ApproxLP: Approximate Multiplication with Linearization and Iterative Error Control

Mohsen Imani*, Alice Sokolova†*, Ricardo Garcia, Andrew Huang, Fan Wu, Baris Aksanli†, Tajana Rosing
Department of Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA
†Electrical and Computer Engineering Department, San Diego State University, San Diego, CA 92182, USA
{moimani,aasokolo,rag023,anh162,tajana}@ucsd.edu;baksanli@sdsu.edu

ABSTRACT

In a data hungry world, approximate computing has emerged as one of the solutions to create higher energy efficiency and faster systems, while providing application tailored quality. In this paper, we propose ApproxLP, an Approximate Multiplier based on Linear Planes. We introduce an iterative method for approximating the product of two operands using fitted linear functions with two inputs, referred to as linear planes. The linearization of multiplication allows multiplication operations to be completely replaced with weighted addition. The proposed technique is used to find the significand of the product of two floating point numbers, decreasing the high energy cost of floating point arithmetic. Our method fully exploits the trade-off between accuracy and energy consumption by offering various degrees of approximation at different energy costs. As the level of approximation increases, the approximated product asymptotically approaches the exact product in an iterative manner. The performance of ApproxLP is evaluated over a range of multimedia and machine learning applications. A GPU enhanced by ApproxLP yields significant energy-delay product (EDP) improvement. For multimedia, neural network, and hyperdimensional computing applications, ApproxLP offers on average 2.4×, 2.7×, and 4.3× EDP improvement respectively with sufficient computational quality for the application. ApproxLP also provides up to 4.5× EDP improvement and has 2.3× lower chip area than other state-of-the-art approximate multipliers.

CCS CONCEPTS

• **Hardware → Integrated circuits**; • **Computer systems organization →** *Architectures*;

KEYWORDS

Approximate Computing, Machine learning acceleration, GPU

ACM Reference Format:
Mohsen Imani*, Alice Sokolova†*, Ricardo Garcia, Andrew Huang, Fan Wu, Baris Aksanli†, Tajana Rosing, . 2019. ApproxLP: Approximate Multiplication with Linearization and Iterative Error Control. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317774

*Mohsen Imani and Alice Sokolova contributed equally to this research.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317774

1 INTRODUCTION

Energy conservation and efficiency maximization is a universal goal for all electronic devices, from high voltage power engines to small portable chip devices. However, power usage is limited by availability and cost, regardless of the source. Designers are always on the lookout for potential trade-offs where performance can be acceptably exchanged for power efficiency. Approximate computing is a promising approach which enables such a trade-off between computational accuracy and efficiency. It occupies an important niche in human sensory applications and statistical applications [1, 2]. Human senses naturally have limited sensitivity and may not be able to distinguish between the output of a mathematically accurate computation and an approximate one [3–6]. Machine learning applications are also error tolerant because of their statistical nature. If error introduced due to approximation is on the same order of magnitude as the application's statistical variance, approximate computing would not affect the quality of the output [7–10].

Application-specific devices are often optimized to meet but not exceed minimum performance and accuracy requirements [11–14]. However, GPUs, a popular and widespread choice for a large variety of applications, often exceed the minimum performance metrics of the application. In such cases, energy and time are wasted to compute highly accurate results for applications where the additional accuracy provides no benefits. Hence, there is a need to develop systems which conserve energy by providing sufficient, rather than excessively accurate computational results. Numerous data processing applications use a large range of values and require high precision. Therefore, computations in many traditional and state-of-the-art computing systems use floating point units (FPUs) [15–17]. Multiplication is one of the most common and costly FPU operations, slowing down the computation in many applications such as signal processing, neural networks, and stream processing. For example, our evaluation on general OpenCL applications from AMD APP SDK v2.5 [18] indicates that over 85% of floating point arithmetic involved multiplication. Thus, we devote special attention to reducing the cost of floating point multiplication operations through approximation.

Prior work tried to improve the efficiency of multiplication by enabling approximation [13, 19–22]. However, designs with fixed approximation accuracy lack the flexibility to accommodate a wide variety of applications [11]. Other designs provide flexibility and tunable error, but lose efficiency by utilizing a hybrid approach, where accuracy is adjusted by performing exact computations when approximation fails to yield enough accuracy, rather than tuning the approximation itself [22–24]. These designs provide variable accuracy, but their efficiency plummets as accuracy requirements increase, negatively impacting cost and computational time.

In this paper, we propose ApproxLP, an approximate multiplier based on the linearization of nonlinear planes, which dramatically

decreases the energy cost of floating point multiplication by permitting a controlled amount of error. Proposed ApproxLP is a GPU enhancement which computes the approximate product of floating point inputs with a tunable degree of accuracy. It is an iterative method, wherein every iteration of the algorithm asymptotically approaches the exact product. ApproxLP carries significant advantages over both emerging and conventional approximation methods, providing greater accuracy at lower energy cost and latency. ApproxLP is a versatile algorithm with the ability to generate both rough and very exact approximations. Our method combats the greatest weakness of state-of-the-art hybrid methods, which is the necessity to resort to exact matrix multiplication when approximation fails to yield enough accuracy. ApproxLP provides a solution for approximating products with very low error tolerances, in the range of 1% and lower, without resorting to exact multiplication, as is the case with hybrid methods. We evaluate the efficiency and accuracy of proposed ApproxLP on a wide range of multimedia and machine learning applications. Our evaluation shows that a GPU enhanced by ApproxLP can provide on average 2.4×, 2.7×, and 4.3× energy-delay product (EDP) improvement on multimedia, deep neural network, and hyperdimensional computing applications respectively, as compared to baseline GPU. Compared to the state-of-the-art approximate multiplier [23], ApproxLP can provide up to 4.5× EDP improvement and has 2.3× lower chip area while providing comparable quality of computation.

2 RELATED WORK

Computational power reduction in GPU's can be accomplished either by limiting the performance of the system, or redesigning hardware blocks for low power operation [11, 14, 25, 26]. Dynamic voltage scaling is a common technique that relies on the reduction of power supply voltage to cut down power consumption. However, under powering the circuitry slows down the whole system and increases timing errors, limiting the extent to which voltage scaling can be practically applied. [25–27]. Another research area is focused on enabling approximation by performing computational reuse [28–30]. A lookup table located next to each GPU cores to exactly/approximately eliminates redundant computation. A variety of designs have also focused on redesigned hardware blocks to conserve energy at the expense of accuracy, which can be used both independently and in conjunction with voltage scaling. Work such as [11, 13, 14, 31, 32] are designs which lower the number of outputs of multiplication building blocks, decreasing energy and area. Work in [11] utilizes truncated multiplication by selecting a fixed number of most significant bits as inputs. Accuracy is conserved by neglecting leading zeros in the operands. However, these methods of approximation are application specific and non-configurable, as the level of accuracy cannot be changed during runtime.

Works in CFPU [22] and RMAC [23] use a hybrid method for runtime configurable approximation. At worst, the error yielded by CFPU is 50%. Energy efficiency drops rapidly for higher desired accuracy. RMAC [23] provides a very fast and energy efficient way of approximating multiplication with an error of up to 11.1%, but loses efficiency once a lower maximum error is desired due to the necessity of resorting to exact multiplication to increase accuracy. Ultimately, despite both being runtime configurable with low energy consumption, both designs are still unable to completely remove the need for exact multiplication from their designs. Unlike RMAC and CFPU, our proposed design, ApproxLP, provides tunable accuracy without ever resorting to exact multiplication – the costly operation which is desirable to avoid.

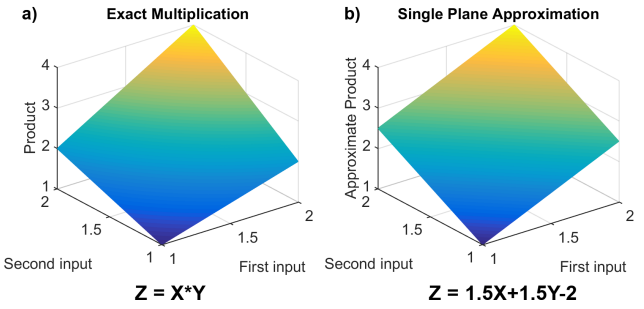

Figure 1: (a) The exact product of the mantissa operands; (b) the product of the mantissa operands linearized with one plane.

3 APPROXLP DESIGN

3.1 Standard IEEE Format

The IEEE floating point standard 754 [33] represents non-integer numbers as the product of a sign bit, two to the power of an exponent, and the significand. The significand is a value between one and two, but only the fractional portion of the significand is stored explicitly. The bits storing the fractional portion are called the mantissa. Floating point multiplication is accomplished by XORing the sign bits, adding the exponent bits, and multiplying the significands. This section explains our algorithm for approximating the product of the significands, which henceforth will be referred to as operands.

3.2 ApproxLP Mathematical Support

The product of two numbers, X and Y, is a nonlinear plane in three-dimensional Cartesian coordinates, where the product lies along the z axis. However, any nonlinear plane can be approximated by a linear plane of the form:

$$A(X - X_0) + B(Y - Y_0) + C(Z - Z_0) = 0 \quad (1)$$

Rearranged and simplified, equation 1 becomes:

$$Z = A \times X + B \times Y + C \quad (2)$$

Linear planes are an attractive method for approximating a nonlinear function of two variables. By its very definition, a linear plane is the sum of two scaled inputs, meaning the multiplication of the two input mantissas can be eliminated and replaced with addition. We will be using linear planes to approximate the product of the significands of two floating point inputs, which henceworth will be denoted as X and Y.

Figure 1 shows a comparison between the nonlinear plane representing the product of the two operands, and a corresponding linear approximation. Both planes have the same output range and similar slope, but there are two evident limitations:

- Error: Error is defined as the difference between the exact plane and its approximation, normalized by the exact plane. A single linear plane provides a highly inaccurate approximation for $X \times Y$, as seen in Figure 1, so a more accurate solution must be found.
- Efficiency: In order for the linear approximation method to be effective, the input variables may only be scaled by powers of two, since scaling by powers of two in binary arithmetic involves a simple left or right bitwise shift. Scaling by any other number reinstates the need for multiplication, meaning that the function in Figure 1b is unsatisfactory.

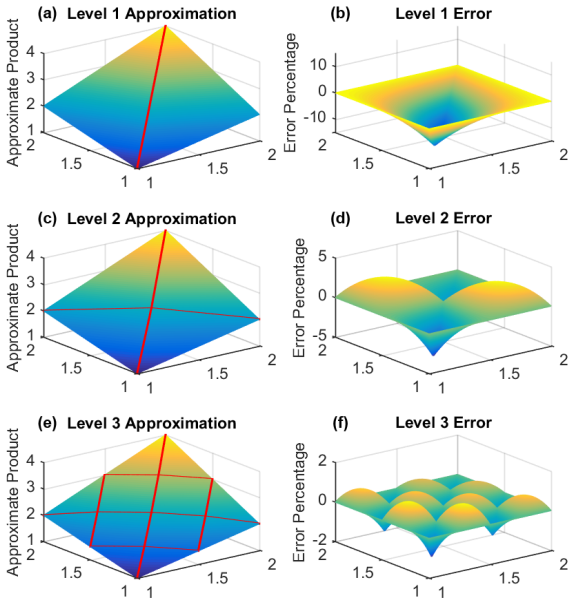

Figure 2: Overview of ApproxLP approximation and the distribution of error in different configurations. (a, c, e) show the multiplication plane subdivided at different levels; (b, d, f) show the error distribution map at different levels.

In the next section, we describe our proposed method for overcoming the challenges listed above and approximating the nonlinear product of operands using linearization. The key to accurately approximating a nonlinear plane with linear ones is to correctly subdivide the nonlinear plane and model each individual section with its own unique linear function, while only using permissible coefficients, powers of two. We have derived a methodology for finding the correct subdivision patterns for continuously decreasing approximation error, while operating within the confines of limited coefficients. ApproxLP provides tailored linear functions which are individually optimized for different input combinations to provide the most accuracy with the least number of operations. This customization is what sets ApproxLP apart from emerging and conventional methods which treat all inputs equally, such as truncated matrix multiplication.

3.3 ApproxLP & Error Configuration

An efficient subdivision pattern is crucial for modeling the nonlinear multiplication plane. We derive the optimal subdivision pattern by examining error maps. The difference between the exact plane and an approximate plane results in another Cartesian plane, which provides insight into the magnitude and location of error. By observing these error maps, we can define the optimal way to subdivide the exact plane and fit each section with a suitable linear function. This process is illustrated in Figure 2.

Figure 2a depicts the initial subdivision. Observation of Figure 1 shows that there is a line of symmetry running down the exact multiplication plane, from the top corner to the bottom corner. After the plane is divided in half in such a manner, the right and left halves are fitted with linear functions, within the constraints described earlier. The resulting error map is shown in Figure 2b. It has a clearly defined inverted pyramidal-like shape, with four distinct 'faces'. This observation indicates to us that the next subdivision should mimic this pattern.

Table 1: ApproxLP configuration and maximum error rate at different levels of approximation.

Approximation	Division Lines	Sections	Maximum Error
Level 1	1	2	12.5%
Level 2	2	4	4.167%
Level 3	6	12	1.25%
Level 4	14	40	0.347%

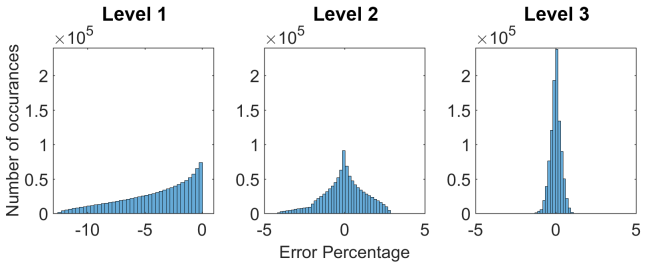

Figure 3: Histograms of ApproxLP error distribution at different levels of approximation.

Figures 2c-f show the continuation of this process. The points of maximum error nonlinearity indicate the optimal placement of new subdivision lines, which are shown in red on the rightmost figures. With every iteration of this method, the points of maximum error are set to zero, and new peaks and troughs are formed, with diminishing amplitude – in other words, the error of approximation decreases. Henceforth, the extent to which the plane is subdivided will be referred to as "level of approximation". The first level, called Level 1, corresponds to one division line, as seen in Figure 2a. Each successive subdivision corresponds to the next level. For an index $n = 1, 2, 3, \ldots$ where n is the level of approximation, the total number of subregions, denoted as N is determined by the equation:

$$N = \begin{cases} 2, & \text{n=1} \\ 2^{n-1} + 2^{2n-3}, & \text{n} \geqslant 2 \end{cases} \tag{3}$$

For $n = 1, 2, 3, \ldots$ where n is the level of approximation, the maximum error is equal to:

$$error(\%) = \frac{2^{-2n}}{1 + 2^{-n+1}} * 100 \tag{4}$$

Table 1 lists the summary of the results for four levels of approximation. Figure 3 shows the histograms of the ApproxLP error distribution running one million randomly generated floating point numbers. At Level 1, ApproxLP provides a maximum error of 12.5%. At Levels 2 and 3, the error distribution narrows and centers around zero. At Level 4, the magnitude of error is less than or equal to 0.347%.The maximum multiplication error significantly decreases with every successive level.

4 IMPLEMENTATION & INTEGRATION

4.1 ApproxLP Hardware Implementation

Our multiplication algorithm uses conventional techniques to find the sign bit and exponent bits of the product, but uses an approximation technique to find the significand. The previous section discussed the mathematical theory of approximating the significand using linear tangential planes. This section describes the practical implementation of the mathematical model. The subdivisions described in Section 3 become conditional statements, and the levels of approximation correspond to layers of nested conditional

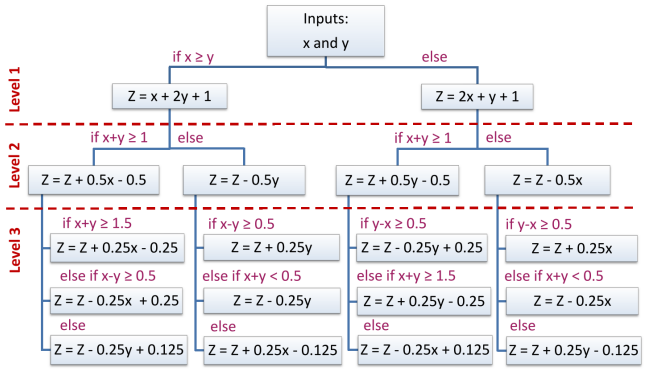

Figure 4: Tree-based diagram of ApproxLP approximation up to Level 3.

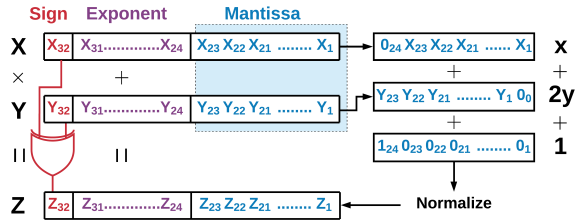

Figure 5: Hardware implementation of ApproxLP in Level 1.

logic. Each successive level builds upon the previous, meaning no computations are wasted, and no unnecessary operations are performed.

Section 3 describes linear equations used to approximate the product of X and Y. However, X and Y are the full significands of the floating point inputs. Conventionally, X and Y are not explicitly stored. Rather, only the fractional portions of the significands are available. The transition from complete significands to mantissas is performed by applying a simple substitution:

$$X = 1 + x \qquad Y = 1 + y \tag{5}$$

Where x and y are the fractional portions of the significands, aka, mantissas. With this substitution applied, a finalized linear model for multiplication approximation can be derived. The equations governing our approximation method are pictured in Figure 4. Figure 4 shows a tree diagram for Levels 1 through 3, complete with conditional statements and equations. The approximate product of the inputs for any given level of approximation is found by evaluating conditional statements and following the corresponding branch. As discussed earlier, inputs are only scaled by powers of two.

Figure 5 shows the implementation example of Level 1 approximation. Conventionally, the sign bits are XORed and the exponents are added. Our tailored linear function is then evaluated. The final step of the process is normalization, which also occurs in conventional floating point multiplication. If the evaluated mantissa from the previous step is greater than two, it is shifted so that it lies between one and two, and the exponent is adjusted accordingly.

4.2 GPU Integration

The proposed ApproxLP was integrated into the AMD GPU architecture, Radeon HD 7970 device. We have implemented ApproxLP on Multi2sim, a cycle accurate CPU-GPU simulator [34]. The GPU consists of 32 computational units, where each consists of a set of four SIMD execution units and a scheduler. The SIMD has 16 execution cores, resulting in a total of 64 cores for each computing unit. We integrated ApproxLP on GPU architecture by replacing the existing floating point units in the Multiplier (MUL) and the multiply-accumulator (MAC) with our design. Since most learning algorithms use a MAC unit, we exploit ApproxLP to provide an approximate MAC unit. The MAC unit performs all multiplication operations approximately using ApproxLP. Additionally, when the accumulation of the multiplier exponents is much greater than the adder operand ($Z_{exp} - (X_{exp} + Y_{exp}) > 4$), the MAC unit ignores the addition and outputs the result of ApproxLP, since the multiplication term dominates the output.

Table 2: 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 [37]
UCIHAR	561	12	10MB	97.3%	98.1%	Activity Recognition[38]
PAMAP	75	5	240MB	95.8%	92.9%	Physical Monitoring[39]
FACE	608	2	1.3GB	96.1%	96.5%	Face Recognition[19]

5 EVALUATION

5.1 Experimental Setup

We developed both software and hardware implementation of ApproxLP for use in software simulation and architectural evaluation. The software approach is implemented using C++ and Python, and can be integrated with any application using multiplication. We calculated the energy consumption of existing FPU's using the FloPoCo [35] library and synthesized them using *Synopsys Design Compiler* in 45-nm ASIC flow. We also used *Synopsys Prime Time* to optimize the FPU's power consumption. The energy consumption and execution time of the proposed ApproxLP are estimated using a circuit level simulator, HSPICE, in 45-nm technology.

5.2 Workloads

We tested the efficiency of the proposed ApproxLP using both circuit-level and application-level simulation. For the application level, we evaluated ApproxLP efficiency/accuracy on a wide range of practical GPU applications including multimedia, Deep Neural Networks (DNN's), and brain-inspired Hyperdimensional computing [36]. Table 2 shows the number of features, number of classes and the baseline DNN and HD computing accuracy for each application. For neural network, each application contains a network with two hidden layers with 512 and 256 neurons respectively. In HD computing, the hypervector are in $D = 10,000$ dimensions.

5.3 ApproxLP & Multimedia Applications

For general GPU applications, we used image data and Peak Signal to Noise Ratio (PSNR) as a metric to check the quality of the computation. Similarly to prior work, we considered computations with higher that 30dB PSNR as visually acceptable quality [11, 23]. Table 3 lists the PSNR of different applications running on enhanced GPU with ApproxLP at different configurations. Our evaluations show that the PSNR value for all applications increases as more levels of approximation are included. For example, *Sobel* application provides 24.8dB PSNR at Level 1, while the quality of computation increases to 31.60dB at Level 2.

Table 3 also lists the energy efficiency and Energy-Delay Product (EDP) improvement as compared to baseline GPU. EDP improvement for each application is found at the level of approximation which satisfies a 30dB PSNR threshold. Our evaluation shows that ApproxLP can provide on average 2.14× and 2.39× energy efficiency and EDP improvements while ensuring acceptable quality loss.

Table 3: The quality of computation, energy and EDP improvement of ApproxLP running different applications.

Applications	PSNR (dB) Level 1	Level 2	Level 3	Energy Improv.	EDP Improv.
Sobel	24.85	31.60	37.03	2.17×	2.41×
Prewitt	26.63	37.08	37.08	2.08×	2.26×
Robert	31.78	39.58	56.42	2.43×	2.77×
GaussBlur	29.34	39.50	47.21	2.27×	2.53×
JPEG	37.70	45.06	50.80	1.85×	2.23×
FFT	34.50	52.51	53.45	2.04×	2.12×

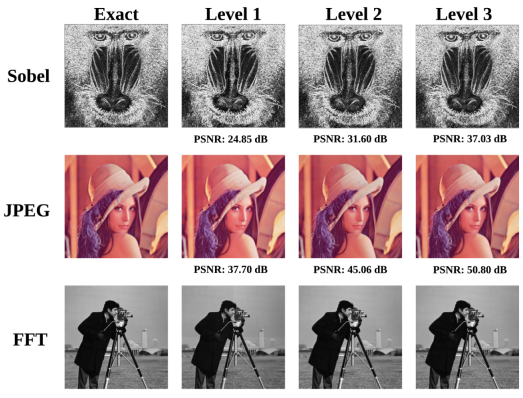

Figure 6: Quality of computation of different applications using ApproxLP at different approximation levels.

Table 4: Quality loss of DNN and HD computing applications running at different levels of ApproxLP.

Applications	DNN Level 1	Level 2	Level 3	HD Computing Level 1	Level 2	Level 3
ISOLET	0.83%	0.31%	0.06%	2.28%	0.86%	0.10%
UCIHAR	1.35%	0.62%	0%	0.94%	0.19%	0.03%
PAMAP	0.80%	0.21%	0%	0.48%	0.26%	0%
FACE	2.07%	0.94%	0.09%	1.32%	0.44%	0.07%

Figure 6 shows an example of computation quality using *Sobel*, *JPEG compression*, and *FFT* applications running on ApproxLP with different configurations. A visual inspection of the images reveals a very small difference between the exact image and the images produced by each of the three different ApproxLP levels. However, the corresponding PSNR values reveal significant improvement in quality when more levels are included while running ApproxLP.

5.4 ApproxLP & Machine Learning Applications

We also compared the accuracy of ApproxLP on neural network and hyperdimensional computing applications. Table 4 shows the impact of ApproxLP configuration on the classification accuracy of DNN and HD computing algorithms. Our evaluation shows that ApproxLP at Level 1 provides low quality losses for both DNN and HD computing applications. For example, DNN and HD on average have 1.06% and 0.98% lower classification accuracy as compared to the exact hardware. Quality loss decreases when using ApproxLP at Level 2 and Level 3 configurations. For example, at Level 2, DNN and HD computing applications lose on average 0.36% and 0.33% accuracy respectively. However, using ApproxLP at Level 3 configuration, both DNN and HD computing applications provide very similar classification accuracy as the baseline algorithm running on exact hardware, with less than 0.04% average quality loss.

Figure 7 shows the impact of ApproxLP configuration on the EDP of DNN and HD computation algorithms. Our evaluations on

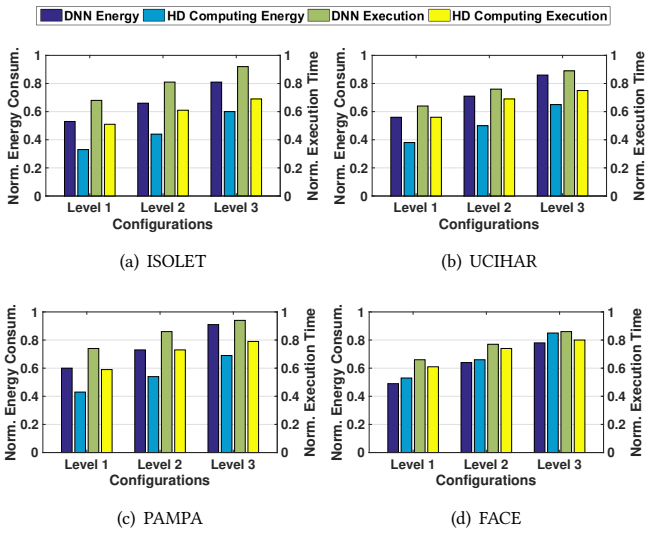

Figure 7: Energy efficiency and speedup of enhanced GPU running DNN and HD computing applications.

Table 5: Maximum error of ApproxLP versus Truncated Multiplication using equivalent number of operations.

# Operations	Truncated Mult	ApproxLP
4	23.44%	12.5%
8	6.152%	4.167%
12	1.556%	1.25%
14	0.78%	0.347 %

four DNN (HD computing) applications show that ApproxLP at Level 1 can provide 1.83× and 1.47× (2.39× and 1.76×) improvement of energy efficiency and speedup as compared to the baseline GPU, with an average of 1% quality loss. ApproxLP in more accurate configurations results in an average of 0.5% (maximum 1%) quality loss, while still providing 1.53× and 1.30× (2.18× and 1.49×) higher energy efficiency and faster computation when running DNN (HD computing) applications.

5.5 ApproxLP vs Prior Work

We compared ApproxLP with truncated multiplication. If N is the length of the truncated mantissas, multiplying two truncated mantissas using the conventional shift and add method requires $2N$ operations – N adds and N shifts. Given the same maximum error, ApproxLP requires less arithmetic operations than truncated multiplication, or conversely, given the same number of operations, ApproxLP yields a lower error. This happens because ApproxLP offers custom fit functions utilizing the minimum number of required operations. Table 5 compares the accuracy of ApproxLP with the accuracy of truncated multiplication. The number of operations at each level of ApproxLP varies slightly depending on the inputs, and the best case is presented in Table 5. Even at worst case, ApproxLP always offers more accuracy than truncated multiplication.

Table 6 compares the Energy-Delay Product (EDP) improvement of ApproxLP with two state-of-the-art configurable approximate multipliers: CFPU [22] and RMAC [23]. All results are produced by multiplying one million random numbers using different approximate hardware. CFPU and RMAC are hybrid designs which use both approximate and exact hardware depending on the desired maximum error value. In contrast, ApproxLP is a stand-alone computing unit with tunable error control. As our results show, RMAC

Table 6: Comparing the EDP improvement of ApproxLP and other approximate multipliers.

Error Rate	12.5%	8%	5%	3%	1.5%	0.5%
CFPU [22]	1.54×	1.32×	1.16×	1.07×	1.03×	1.01×
RMAC [23]	7.25×	11.07×	2.84×	1.63×	1.29×	1.06×
ApproxLP	13.69×	9.34×	9.34×	5.81×	5.81×	2.84×

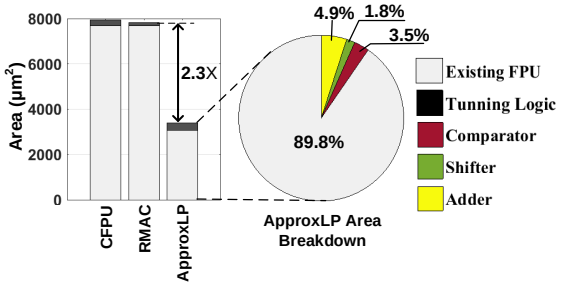

Figure 8: Area breakdown of ApproxLP and the state-of-the-art approximate multipliers.

and ApproxLP provide similar EDP improvement when the desired error rate is higher than 7%. However, due to the large error rate of RMAC in approximate mode, it needs to primarily run in exact mode in order to provide error rates of less than 7%. In contrast, as we show in Section 3.3, ApproxLP has runtime control of the computation error rate. Therefore, it can provide significantly higher EDP improvement even at low desired error rates. Our evaluation shows that ApproxLP can provide at least 4.5× and 2.7× EDP improvement compared to other approximate multipliers [22, 23] at 1% and 0.5% precision requirements.

5.6 Area Comparison

Finally, we compare ApproxLP and prior work in terms of area overhead. Figure 8 shows the area of each floating point unit in CFPU, RMAC and proposed ApproxLP. Both CFPU and RMAC are hybrid designs, requiring the inclusion of exact FPU hardware in addition to extra approximation hardware. Thus, the area of these FPU's are greater than that of exact hardware. For example, CFPU requires 3.4% (261.4 μm^2) more area to enable adaptive input selection and tuning. Similarly, RMAC takes 1.7% area overhead on top of exact FPU to enable tuning. By contrast, we propose ApproxLP as a stand alone approximate FPU. ApproxLP removes the necessity of using exact hardware, thus eliminating the large exact mantissa multiplier. Our evaluation shows that the ApproxLP main module requires about 60% less area than exact FPU. The peripheral circuitry to support different ApproxLP configurations, including an adder, shifter and comparator, take 101.0μm^2, 56.4μm^2, and 169.7μm^2 respectively. Our evaluation shows that in comparison with RMAC (CFPU), ApproxLP can achieve 2.3× (2.4×) lower area.

6 CONCLUSION

ApproxLP is a highly efficient multiplication approximator, built upon the principle of function linearization. The design features multiple levels of approximation, offering an increase in accuracy for the price of an increase in energy cost. ApproxLP provides EDP improvement as high as 5.3× compared to other state of the art approximate multipliers, while occupying up to 2.3× less chip area. ApproxLP completely removes the necessity for a hardware multiplier, as it only uses comparators, shifters, and adders, ensuring low energy consumption. ApproxLP also has no upper accuracy limit, and practical application of ApproxLP is only limited by a device's energy constraints.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] M. Shafique *et al.*, "Cross-layer approximate computing: From logic to architectures," in *DAC*, p. 99, ACM, 2016.
[2] M. Imani *et al.*, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.
[3] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE ETS*, pp. 1–6, IEEE, 2013.
[4] V. Gupta *et al.*, "Impact: imprecise adders for low-power approximate computing," in *ISLPED*, pp. 409–414, IEEE Press, 2011.
[5] S. Venkataramani *et al.*, "Quality programmable vector processors for approximate computing," in *MICRO*, pp. 1–12, IEEE, 2013.
[6] O. Akbari *et al.*, "Px-cgra: Polymorphic approximate coarse-grained reconfigurable architecture," in *DATE*, pp. 413–418, IEEE, 2018.
[7] S. Hashemi *et al.*, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *DATE*, pp. 1474–1479, IEEE, 2017.
[8] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, ACM, 2016.
[9] S. Li *et al.*, "Fpga acceleration of recurrent neural network based language model," in *FCCM*, pp. 111–118, IEEE, 2015.
[10] J. H. Ko *et al.*, "Adaptive weight compression for memory-efficient neural networks," in *DATE*, pp. 199–204, IEEE/ACM, 2017.
[11] S. Hashemi *et al.*, "tldrum: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, pp. 418–425, IEEE Press, 2015.
[12] S. Vahdat *et al.*, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *DATE*, pp. 1639–1642, IEEE, 2017.
[13] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *IVLSI*, pp. 346–351, IEEE, 2011.
[14] K. Bhardwaj *et al.*, "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems," in *ISQED*, pp. 263–269, IEEE, 2014.
[15] M. Courbariaux *et al.*, "Low precision arithmetic for deep learning," *arXiv:1412.7024*, 2014.
[16] M. Samragh *et al.*, "Lookn: Neural network with no multiplication," in *IEEE/ACM DATE*, 2017.
[17] M. Imani *et al.*, "Canna: neural network acceleration using configurable approximation on gpgpu," in *IEEE ASPLOS*, IEEE, 2018.
[18] "Amd app sdk v2.5 [online] available." http://www.amd.com/stream.
[19] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *IEEE/ACM ICCAD*, pp. 25–32, IEEE, 2017.
[20] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *DAC*, p. 6, ACM, 2017.
[21] M. Imani *et al.*, "Remam: low energy resistive multi-stage associative memory for energy efficient computing," in *IEEE ISQED*, pp. 101–106, IEEE, 2016.
[22] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *IEEE/ACM DAC*, pp. 1–6, IEEE, 2017.
[23] M. Imani *et al.*, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *ISLPED*, p. 12, ACM, 2018.
[24] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *IEEE/ACM ISLPED*, pp. 162–167, 2016.
[25] D. Jeon *et al.*, "Design methodology for voltage-overscaled ultra-low-power systems," *TCAS II*, vol. 59, no. 12, pp. 952–956, 2012.
[26] 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.
[27] D. Mohapatra *et al.*, "Design of voltage-scalable meta-functions for approximate computing," in *DATE*, pp. 1–6, IEEE, 2011.
[28] M. Imani *et al.*, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *IEEE/ACM DATE*, pp. 373–378, IEEE, 2017.
[29] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *IEEE TETC*, 2016.
[30] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *DATE*, pp. 1327–1332, IEEE, 2016.
[31] S. Narayanamoorthy *et al.*, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *TVLSI*, vol. 23, no. 6, pp. 1180–1184, 2015.
[32] M. Imani *et al.*, "Cade: Configurable approximate divider for energy efficiency," in *DATE*, IEEE/ACM, 2019.
[33] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.
[34] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *PACT*, pp. 335–344, ACM, 2012.
[35] "Aflopoco [online]. available:http://flopoco.gforge.inria.fr/,"
[36] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.
[37] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[38] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *IWAAL*, pp. 216–223, Springer, 2012.
[39] A. Reiss *et al.*, "Introducing a new benchmarked dataset for activity monitoring," in *ISWC*, pp. 108–109, IEEE, 2012.