EERA-DNN: An energy-efficient reconfigurable architecture for DNNs with hybrid bit-width and logarithmic multiplier

Zhen Wang, Mengwen Xia, Bo Liu^{a)}, Xing Ruan, Yu Gong, Jinjiang Yang, Wei Ge, and Jun Yang

National ASIC System Engineering Technology Research Center,
Southeast University, Nanjing 210096, P.R.C

a) liubo_cnasic@seu.edu.cn

Abstract: This paper proposes an energy-efficient reconfigurable architecture for deep neural networks (EERA-DNN) with hybrid bit-width and logarithmic multiplier. To speed up the computing and achieve high energy efficiency, we first propose an efficient network compression method with hybrid bit-width weights scheme, that saves the memory storage of network LeNet, AlexNet and EESN by 7x–8x with negligible accuracy loss. Then, we propose an approximate unfolded logarithmic multiplier to process the multiplication operations efficiently. Comparing with state-of-the-art architectures EIE and Thinker, this work achieves over 1.8x and 2.7x better in energy efficiency respectively.

Keywords: deep neural networks, approximate computing, hybrid bit-width

Classification: Integrated circuits

References

- [1] A. Krizhevsky, *et al.*: “ImageNet classification with deep convolutional neural networks,” *Commun. ACM* **60** (2017) 84 (DOI: [10.1145/3065386](https://doi.org/10.1145/3065386)).
- [2] K. Simonyan, *et al.*: “Very deep convolutional networks for large-scale image recognition,” *Computer Science* (2014) (arXiv:1409.155).
- [3] A. Graves, *et al.*: “Hybrid speech recognition with Deep Bidirectional LSTM,” *ASRU* (2014) 273 (DOI: [10.1109/ASRU.2013.6707742](https://doi.org/10.1109/ASRU.2013.6707742)).
- [4] P.-S. Huang, *et al.*: “Learning deep structured semantic models for web search using clickthrough data,” *ICCIKM* (2013) 2333 (DOI: [10.1145/2505515.2505665](https://doi.org/10.1145/2505515.2505665)).
- [5] G. E. Dahl, *et al.*: “Improving deep neural networks for LVCSR using rectified linear units and dropout,” *ICCA* **26** (2013) 8609 (DOI: [10.1109/ICASSP.2013.6639346](https://doi.org/10.1109/ICASSP.2013.6639346)).
- [6] B. Shakibi, *et al.*: “Predicting parameters in deep learning,” *NIPS* **2** (2013) 2148 (arXiv:1306.0543).
- [7] S. Han, *et al.*: “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” *NIPS* **56** (2015) 3

- (arXiv:1306.0543).
- [8] B. Reagen, *et al.*: “Minerva: Enabling low-power, highly-accurate deep neural network accelerators,” ISCA (2016) 267 (DOI: [10.1109/ISCA.2016.32](https://doi.org/10.1109/ISCA.2016.32)).
 - [9] S. Han, *et al.*: “EIE: Efficient inference engine on compressed deep neural network,” ISCA (2016) 243 (DOI: [10.1109/ISCA.2016.30](https://doi.org/10.1109/ISCA.2016.30)).
 - [10] S. Zhang, *et al.*: “Cambricon-X: An accelerator for sparse neural networks,” MICRO (2016) 1 (DOI: [10.1109/MICRO.2016.7783723](https://doi.org/10.1109/MICRO.2016.7783723)).
 - [11] T. Chen, *et al.*: “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” ISBN **49** (2014) 269 (DOI: [10.1145/2541940.2541967](https://doi.org/10.1145/2541940.2541967)).
 - [12] Z. Du, *et al.*: “ShiDianNao: Shifting vision processing closer to the sensor,” ISCA (2015) 92 (DOI: [10.1145/2749469.2750389](https://doi.org/10.1145/2749469.2750389)).
 - [13] K.-L. Tsai, *et al.*: “A low power high performance design for JPEG Huffman decoder,” ICECS (2008) 1151 (DOI: [10.1109/ICECS.2008.4675062](https://doi.org/10.1109/ICECS.2008.4675062)).
 - [14] Z. Babić, *et al.*: “An iterative logarithmic multiplier,” Microprocess. Microsyst. **35** (2011) 23 (DOI: [10.1016/j.micpro.2010.07.001](https://doi.org/10.1016/j.micpro.2010.07.001)).
 - [15] B. Liu, *et al.*: “E-ERA: An energy-efficient reconfigurable architecture for RNNs using dynamically adaptive approximate computing,” IEICE Electron. Express **14** (2017) 20170637 (DOI: [10.1587/elex.14.20170637](https://doi.org/10.1587/elex.14.20170637)).
 - [16] P. Lucey, *et al.*: “The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression,” CVPRW (2010) 94 (DOI: [10.1109/CVPRW.2010.5543262](https://doi.org/10.1109/CVPRW.2010.5543262)).
 - [17] Y. Miao, *et al.*: “EESSEN: End-to-end speech recognition using deep RNN models and WFST-based decoding,” ASRU (2015) 167 (DOI: [10.1109/ASRU.2015.7404790](https://doi.org/10.1109/ASRU.2015.7404790)).
 - [18] Y. Jia, *et al.*: “Caffe: Convolutional architecture for fast feature embedding,” AICM (2014) 675 (DOI: [10.1145/2647868.2654889](https://doi.org/10.1145/2647868.2654889)).
 - [19] D. Wang, *et al.*: “A free Chinese speech corpus,” Computer Science (2015) (arXiv.org/abs/1512.01882).
 - [20] S. Li, *et al.*: “FPGA acceleration of recurrent neural network based language model,” FCCM (2015) 111 (DOI: [10.1109/FCCM.2015.50](https://doi.org/10.1109/FCCM.2015.50)).
 - [21] S. Yin, *et al.*: “A high energy efficient reconfigurable hybrid neural network processor for deep learning applications,” IEEE J. Solid-State Circuits **53** (2018) 968 (DOI: [10.1109/JSSC.2017.2778281](https://doi.org/10.1109/JSSC.2017.2778281)).

1 Introduction

Deep neural networks have evolved to the state-of-the-art technique for computer vision [1, 2], speech recognition [3], automatic translation, advertisement recommendation, and so on [4, 5]. As the large amounts of synaptic weights incur intensive computation and memory accesses, efficiently processing large-scale neural networks with limited resources remains a challenging problem.

In order to overcome the challenge caused by overwhelming neurons and synapses, Denil et al. demonstrated that there are a large number of parameter redundancy problems in the learning model and further that some of these parameters can be pruned directly [6]. Han et al. [7] have proposed a pruning technique to shrink the amount of synaptic weights by about 10x with negligible accuracy loss. Reagen et al. have explored minimum number of bits requirements for each data path while preserving model accuracy within established negligible error bound [8].

On the basis of sparse and reduced network, some accelerators on compressed deep neural network have been proposed [9, 10].

Interestingly, dramatically reducing the amount of synapses and using traditional sparse matrix calculation scheme do not necessarily improve the performance and energy efficiency of existing accelerators. The computationally intensive model of the neural network is mainly embodied in a large number of multiplication operations. Although a variety of neural network accelerator solutions have been proposed [8, 9, 10, 11, 12], most of the work remain in the solution to the problem of access and data path design, and do not really go deep into the underlying multiplication unit.

In this paper, we propose an energy-efficient reconfigurable engine for hybrid bit-width DNNs. An efficient network compression method with hybrid bit-width weights scheme is proposed. Based on that, an energy-efficient approximate unfolded logarithmic multiplier is also adopted in this work to further reduce the calculation power consumption. The rest of this paper is organized as followed. In section 2, we present the hybrid bit-width weights scheme, the approximate unfolded logarithmic multiplier and a DNN accelerator architecture. In section 3, several case studies and the implementation results are shown. Section 4 concludes this paper.

2 Energy-efficient reconfigurable architecture for DNNs with hybrid bit-width and logarithmic multiplier

2.1 Hybrid bit-width weights scheme

The the weight magnitude of neural network reflects the importance of synapses. In terms of its distribution, the closer to the zero point, the more densely the weight is distributed. Fig. 1 shows the weight distribution of typical DNN network LeNet.

Fig. 1. The weight distribution of LeNet

For the weight grading, which is shown in Table I, there are several pieces of information available to characterize weight size as efficiently as possible. First is leading “1” position, requires only 4 bits for the weight represented by the 16-bit binary number, since the level of the weight is set according to the leading “1” index, knowing the leading “1” index will naturally be able to know the level and the effective length of the weight; the second is the sign bit; the third is the significant bit used to characterize the size of the weight. Fig. 2 shows an example of reducing the complexity of weights.

Table I. Weight grading table

Level	Leading “1”	Length of bits
0	<2	0
1	2–6	2
2	7–12	4
3	13–14	6
4	15	8

Fig. 2. Example of reducing the weights complexity

The storage of the weight is divided into two parts, one part is the mixed bit stream, the other part is the leading “1” index, the leading “1” index indicates the length of significant bits for weights and the position of leading “1” in a weight. Fig. 3 shows the principle of weight storage and parsing. In the upper part of the D_{ij} , i represents the i -th number, and j represents the j -th significant bit of the data i . For example, D_{62} represents the second significant bit of the sixth number. D_{63} represents the sign bit of the sixth number. The significant bits and the number of leading “1” index for each weight are represented by the same color. The process of weight parsing is as follows: The decoder reads a leading “1” index, for example, 0111, by looking up Table I, the most significant bit position is 7 and the corresponding level is 2, so the decoding result is $D_{63}000_0000_1D_{62}D_{61}D_{60}_0000$. For the leading “1” index 1110, the most significant bit position is 14 and the corresponding level is 6, so the decoding result is $D_{75}01D_{74}_D_{73}D_{72}D_{71}D_{70}_0000_0000$. The third, fourth and fifth leading “1” index are 0000, which indicates that these three weights are all 0, and therefore the decoding result is 0000_0000_0000_0000. In addition, the weights with value of 0 are only needed to store the part of leading “1” index of each.

In this paper, we use the 4-bit leading “1” index to characterize the highest nonzero position, the level of weight and so on. Fig. 4 is the hardware implementation for decoding the leading “1” index. The leading “1” index of each weight are pre-processed with Huffman encoding off-line [13], and dynamically decoded on-chip as shown in Fig. 4. The selector determines whether the input data enters LUT1 or LUT2. LUT1 is selected when the first six digits of the input codeword (CW) are not all 1, otherwise LUT2 is selected. Finally, the MUX outputs code length (CL) which is returned to the shift register and symbol (position of the first bit of weight).

Fig. 3. Weights storage scheme with hybrid bit-width

Fig. 4. Decoder for the leading “1” index of weights

2.2 Approximate unfolded logarithmic multiplier

The iterative logarithmic multiplier scheme is proposed in paper [14]. But it is not suitable for DNN with the following factors: first, iterative computing limits the data processing throughput; second, iterative computing seriously affects the pipeline of DNN scheduling. So an approximate unfolded logarithmic multiplier is proposed in this paper.

The binary representation of the number N can be written as (1):

$$N = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{i-k} Z_i \right) = 2^k (1 + x) \quad (1)$$

From (1), where k is the characteristic number or place of the most significant bit with the value of ‘1’, Z_i is the bit value at i -th position, x is a mantissa (fraction) and j depends on the number precision. A full precise expression for the multiplication can be written as:

$$P_{true} = N_1 N_2 = 2^{k_1+k_2} (1 + x_1 + x_2) + 2^{k_1+k_2} (x_1 x_2) \quad (2)$$

The formula (3) derived from (1) must be taken into account for avoiding the approximation error:

$$x * 2^k = N - 2^k \quad (3)$$

Combining (2) with (3), we can get:

$$P_{true} = 2^{k_1+k_2} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} + (N_1 - 2^{k_1})(N_2 - 2^{k_2}) \quad (4)$$

Let

$$P_{approx}^{(0)} = 2^{k_1+k_2} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} \quad (5)$$

be the first order approximation. We can get:

$$P_{true} = P_{approx}^{(0)} + (N_1 - 2^{k_1})(N_2 - 2^{k_2}) \quad (6)$$

The two multiplicands in (6) can be obtained simply by removing the leading 1 in the numbers N_1 and N_2 . Therefore, we can easily repeat the multiplication procedure with multiplicands. Note that the first term can be seen as a binary number that the $k_1 + k_2$ bit is 1, the other bits are zero; the second item can be seen as the first multiplier to be removed the leading 1 and then shift k_2 bit; the third term can be seen as the second multiplier to be removed the leading 1 then shift k_1 bit.

Considering the fault tolerance of neural networks, only 3 iterations of logarithmic multiplier are enough for the network accuracy [15]. So an approximate unfolded logarithmic multiplier is designed, as shown in Fig. 5. It is composed of three stages, each stage generates the multiplication results of current iteration, and also the input of next stage (the third iteration only generates the multiplication results). The results of the three stages are finally merged to the final result. To promote the throughput, a group of pipeline registers are inserted between stage 2 and stage 3. Two 16-bit shifter SH, a 16-bit adder, a 16-bit S-A and an optional priority encoder Pri-Enc are included in one stage. The Pri-Enc module replaces the complex LOD (leading one detector) module in the original design [14] to achieve a shorter delay without affecting the multiplier accuracy, The maximum for the highest nonzero bit of the last two terms are only possible for the $(k_1 + k_2 - 1)$ -th

Fig. 5. Approximate unfolded logarithmic multiplier

bit, the effective bit overlap region is less than 16 bits, so that it is a 16-bit adder. In the first iteration, k_1^{-1} , k_2^{-1} in the figure represent $(16 - k_1)$ and $(16 - k_2)$, and the SH module is a shifter. k_1^{-1} and k_2^{-1} are used as control signals to shift the two multipliers so that they are aligned to the left most significant bit. The shifted two numbers are summed by the adder. Since the result can be affected by the previous shift operation, it needs to be readjusted by the S-A module, which is composed of several sets of shift operation arrays. In the second and third operations, Pri-Enc module obtains a new multiplier k_1^{-1} , k_2^{-1} , while the functionality of other modules is the same as in the first stage.

2.3 DNN accelerator with hybrid bit-width and logarithmic multiplier

The top level architecture of an EERA-DNN prototype system is as shown in Fig. 6. It consists of an ARM7TDMI used as system controller, a 32 KB scratch-pad memory (SPM) used as system buffer, two EERA-DNN accelerators for accelerating DNN networks, and other modules, including an Interrupt Controller (IntCtrl), a Direct Memory Access Controller (DMAC), and an External Memory Interface (EMI).

As shown in Fig. 6, each EERA-DNN accelerator consists of a controller, a sigmoid function module, a Huffman decoder, 16 process elements (PEs), a Conf SRAM and a Dest SRAM. EERA-DNN accelerator can directly access both the input data and the weights via DMA. The leading “1” index of the weighted values is stored in the Conf SRAM. The Huffman decoder decodes the weights in the Conf SRAM and then transmit it to the control module; the control module distributes the weights to the PEs. The leading “1” index of the input vector is obtained through an LOD module [14], which is distributed to the PE along with the input vector for the multiply-accumulate operation in PEs.

Each PE consists of an approximate unfolded logarithmic multiplier proposed above, eight 16-bit data input/output registers and multiplexers, a weight SRAM with a size of 2 KB, a data input FIFO, and a resolver module as denoted in Fig. 3 for parsing the weights. The PE can be configured to process multiplication operations, or addition operations with the adder adopted in the proposed multiplier architecture. Besides, the interconnections between PEs can also be reconfigured by setting the data input/output multiplexers of each PE, and therefore EERA-DNN

Fig. 6. DNN accelerator with hybrid bit-width and logarithmic multiplier

can process matrix multiplication and addition with various sizes of different DNN layers.

3 Case study and simulation results

To compress the network, we first adopt the typical pruning method [9], and then use the proposed hybrid bit-width scheme to further compress the network. Two models for CNN were trained and tested on MNIST and ImageNet data-sets, the LeNet is on MNIST and the AlexNet is on ImageNet. A RNN is trained using EESSEN [17]. We use the THCHS-30 Speech Corpus [18], including 750 sentences, as the training set. All experiments were performed with the Caffe framework [19] and ESSEN [17]. The network parameters and accuracy before and after compressing are shown in Table II, where HB represent the proposed hybrid bit-width scheme. Taking LeNet as an example, the top-1 error is 0.80% originally, when adopted with the typical pruning method [9], the network is compressed by 2.7X and the top-1 error is 0.89%. When the proposed hybrid bit-width scheme is further adopted, the network can be compressed 7.0X as much, and the the top-1 error is 1.10%. The compression effects of other networks including AlexNet and ESSEN are similar. The compression schemes save network storage by 7x and 8x across these three different networks with negligible accuracy loss. The total size of LeNet decreased from 1720k to 246k, AlexNet decreased from 240 MB to 30 MB and EESSEN decreased from 17 MB to 0.23 MB, this is of great significance to the optimization of neural networks.

Table II. Parameter size after compression

Network	Scheme	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet	original	0.80%	-	1.72 MB	-
	pruning	0.89%	-	0.64 MB	2.7
	pruning+proposed HB	1.10%	-	0.25 MB	7.0
AlexNet	original	42.78%	19.73%	240 MB	-
	pruning	43.18%	19.92%	78 MB	3.1
	pruning+proposed HB	43.58%	20.12%	30 MB	8.0
ESSEN	original	8%	0	17 MB	-
	pruning	8.1%	0	5.86 MB	2.9
	pruning+proposed HB	11.6%	0.1%	0.23 MB	7.0

Table III is results of the extend Cohn-Kanaded database (CK+) [16] and ImageNet respectively on AlexNet. From experimental results, it can be seen that the approximate multiplier achieves about 94.55% and 67.12% recognition accuracy while the standard multiplier achieve 94.63% and 67.26% when trained and tested respectively. Therefore, the proposed logarithmic multiplier can achieve negligible loss of recognition accuracy, and is suitable for DNN applications.

The EERA-DNN prototype system is simulated at netlist level with TSMC 45 nm LP process technology. The timing and power consumption are evaluated at

Table III. Comparison of the recognition accuracy using different multipliers

Database (on AlexNet)	Recognition accuracy	
	CK+	ImageNet
Standard Multiplier	94.63%	67.26%
Approximate Multiplier	94.55%	67.12%

1.1 V 25°C TT process corner. Static timing analysis by PrimeTime after placement and routing shows that its critical path is 1.25 ns, so its clock frequency reaches 800 MHz. The area is about 4.34 mm² after place and routing, and the power is 120 mW when the throughput is 51.2GOPS. The area and power consumption is shown in Table IV.

Table IV. The implementation details of proposed EERA-DNN

Module	Energy consumption		Hardware overhead	
	Power (mW)	Percent (%)	Area (um ²)	Percent (%)
Array	40.15	33.39	1035812	23.84
Memory	58.23	48.41	2807823	64.63
Others	21.90	18.20	500915	11.53
Total	120.28	100	4344550	100

The comparisons with other accelerators are shown in Table V. The result shows that EERA-DNN can achieve much higher energy efficiency comparing with FPGA in the field of DNN accelerator. In work [20] and [21], the network are implemented and evaluated on their original form without compression. In work [9] and our work, all the networks are first compressed, so the equivalent throughput should be multiplied by its compression rates which is as same as [9]. For our works, the networks are firstly compressed with a typical pruning method similar as in work [9], and then further compressed by our proposed hybrid bit-width scheme.

Table V. Comparison with other accelerators

	RNN-LM [20]	EIE [9]	Thinker [21]	This work
Architecture	FPGA	Reconfigurable Architecture		
Technology (nm)	-	45	65	45
Frequency (MHz)	150	800	200	800
Data Width (bits)	16	16	8/16	16
Power Consumption (W)	25	0.59	0.29	0.12
Peak Throughput (GOPS)	9.6	401.2–1103.3*	368	409.6*
Power Efficiency (GOPS/W)	0.38	680–1870	1268	3413

*The equivalent throughput are multiplied by their compression rates (4x–11x for EIE and 8x for our work on AlexNet, respectively).

For example, in our work, the compression rate is 8.0 for AlexNet, and that means 87.5% of computation are eliminated in our work. The results show that, comparing with state-of-the-art architectures EIE which is also implemented on compressed networks, this work achieves over 1.8x better in energy efficiency, since the approximate unfolded logarithmic multiplier proposed is much more energy efficient than the standard multiplier in EIE; And comparing with Thinker which is implemented on uncompressed networks, this work not only improve the equivalent throughput with network compression but also improve the energy efficiency with the proposed approximate unfolded logarithmic multiplier, and achieves over 2.7x better in energy efficiency.

4 Conclusions

This paper proposed an energy-efficient DNN accelerator with Hybrid Bit-Width scheme and approximate unfolded logarithmic multiplier. With the Hybrid Bit-Width scheme, the networks LeNet, AlexNet and ESSEN can be compressed by 7x–8x. By further adopting the proposed energy-efficient approximate unfolded logarithmic multiplier, the accelerator proposed in this paper can achieve over 1.8x and 2.7x better in power efficiency with comparison to state-of-the-art architecture EIE and Thinker, respectively.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61404028 and 61574033) and the National High Technology Research and Development Program of China (863 Program) (No. 2012AA012703).