2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)

A Cost-Efficient Iterative Truncated Logarithmic Multiplication for Convolutional Neural Networks

HyunJin Kim
School of EEE
Dankook University
Yongin, South Korea
Email: hyunjin2.kim@gmail.com

Min Soo Kim
Dept. of EECS
University of California
Irvine, CA, USA
Email: minsk1@uci.edu

Alberto A. Del Barrio
Dept. of Computer Architecture and Automation
Universidad Complutense de Madrid, Spain
Email: abarriog@ucm.es

Nader Bagherzadeh
Dept. of EECS
University of California
Irvine, CA, USA
Email: nader@uci.edu

***Abstract*—This paper proposes a cost-efficient approximate logarithmic multiplication for convolutional neural networks (CNNs), where two truncated logarithmic multipliers are connected for error correction. The proposed iterative logarithmic multiplication achieves low and unbiased average error while the hardware cost is significantly reduced by utilizing the truncated Mitchell multiplier and approximating error terms from the first stage. The proposed design has error characteristics that are suitable for neural network inferences, and the experiments on contemporary CNNs show that the proposed multiplier does not cause significant degradation on accuracy compared to exact multiplication.**

***Keywords*-approximate multiplier; convolutional neural network; logarithmic multiplication; Mitchell multiplier**

I. Introduction

CNNs are a type of artificial neural network with many layers between the input and output, and they have been widely applied to many fields with significant achievements. There have been various studies that seek to reduce the cost of neural network inferences. The neural network inferences are found to be resilient against the acceptable errors in the computation because of discrete classification characteristics [1]. While the previously proposed approximate multipliers can be well applied to the inference of simple networks, there was a clear trend that complex networks required high accurate computations, which motivate developing accurate and affordable approximate multiplication. The iterative logarithmic structure has the capability to enhance accuracy beyond the Mitchell multiplier [2], but repeating basic blocks adds significant costs to the design [3]. There is a strong need to reduce the cost of basic blocks so that the iterative logarithmic multiplication becomes affordable.

This paper proposes a low-cost two-stage approximate logarithmic multiplication for CNNs, where each stage adopts the truncated Mitchell multiplier. We present the detailed description of the proposed two-stage multiplier, including the techniques to convert the truncated Mitchell multiplier into an iterative design and transfer error terms between stages. The effect of the proposed design is simulated on multiple contemporary neural networks and discuss why our design performs well for CNNs.

II. Preliminaries

An n-bit unsigned integer number A can be expressed in logarithmic number system (LNS) format as:

$$A = (1 + x_A) \cdot 2^{k_A}, x_A \in [0, 1). \tag{1}$$

In (1), k_A and x_A are the characteristic number for indicating the location of the most significant bit (MSB) with the value of '1' and the fraction or mantissa of A, respectively. When $C = (1+x_C)\cdot 2^{k_C}$ and $C = A \cdot B$, the logarithmic conversion is approximated as follows:

$$\begin{cases} k_C = k_A + k_B + 1,\ x_C = x_A + x_B - 1, \\ \text{if } x_A + x_B \geq 1 \\ \\ k_C = k_A + k_B,\ x_C = x_A + x_B, \\ \text{if } x_A + x_B < 1. \end{cases} \tag{2}$$

It is denoted that MUL_{exact} and MUL_{appr} are results of the exact multiplication and approximate multiplication for the same input. When error is defined as $MUL_{exact} - MUL_{appr}$, formula for calculating the relative error (called $rerr$ later) is defined in [3] as:

$$rerr = \frac{MUL_{exact} - MUL_{appr}}{MUL_{exact}}. \tag{3}$$

In Mitchell multiplier [2], $rerr$ is calculated as:

$$rerr = \begin{cases} \frac{(1-x_A)\cdot(1-x_B)}{(1+x_A)\cdot(1+x_B)}, & \text{if } x_A + x_B \geq 1 \\ \frac{x_A \cdot x_B}{(1+x_A)\cdot(1+x_B)}, & \text{if } x_A + x_B < 1. \end{cases} \tag{4}$$

By truncating fractions in the approximate multipliers, hardware cost can be reduced with the loss of accuracy in [4], [5]. In addition, average error can be small by adding compensating value to the truncated fractions for unbiased errors, so that unbiased error can be positive or negative [4], [5].

978-1-7281-3366-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ARITH.2019.00029

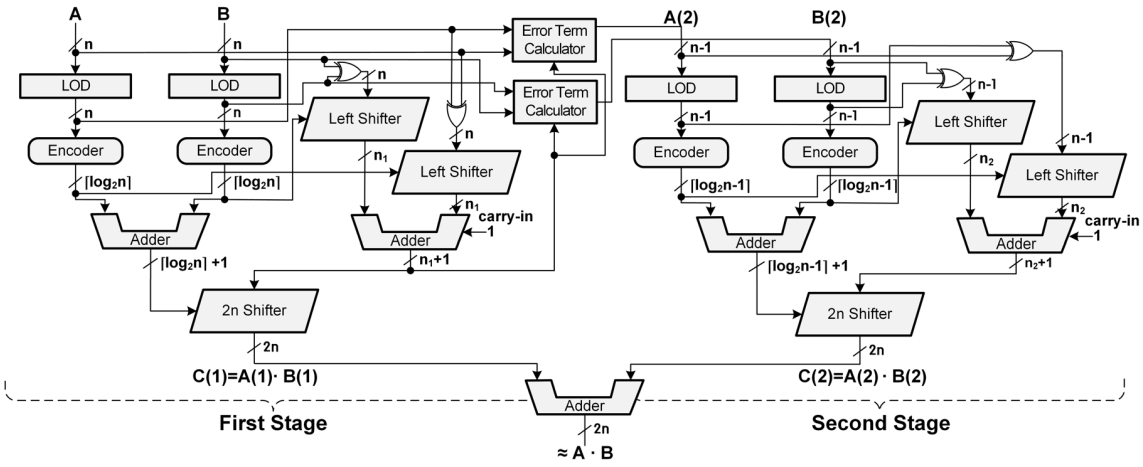

Figure 1. Structure of the proposed two-stage logarithmic multiplier.

III. Proposed Logarithmic Multiplier

Fig. 1 shows the structure of proposed n-bit two-stage logarithmic multiplier. In the first stage, leading-one detector (LOD) is needed to find characteristic numbers k_A and k_B for A and B. For n-bit A, the output of LOD also consists of n bits, where the output bit in the same position with the MSB of '1' in A has the value of '1' and other output bits from LOD are '0.' By encoding the n-bit output of LODs, $\lceil \log_2 n \rceil$-bit k_A and k_B are obtained.

In the truncated Mitchell multiplier for the first stage, n_1-bit fractions $x_{A(1)}$ and $x_{B(1)}$ are obtained by shifting $A - 2^{k_A}$ and $B - 2^{k_B}$ from XOR gates to the left by $n - k_A$ and $n - k_B$. For the truncations, n_1 high-order bits of fractions are output from the two left shifters in the right part of the first stage. In addition, carry-in '1' for averaging truncated values is taken in the n_1-bit adder. Another truncated Mitchell multiplier is connected iteratively in the second stage to increase accuracy in Fig. 1. In the proposed design, because of the truncation in fractions and addition of carry-in to the fractions, the error terms from the first stage depend on the range of $x_{A(1)} + x_{B(1)} + 2^{-n_1}$. In Fig. 1, two *Error Term Calculator*s calculate error terms $A(2)$ and $B(2)$, where the carry-out from the adder is connected into port *range*. By adopting 1's complement conversion, the usage of subtractors can be avoided. Therefore, $A(2)$ and $B(2)$ are formulated in (5) as:

$$\begin{cases} A(2) = (1 - x_A) \cdot 2^{k_A} - 1, \\ B(2) = (1 - x_B) \cdot 2^{k_B} - 1, \\ \text{if } x_{A(1)} + x_{B(1)} + 2^{-n_1} \geq 1 \\ \\ A(2) = x_A \cdot 2^{k_A}, \ B(2) = x_B \cdot 2^{k_B}, \\ \text{if } x_{A(1)} + x_{B(1)} + 2^{-n_1} < 1. \end{cases} \tag{5}$$

In the second stage, another truncated Mitchell multiplier takes $(n-1)$-bit error terms from the first stage to calculate error from the first stage. In the right part of Fig. 1, two left shifters output n_2 bits, where $n_2 < n - 1$. Like the first stage, carry-in '1' for averaging truncated values is taken in the n_2-bit adder. Finally, $C(1) = A(1) \cdot B(1)$ and $C(2) = A(2) \cdot B(2)$ from the first and second stages are summed to generate the approximate output for $A \cdot B$. In the first stage, zero detectors in LODs are adopted like [6]. When any input is zero, zero detectors make the final output zero, which is not shown for clarity in Fig 1.

In ideal two-stage multiplication, error terms from the first stage are transferred to the second stage without any change. Unlike the ideal two-stage iterative multiplication above, several other errors are considered in the proposed design. When fractions are truncated and carry-in value is added, additional errors are generated. When n_1 high-order bits in fractions are only adopted, the truncated fractions of A and B are denoted as $x_{A_{n_1}}$ and $x_{B_{n_1}}$, respectively. When n_1 and n_2 high-order bits are adopted in the fractions of the first and second stages, the error is formulated as:

$$\begin{aligned} &(x_{A_{n_1}} + x_{B_{n_1}} - 2^{-n_1}) \cdot 2^{k_A + k_B} \\ &\quad + (x_{A_{n_2}} + x_{B_{n_2}} - 2^{-n_2}) \cdot 2^{k_{A(2)} + k_{B(2)}}. \end{aligned} \tag{6}$$

In Table I, relative errors are summarized according to n and n_1 when $n_2 = 2$. Term $rerr_{min}$ stands for the minimum $rerr$. Average and average absolute relative errors are denoted as $rerr_{avg}$ and $|rerr|_{avg}$, respectively. During simulations, one million pairs of two inputs were randomly selected to obtain $rerr_{avg}$ and $|rerr|_{avg}$. Because of carry-in '1' in the truncated Mitchell multiplier, $rerr_{min}$ can be negative, where absolute value of $rerr_{min}$ can decrease by increasing n_1. Because $rerr$ can be negative, $|rerr|_{avg}$ is somewhat larger than $rerr_{avg}$. But the difference between $rerr_{avg}$ and $|rerr|_{avg}$ decreases with n_1.

Several existing and proposed designs were coded as combinational multipliers and were evaluated in terms of

Table I
SUMMARY OF RELATIVE ERRORS

n	n_1	$rerr_{max}$	$rerr_{min}$	$rerr_{avg}$	$\|rerr\|_{avg}$
8	4	11.1%	-6.25%	-1.09%	1.77%
	5	11.1%	-3.33%	-0.83%	1.11%
	6	11.1%	-2.50%	-0.58%	0.76%
	7	11.1%	-2.08%	-0.32%	0.57%
	8	11.1%	-1.88%	-0.06%	0.44%
16	4	11.1%	-6.25%	0.10%	1.44%
	5	11.1%	-3.33%	0.11%	0.77%
	6	11.1%	-2.50%	0.11%	0.46%
	7	11.1%	-2.08%	0.12%	0.33%
	8	11.1%	-1.88%	0.12%	0.28%
32	4	11.1%	-6.25%	0.11%	1.44%
	5	11.1%	-3.33%	0.12%	0.77%
	6	11.1%	-2.50%	0.12%	0.46%
	7	11.1%	-2.08%	0.13%	0.33%
	8	11.1%	-1.88%	0.13%	0.28%

Table II
COMPARISON OF RELATIVE ERROR AND COST

n	design	$rerr_{max}$ (%)	$rerr_{avg}$ (%)	critical path (ns)	area (um^2)	power (uW)
8	Booth[a]	0	0	1.3	613	403
	MM[b]	11.11	3.76	1.3	446	217
	IM[c]	6.25	0.83	1.9	1,133	590
	PROP [d]	11.11	-1.09	2.6	786	370
16	Booth[a]	0	0	2.8	2,507	1,760
	MM[b]	11.11	3.85	2.3	1,168	602
	IM[c]	6.25	0.99	3.7	2,901	1,410
	PROP [e]	11.11	0.11	5.1	1,638	739
32	Booth[a]	0	0	5.4	10,139	6,750
	MM[b]	11.11	3.85	4.2	3,418	1,640
	IM[c]	6.25	0.99	6.5	7,674	3,680
	PROP [e]	11.11	0.12	7.9	3,102	1,370

[a] Radix-4 Booth multiplier
[b] Mitchell multiplier [2]
[c] Two-stage Babic's iterative multiplier [7]
[d] Proposed two-stage multiplier with $n_1 = 4, n_2 = 2$
[e] Proposed two-stage multiplier with $n_1 = 6, n_2 = 2$

area and power. In all designs, LODs and zero detector in [6] were adopted for apple-to-apple comparison. Then, the codes were synthesized using Synopsys Design Compiler and 32nm standard cell library from Synopsys, where 100MHz frequency was the targeted goal in *Ultra* mode. In Table II, relative errors and costs are compared according to bit width n. Unlike the cases of $n = 16$ or $n = 32$, the maximum number of bits that can be truncated was small when $n = 8$, so that the comparison in Table II assumes just $n_1 = 4$ for $n = 8$ and $n_1 = 6$ for $n = 16$ and $n = 32$. For all proposed designs in Table II, $n_2 = 2$ was adopted for the second stage. In addition, exact radix-4 Booth multiplier (*Booth*), Mitchell multiplier [2] (*MM*), and two-stage iterative multiplier [7] (*IM*) were also coded.

When $n = 8$, delay and area were greater than those of the exact multiplier. On the other hand, when $n = 16$ and $n = 32$, area and power were reduced by 69.4-34.7% and 79.7-58.0% compared to those of exact Booth multiplier. The two-stage structure of the proposed design made overall costs greater than that of Mitchell multiplier for $n = 8$ and $n = 16$. However, the average relative error was significantly reduced. When $n = 32$, both area and power were reduced by 9.2% and 16.5%, respectively. Compared to the two-stage iterative multiplier in [7], the proposed design reduced area and power on average by 44.6% and 48.1%, respectively.

Because error terms from the first stage are determined depending on the sum of fractions, delay can increase in the proposed design. In our experiments, when applying the pipelining to the 16-bit multiplier of [3], delay was reduced up to 2.35 (ns). However, area was increased to 7,273 (um^2). In the example with four registered stages for the proposed design, delay and area of the synthesized result were 2.24 (ns) and 2,999 (um^2), respectively. Compared to the pipelined design [3], because the truncated Mitchell multiplier in each stage can be simply implemented, smaller area was required with reduced delay.

IV. EXPERIMENTS ON CNNS

The evaluation of the applicability of our design to neural networks was done by testing the proposed multiplier in the Caffe deep learning framework [8]. In the experiments, those evaluations using logarithmic multiplications adopted 16 integer bits and 16 fractional bits for target neural networks. All approximate multipliers were assumed to perform multiplication with unsigned numbers. For the computations with signed numbers in neural networks, 2's complement conversion was used for inputs and outputs of multipliers. For experimentations, CIFAR-10 [9] and ImageNet [10] ILSVRC2012 validation datasets were adopted using 5,000 images for each run.

In [4], it was explained that the approximate multipliers were not suitable because large error can be critical problem in the convergence of the gradient descent of backpropagation. Therefore, we focused just on the experiments of inference, where the neural networks were trained using floating-point arithmetic operations and then adopted the approximate multiplier in the inference.

To compare with other existing multipliers, inferences on neural network models in Table III were performed. The performance of inferences of the fixed-point multiplications was nearly close to that of floating-point multiplications. Even though $rerr_{max}$ of the proposed design is greater than that of the Mitchell multiplier, Table III shows the proposed logarithmic multiplier outperforms the Mitchell multiplier. Therefore, it was concluded that large $rerr_{max}$ from the corner cases in the proposed design was not critical in the inferences. While $rerr_{max}$ of the proposed design was greater than that of the two-stage iterative multiplier [7],

Table III
COMPARISON WITH EXISTING MULTIPLIERS

DataSet	Model	Multiplier	Top-1 (%)	Top-5 (%)
NiN [11]	CIFAR-10	FLOAT[a]	89.4	-
		FIXED[b]	89.4	-
		MM[c]	88.7	-
		IM[d]	89.4	-
		PROP[e]	89.5	-
AlexNet [12]	ImageNet	FLOAT[a]	57.0	81.3
		FIXED[b]	57.0	81.3
		MM[c]	56.8	80.8
		IM[d]	56.8	81.3
		PROP[e]	56.9	81.3
GoogLeNet [13]	ImageNet	FLOAT[a]	68.3	88.4
		FIXED[b]	68.3	88.4
		MM[d]	67.1	87.5
		IM[d]	68.3	88.2
		PROP[e]	68.3	88.3
ResNet-50 [14]	ImageNet	FLOAT[a]	74.3	90.9
		FIXED[b]	74.2	90.9
		MM[c]	72.4	90.0
		IM[d]	73.9	90.9
		PROP[e]	73.8	90.6

[a] Original Caffe using floating-point multiplications
[b] Fixed-point multiplications
[c] Mitchell multipliers [2]
[d] Two-stage Babic's iterative multipliers [7]
[e] Proposed two-stage multiplier with $n_1 = 6, n_2 = 2$

the performance was slightly better in several models. In the evaluation with ResNet model for $n_1 = 6$, Top-5 accuracy was dropped by just 0.3%. For $n_1 = 8$, Top-1 and Top-5 accuracies with ResNet model were 74.1% and 90.9%, which were nearly close to those of the fixed-point multiplications. As shown in Table I and Table II, $rerr_{max}$ and $rerr_{min}$ in [7] were just 6.25% and 0%, but $rerr_{avg}$ and $|rerr|_{avg}$ of the proposed design were smaller than that of the two-stage iterative multiplier in [7]. In the proposed design, carry-in '1' was taken in the proposed design to compensate and unbiasing error from the bit truncation, while the iterative multiplier in [7] had the biased positive error. The comparison with the existing two-stage iterative multiplier led to the conclusion that the unbiased error and small $rerr_{avg}$ and $|rerr|_{avg}$ of the proposed design can be helpful for applying to neural networks.

Overall, the inference accuracies for the proposed design in Table III were close to those of fixed-point multiplications and floating-point multiplications without significant performance degradation. By adopting the approximation of error term using 1's complement conversion and the truncations of fractions, the worst-case value can increase in the design. But it did not incur substantial impact on the inference accuracies for the models of Table III.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2017R1D1A1B03030348), grants S2018/TCS-4423 and TIN 2015-65277-R, and Center for Pervasive Communications and Computing (CPCC) at UC, Irvine.

REFERENCES

[1] M. S. Kim, A. A. Del Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of mitchell's approximate logarithmic multiplication for convolutional neural networks," Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, 2018, pp. 617–622.

[2] J. N. Mitchell, "Computer multiplication and division using binary logarithms," IRE Transactions on Electronic Computers, no. 4, pp. 512–517, 1962.

[3] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," Microprocessors and Microsystems, vol. 35, no. 1, 2011, pp. 23–33.

[4] M. S. Kim, A. A. D. B. Garcia, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient mitchell's approximate log multipliers for convolutional neural networks," IEEE Transactions on Computers, 2018, doi: 10.1109/TC.2018.2880742.

[5] S. Hashemi, R. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," Proc. IEEE/ACM International Conference on Computer-Aided Design, IEEE Press, 2015, pp. 418–425.

[6] K. H. Abed and R. E. Siferd, "Cmos vlsi implementation of a low-power logarithmic converter," IEEE Transactions on Computers, vol. 52, no. 11, pp. 1421–1433, 2003.

[7] Z. Babic, A. Avramovic, and P. Bulic, "An iterative mitchell's algorithm based multiplier," Proc. IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2008), IEEE, 2008, pp. 303–308.

[8] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," Proc. 22nd ACM international conference on Multimedia, ACM, 2014, pp. 675–678.

[9] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[10] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision (IJCV), vol. 115, no. 3, 2015, pp. 211–252.

[11] M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv preprint arXiv:1312.4400, 2013.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, 2012, pp. 1097–1105.

[13] C. Szegedy et al., "Going deeper with convolutions," Proc. IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv preprint arXiv:1512.03385, 2015.