Approximate Logic Synthesis in the Loop for Designing Low-Power Neural Network Accelerator

Yifan Qian[1], Chang Meng[1], Yawen Zhang[2], Weikang Qian[1,3,4], Runsheng Wang[2], Ru Huang[2]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]Institute of Microelectronics, Peking University, Beijing, China
[3]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
[4]State Key Laboratory of ASIC & System, Fudan University, Shanghai, China

***Abstract*—Approximate computing is an emerging circuit design paradigm. It improves the energy efficiency of circuits by introducing some errors. Recent works propose to apply approximate multipliers to design low-power neural network (NN) accelerators. Different from existing methods, in this paper, we advocate a method that integrates approximate logic synthesis (ALS) into the design loop of low-power NN accelerators. ALS automatically synthesizes a good approximate circuit and can take input distribution into consideration. With the help of ALS, the NN computation pattern can be exploited to design an approximate multiplier that fits better with the NN. The experimental results show that the proposed method can generate an extremely small approximate multiplier with area only 4.2% of the accurate version, while it can still achieve a high accuracy of 97.9% for LeNet-5 on MNIST dataset.**

***Index Terms*—Approximate Computing, Approximate Logic Synthesis, Neural Network, Neural Network Accelerator**

I. Introduction

As the feature size of CMOS transistors has decreased exponentially, there is an increasing challenge to improve circuit performance and energy efficiency. New design paradigms are needed to produce more energy-efficient circuits. At the same time, many applications, such as multimedia, data mining, and machine learning, can tolerate certain errors occurred in their internal computation. Such error tolerance is exploited in a new design paradigm, *approximate computing*, to address the energy efficiency issue [1]. Its key idea is to deliberately trade off accuracy for energy consumption by changing the Boolean function of a circuit. If the new approximate function is carefully chosen, the application-level correctness is minimally affected, while the energy consumption of the circuit can be reduced dramatically.

One application that approximate computing is particularly suitable is neural network (NN) computation. NN is an effective machine learning method that is widely used in various fields nowadays, such as computer vision and natural language processing [2]. In order to handle more and more complex learning tasks, NNs with more hidden layers are proposed. They involve an enormous number of additions and multiplications and hence, consume a large amount of energy. However, many studies have shown that there is always redundancy within the NNs [3], and hence, they are error tolerant and suitable for approximate computing. Some popular approaches to design energy-efficient NN accelerators like weight pruning and weight quantization indeed can be viewed as approximate computing methods.

In this paper, we focus on another type of approximate computing method for designing NN accelerators, that is, replacing arithmetic circuits by their approximate versions. Since multiplier is the most area- and energy-consuming arithmetic circuit in an NN accelerator, most works in this category target at designing NN accelerator with approximate multipliers. Some works propose new approximate multipliers specifically for NN accelerators, such as the logarithm-based approximate multiplier [4]. Some other works use existing approximate multiplier to build NN accelerators. Ansari *et. al.* apply the circuits in the EvoApprox8b library [5] and compare their hardware and software efficiency [6]. Then, they identify the features in approximate multipliers that make them better than others with respect to NN accuracy and build a predictor to forecast how well an approximate multiplier will perform. Liu *et. al.* proposes INA, which incrementally replaces some accurate multipliers in the NN by the approximate ones [7]. Mrazek *et. al.* introduces ALWANN, an automatic layer-wise approximation of NNs [8]. It uses one approximate multiplier in each layer.

One common feature of these existing works is that the approximate multiplier is first chosen and then the weights of the NNs are properly trained. This creates a one-way link from approximate multiplier to NN, as shown in Fig. 1(a). For this approach, if the initial approximate multiplier is not chosen properly, the NN accuracy may not reach a value close to the ideal accuracy by later training. To address this problem, in the paper, we propose an approach of integrating approximate logic synthesis (ALS) into the design loop, which is shown in Fig. 1(b).

Given the large design space of an approximate circuit, ALS tries to systematically explore the design space and automatically synthesize a good approximate circuit under user-specified error constraints. Many ALS methods have been proposed so far [9]–[15]. One feature of some ALS methods is that they can take input distribution into consideration to

This work is supported in part by the National Key R&D Program of China under Grant 2020YFB2205500 and the State Key Laboratory of ASIC & System Open Research Grant 2019KF004. Corresponding authors: Weikang Qian (qianwk@sjtu.edu.cn) and Runsheng Wang (r.wang@pku.edu.cn).

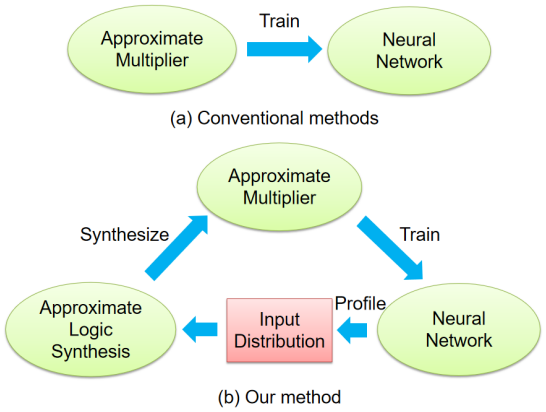

Fig. 1. Conventional methods versus our proposed method for designing approximate computing-based lower-power neural network accelerators.

design proper approximate circuits [12], [14], [15]. Given this feature, we advocate an approach that *integrates ALS into the design loop of the approximate computing-based low-power NN accelerators*. As shown in Fig. 1(b), the key idea is to take into consideration the operand distribution in the NN computation to design the approximate multiplier. By profiling the NN computation, an operand distribution is obtained. It is then fed into an ALS tool to synthesize an approximate multiplier. Then, the NN with the new approximate multiplier is properly trained. This forms a profiling-synthesizing-training loop, which can be iterated until convergence. With the help of ALS, the approximate multiplier can fit better with the NN, since the knowledge on the NN computation pattern is used to design the multiplier. In some sense, this realizes a genuine *software-hardware co-design*.

This paper demonstrates the above idea by using a state-of-the-art ALS method, ALSRAC [15], and testing it on LeNet-5 with MNIST dataset. The experimental results show that by using the proposed approach, we can generate an extremely small approximate multiplier with area only 4.2% of the accurate version, while it can still enable a high NN accuracy. Compared to a low-precision accurate multiplier, i.e., a 2-bit rounded multiplier, the final approximate multiplier generated by ALSRAC is both smaller and faster, while the NN accuracy by the approximate multiplier is higher than that by the 2-bit rounded multiplier.

The rest of the paper is organized as follows. Section II introduces the basic idea of ALSRAC. Section III describes the details of integrating ALS into the design loop of NN accelerators. Section IV shows the experimental results. Finally, Section V concludes the paper.

II. Background: ALSRAC

The basic idea of ALSRAC is to implement an internal signal by a new function that takes some other existing internal signals as input. With this, the original sub-circuit that generates the signal is replaced by a new sub-circuit that realizes the new function. If the new sub-circuit is smaller than the original one, the entire area of the circuit is reduced. This operation is called *resubstitution* in traditional logic synthesis [16]. An example of resubstitution is shown in Fig. 2. The figure shows two AND-inverter graphs (AIGs) where a node represents an AND gate and a dashed edge represents a negation of the signal on the edge. The signal g in the left sub-figure has the function $g = a(b+c+d)$. It can be realized by a new function $f(n, m) = n+m$ on two other existing internal signals n and m, where $n = a(b + c)$ and $m = ad$ (see the right sub-figure). By this resubstitution, the node number in the circuit is reduced by 1.

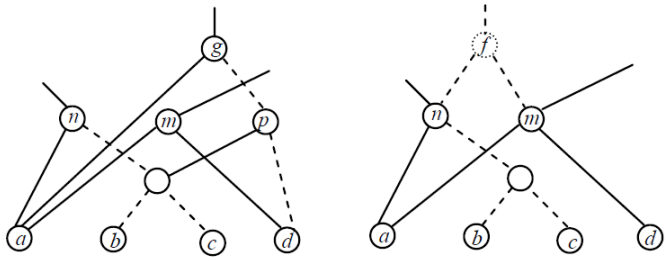

Fig. 2. Illustration of the resubstitution operation [16].

However, for traditional logic synthesis where resubstitutions should maintain the given Boolean function, the resubstitution opportunity is limited. ALSRAC extends the resubstitution technique in the approximate computing context and proposes an approximate resubstitution method [15]. With that, by allowing a small amount of error, more resubstitution candidates can be identified, which leads to further simplification of the original circuit. ALSRAC uses Monte Carlo simulation to guide the finding of the resubstitution candidates. It can take input distribution into consideration. It also supports various error metrics such as error rate, normalized mean error distance (NMED), etc.

III. Methodology

In this section, we elaborate the method of integrating ALSRAC into the design loop of low-power NN accelerators built with approximate multipliers.

The NN accelerator with approximate multipliers is modified from an NN accelerator with accurate integer multipliers by replacing the accurate multipliers with the approximate ones. We first describe how the NN accelerator with accurate integer multipliers is designed. Typically, the original NN model is real-value computation. The accelerator with accurate integer multipliers are designed from the original NN model by properly mapping the weights and activations into integers, using a method modified from the one presented in [17]. Specifically, the weights in the k-th layer share a common scaling factor W_k. For a real-valued weight r in that layer, it is converted into an n-bit signed integer q by the following equation:

$$q = \text{round}(r/W_k).$$

Note that the scaling factor W_k should be properly chosen to map the range of the weights in the k-th layer into the range of n-bit signed integers. The same mapping method is also applied to the activations in the k-th layer, which share a common scaling factor A_k. For different layers of an NN, the scaling factors W_k and A_k are different. With the weights and the activations mapped to the integers, the convolution can be implemented by integer multipliers and adders. After the convolution is finished, the integer result will be multiplied by the corresponding scaling factors W_k and A_k in this layer and sent to the following activation layer. Although this step involves floating-point multiplications, such operations are much fewer than the integer multiplications in the convolution. Thus, its energy cost is negligible.

Our proposed method designs the approximate multipliers to be used in the NN accelerator. The procedure of our method is shown in Fig. 3.

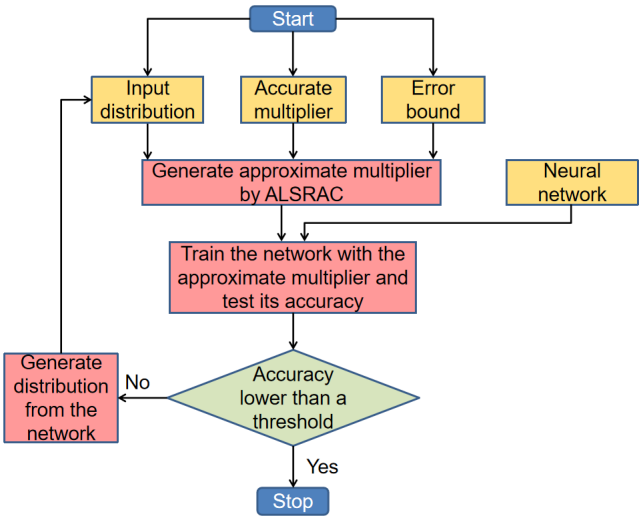

Fig. 3. The flow chart of our proposed method.

First, we use ALSRAC to generate an approximate integer multiplier from the original accurate integer multiplier. Besides the original circuit, ALSRAC takes two additional inputs, a bound on an error metric and an input distribution, as shown in Fig. 3. ALSRAC will generate an approximate circuit with the error no more than the bound under the input distribution. The error metric we use in this work is NMED, which is defined as the mean absolute difference between the accurate result and approximate result normalized to the maximum output value. The bound on NMED is initially set as a small value E_0. The input distribution is initially set as the uniform distribution. After the approximate multiplier is generated, we use it to replace each accurate multiplier in the NN accelerator and train the NN with the approximate multiplication to obtain the new weights. Then, we do inference on the test dataset and count the number of occurrences for each input pattern of the approximate multiplier. We count all the occurring input patterns in the process of forward propagation and normalize the results to obtain the probability distribution of the input patterns.

In the next round, we use the input distribution just derived to generate a new approximate multiplier using ALSRAC, as shown in Fig. 3. The following steps are the same as those in the first round. We repeat the process until the accuracy of the NN drops below a threshold. Then, we use the approximate multiplier of the previous round as the final one. Note that in each round, the error bound is increased over that of the previous round to enable ALSRAC to generate a smaller approximate multiplier.

IV. Experimental Results

In this section, we present the experimental results on the proposed method.

A. Experiment Setup

We use LeNet-5 and MNIST dataset [18] to study the effect of the proposed method. There are 50000 images in the training dataset and 10000 images in the test dataset. We use *tiny-dnn* [19], which is an open source C++14 implementation of deep learning, to simulate the accuracy of various LeNet-5 accelerators built with different multipliers. For hardware cost study, we focus on the multipliers in the accelerators. The accurate and the approximate multipliers are mapped with the MCNC standard cell library [20] using the logic synthesis tool ABC [21]. The area and delay of the circuits are reported by ABC.

The baseline accelerator uses accurate 8-bit signed multipliers. Its accuracy is 99.00%. The area and delay of an accurate 8-bit signed multiplier are listed in Table I.

Some important parameters of the training are listed below. The epoch is 50. The learning rate is set to be 0.001. The mini-batch size is 16. The accuracy threshold used in the proposed procedure is 97%.

B. Performance of the Proposed Method

We use the procedure described in Section III to generate the approximate multipliers and retrain the NN. We run 5 rounds in total. The NMED bounds for the 5 rounds are set as 0.001, 0.003, 0.006, 0.012, and 0.024, respectively. The area and delay of the multiplier generated in each round are listed in Table I. In the table, the accuracy of the NN accelerator using each approximate multiplier is also listed. For comparison purpose, we also consider a 2-bit rounded multiplier for 8-bit signed multiplication. It consists of a 2-bit signed multiplier and the input rounding part, which rounds the 3 most significant bits of each 8-bit input into a 2-bit signed number. Table I also lists the area and delay of that multiplier together with the accuracy of the LeNet-5 accelerator built with it.

We can see from Table I that after 5 rounds, we can finally obtain an extremely small approximate multiplier. Its area is only 4.2% of the accurate 8-bit multiplier. After retraining, the accelerator with that multiplier still achieves an accuracy of 97.86%, which is only 1.14% less than the

TABLE I
THE AREAS AND DELAYS OF VARIOUS MULTIPLIERS AND THE ACCURACIES OF THE ACCELERATORS BUILT WITH THESE MULTIPLIERS.

Circuit type	area	delay	error bound	accuracy
Accurate 8-bit multiplier	1326	27.1	-	99.00%
Approximate multiplier 1	966	27	0.001	98.86%
Approximate multiplier 2	786	26	0.003	98.74%
Approximate multiplier 3	202	18.1	0.006	98.67%
Approximate multiplier 4	73	10.7	0.012	98.44%
Approximate multiplier 5	56	6.7	0.024	97.86%
2-bit rounded multiplier	72	7.7	-	97.52%

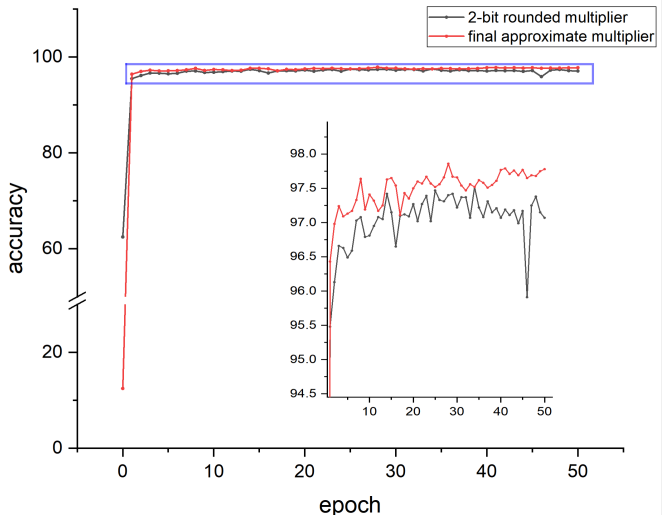

Fig. 4. Accuracy for retraining LeNet-5 accelerators built with the final approximate multiplier and with the 2-bit rounded multiplier.

baseline accuracy. Compared to the 2-bit rounded multiplier, the final approximate multiplier has smaller area and shorter delay. Yet, the accelerator with the approximate multiplier achieves a higher accuracy than the one with the 2-bit rounded multiplier.

In Fig. 4, we show more details on the accuracy of the LeNet-5 accelerators with the final approximate multiplier (i.e., approximate multiplier 5 in Table I) and with the 2-bit rounded multiplier after retraining. We can see that the accelerator with the final approximate multiplier always has a higher accuracy than the one with the 2-bit rounded multiplier.

Besides, as the approximate multiplier is generated by ALSRAC using the specific distribution from the NN computation, it is easy to retrain the NN model with the approximate multiplier. Thus, the training process does not require much time. Before retraining, the accuracy of the NN with the approximate multiplier is 12.03%. However, after one epoch, the NN accuracy improves significantly. More details are shown in the inset of Fig. 4. After the first epoch, the accuracy of the NN increases to more than 96.0%. Then, it fluctuates between 96.9% and 97.86%. From this, we can see that very few epochs are needed to retrain the NN built with the final approximate multiplier.

V. CONCLUSION

This paper proposes a new approach for designing low-power approximate computing-based NN accelerators: integrating ALS into the design loop. This approach potentially can obtain approximate computing components that fit better with the given NN, since ALS can take into account the NN computation pattern in designing the approximate computing components. We use a state-of-the-art ALS method, ALSRAC, to demonstrate the effectiveness of the proposed method. The experimental results show that the proposed method leads to an extremely small approximate multiplier with negligible accuracy loss for LeNet-5 on MNIST dataset.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2016.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition Conference*, 2016, pp. 770–778.

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations*, 2016, pp. 1–7.

[4] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[5] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe*, 2017, pp. 258–261.

[6] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2020.

[7] Z. Liu, K. Jia, W. Liu, Q. Wei, F. Qiao, and H. Yang, "INA: Incremental network approximation algorithm for limited precision deep neural networks," in *International Conference on Computer-Aided Design*, 2019, pp. 1–7.

[8] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *International Conference on Computer-Aided Design*, 2019, pp. 1–8.

[9] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *Design Automation Conference*, 2012, pp. 796–801.

[10] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation, and Test in Europe*, 2013, pp. 1367–1372.

[11] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Design Automation Conference*, 2016, pp. 128:1–128:6.

[12] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2017, pp. 344–351.

[13] S. Hashemi, H. Tan, and S. Reda, "BLASYS: Approximate logic synthesis using boolean matrix factorization," in *Design Automation Conference*, 2018, pp. 55:1–55:6.

[14] Z. Zhou, Y. Yao, S. Huang, S. Su, C. Meng, and W. Qian, "DALS: delay-driven approximate logic synthesis," in *International Conference on Computer-Aided Design*, 2018, pp. 86:1–86:7.

[15] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *Design Automation Conference*, 2020, pp. 1–6.

[16] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure," in *International Workshop on Logic and Synthesis*, 2006, pp. 15–22.

[17] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Computer Vision and Pattern Recognition Conference*, 2018, pp. 2704–2713.
[18] Y. Lecun and L. Bottou, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[19] "tiny-dnn," https://github.com/tiny-dnn/tiny-dnn.
[20] S. Yang, "Logic synthesis and optimization benchmarks," Microelectronics Center of North Carolina, Tech. Rep., 1991.
[21] A. Mishchenko *et al.*, "ABC: a system for sequential synthesis and verification, release 90703," http://people.eecs.berkeley.edu/~alanmi/abc/, accessed July 3, 2019.