Original scientific paper
https://doi.org/10.33180/InfMIDEM2023.203

Informacije MIDEM
Journal of Microelectronics, Electronic Components and Materials
Vol. 53, No. 2(2023), 79 – 86

An Energy-efficient and Accuracy-adjustable bfloat16 Multiplier

Ratko Pilipović[1], Patricio Bulić[1], Uroš Lotrič[1]

[1]Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

Abstract: The approximate multipliers have been extensively used in neural network inference, but due to the relatively large error, they have yet to be successfully deployed in neural network learning. Recently, the bfloat16 format has emerged as a viable number representation for neural networks. This paper proposes a novel approximate bfloat16 multiplier with on-the-fly adjustable accuracy for energy-efficient learning in deep neural networks. The size of the proposed multiplier is only 62% of the size of the exact bfloat16 multiplier. Furthermore, its energy footprint is up to five times smaller than the footprint of the exact bfloat16 multiplier. We demonstrate the advantages of the proposed multiplier in deep neural network learning, where we successfully train the ResNet-20 network on the CIFAR-10 dataset from scratch.

Keywords: approximate computing; deep neural networks; energy-efficient processing; bfloat16 multiplier

Energijsko učinkovit približni množilnik v zapisu bfloat16 z nastavljivo natančnostjo

Izvleček: Približni množilniki so se izkazali za zelo primerne pri sklepanju z nevronskimi mrežami, vendar zaradi relativno velike napake še niso bili uspešno uporabljeni pri učenju globokih nevronskih mrež. Pred kratkim se je za predstavitev realnih števil v nevronskih mrežah začel uveljavljati zapis bfloat16. V članku predlagamo nov približni množilnik v zapisu bfloat16 s sprotno nastavljivo natančnostjo za energetsko učinkovito učenje v globokih nevronskih mrežah. Velikost predlaganega množilnika je samo 62 % velikosti natančnega množilnika v zapisu bfloat16. Poleg tega je njegov energijski odtis do petkrat manjši od odtisa natančnega množilnika bfloat16. Uporabnost predlaganega množilnika predstavimo na primeru učenja globokih nevronskih mrež, kjer uspešno naučimo mrežo ResNet-20 na naboru podatkov CIFAR-10.

Ključne besede: približno računanje; globoke nevronske mreže; energijsko učinkovito računanje; množilnik v zapisu bfloat16

* Corresponding Author's e-mail: patricio.bulic@fri.uni-lj.si

1 Introduction

Neural network capability of learning from data and generalising the gained knowledge makes them a very popular modelling tool in various application fields. The popularity growth in the last years can be attributed to the deep models, which pose considerable requirements to the processing hardware. Thus, new hardware solutions are being developed continuously to keep the processing hardware on par with the computing demands.

Approximate computing has emerged as a popular strategy for area- and energy-efficient circuit design, where the challenge is to achieve the best trade-off between design efficiency and accuracy. Efficient designs come at the cost of accuracy reduction and vice versa. Nevertheless, approximate computing perfectly fits neural networks, which, to a certain extent, tolerate or even adapt to an error caused by noisy input data or erroneous computation. Widely used approaches in approximate computing are precision scaling and approximate arithmetic.

In precision scaling [1], we use fewer bits to represent numeric values rather than executing all the required mathematical operations with the full representation. Several standards for the floating-point presentation recently appeared: IEEE 754-2019 for half-precision

How to cite:
R. Pilipović et al., "An Energy-efficient and Accuracy-adjustable bfloat16 Multiplier", Inf. Midem-J. Microelectron. Electron. Compon. Mater., Vol. 53, No. 2(2023), pp. 79–86

[2], posit format with dynamic range and mantissa [3] and Google's bfloat16, targeting the machine-learning workloads [4]. Storing the numeric values with fewer bits reduces the size of arithmetic circuits and their complexity. Besides, it saves on-chip memory and reduces the amount of data that must be transferred, improving speed.

Multiplication represents a ubiquitous arithmetic operation in neural network processing. Moreover, multipliers are complex circuits that importantly affect a processing hardware's area and energy footprint. Hence, the applications can benefit in terms of power and area consumption by replacing the exact multiplier with an approximate one. The approximate multiplier design can originate in the logarithmic approximation of numerical values [5-8] or non-logarithmic approaches, like discarding some stages in Booth multipliers [9-11]. Although most approximate multipliers are designed for fixed-point arithmetic, many floating-point designs, capable of presenting numerical values in a wider range, have appeared lately.

There have been several attempts to use approximate integer multipliers in neural network learning [12-14]. The authors of these studies report that the learning was successful, but they mainly worked with tiny neural networks. To the best of our knowledge, there has yet to be a successful attempt to train large-scale neural networks using approximate multipliers. In neural network learning, we need higher precision arithmetic, so until now, neural networks have mainly been trained using the exact floating-point multipliers [3], [15].

Common to most of the existing designs is that their accuracy can be adjusted at the design time. As such, they can perfectly fit the targeting application but fail for many others. However, many applications need adjustable accuracy during run time. In neural network processing, for example, we can use lower accuracy during the inference phase but need much higher accuracy during the learning phase. Moreover, some parts of an application may still require exact multiplication. For such an application, it would be beneficial to design a multiplier capable of handling all accuracy requirements, thus avoiding putting a plethora of multipliers on a chip and not exploiting them simultaneously.

Several precision-tuning 32-bit floating-point multipliers for deep neural network processing have recently been proposed. The work [16] proposes the 32-bit floating-point approximate PAM multiplier with run-time customisation, which can successfully replace a single-precision floating-point multiplier in some deep neural networks and image-processing applications. In [17], the authors proposed a 32-bit iterative approximate floating-point multiplier based on two-dimensional pseudo-Booth encoding. The accuracy of the proposed multiplier is tuned by three parameters: iteration, encoder's radix, and word length after truncation. To our knowledge, the only state-of-the-art approximate 16-bit bfloat multiplier is proposed in [15]. This variable-precision approximate multiplier uses the bfloat16 format for operand representation and the intermediate conversion of product exponent to the posit encoding to control the mantissa multiplication accuracy. All these multipliers were used only in the inference phase in deep learning models and in image-processing applications, where neglectable degradation in accuracy was observed.

A design that would suit most applications should be able to multiply with the required accuracy, not excluding exact computation, and accept a wide range of numeric values. In this paper, we propose an efficient and accuracy-adjustable approximate 16-bit multiplier for operands presented in the bfloat16 format, which does not require any hardware reconfiguration to adapt accuracy and demonstrates its applicability in the neural network inference and learning phases.

In the remainder of the paper, we first detail the proposed BFILM multiplier design. Section 3 shows the hardware characteristics of the design and demonstrates the BFILM multiplier usability in neural network inference and learning. Lastly, we conclude the paper with the main findings.

2 The design of BFILM multiplier

The proposed brain float iterative logarithmic multiplier (BFILM) operates on numerical values in the bfloat16 format. The advantage of representing the numerical value 0 in the bfloat16 format is, that it keeps one sign bit $s(0)$ and the 8-bit exponent $e(0)$ equal to the IEEE 754 single-precision floating-point format but shortens the mantissa $m(0)$ to 7 bits. Thus, it enables using tiny numerical values, important in the neural network learning phase [18] for example. While the multiplier determines the sign and the exponent exactly, it follows the idea of the approximate iterative logarithmic multiplier to compute the mantissa. The number of steps, which determine the accuracy of the multiplier, can be changed on the fly.

Fig. 1 shows the structure of the BFILM multiplier, which takes operands 0_1 and 0_2 to compute the approximate product P_{approx}. The multiplier consists of a straightforward circuit for determining the sign of the product

and two loosely connected circuits for determining the product's exponent and mantissa.

2.1 The exponent circuitry

The exponent circuity in Fig. 1 incorporates two adders. We must add both operands' exponents to get the product's exponent. However, the bfloat16 format uses the offset-binary representation of the exponent, with the zero offset being 127. To correctly code the product's exponent, we need an additional adder to subtract the offset. The logic connected to the carry input c_{in} of the first adder covers the situations when the product's exponent must be normalised due to the large approximate product P_a obtained from the mantissa multiplier.

2.2 The mantissa circuitry

The mantissa circuitry in Fig. 1 comprises the mantissa multiplier and the mantissa normalizer. The mantissa stores only the fractional bits, to which we must prepend the leading one to get an 8-bit fixed point unsigned number at the input to the mantissa multiplier. The multiplication results is a product, given in 16-bit unsigned fixed-point format with two integer bits and 14 fractional bits, of which we take only the nine most significant bits to the output P_a of the mantissa multiplier. We form the product's mantissa m(P_{approx}) regarding the integer part of the output P_a. When it is greater than one with P_a [8] set, we normalise the result by shifting the radix point one place to the left. To do so, we increment the product's exponent and take the middle seven bits P_a [7:1]. In all other cases, normalisation is unnecessary, and the product's mantissa equals the seven least significant bits P_a [6:0].

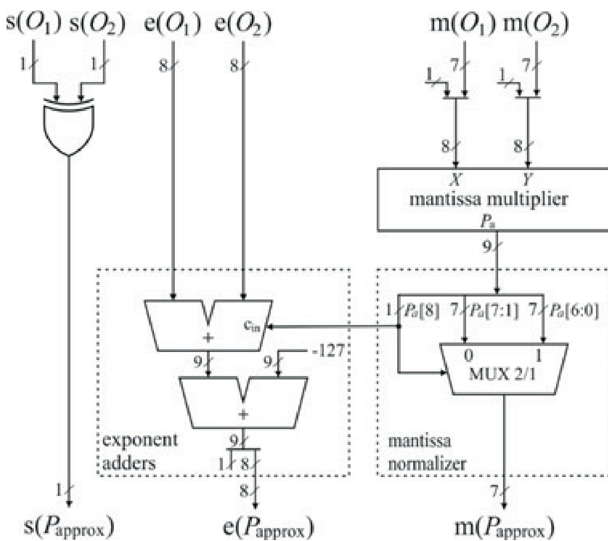

Figure 1: The circuitry of the 16-bit bfloat multiplier.

An important component of the BFILM multiplier is the approximate mantissa multiplier that relies on the iterative logarithmic multiplier (ILM) [7]. Suppose we have two non-negative 8-bit operands *x* and *y*, expressed as the sum of the leading bit and the residuum, $x = 2^{k_x} + r_x$ and $y = 2^{k_y} + r_y$, which multiply to the product

$$p = xy = x\left(2^{k_y} + r_y\right) = x2^{k_y} + xr_y \\ = x2^{k_y} + 2^{k_x} r_y + r_x r_y \quad . \tag{1}$$

By summing up the first-order Taylor expansions of

$$\log_2 x = k_x + \log_2\left(1 + r_x 2^{-k_x}\right) \\ = k_x + \ln\left(1 + r_x 2^{-k_x}\right)\log_2 \mathrm{e} \\ \approx k_x + r_x 2^{-k_x} \log_2 \mathrm{e} \tag{2}$$

and $\log_2 y \approx k_y + r_y 2^{-k_y} \log_2 \mathrm{e}$, we get the approximation

$$\log_2 p \approx \left(k_x + k_y\right) + 2^{-\left(k_x + k_y\right)}\left(r_x 2^{k_y} + r_y 2^{k_x}\right)\log_2 \mathrm{e} \\ \approx \left(k_x + k_y\right) + \log_2\left[1 + 2^{-\left(k_x + k_y\right)}\left(r_x 2^{k_y} + r_y 2^{k_x}\right)\right] \tag{3}$$

By taking the antilogarithm of $\log_2 p$ approximation, we obtain an approximate product

$$p_a = 2^{\left(k_x + k_y\right)}\left[1 + 2^{-\left(k_x + k_y\right)}\left(r_x 2^{k_y} + r_y 2^{k_x}\right)\right] \\ = 2^{\left(k_x + k_y\right)} + r_x 2^{k_y} + r_y 2^{k_x} \\ = \left(2^{k_x} + r_x\right)2^{k_y} + r_y 2^{k_x} \\ = x2^{k_y} + r_y 2^{k_x} \tag{4}$$

which equals equation (1) with the last term omitted. Thus, computing the product approximation p_a requires only two shifts and an addition, completely avoiding multiplication of the term $r_x r_y$.

The ILM core circuitry in Fig. 2 computes the approximate product and both residua. The leading one detectors extract the leading one bits 2^{k_x} and 2^{k_y} and their characteristic numbers k_x and k_y from operands *x* and *y*. We need both leading one bit to compute the residua and the characteristic numbers to do the required shifts of the operand *x* and the residuum r_y. The truncated barrel shifters output only the nine most significant bits required in further processing, thus importantly reducing their size and the size of the adder.

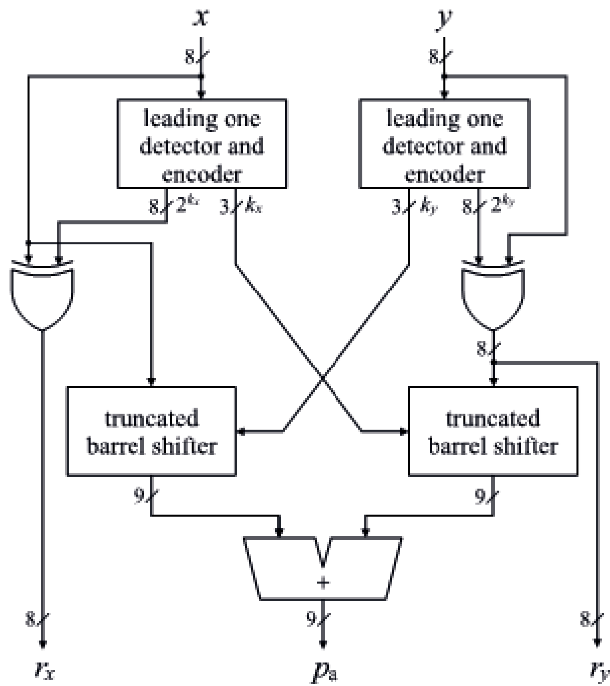

Figure 2: The circuitry of the ILM core.

The relative error of the product $(p - p_a)/p = r_x r_y /p$ can be as high as 25 %. To reduce it, we can iteratively repeat the above procedure by multiplying residua r_x and r_y and adding the result to the current approximation. The procedure can be repeated until at least one residuum becomes zero, thus achieving an error as small as necessary.

The mantissa multiplier shown in Fig. 3 comprises the ILM core, two multiplexers, and an accumulator to iteratively refine the approximate mantissa product P_a. In the initial ILM step ($I = 1$), the multiplexers pass the operands *X* and *Y* to the ILM core, while in the next ILM steps (I>1), the multiplexers feed the ILM core with residua r_x and r_y from the previous ILM step. The accumulator keeps the approximation of the mantissa product, which is in each ILM step increased by the value p_a. To comply with the circuitry presented in Fig. 1, the accumulator needs to keep only the nine most significant bits.

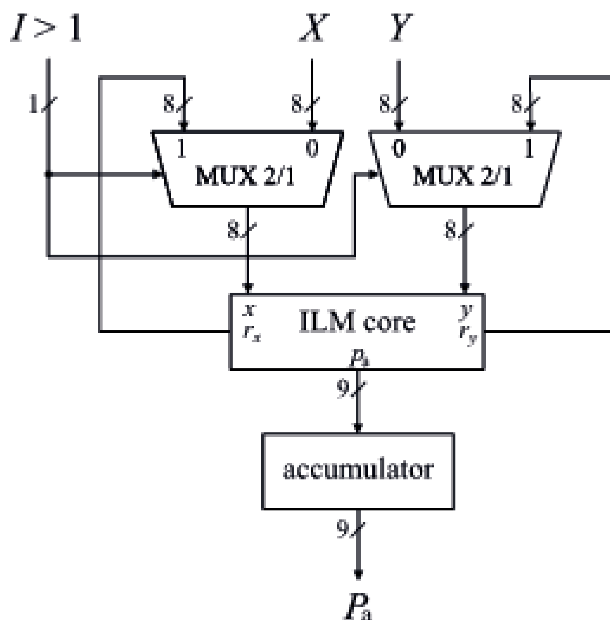

Figure 3: The circuitry of the approximate mantissa multiplier.

At this point, we would like to emphasize that the proposed multiplier does not require any hardware reconfiguration if we want to perform more than one ILM step. For example, when more ILM steps are required, we only need to feed the residua r_x and r_y (Fig. 2) back to the input of the ILM core as presented in Fig. 3. In this case, the multiplexers choose what goes to the ILM core: the new operands, *X* and *Y*, or the residua from the previous iteration, r_x and r_y. In the actual implementation, of course, we must add registers at the input of multiplexers, but these are not shown for simplicity.

3 Results

3.1 Hardware performance

We implement the multipliers in Verilog and synthesise them to the SkyWater PDK 130 cell library using OpenLane [19-21]. The library consists of a 130 nm technology with an operating voltage of 1.8 V, and five metal layers [22-23]. The timing constraints, used for all evaluated designs, specify clock-related parameters, which affect synthesis and timing analysis. We set a clock signal with a period of 10 ns, hence not violating a critical path. To evaluate the power, we use timing with a 100 MHz virtual clock (by definition, a virtual clock is a clock that has no real source in the design and is commonly used to specify delay constraints during static timing analysis), load capacitance equal to 33.442 fF (PDK default) and supply voltage equal to 1.8 V.

We analysed the hardware performance of the BFILM multiplier in terms of power, area, delay, and power-delay-product (PDP) and compare it with the exact bfloat16 multiplier. Table 1 shows that the BFILM multiplier outperforms the exact multiplier in all hardware metrics; its energy consumption estimated through PDP is even more than five times smaller.

Table 1: The synthesis results for the examined multipliers.

Multiplier	Delay [ns]	Power [*uW*]	Area [*um*^{2}]	PDP [fJ]
exact bfloat16	2.89	869	6120	2590
BFILM	**1.67**	**298**	**3796**	**498**

Table 2 compares hardware characteristics of the state-of-the-art variable-accuracy bfloat16 multipliers. The results are given as relative values to the standard reference implementations of the exact bfloat16 multiplier. The BFILM multiplier, with its very slim design, outperforms the recently proposed BFLP16-prop multiplier [15] in all aspects.

Table 2: Comparison of the bfloat16 multipliers regarding hardware gains relative to the exact bfloat16 multiplier.

Multiplier	Delay [%]	Power [%]	Area [%]	PDP [%]
exact bfloat16	100	100	100	100
BFLP16-prop [15]	104	58	81	59
BFILM	**58**	**33**	**62**	**19**

Since the BFILM multiplier does not require reconfiguration or additional hardware for more accurate processing, the multiplier's size (area) and power are preserved for an arbitrary number of the ILM steps. Of course, with the additional ILM steps, it is necessary to observe that residua r_x and r_y must be multiplied once or twice and added to the final product. Therefore, in this case, the processing time required to calculate the product increases linearly with the number of the ILM steps and thus does also the energy consumption. We assess different configurations of the BFILM multiplier in terms of delay, energy consumption (PDP) and the mean relative error distance (MRED), and present them in Table 3. For easier comparison, the delay and energy consumption are given relative to the values of the exact bfloat16 multiplier.

The proposed multiplier with two or three ILM steps has a lower energy consumption than the exact bfloat16 multiplier and the BFLP16-prop multiplier [15]. Moreover, the BFILM multiplier with two ILM steps is not much slower than the state-of-the-art BFLP16-prop multiplier [15]. However, the BFILM multiplier with only one ILM step has a rather large error, which with two ILM steps comes close to the BFLP16-prop multiplier's MRED, and then drops by order of magnitude with each additional ILM step.

Table 3: Comparison of delay, PDP, and the MRED error for the different number of ILM steps in the BFILM multiplier.

Multiplier	Delay [%]	PDP [%]	MRED [10^{-3}]
exact bfloat16	100	100	0
BFLP16-prop [15]	104	59	3.50
BFILM, 1 ILM step	**58**	**19**	**91.21**
BFILM, 2 ILM steps	**115**	**38**	**9.08**
BFILM, 3 ILM steps	**173**	**58**	**0.86**

These results suggest that the BFILM multiplier should fit well with error-resilient applications where low-energy consumption is an important goal and where most of the time the BFILM multiplier with a small number of ILM steps could be used. An important feature of the BFILM multiplier is that we can control the product accuracy by adjusting the number of ILM steps without hardware modification, ultimately leading even to removing the exact multiplier from the circuitry.

3.2 Impact on neural network learning

Convolutional neural networks achieve remarkable performance in visual recognition tasks [24]. However, the learning and inference of convolutional neural networks are computationally demanding tasks that involve many multiplications. Nevertheless, convolutional neural networks are error-tolerant models, making them perfect candidates for employing approximate multipliers. Therefore, we assess the influence of the proposed multiplier on the performance of the inference and learning phases.

To evaluate the BFILM multiplier, we select the ResNet-20 convolutional neural network [25-26] and the CIFAR-10 dataset [27]. We change the number representation in the ResNet-20 convolutional neural network from the single-precision floating-point format to the bfloat16 format. In the experiments, we use the Caffe framework [28], where we replace the calls to the cuBLAS multiplication routines with the calls to our own GPU kernels, which emulate the proposed BFILM multiplier.

The neural network learns using the predetermined split of the dataset to train and test sets [27]. Before learning, we preprocess the images by subtracting their mean value. Besides, we quantify the ResNet-20 single-precision floating-point weights to the bfloat16 format representation by simply discarding the last 16 bits of the floating-point mantissa. In the learning phase, we optimize the multinomial logistic loss function [29] with the Nesterov momentum algorithm [30]. The learning starts with randomly initialised weights. In all experiments, we train the network for 64000 epochs.

In the first experiment, we evaluate the influence of the proposed multiplier on the ResNet-20 classification accuracy. As the BFILM multiplier is configurable in terms of the number of steps affecting the multiplication error, we test several BFILM configurations. In the tested configurations, BFILM-1-1, BFILM-1-2, BFILM-2-2 and BFILM-2-3, the first number denotes the number of ILM steps in the inference phase, while the second number denotes the number of ILM steps used in the learning phase.

Table 4 shows the classification accuracy of the CIFAR-10 dataset. For each configuration, we list the average value and standard deviation over five runs. Significant multiplication error of BFILM-1-1 leads to low classification accuracy. Increasing the number of the ILM steps in the inference and learning phase improves classification accuracy. For example, with BFILM-2-2 and BFILM-2-3, the classification accuracy is almost the same as with the exact bfloat16 multiplier.

Table 4: Performance of the ResNet-20 convolutional neural network on the CIFAR-10 dataset using bfloat16 multipliers.

Multiplier	Test set classification accuracy [%]
exact bfloat16	91.50 ± 0.10
BFILM-1-1	86.32 ± 1.26
BFILM-1-2	90.98 ± 0.15
BFILM-2-2	91.30 ± 0.30
BFILM-2-3	91.40 ± 0.20

Also, we can see from the results for BFILM-1-1 and BFILM-1-2 that increasing the number of the ILM steps in the learning phase positively affects classification performance. On the other hand, a further increase in the number of steps in the inference phase from BFILM-1-2 to BFILM-2-2 has much less impact. Moreover, according to Table 3, BFILM-1-2 has a very small energy footprint and thus could be sufficient for neural network inference and learning.

The second experiment highlights the advantage of the on-the-fly accuracy adaptation of the BFILM multiplier, which can help in faster and more energy-efficient neural network learning. The idea is to start with one ILM step in the inference and learning phase to save energy and later, when model performance improves, increase the number of the ILM steps to further refine the result.

Fig. 4 shows the outcome of the learning process on the training and testing set for five separate runs, each with randomly initialised neural network weights. For the loss (red) and the accuracy (green), we show the span of obtained values and the curve averaged over all runs. We see that with the BFILM-1-1 configuration, the model improves rapidly and reaches a classification accuracy of more than 60 % in only 10000 epochs. At this point, we use an additional ILM step in the learning phase (BFILM-1-2) to improve the model's convergence and achieve more than 99.4 % of the accuracy of the exact bfloat16 multiplier. However, if the accuracy still needs to be increased for some applications, we can enhance the model by training it with additional ILM steps.

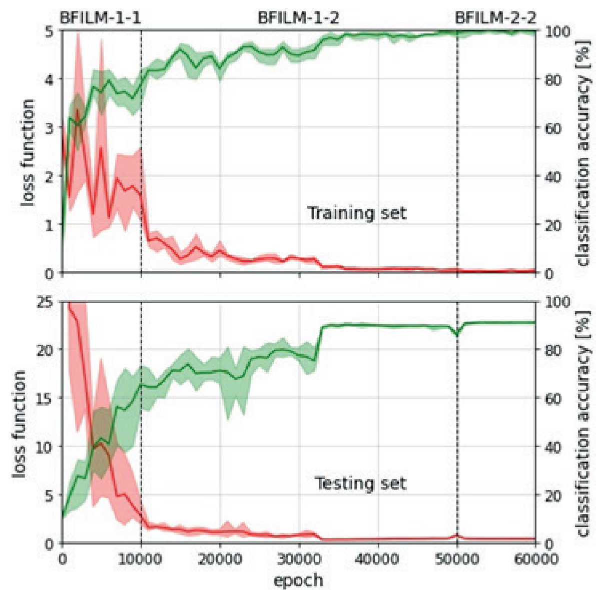

Figure 4: Varying configuration of BFILM during the learning phase.

4 Conclusion

In this paper, we proposed a novel approximate bfloat16 multiplier with adjustable accuracy, which can be achieved without any hardware reconfiguration. Instead, the proposed BFILM multiplier iteratively uses an approximate logarithmic multiplier core to reduce the error. This way, we avoid using additional error refinement circuits, keeping the design small and energy efficient. The primary purpose of the proposed design is to use it in deep neural network processing in the inference and learning phases. We apply the BFILM multiplier in the ResNet-20 convolutional neural network to classify the CIFAR-10 dataset. We demonstrate the impact of various BFILM configurations on the neural network learning process and classification accuracy. The results show that we can easily adjust the multiplier's accuracy according to the application's requirements. The main advantage of the on-the-fly adaptation of the BFILM multiplier comes to expression during the learning phase. The results prove that we can start with one ILM step in the inference and learning phase to save energy and later, when model performance improves, increase the number of the ILM steps to refine the result further. In future work, we aim to develop an algorithm that could optimize the learning process in terms of speed and efficiency by automatically adapting the ILM steps to the BFILM multiplier when needed.

5 Acknowledgments

This research was supported by Slovenian Research Agency under Grants P2-0359 (National research program Pervasive computing), P2-0241 (Synergy of the technological systems and processes) and by Slovenian Research Agency and Ministry of Civil Affairs, Bosnia and Herzegovina, under Grant BI-BA/21-23-033 (Bilateral Collaboration Project).

6 Conflict of Interest

The authors declare no conflict of interest.

The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results.

7 References

1. G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, mar 2022. https://doi.org/10.1145/3527156.
2. "IEEE standard for floating-point arithmetic,"" 2019, IEEE Std 754-2019 (Revision of IEEE 754-2008).
3. R. Murillo, A. A. Del Barrio Garcia, G. Botella, M. S. Kim, H. Kim, and N. Bagherzadeh, "Plam: a posit logarithm-approximate multiplier," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.
4. H. Kim, "A low-cost compensated approximate multiplier for bfloat16 data processing on convolutional neural network inference," *ETRI Journal*, vol. 43, no. 4, pp. 684–693, 2021. https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.2020-0370.
5. J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, Aug. 1962.
6. V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition," *IEEE Transactions on Computers*, vol. 55, no. 12, pp. 1523–1535, Dec. 2006. https://doi.org/10.1109/TC.2006.198.
7. Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011. https://doi.org/10.1016/j.micpro.2010.07.001.
8. M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 660–675, Dec. 2019. https://doi.org/10.1109/TC.2018.2880742.
9. V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 421–430, Nov. 2018. https://doi.org/10.1109/TVLSI.2017.2767858.
10. H. Waris, C. Wang, and W. Liu, "Hybrid low radix encoding-based approximate Booth multipliers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3367–3371, Feb. 2020. https://doi.org/10.1109/TCSII.2020.2975094.
11. H. Waris, C. Wang, W. Liu, J. Han, and F. Lombardi, "Hybrid partial product-based high-performance approximate recursive multipliers," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 507–513, 2022. https://doi.org/10.1109/TETC.2020.3013977.
12. U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," Neurocomputing, vol. 96, pp. 57–65, 2012 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231212003311
13. T. Y. Cheng, Y. Masuda, J. Chen, J. Yu, and M. Hashimoto, "Logarithm-approximate floating-point multiplier is applicable to power-efficient neural network training," *Integration*, vol. 74, pp. 19–31, 2020. https://doi.org/10.1016/j.vlsi.2020.05.002.
14. R. Pilipović, V. Risojević, J. Božič, P. Bulić, and U. Lotrič, "An approximate GEMM unit for energy-efficient object detection," *Sensors*, vol. 21, no. 12, 2021. https://doi.org/10.3390/s21124195
15. H. Zhang and S. B. Ko, "Variable-precision approximate floating-point multiplier for efficient deep learning computation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, pp. 2503–2507, 5 2022. https://doi.org/10.1109/TCSII.2022.3161005.
16. C. Chen, W. Qian, M. Imani, X. Yin, and C. Zhuo, "PAM: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Transactions on Computers*, vol. 71, pp. 2473–2486, 10 2022. https://doi.org/10.1109/TC.2021.3131850.
17. A. Towhidy, R. Omidi, and K. Mohammadi, "On the design of iterative approximate floating-point multipliers," *IEEE Transactions on Computers*, 2022. https://doi.org/10.1109/TC.2022.3216465.
18. A. Y. Romanov, A. L. Stempkovsky, I. V. Lariushkin, G. E. Novoselov, R. A. Solovyev, V. A. Starykh, I. I.

Romanova, D. V. Telpukhov, and I. A. Mkrtchan, "Analysis of posit and bfloat arithmetic of real numbers for machine learning," *IEEE Access*, vol. 9, pp. 82 318–82 324, 2021. https://doi.org/10.1109/ACCESS.2021.3086669.

19. A. A. Ghazy and M. Shalan, "OpenLANE: The Open-Source Digital ASIC Implementation Flow," in 2020 Workshop on Open-Source EDA Technology (WOSET), 2020, last accessed 27 September 2022 .Available: https://woset-workshop.github.io/PDFs/2020/a21.pdf
20. OpenLane, "Openlane EDA Toolset." 2022, last accessed 27 September 2022. Available: https://github.com/The-OpenROAD-Project/OpenLane
21. M. Chupilko, A. Kamkin, and S. Smolov, "Survey of open-source flows for digital hardware design," in 2021 Ivannikov Memorial Workshop (IVMEM), 2021, pp. 11–16.
22. T. Edwards, "Google/SkyWater and the Promise of the Open PDK," in 2020 Workshop on Open-Source EDA Technology (WOSET), 2020, last accessed 27 September 2022. Available: https://woset-workshop.github.io/PDFs/2020/a03.pdf
23. "Google SkyWater Open Source PDK." 2022, last accessed 27 September 2022. Available: https://github.com/google/skywater-pdk
24. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Lake Tahoe, NV, USA: Curran Associates, Inc., Dec. 2012, pp. 1097–1105.
25. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
26. Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in 2017 IEEE International Conference on Computer Vision (ICCV), Oct. 2017, pp. 1398–1406.
27. A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Tech. Rep., Apr. 2009.
28. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in Proceedings of the 22nd ACM International Conference on Multimedia, ser. MM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 675–678. Available: https://doi.org/10.1145/2647868.2654889
29. J. S. Long and J. Freese, Regression Models for Categorical Dependent Variables using Stata, 3rd Edition. StataCorp LP, 2014. Available: https://www.stata.com/bookstore/regression-models-categorical-dependent-variables
30. I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in Proceedings of the 30th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. Available: https://proceedings.mlr.press/v28/sutskever13.html 6 VOLUME 11, 2023

Copyright © 2023 by the Authors. This is an open access article distributed under the Creative Commons Attribution (CC BY) License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Arrived: 22. 06. 2023
Accepted: 21. 07. 2023