MACcelerator: Approximate Arithmetic Unit for Computational Acceleration

Alice Sokolova*‡, Mohsen Imani†, Andrew Huang*, Ricardo Garcia*, Justin Morris*‡, Tajana Rosing*, and Baris Aksanli‡
*Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093
†Department of Computer Science, University of California Irvine, Irvine, CA 92697
‡Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182
{*aasokolo, anh162, rag023, j1morris, tajana*}*@ucsd.edu; m.imani@uci.edu; baksanli@sdsu.edu*

2021 22nd International Symposium on Quality Electronic Design (ISQED) | 978-1-7281-7641-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ISQED51717.2021.9424293

***Abstract*—As computationally expensive applications such as neural networks gain popularity, approximate computing has emerged as a solution for significantly reducing the energy and latency costs of extensive computational workloads. In this paper, we propose a highly accurate approximate floating point Multiply-and-Accumulate (MAC) unit for GPUs which significantly decreases power and delay costs of a MAC operation. We propose an intelligent input analysis scheme to approximate the addition stage of a MAC operation and an efficient Approximate Multiplier to simplify the multiplication stage. Our design has tunable accuracy, offering the flexibility of exchanging accuracy for increased efficiency. We evaluated our proposed design over a range of multimedia and machine learning applications. Our design offers up to 2.18× and 3.21× Energy-Delay Product improvement for machine learning and multimedia applications respectively while providing comparable quality to an exact GPU.**

***Index Terms*—Approximate computing, Energy Efficiency, Multiply-accumulator, Machine learning acceleration**

I. Introduction & Related Work

Machine learning tools, such as Deep Neural Networks (DNNs), are a powerful tool for many complex applications, but their beneficial capabilities are offset by their latency and power consumption, which stifle the growth of many practical DNN applications. Due to the inherent robustness of machine learning algorithms, approximate computing has emerged as a promising approach for accelerating and decreasing the power consumption of DNNs.

Typical approximate computing approaches include dynamic voltage scaling, imprecise gate-level logic units, and approximation algorithms. Voltage over-scaling, a technique for under-powering digital circuitry, causes an increasing number of timing errors, limiting the extent to which it can be practically applied [1]. Works such as [2] and [3] redesign logic blocks at the transistor level to simplify arithmetic at the expense of accuracy. A number of algorithms have been developed to approximate multiplication. Some designs, such as DRUM [4], use intelligent truncation. Others, such as CFPU[5] and RMAC [6], are non-conventional algorithmic approaches to approximate multiplication. These designs will be further discussed and compared with our proposed design in Section V. Other designs, such as [7] and [8], are specifically tailored for DNNs. However, we strive to create a design which can be used for a wider, more general set of applications.

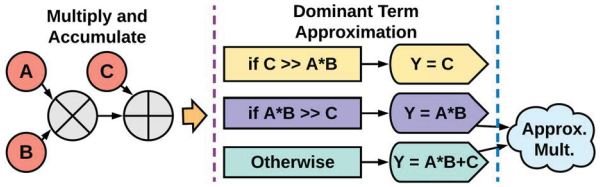

Fig. 1. Overview of the Maccelerator operation flow.

This paper proposes a delay and energy efficient floating-point MAC unit, Maccelerator, which approximates error-tolerant computations, such as machine learning algorithms and image processing. Maccelerator performs an evaluation of the input arguments and determines an optimal approximation scheme to provide minimal error with reduced computational costs. Depending on the inputs to the MAC unit, Maccelerator approximates either the addition stage, the multiplication stage, or both. The approximation accuracy of Maccelerator is adjustable, offering a trade-off between power and computational accuracy. The contributions of our paper include:

1) Dominant Term Approximation: A methodology for determining the presence of a dominant term affecting the solution of a MAC operation. If a dominant term is identified, all other terms can be omitted with an acceptable degree of known error. Our Dominant Term Approximation algorithm has adjustable accuracy and can provide computations with error of 1% and lower.
2) Approximate Multiplier: An algorithm for approximating the product of two floating point inputs. Our design performs low energy and latency multiplication with a maximum error of 1.57% and near-zero average error.

We evaluated the impact of Maccelerator on two machine learning topologies, Neural Nets (DNNs) and Hyperdimensional Computing (HD), as well as a number of multimedia applications. Our evaluations show that Maccelerator can offer up 1.78× and 2.18× Energy-Delay Product (EDP) improvement for DNNs and HD respectively with low quality losses. In more accurate configurations, Maccelerator offers up to 1.64× and 1.83× EDP improvement for DNNs and HD respectively with 0% classification accuracy loss. Maccelerator also offers up to 3.21× EDP improvement for multimedia applications with uncompromised visual quality.

II. Dominant Term Approximation

A. Overview

Whenever two numbers are summed together, it is possible for one of the additives to dominate the output. In such a case, the non-dominant input has little influence on the final sum. Thus, if we can identify the presence of a dominant term, the addition operation can simply be omitted, and the dominant term can replace the sum. Let A and B be the multiplicands of a MAC operation, and let C be the additive term. We can define three distinct input combinations and their respective outputs, as shown in Figure 1:

1) $if |C| \gg |A \times B| : Y = C$
2) $if |A \times B| \gg |C| : Y = A \times B$
3) $if |A \times B| \approx |C| : Y = A \times B + C$

It is evident that the first scenario is most efficient, since no multiplication or addition is required. However, the second and third scenarios require multiplication, which is known to be one of the slowest and costliest arithmetic operations. We address this apparent drawback by proposing an Approximate Multiplier as part of Maccelerator. As such, Maccelerator can be summarized as an approximation algorithm with three modes of operation, where approximation can be applied both at the summation and multiplication stages.

B. Identifying the dominant term

Recall that in floating point format, a number is stored as the product of a sign, two raised to the power of an exponent, and a significand lying between 'one' and 'two'. It is important to note that only the fractional portion of significand is stored explicitly. The fractional portion of the significand is generally called the mantissa, and its value lies between zero and one.

The presence of a dominant term is easily determined by comparing the exponents of $A \times B$ and C. The exponent of $A \times B$ is quickly determined by adding the exponent of A to the exponent of B. If the exponent of either $A \times B$ or C sufficiently exceeds that of the other, the term will dominate the output. As the difference between the exponents of $A \times B$ and C increases, the influence of the non-dominant term decreases, meaning Dominant Term Approximation accuracy is a function of the difference of exponents.

Let $A \times B$ and C be rewritten below, where X and Y are the significands of the two floating point numbers:

$$A \times B = \pm 2^M \times X \text{ and } C = \pm 2^N \times Y \tag{1}$$

Approximating the sum of $A \times B$ and C with either $A \times B$ or C will result in approximation error. Suppose $C \gg A \times B$, which is the first scenario described earlier. Then the expression for relative error is:

$$error = \frac{A \times B}{A \times B + C} = \frac{\pm 2^M \times X}{\pm 2^M \times X \pm 2^N \times Y} \tag{2}$$

We are interested in identifying the maximum absolute value of the error, which is found when the denominator is minimized and the numerator is maximized. The denominator of Equation 2 is minimized when the terms $2^M \times X$ and $2^N \times Y$ have opposite signs. Since we are examining the first scenario, we know that $N > M$ and the maximum error becomes:

TABLE I
Maximum and average error of Dominant Term Approximation at various values of Delta

Delta	2	3	4	5	6	7	8	9	10
Max. Error (%)	100	33.33	14.29	6.67	3.23	1.59	0.79	0.39	0.20
Avg. Error (%)	0.45	0.21	0.11	0.053	0.026	0.013	0.007	0.003	0.002

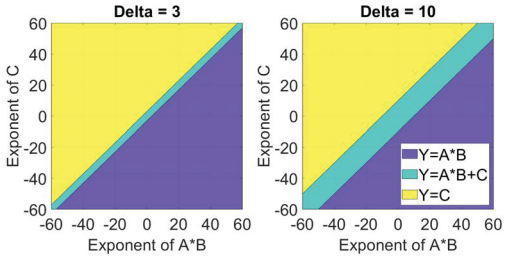

Fig. 2. The three scenarios of Dominant Term Approximation relative to the input exponents at different values of Delta.

$$\begin{aligned} max(error) &= \frac{\pm 2^M \times X}{\pm |2^N \times Y - 2^M \times X|} \\ &= \pm \left| \frac{2^M \times X}{2^N \times Y - 2^M \times X} \right| \end{aligned} \tag{3}$$

Recalling that a significand of a floating point number lies between one and two, the absolute value of Expression 3 is greatest when X is equal to two, and Y is equal to one, regardless of the values of M and N. This is easily proven by finding the partial derivatives of Expression 3 with respect to X and Y. (Proof omitted for space.)

$$\begin{aligned} |\, max(error) \,| &= \frac{2^M \times 2}{2^N \times 1 - 2^M \times 2} = \frac{2^{M+1}}{2^N - 2^{M+1}} \\ &= \frac{2^{M+1}}{2^{M+1}(2^{N-M-1} - 1)} = \frac{1}{2^{N-M-1} - 1} \end{aligned} \tag{4}$$

This important result provides us with an expression for absolute error as a function of the difference between N and M. In other words, this expression determines the extent to which one term in the MAC unit must dominate over the other to provide a highly accurate approximation.

A similar expression can be derived for Case 2, except that the variables N and M are swapped. The maximum error depends only on the absolute difference between the exponents of $A \times B$ and C, hence referred to as Delta (Δ).

Table I lists maximum Dominant Term Approximation error for various values of Delta. Figure 2 shows a comparison between two different values of Delta. For plotting purposes, only exponents between -60 and 60 are shown. Even for large values of Delta, only a small portion of computations need to perform both addition and multiplication. Table II shows the exact percentage of all calculations falling into each of the three Maccelerator modes for various values of Delta.

The efficiency of Dominant Term Approximation also benefits from the fact that the maximum error only occurs at the boundaries of the three modes, as shown in Figure 3 (zoomed in on exponents between -10 and 10 for visibility). For areas

TABLE II
THE DISTRIBUTION OF ALL POSSIBLE MAC CALCULATIONS AMONG THE THREE MODES OF MACCELERATOR AS A FUNCTION OF DELTA (%)

Delta	3	4	5	6	7	8	9	10
Y=A*B+C	2.37	3.16	3.95	4.74	5.53	6.32	7.11	7.91
Y=A*B	48.81	48.42	48.02	47.63	47.23	46.84	46.44	46.05
Y=C	48.81	48.42	48.02	47.63	47.23	46.84	46.44	46.05

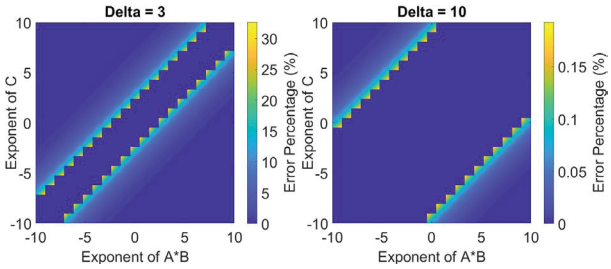

Fig. 3. Error distribution of Dominant Term Approximation.

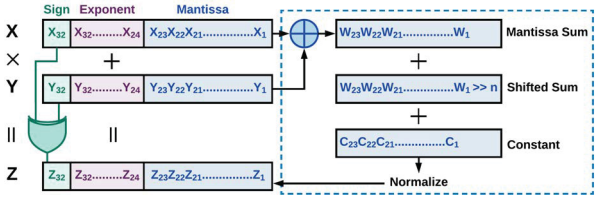

Fig. 4. Overview of Multiplication Approximation operation flow.

further from the boundary region, the error approaches zero. The majority of possible MAC calculations have near-zero Dominant Term Approximation error. The average computational error for different values of Δ is also shown in Table I.

III. MULTIPLICATION APPROXIMATION

In order to accelerate multiplication computations, we propose a highly accurate Approximate Multiplier, derived from RMAC [6]. RMAC can approximate multiplication with very low latency and energy, but has an error range up to 11%, leading to reduced efficiency when higher accuracy is desired. We have designed an improved multiplication algorithm which retains the excellent speedup and energy efficiency of RMAC while reducing maximum multiplication error to 1.57%.

Essentially, RMAC is a two-case algorithm. Similarly to standard floating point multiplication, the sign bits of floating point inputs A and B are XORed and the exponents are added. However, instead of multiplying the mantissa bits, the bits are summed together. If the sum results in a carry bit, it is added to the result of the exponent. Recall that the mantissa bits of a floating point number store only the fractional portion of the full significand lying between one and two. Let a and b be the values stored in the mantissa bits of A and B respectively. Then the mathematical expression for RMAC is written as:

$$\begin{aligned} if\, a+b<1 : Y &= \pm 2^{exp_A+exp_B} \times (1+a+b) \\ if\, a+b\geq 1 : Y &= \pm 2^{exp_A+exp_B} \times 2 \times (a+b) \end{aligned} \tag{5}$$

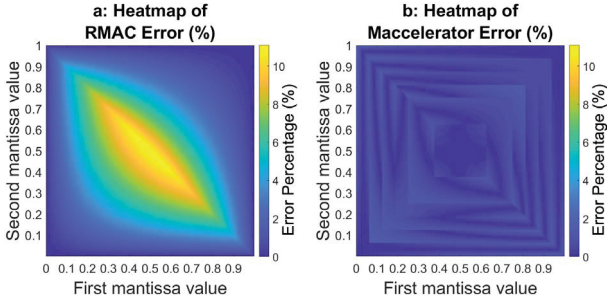

Fig. 5. (a) Error distribution of RMAC and (b) Error distribution of the Maccelerator Approximate Multiplier as functions of input mantissas.

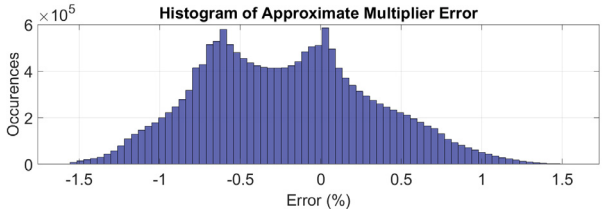

Fig. 6. Error range/distribution of the Maccelerator Approximate Multiplier

The error of RMAC is represented by the heatmap in Figure 5a. We have significantly reduced this error by partitioning RMAC into regions and applying error compensation to each region so that the maximum value does not exceed 1.57%. The error heatmap of our design is shown in Figure 5b. Our design adds a few steps to RMAC to achieve this accuracy improvement. Figure 4 shows the flow of the algorithm.

1) After the sign bits are XORed and the exponent bits are added, the mantissas a and b are added, and the presence of a carry bit is identified.
2) Depending on whether a carry bit is present or absent, the algorithm accesses one of two look-up tables which accept a and b as inputs. Each entry of the look-up tables contains a constant C and an integer value n.
3) The final error-corrected mantissa is found by summing three values: the sum of a and b, the sum of a and b shifted right by n, and the constant C.
4) Similarly to conventional floating-point multiplication, the mantissa may need to be normalized so that is lies between one and two, which involves shifting the mantissa and adjusting the exponent accordingly.

This procedure is described by the equations below:

$$\begin{aligned} if\, a+b<1 : Y &= \pm 2^{exp_A+exp_B} \times (a+b+\frac{a+b}{2^n}+C) \\ if\, a+b\geq 1 : Y &= \pm 2^{exp_A+exp_B} \times (2(a+b)+\frac{a+b}{2^n}+C) \end{aligned} \tag{6}$$

1) Look-up Table: Inspection of the error of RMAC shown in Figure 5a reveals that there is a geometric pattern in the error distribution. To target each region individually, we partitioned the plane into a 16×16 grid and derived an optimal compensation function for each region. A finer grid would

increase approximation accuracy, but also complexity. We have concluded that a 16×16 grid strikes an optimal balance.

2) Compensation Functions: Since n is an integer, the division by 2^n becomes a right shift by n bits. We experimentally determined the values of n and C by sweeping all possible values of n and C for each region. The error yielded by our Approximate Multiplier can be either positive or negative, and the histogram of error distribution is shown in Figure 6. Our multiplication approximation algorithm yields a maximum absolute error of 1.57%, an average signed error of -0.21%, and an absolute average error of 0.47%.

3) Computational Reuse: An important feature of our algorithm is computational reuse. The sum of a and b is used both as the branching criterion and function input. Every calculation consists of only three addition operations, and up to two bitwise shift operations. Our algorithm makes use of a look-up table, but the look-up table is of a small dimension, and therefore does not hinder computational acceleration.

IV. MAC System

1) System Accuracy: Maccelerator has two stages of approximation: Dominant Term Approximation and Approximate Multiplication. The yellow region in Figure 3 experiences only Dominant Term Approximation error, the purple region experiences only Approximate Multiplication error, but in the teal region, error depends on both. Dominant Term Approximation always yields negative error, but since the error of Approximate Multiplication can be positive or negative, the sum may lead to either error cancellation or error accumulation. Figure 7 illustrates three possible scenarios.

1) Multiplication Approximation error is near zero, and error is only introduced by Dominant Term Approximation.
2) Multiplication Approximation error and Dominant Term Approximation error have **opposite signs** such that the former is canceled out by the latter. We demonstrate this by setting multiplication approximation error to its maximum value, 1.57%, in Figure 7b.
3) Multiplication Approximation error and Dominant Term Approximation error have the **same sign**. In this case, the error from both stages accumulates. We demonstrate this by setting multiplication approximation error to its minimum value, -1.57%, in Figure 7c. This scenario has the greatest possible error. As such, we can identify the worst possible error which can occur in the Maccelerator system for every value of Delta, shown in Table III.

The average error is, however, much lower than the maximum possible error. Firstly, Multiplication Approximation error tapers toward the positive and negative extremes as seen in Figure 6, and secondly, as seen from Figure 7, Dominant Term Approximation only induces error in a very narrow boundary region. We can calculate the average error of Maccelerator by averaging both Multiplication and Dominant Term Approximation error. The results are found in Table III.

2) System Efficiency: The efficiency of Maccelerator (the latency and energy cost of a computation) is also a function of Delta. The distribution percentages shown in Table II are also the probabilities of a single calculation occurring in each of the three design modes. For example, at Delta equal to 10, there is a 46.05% chance that both multiplication and addition can be omitted, and only a 7.11% chance that both must be performed. The smaller the value of Delta, the more likely one or both operations can be omitted, increasing the efficiency.

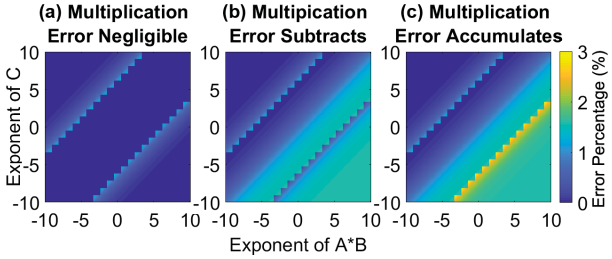

Fig. 7. Various error distribution patterns of Maccelerator for different input combinations ($\Delta = 7$).

TABLE III
Maximum and Average Error of the Maccelerator System at Various Values of Delta

Delta	3	4	5	6	7	8	9	10
Max. Error (%)	34.90	15.86	8.24	4.80	3.16	2.358	1.96	1.77
Avg. Error (%)	0.29	0.19	0.15	0.12	0.11	0.108	0.106	0.106

V. Experimental Results

A. Experimental Setup

We integrated Maccelerator into the floating point units of the AMD Southern Island Radeon HD 7970 GPU. To evaluate the functionality of Maccelerator, we modified Multi2sim, a cycle accurate CPU-GPU simulator [9] to model the three main floating point operations in GPU architecture: multiplication, multiplication-accumulation (MAC), and multiplication-addition (MAD). We analyzed the execution time and energy consumption of Maccelerator by performing a circuit-level simulation using *HSPICE* in 45nm technology. We calculated the energy consumption of conventional FPUs using the FloPoCo [10] library and synthesized them using *Synopsys Design Compiler* in 45-nm ASIC flow [11]. We used *Synopsys Prime Time* to optimize the FPU's power consumption. We evaluated the efficiency and accuracy of Maccelerator on general multimedia applications and various machine learning algorithms. Roughly 85% of the floating point operations in these applications involve multiplication or MAC operations, which can be approximated using Maccelerator.

B. Acceleration Performance

Multimedia: We tested our Approximate Multiplier on six general Open*CL* applications, including *BoxSharpen, Sobel, Prewitt, Roberts, JPEG Compression*, and *Gaussian Blur* from AMD APP SDK v2.5 [12] using images from the Caltech 101 dataset [13]. We used Peak Signal to Noise Ratio (PSNR) as a metric for accuracy. Similarly to prior work, computations with a PSNR value of 30dB and higher are considered to be of acceptable quality for the human eye [4], [6].

TABLE IV
PSNR (DB) OF MULTIMEDIA APPLICATIONS RUNNING ON MACCELERATOR

Images	Applications					
	BoxSharp	Sobel	Prewitt	Robert	JPEG	GaussBlur
Lichtenstein	33.10	36.85	40.78	42.47	50.82	47.89
Baboon	33.63	34.92	39.09	39.41	48.77	46.07
Camera	32.09	36.08	39.99	41.37	52.53	49.87
Lena	32.53	35.13	39.36	40.78	49.94	45.77
EDP Improv.	2.38×	1.93×	3.21×	2.14×	2.76×	2.41×

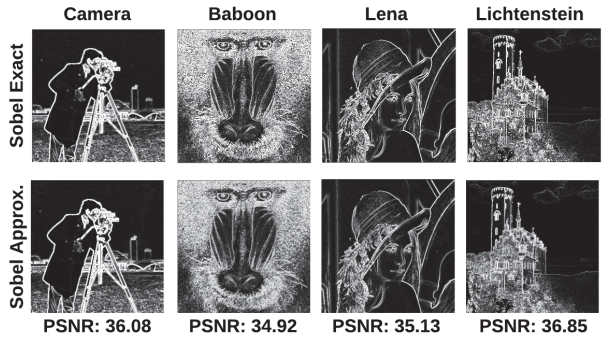

Fig. 8. A comparison between images generated using exact arithmetic and images generated by Maccelerator for the application *Sobel*.

TABLE V
DATASETS (n: FEATURE SIZE, K: NUMBER OF CLASSES)

	n	K	Data Size	DNN Accuracy	HD Accuracy	Description
ISOLET	617	26	19MB	96.3%	96.1%	Speech Recognition [16]
UCIHAR	561	12	10MB	97.3%	98.1%	Activity Recognition[17]
PAMAP	75	5	240MB	95.8%	92.9%	Physical Monitoring[18]
FACE	608	2	1.3GB	96.1%	96.5%	Face Recognition[19]

Accuracy: Table IV shows the PSNR values obtained by running six applications on four different benchmark images using Maccelerator, none of which show a PSNR value less than 30dB. Figure 8 compares images processed through *Sobel* using exact arithmetic and our Approximate Multiplier.

Efficiency: Table IV also shows the average Energy-Delay Product (EDP) improvement of Maccelerator running 500 randomly chosen images from the Caltech 101 database on each of the six applications. Since multimedia applications utilize multiplication operations but not MAC operations, efficiency is not a function of Delta.

Machine Learning: We evaluated the efficiency of proposed Maccelerator on four popular neural network applications, listed in Table V. We also evaluated the efficiency of Maccelerator on another machine learning topology – Brain-inspired Hyperdimensional Computing (HD). HD [14], [15] is a new computing paradigm which emulates cognition tasks by exploiting long-sized vectors instead of working with numeric values used in contemporary processors. The baseline accuracy of HD on four applications is also shown in Table V.

Accuracy: We ran two tests to evaluate the performance of Maccelerator on machine learning applications: 1) Approximating only the inference stage using a GPU enhanced with Maccelerator, and 2) Approximating both training and inference. (For training, approximation is only used during the feed-forward phase, while back-propagation is still performed using exact arithmetic.) Figure 9 shows that quality losses decrease and approach zero as the value of Delta increases. Due to the machine learning algorithms' inherent ability to adapt to consistent patterns, quality loss decreases when both training and inference is performed using Maccelerator, which is advantageous for decreasing the extensive time and energy resources required for training.

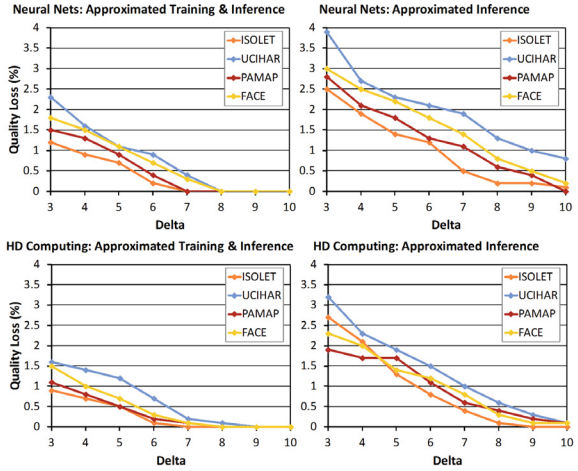

Fig. 9. Quality loss of NNs and HD computing as a function of Delta.

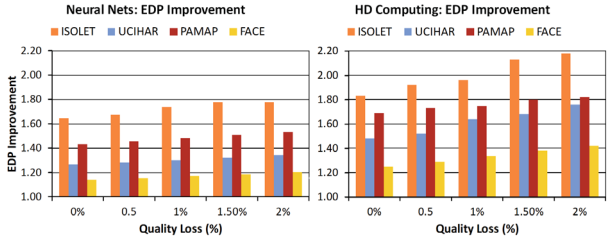

Fig. 10. EDP Improvement of NNs and HD computing based on quality loss.

Efficiency: We evaluated the EDP improvement of Maccelerator over exact GPU hardware as a function of quality loss by running Maccelerator at whichever Delta value was necessary to achieve the given accuracy level. Figure 10 shows that different applications provide varying levels of efficiency depending on their robustness to approximation. Maccelerator offers best results when the input data has a large range of values, which maximizes the efficiency of Dominant Term Approximation. Maccelerator provides on average 1.42× and 1.67× EDP improvement for NNs and HD, respectively, with less than 1% quality loss as compared to baseline GPU.

C. Comparison with Prior Work

To our knowledge, no other approximate standalone floating point MAC units have yet been proposed. Thus, we compare Maccelerator with approximate multipliers integrated into a conventional MAC unit. State-of-the-art approximate multipliers include Kulkarni [20], ESSM [21], DRUM [4], CFPU [5], and RMAC [6]. Kulkarni [20] modifies the logical structure of a partial product generator to induce approximation;

TABLE VI
A COMPARISON OF STATE-OF-THE-ART APPROXIMATE MULTIPLIERS.

	Kulkarni[20]	ESSM8[21]	DRUM6[4]	CFPU[5]	RMAC[6]	Maccelerator
Max. Error	22.2%	11.1%	6.3%	6.3%	6.3%	1.57%
EDP (pJs)	10	1.2	1.04	0.44	0.08	0.07
Delay (ns)	3.5	2.1	1.9	1.6	0.52	0.58
Energy (pJ)	2.9	0.58	0.55	0.27	0.15	0.12

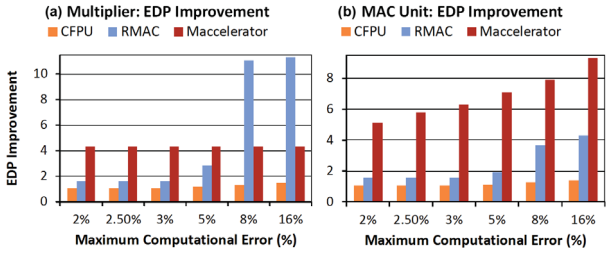

Fig. 11. A comparison of EDP improvement between different approximate computing designs.

ESSM [21] intelligently truncates multiplicands; DRUM [4] expands the concept of intelligent truncation to floating point arithmetic; CFPU [5] and RMAC [6] are hybrid approximate-and-exact designs with runtime-configurable error which reduce multiplication to a set of simpler arithmetic operations. We compare these designs with Maccelerator. Table VI shows the maximum error, energy, delay, and EDP offered by these designs in their most accurate configurations Maccelerator's Approximate Multiplier offers the lowest approximation error and lowest energy-delay product.

CFPU [5] and RMAC [6] are run-time configurable, which means they offer a trade-off between approximation accuracy and energy consumption. We compare Maccelerator to CFPU and RMAC by evaluating the EDP improvement of each design as a function of accuracy for multiplication and MAC operations, as compared with baseline GPU.

1) Multiplication: RMAC offers very high EDP improvement for multiplication operations within an error threshold of 8% or greater. Due to its hybrid exact-and-approximate hardware design, the EDP improvement plummets for accuracies of 5% or less. The Maccelerator Multiplier, on the other hand, offers greater EDP improvement at higher accuracy thresholds.

2) Multiply-and-Accumulate: The advantages of Maccelerator are most prevalent for MAC operations. It offers at least 5.1× and up to 9.3× EDP improvement for a single MAC operation, greatly exceeding that of conventional MAC units, with approximate multipliers such as CFPU or RMAC.

D. Overhead

Maccelerator is a configurable MAC unit offering computational acceleration and power reduction with known precision. However, certain applications may require full precision. We propose Maccelerator as an enhancement to GPU cores such that an application has access both to exact FPU hardware and approximate hardware. Maccelerator requires the use of some macro blocks, which include three adders, a logic unit, one comparator, one shifter, six multiplexers, and a small amount of memory storage to hold the two look-up tables. The area overhead of Maccelerator is 6.7mm^2, which is less than 1.4% of the entire GPU area (471mm^2).

VI. CONCLUSION

We proposed Maccelerator, a configurable approximate MAC unit with tunable accuracy. Based on intelligent input analysis, our design uses two approximation schemes, Dominant Term Approximation, and Approximate Multiplication. Maccelerator produces calculations with a near-zero average error, and as low as 1.57% maximum error. Maccelerator offers up to 9.3× EDP improvement per MAC operation, up to 1.78× and 2.18× EDP improvement on Neural Network and HD machine learning applications respectively, and up to 3.21× EDP improvement on image processing applications. For machine learning algorithms, Maccelerator offers 0% quality loss in highly accurate configurations, and up to 2.3% quality loss in the least accurate configuration. The area overhead of Maccelerator is 1.4% when integrated into a GPU core.

ACKNOWLEDGMENTS

This work was supported in part by SRC-Global Research Collaboration Task No. 2988.001 and also NSF grants 1527034, 1730158, 1826967, 1830331, 1911095, 2003277, and 2003279.

REFERENCES

[1] K. He *et al.*, "Circuit-level timing-error acceptance for design of energy-efficient dct/idct-based systems," *TCSVT*, vol. 23, no. 6, pp. 961–974, 2013.
[2] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *13th IEEE International Conference on Nanotechnology*, pp. 690–693, 2013.
[3] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *IEEE/ACM DATE*, 2014.
[4] S. Hashemi *et al.*, "tldrum: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, pp. 418–425, IEEE Press, 2015.
[5] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *IEEE/ACM DAC*, pp. 1–6, IEEE, 2017.
[6] M. Imani *et al.*, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *ISLPED*, p. 12, ACM, 2018.
[7] H. Tann *et al.*, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *11th IEEE/ACM/IFIP CODES+ISSS*, 2016.
[8] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *IEEE Winter Conference on Applications of Computer Vision*, 2016.
[9] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *PACT*, pp. 335–344, ACM, 2012.
[10] "Aflopoco [online]. available:http://flopoco.gforge.inria.fr/,"
[11] D. Compiler, "Synopsys inc," 2000.
[12] "Amd app sdk v2.5 [online]. available: http://www.amd.com/stream,"
[13] "Caltech 101 [online]. http://www.vision.caltech.edu/image_datasets/caltech101/,"
[14] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.
[15] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, 2016.
[16] "Uci ML repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[17] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *IWAAL*, pp. 216–223, Springer, 2012.
[18] A. Reiss *et al.*, "Introducing a new benchmarked dataset for activity monitoring," in *ISWC*, pp. 108–109, IEEE, 2012.
[19] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *IEEE/ACM ICCAD*, 2017.
[20] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *IVLSI*, pp. 346–351, IEEE, 2011.
[21] S. Narayanamoorthy *et al.*, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *TVLSI*, vol. 23, no. 6, pp. 1180–1184, 2015.