times T_{BB})$ in conventional bit-serial design. On the other hand, if the aforementioned condition can be met, *TALIPOT* processes the same cascaded network with a latency factor of $(N \times T_A \times T_B) + (T_{AA} \times T_{BB})$, thus boosting the cascaded network by $2N/(N+1)$.

TALIPOT employs MSB-first computing, so boosting strategy is valid through the cascaded convolutional layers of CNN as individually or conjointly such that *TALIPOT* can handle the CONV1 and CONV2 at once and then, CONV3 and CONV4 at once, or CONV1, CONV2, CONV3 and CONV4 at once. So, as long as the predefined boosting condition is met, the overall boosting factor of *TALIPOT* in CNN design can be defined as $\frac{N \times k}{N + k - 1}$ where k is defined as the number of convolutional layers into which *TALIPOT* is applied.

IV. Experimental Results

TALIPOT and the state-of-the-arts are compared using MNIST [22] and CIFAR-10 [27] datasets. In our implementations, we consider both constant and variable weights. When the weights are constant, they are embedded into the network as wires. When they are variable, they are fed into the network as inputs from memory which allows future weight updates. We consider six different performance metrics for evaluations: Area (A), Latency (L), Power Consumption (P), Maximum Frequency (F), Misclassification Rate (MR), and Energy Consumption (E) per inference.

A. Methodology

1) Training and Topology: For training, we use TensorFlow. For MNIST, 6 different MLP topologies including 784-50-10, 784-16-16-16-10, 784-32-32-32-10, 784-64-64-64-10, 784-128-128-128-10, 784-256-256-256-10 and LeNet [22] architecture are implemented.For CIFAR-10, we use VGG model of 6 convolutional layers.

2) Quantization: 4-bit and 8-bit precisions are applied for weights and inputs. 4-bit quantization generally meets the need, but 8-bit quantization is done for better accuracy.

3) Source Code and Memory: System Verilog is used for implementations. Circuits are complied and simulated in Vivado Design Suite. SRAM on chip is used for designs to store weights.

4) Synthesis and Performance Measurements: Cadence GENUS low-power design flow is used for performance estimation. For accurate power estimation, backward annotated switching activity file is included. TSMC 40nm library is used for gate-level analysis. GENUS optimizes the power estimation in each iteration of design flow loop with newly synthesized file until the power constraint is met. Therefore, the power is estimated from the first iteration for not violating the estimations from the post-layout optimization.

5) TALIPOT Modes: TALIPOT produces output bits in serial. It means that output bits can be cut-off for better latency and relatively for better energy consumption at the cost of accuracy. Some of *TALIPOT*'s operating modes are stated below:

a) TALIPOT-U: unfolded activation rounding w/o cut-off,

b) TALIPOT-UC: unfolded activation rounding w/ cut-off,

c) TALIPOT-F1: 1-bit folded activation rounding w/o cut-off,

d) TALIPOT-F2: 2-bit folded activation rounding w/o cut-off.

B. Implementation and Results

We compare the performance of the state-of-the-arts which we mentioned in Section III with our proposed method, *TALIPOT*. We use a scaled network of MNIST (784-50-10) for comparison. By doing so, we aim to test and compare optimal performance of methods in the same league. As discussed in section III-D, *TALIPOT*'s first strategy is using the largest size of batching, specially for inputs of MLP, to utilize data reuse and to avoid energy consuming memory read/write operations. To keep up with *TALIPOT*, state-of-the-arts will need to use same size of batching, otherwise energy consumption of them becomes even worse comparing with *TALIPOT*. However, operating on relatively larger networks, this strategy is hardly available for the state-of-the-arts as computation bandwidth needed to process layers can go beyond practical limits of the hardware.

Tables IV and V show detailed performance analysis of the DNN accelerators for aforementioned networks; Fig. 16 summarizes them according to the energy efficiency.

We consider the same largest batch size for all designs with different tile structures as detailed in Section III-B in our comparison. This is done to maximize the energy efficiency, so the reported maximum frequencies of the methods are decreased according to the increased delay as a result of increased bandwidth. *TALIPOT* increases energy efficiency up to 62× over *MAC*, and 3× over *DaDianNao* and *STRIPE* at best. While *TALIPOT* has better energy efficiency in all cases over *STRIPE* and better energy efficiency over *DaDianNao*

TABLE IV
PERFORMANCE OF DNN ACCELERATORS FOR MNIST WITH 4-BIT PRECISION

	DNN Accelerator	Constant Weight A (mm^2)	L (ns)	P (mW)	F (Mhz)	MR (%)	E (nJ)	Variable Weight A (mm^2)	L (ns)	P (mW)	F (Mhz)	MR (%)	E (nJ)
MNIST 784-50-10 (*N*=4)	*DaDianNao*	2.18	50	214.7	57	4.1	10.7	5.7	50	377.1	44	4.1	18.9
	MAC	N/A	N/A	N/A	N/A	N/A	N/A	0.03	1985000	0.113	275	4.1	224
	STRIPE	0.75	500	21.4	122	4.1	10.7	1.52	500	61	53	4.1	30.5
	TALIPOT-F1	0.75	350	14.8	139	4.3	5.2	1.51	350	35	55	4.3	12.5
	TALIPOT-U	0.74	300	12	139	4.6	3.6	1.5	300	31.6	55	4.6	9.5

TABLE V
PERFORMANCE OF DNN ACCELERATORS FOR MNIST WITH 8-BIT PRECISION

	DNN Accelerator	Constant Weight A (mm^2)	L (ns)	P (mW)	F (Mhz)	MR (%)	E (nJ)	Variable Weight A (mm^2)	L (ns)	P (mW)	F (Mhz)	MR (%)	E (nJ)
MNIST 784-50-10 (*N*=8)	*DaDianNao*	6.4	50	768	37	3.7	38.4	16	50	1152	25	3.7	58
	MAC	N/A	N/A	N/A	N/A	N/A	N/A	0.056	1985000	0.206	263	3.7	408
	STRIPE	1.53	900	31.6	104	3.7	28.4	2.92	900	131	45	3.7	118
	TALIPOT-F1	1.51	550	40.5	112	3.7	22.3	2.91	550	150	49	3.7	83
	TALIPOT-U	1.5	500	35.6	112	4.1	17.8	2.9	500	131	49	4.1	66

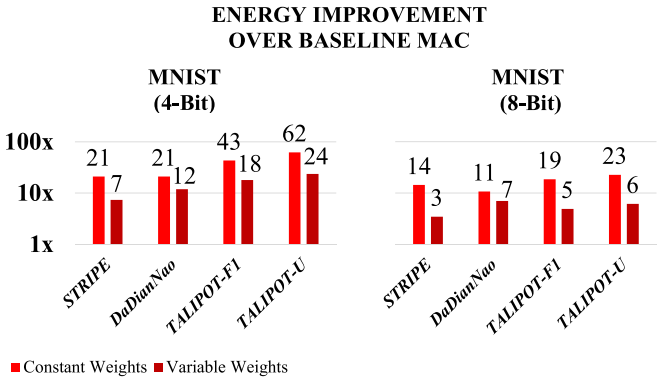

Fig. 16. Energy improvement of *TALIPOT* and the state-of-the-arts over baseline MAC.

on average, there are few cases that *DaDianNao* performs better with up to 1.4× larger energy efficiency. Note that *DaDianNao* has 4× area overhead on average to keep up with *TALIPOT*'s energy efficiency. This overhead is due to the batch size of inputs that is used same for all of the networks. While *TALIPOT* and *STRIPE* process inputs in serial, *DaDianNao* processes them all in parallel. However, *TALIPOT-U* and *TALIPOT-F1* have reduced-latency superiority over *STRIPE* which brings energy efficiency. On the other hand, *MAC* based MLP design falls behind others in terms of energy due to control and latency overhead that it has.

In order to test performance of *TALIPOT* in a more compelling workload, we considered DNN with 3-hidden layers with doubled neuron size in each layer from 16 to 256. We aim to reveal minimum energy point versus accuracy of *TALIPOT* in Table VI. Two different activation functions of *Tanh* and *ReLU*, and three different activation techniques of *U*, *UC*, and *F1* are used. We use 784-bit input MSB buffer and 3 256-bit MSB buffers for the hidden layers of the network which has topology of 784-256-256-256-10. *TALIPOT*'s computation bandwidth is roughly 1.5 Mb/cycle for one time batching of MSB buffers (the largest batch size available in this topology) when inputs and weights have 4-bit precision. This is doubled when *TALIPOT* uses 1-bit folded rounding activation.

We see that in general, *Tanh* performs better than *ReLU* in terms of accuracy in relatively small networks. However, with the increased size of neurons and heavy workload, *TALIPOT*'s unfolded activation rounding achieves the optimal accuracy/energy performance which eliminates the use of 1-bit folded activation rounding mode (*TALIPOT-F1*) and thus, avoids the latency overhead which limits the energy efficiency.

Fig. 17 collects reported MNIST ASIC implementations in the literature in terms of energy consumption and accuracy. Here, among digital MLPs that use similar training techniques and architectures, *TALIPOT* is superior in terms of energy consumption, e.g., 3× better than the nearest one. If we apply technology scaling as 16nm : 28nm : 40nm to 1 : 2.43 : 3.66, *TALIPOT*'s energy efficiency becomes more distinct. Also, *TALIPOT* operates in similar energy points with MLP IMC (In-Memory Computing) [10] with slight accuracy loss for the worst case scenario.

More detailed comparisons among digital MLPs are given in Table VII. Here, *TALIPOT* performs a single inference with 4 clock cycles at a maximum frequency of 0.111 GHz within the 784-256-256-256-10 topology, so 27.8M inferences per second is achieved. When inputs/weights are quantized in 4-bits, *TALIPOT* achieves energy and area efficiencies of 25.3 TOPS/W and 2.87 TOPS/mm^2, respectively. Considering technology scaling, *TALIPOT* achieves 9× and 15× better energy consumption compared to [20] and [25], respectively, 3× better acceleration compared to [10].

We tested our proposed method, *TALIPOT*, in CNN of LeNet. LeNet has the architecture of two stacked convolutional layers (CONV) which have 6 and 16 features, respectively, filtered by 5×5 kernels. Each convolutional layer is followed by average pooling (AP) layer of 2×2. Finally, 2 fully-connected (FC) layers with the size of 120 and 84 neurons and output layer of 10 neurons are set as classifier of the architecture. So, LeNet has the topology of 6C5-AP2-16C5-AP2-120FC-84FC-10FC. We will consider Lenet with CONV1(6C5-AP2), CONV2(16C5-AP2), FC1(120), FC2(84) and FC3(10) in the

TABLE VI
TALIPOT MNIST PERFORMANCE

Network	Activation Function	TALIPOT Mode	Train Acc.	Test Acc.	4-bit Acc.	*TALIPOT* Acc.	A (mm^2)	L (ns)	P (mW)	F (Mhz)	E (nJ)
784-16-16-16-10	*ReLU*	*U*	96	94.5	94	86	0.17	400	3.9	143	1.6
784-16-16-16-10	*ReLU*	*F1*	96	94.5	94	93	0.18	550	4.5	143	2.5
784-16-16-16-10	*Tanh*	*U*	97.5	95.3	95	93.5	0.2	400	5	143	2
784-16-16-16-10	*Tanh*	*F1*	97.5	95.3	95	94.5	0.21	550	5.8	143	3.2
784-32-32-32-10	*ReLU*	*U*	99	97	96.8	92	0.42	400	9.4	133	3.8
784-32-32-32-10	*ReLU*	*F1*	99	97	96.8	96	0.43	550	11	133	6.1
784-32-32-32-10	*Tanh*	*U*	99	96.9	96.8	96.5	0.45	400	10	133	4
784-32-32-32-10	*Tanh*	*F1*	99	96.9	96.8	96.7	0.47	550	12.3	133	6.8
784-64-64-64-10	*ReLU*	*U*	99.5	97.9	97.8	95.5	1	400	20	125	8
784-64-64-64-10	*ReLU*	*F1*	99.5	97.9	97.8	97.5	1.03	550	24.5	125	13
784-64-64-64-10	*Tanh*	*U*	99.6	97.8	97.7	97.5	1.07	400	21	125	8.4
784-64-64-64-10	*Tanh*	*F1*	99.6	97.8	97.7	97.7	1.1	550	26	125	14
784-128-128-128-10	*ReLU*	*U*	99.7	98.3	98.2	97.5	2.1	400	42	118	17
784-128-128-128-10	*ReLU*	*UC*	99.7	98.3	98.2	97	2.1	300	42	118	12.6
784-128-128-128-10	*ReLU*	*F1*	99.7	98.3	98.2	98.2	2.11	550	45	118	25
784-128-128-128-10	*Tanh*	*U*	99.8	98.3	98.2	98.2	2.14	400	43	118	17
784-128-128-128-10	*Tanh*	*UC*	99.8	98.3	98.2	98	2.14	300	43	118	13
784-128-128-128-10	*Tanh*	*F1*	99.8	98.3	98.2	98.2	2.17	550	47	118	26
784-256-256-256-10	*ReLU*	*U*	99.9	98.6	98.5	98.3	4.3	400	86	111	34
784-256-256-256-10	*ReLU*	*UC*	99.9	98.6	98.5	98	4.3	300	86	111	26
784-256-256-256-10	*ReLU*	*F1*	99.9	98.6	98.5	98.5	4.32	550	90	111	50
784-256-256-256-10	*Tanh*	*U*	100	98.5	98.4	98.4	4.35	400	88	111	35
784-256-256-256-10	*Tanh*	*UC*	100	98.5	98.4	98.2	4.35	300	88	111	26
784-256-256-256-10	*Tanh*	*F1*	100	98.5	98.4	98.4	4.41	550	95	111	52

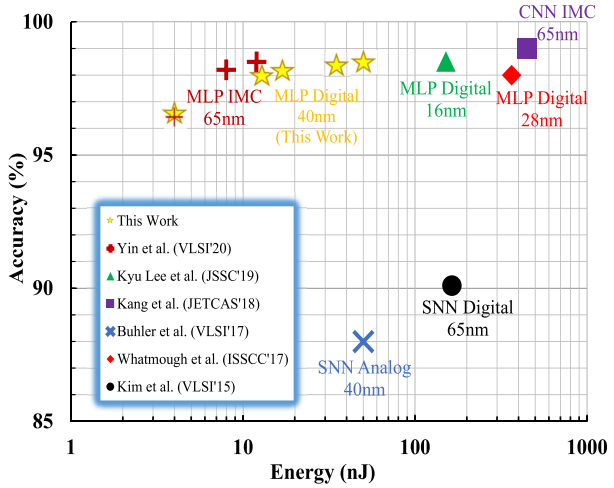

Fig. 17. Energy versus accuracy comparison for published MNIST results. Data are collected from: Yin et al. (VLSI'20) [10], Kyu Lee et al. (JSSC'19) [20], Kang et al. (JETCAS'18) [23], Buhler et al. (VLSI'17) [24], Whatmough et al. (ISSCC'17) [25], and Kim et al. (VLSI'15) [26].

TABLE VII
PERFORMANCE COMPARISON OF HARDWARE ACCELERATORS IN MNIST (W/O NORMALIZATION)

Reference	[10]	[25]	[20]	This Work		
Tech.(nm)	65	28	16	40		
Domain	IMC	Digital	Digital	Digital		
Supported Applications	MLP/CNN	MLP	MLP	MLP/CNN		
Weight Precision (Bits)	1	8	8	4		
Area (mm^2)	15	5.76	6.25	4.3		
Maximum Frequency (GHz)	0.55	0.67	1	0.111		
Accuracy	98.5	98.4	98.5	98.5	98.4	98.2
Througput (MFPS)	8.6	0.015	-	27.8	27.8	27.8
Energy (nJ)	12	360	151	50	35	26
1-Bit Energy (nJ)	12	45	18.9	12.5	8.8	6.5
Energy Efficiency (TOPS/W)	-	-	2.44	13.5	19	25.3
Energy Efficiency (TOPS/W*Bit)	-	-	19.5	53	78	101.2
Area Efficiency ($TOPS/mm^2$)	-	-	-	1.57	2.15	2.87

following paragraphs.

Table VIII details implementation of LeNet according to bandwidth (BW) requirement of the hardware. For example, there are total of 150×784 nodes needed to be calculated to be done with CONV1. Considering that each node has 8-bit precision, the total bandwidth of the layer is about 1 Mb. The overall architecture has a total bandwidth of 3.33 Mb. We assumed that the maximum allowed bandwidth of the hardware is below 1 Mb/cycle. In order to ensure this, we have reasonably scaled nodes of each layer such that the total bandwidth of the tile configurations doesn't exceed the hardware capacity.

Fig. 18 displays the timeline of *TALIPOT* in configured hardware of LeNet implementation. The model details are given in Table IX. In Hybrid-2 mode, *TALIPOT* using unfolded activation mode (U) is applied on every layer of LeNet architecture. The MSB of the output is generated at 16th clock cycle. As soon as the MSB is generated at the output of FC3, 3 MSBs of the CONV1, 2 MSBs of the CONV2 and 1 MSB of FC1 and FC2 are generated (highlighted in red). Then, in every 5 clock cycles, the following MSBs of the output are generated until full precision is achieved. *TALIPOT* has latency of 51 clock cycles in this model with accuracy drop of 0.9%. This can be trimmed by well calibrating or using folded mode of *TALIPOT*. In order to provide same throughput, baseline conventional bit-serial model needs 128 clock cycles. So, the Estimated Energy Efficiency (EEE) of *TALIPOT* over baseline

TABLE VIII
LENET ARCHITECTURE AND HARDWARE CONFIGURATION WITH 8-BIT PRECISION

	CONV1	CONV2	FC1	FC2	FC3	BW (Mb)
Matrix Mult.	5*5*28*28*6	5*5*6*10*10*16	400*120	120*84	84*10	
Nodes	150×784	150×1600	400×120	120×84	84×10	3.33
Scaling Factor	5	5	4	1	1	
Tile Config.	150x160	150×320	100×120	120×84	84×10	0.76

Scaling Factors					Model
5	**5**	**4**	**1**	**1**	**Hybrid-2**
CONV1	**CONV2**	**FC1**	**FC2**	**FC3**	**#CLK**
MSB	MSB	MSB	MSB	MSB	16
MSB-1	MSB-1	MSB-1	MSB-1	MSB-1	+5
MSB-2	MSB-2	MSB-2	MSB-2	MSB-2	+5
MSB-3	MSB-3	MSB-3	MSB-3	MSB-3	+5
MSB-4	MSB-4	MSB-4	MSB-4	MSB-4	+5
MSB-5	MSB-5	MSB-5	MSB-5	MSB-5	+5
MSB-6	MSB-6	MSB-6	MSB-6	MSB-6	+5
LSB	LSB	LSB	LSB	LSB	+5

Fig. 18. LeNet timeline with hybrid processing.

TABLE IX
PERFORMANCE COMPARISION OF HYBRID MODELS WITH BASELINE CONVENTIONAL BIT-SERIAL METHOD IN LENET WITH 8-BIT PRECISION

Models	CONV1-CONV2	FC1	FC2-FC3	Latency (#Clk)	Accuracy (%)	EEE (×)
Baseline	X	X	X	128	99.01	1
Hybrid-1	U	U	X	65	98.45	1.97
Hybrid-2	U	U	U	51	98.14	2.51

in this architecture is about 2.5×.

We compared *TALIPOT*'s hybrid models with the state-of-the-art design using modified VGG model [28] on CIFAR-10 dataset. The architecture starts with an input layer of 32×32 image with 3 channels. 32, 32, 64, 64, 128, 128 kernels with size of (3×3) are used for 6 convolutional layers with following MLP of 128-10. The size of features are halved by 2×2 max pooling (MP) layers after each double convolutional layers. So, the architecture is CONV1 (32C3), CONV2 (32C3-MP2), CONV3 (64C3), CONV4 (64C3-MP2), CONV5 (128C3), CONV6 (128C3-MP2), FC1 (128), FC2 (10). The model details are given in Table X. The total bandwidth of the architecture is about 331 Mb when 8-bit precision is used for inputs and weights. 64×64 and 128×64 tile structures are used for relatively smaller and bigger layers, respectively, for both of the designs. After tiling, total bandwidth is reduced to 0.36 Mb per cycle according to hardware capacity, consuming the same power, having the same throughput. So, the energy efficiency estimation is predicted from the latency factors.

STRIPE [2] has normalized latency factor of 42 clock cycles in this architecture. On the other hand, *TALIPOT* employing unfolded activation mode (U) in Hybrid-1 model has latency factor of 14 clock cycles which brings about 3× EEE over with 14.6% accuracy drop. Note that *TALIPOT* doesn't apply boosting on CONV1 and MLP because the processing time is dominated by CONV2 to CONV6 layers. 1-bit folded activation mode (F1) is applied from CONV2 to CONV6 in Hybrid-2 model. 2.3× EEE is achieved over 4.8% accuracy overhead. We see that the accuracy gap is mostly closed by using Hybrid-3 model in which 2-bit activation folded mode (F2) is applied on CONV2. *TALIPOT* has normalized latency factor of 17 clock cycles which results in 2.5× EEE over STRIPE. Finally, if we further increase folding bits on CONV3 as well as CONV2 as depicted in Hybrid-4 model, *TALIPOT* achieves prime accuracy with EEE of 2.3× over STRIPE.

V. DISCUSSION AND CONCLUSION

We propose *TALIPOT* as an energy efficient hardware implementation technique of DNNs. *TALIPOT* uses hybrid bit parallel-serial processing which boosts the computation and reduces the energy. Energy efficiency of DNNs suffer from limited bandwidth and data reuse [1], [15]. For example, in order to untangle this problem, [1] utilizes multi-chip support. However, *TALIPOT* relaxes computation bandwidth significantly by $\frac{1}{N}$ where N-bit activation is used for inputs, with the help of hybrid processing. On the other hand, *TALIPOT* operates in real-time, generates output in serial, which is different from other type of accelerators that we have seen in literature to the best of our knowledge. This should give superiority to *TALIPOT* over accelerators that utilize conventional bit-serial processing for real-time applications such as object recognition and face identification. So, we see that *TALIPOT* has great potential to replace conventional bit-serial processing of DNN accelerators which are widely used in literature.

REFERENCES

[1] T. Luo et al., "DaDianNao: A Neural Network Supercomputer," in IEEE Transactions on Computers, vol. 66, no. 1, pp. 73-88, 1 Jan. 2017.

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

[3] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim and H. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 173-185, Jan. 2019.

[4] H. O. Johansson, P. Larsson, P. Larsson-Edefors and C. Svensson, "A 200-MHz CMOS bit-serial neural network," Proceedings Seventh Annual IEEE International ASIC Conference and Exhibit, Rochester, NY, USA, 1994, pp. 312-315.

[5] S. M. Kueh and T. Kazmierski, "A dedicated bit-serial hardware neuron for massively-parallel neural networks in fast epilepsy diagnosis," 2017 IEEE Healthcare Innovations and Point of Care Technologies (HI-POCT), Bethesda, MD, 2017, pp. 105-108.

[6] E. Charles, W. Xiaowei, W. Jingcheng, S. Arun, I. Ravi, S. Dennis, B. David, D. Reetuparna " Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks", 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 383-396, 2018.

[7] Afzal Ahmad, Muhammad Adeel Pasha, "Towards Design Space Exploration and Optimization of Fast Algorithms for Convolutional Neural Networks (CNNs) on FPGAs", 22nd IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE'19), Florence, Italy, March 2019.

[8] K. Yoshioka et al., "PhaseMAC: A 14 TOPS/W 8bit GRO Based Phase Domain MAC Circuit for in-Sensor-Computed Deep Learning Accelerators," 2018 IEEE Symposium on VLSI Circuits, Honolulu, HI, 2018, pp. 263-264, doi: 10.1109/VLSIC.2018.8502291.

[9] T. Szabo, L. Antoni, G. Horvath and B. Feher, "A full-parallel digital implementation for pre-trained NNs," Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, 2000, pp. 49-54 vol.2.

TABLE X
PERFORMANCE COMPARISON OF TALIPOT HYBRID MODELS WITH STATE-OF-THE-ART DESIGN ON CIFAR-10 DATASET

CNN Model (N=8)	CONV1	CONV2	CONV3	CONV4	CONV5	CONV6	FC1	FC2	BW (Mb)	Norm. Latency (#Clk)	Accuracy (%)	EEE (×)
Matrix Mult.	3*3*3* 32*32*32	3*3*32* 32*32*32	3*3*32* 16*16*64	3*3*64* 16*16*64	3*3*64* 8*8*128	3*3*128* 8*8*128	2048*128	128*10				
Architecture	864×1024	9216×1024	4608×1024	9216×1024	4608×1024	9216×1024	256×1024	128×10	311			
Tile Structure	64×64	128×64	64×64	128×64	64×64	128×64	64×64	64×64	0.36			
Scaling Factor	216	1152	1152	1152	1152	1152	64	1				
Normalized Scaling Factor	0.19	1	1	1	1	1	0.056	0				
STRIPE [2]	X	X	X	X	X	X	X	X	0.36	42	90.4	1
Hybrid-1	X	U	U	U	U	U	X	X	0.36	14	75.8	3
Hybrid-2	X	F1	F1	F1	F1	F1	X	X	0.36	18	85.6	2.3
Hybrid-3	X	F2	F1	U	U	U	X	X	0.36	17	90.1	2.5
Hybrid-4	X	F2	F2	U	U	U	X	X	0.36	18	90.4	2.3

X: No activation rounding is applied to layer.
U: Unfolded activation rounding is applied to layer.
F1: 1-bit folded activation rounding is applied to layer.
F2: 2-bit folded activation rounding is applied to layer.

[10] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok and J. Seo, "Vesti: Energy-Efficient In-Memory Computing Accelerator for Deep Neural Networks," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 1, pp. 48-61, Jan. 2020.

[11] J. Lee, J. Lee, D. Han, J. Lee, G. Park and H. Yoo, "An Energy-Efficient Sparse Deep-Neural-Network Learning Accelerator With Fine-Grained Mixed Precision of FP8–FP16," in IEEE Solid-State Circuits Letters, vol. 2, no. 11, pp. 232-235, Nov. 2019.

[12] J. L. Holt and J. -. Hwang, "Finite precision error analysis of neural network electronic hardware implementations," IJCNN-91-Seattle International Joint Conference on Neural Networks, Seattle, WA, USA, 1991, pp. 519-525 vol.1.

[13] N. Samimi, M. Kamal, A. Afzali-Kusha and M. Pedram, "Res-DNN: A Residue Number System-Based DNN Accelerator Unit," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 2, pp. 658-671, Feb. 2020.

[14] Ritchie Zhao and Weinan Song "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs", Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2017.

[15] Y. Chen, T. Yang, J. Emer and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 292-308, June 2019.

[16] Y. Wang, H. Li, L. Cheng and X. Li, "A QoS-QoR Aware CNN Accelerator Design Approach," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 11, pp. 1995-2007, Nov. 2019.

[17] D. Kim, T. Na, S. Yalamanchili and S. Mukhopadhyay, "DeepTrain: A Programmable Embedded Platform for Training Deep Neural Networks," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2360-2370, Nov. 2018.

[18] D. Shin, W. Choi, J. Park and S. Ghosh, "Sensitivity-Based Error Resilient Techniques With Heterogeneous Multiply–Accumulate Unit for Voltage Scalable Deep Neural Network Accelerators," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 3, pp. 520-531, Sept. 2019.

[19] H. Wu and C. Huang, "Data Locality Optimization of Depthwise Separable Convolutions for CNN Inference Accelerators," 2019 Design, Automation and Test in Europe Conference and Exhibition (DATE), Florence, Italy, 2019, pp. 120-125.

[20] S. K. Lee, P. N. Whatmough, D. Brooks and G. Wei, "A 16-nm Always-On DNN Processor With Adaptive Clocking and Multi-Cycle Banked SRAMs," in IEEE Journal of Solid-State Circuits, vol. 54, no. 7, pp. 1982-1992, July 2019.

[21] Y. Yamada et al., "A 20.5 TOPS Multicore SoC With DNN Accelerator and Image Signal Processor for Automotive Applications," in IEEE Journal of Solid-State Circuits, vol. 55, no. 1, pp. 120-132, Jan. 2020.

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

[23] Kang *et al.*, "An In-Memory VLSI Architecture for Convolutional Neural Networks," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 3, pp. 494-505, Sept. 2018

[24] Buhler *et al.*, "A 3.43TOPS/W 48.9pJ/pixel 50.1nJ/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm CMOS," 2017 Symposium on VLSI Circuits, Kyoto, 2017, pp. C30-C31.

[25] Whatmough *et al.*, "14.3 A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with ¿0.1 timing error rate tolerance for IoT applications," 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, 2017, pp. 242-243.

[26] Kim *et al.*, "A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning," 2015 Symposium on VLSI Circuits (VLSI Circuits), Kyoto, 2015, pp. C50-C51, doi: 10.1109/VLSIC.2015.7231323.

[27] Krizhevsky, Alex. (2012). Learning Multiple Layers of Features from Tiny Images. University of Toronto.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, 2014.

Mahmut Burak Karadeniz received the B.S degree in electronics engineering from Turkish Naval Academy, Istanbul, Turkey, in 2011 and the M.S. degree in electrical engineering from University of Washington, Seattle, USA, in 2015. He is currently pursuing the Ph.D. degree in electronics and communication engineering at Istanbul Technical University (ITU), Istanbul, Turkey.

Since 2016, he has been director of electronic warfare systems in Istanbul Naval Shipyard, Istanbul, Turkey. He is the member of the Emerging Circuits and Computation (ECC) Group at ITU. His research interest includes very large-scale integration circuits, stochastic computing and energy efficient artificial intelligence hardware designs.

Mustafa Altun received his BSc and MSc degrees in electronics engineering at Istanbul Technical University in 2004 and 2007, respectively. He received his PhD degree in electrical engineering with a PhD minor in mathematics at the University of Minnesota in 2012.

Since 2012, he has served as a faculty member of electrical engineering at Istanbul Technical University. Dr. Altun runs the Emerging Circuits and Computation (ECC) Group in the same university. Dr. Altun has been served as a principal investigator/researcher of various projects including EU H2020 RISE, National Science Foundation of USA (NSF), and TUBITAK projects. He is an author of more than 60 peer reviewed papers and a book chapter, and the recipient of the Science Academy Young Scientist, TUBITAK Success, TUBITAK Career, and Werner von Siemens Excellence awards.