Using a Configurable Floating Point Multiplier to Trade-Off Runtime Efficiency and Accuracy

[#1]Gurram Pranitha, M.Tech Student,

[#2]G. Karthick, [#3]N .Umapathi ***[#2, 3]Associate Professor,***

[#1,2,3] Department of Electronics and Communications Engineering,

***[#1,2,3]* Jyothishmathi Institute of Technology & Science, Karimngar, Telangana.**

ABSTRACT: Some degree of calculation inaccuracy is common in statistical applications, such as machine learning and data sensing. Approximate computations can be utilised to conserve energy and increase performance. The output error rate of some approximation solutions is uncontrollable, therefore they can only be employed in a limited number of situations. An approximation floating point multiplier called the CFPU is the primary purpose of this research. Our approach approximates multiplication by omitting the operation's most energy-intensive step and replacing it with a less energy-intensive one. CFPU dynamically picks the inputs that create the biggest approximation error and processes them in exact CFPU mode in order to fine-tune the amount of approximation. When completing at least 4% of multiplications in approximate mode, our CFPU outperforms a typical FPU. These multiplications save a large amount of energy in our evaluated applications. We found that by replacing the CFPU with FPUs on the AMD Southern Island GPU, we were able to reduce power consumption by 77% and improve latency by 3.5 percent for eight common Open CL applications. CFPU also enhances the energy-delay product by 2.4 over the best approximate multipliers currently available.

Index Terms— Approximation-based computation in graphics processing units (GPUs) that consume less power.

1. INTRODUCTION

The processing power required to run machine learning algorithms or multimedia applications on general-purpose processors like GPUs is considerable. Making allowances for slight errors rather than executing all calculations precisely can save energy and boost efficiency across many different applications. Due to the lack of control over the output error rate, approximate solutions can only be used in a restricted number of cases[1].

High-precision and wide-range applications are popular in the data-processing business. As a result, a wide range of computer systems, both classic and cutting-edge, make use of floating-point units (FPUs). Frame rendering and high-performance scientific computation, for example, both require a large amount of GPU resources to conduct FPU operations. More than 85% of Open CL's floating point arithmetic made use of multiplication. To match the dynamic range of a floating point unit, a fixed point unit would have to be five times larger and 40% slower.

A range of applications, including signal processing, neural networks, and real-time data processing, use multiplication inefficiently and time-consuming FP operations. There are numerous ways in which accurate multiplication units save energy. There have previously been publications that employed truncated or block-sized operands for multiplication to allow for approximation

multiplication. Although approximation designs provide some advantages, the absence of exact controls and the large area overhead outweigh these advantages..
Floating point multiplication's energy usage can be reduced with CFPU, a customizable floating point multiplication, which we recommend. There are two ways the CFPU avoids multiplication when dealing with floating point numbers[2]:

1) Mantissa Discarding -
In floating point multipliers, the mantissa is represented by the bottom 23 bits. There is a bottleneck in multiply operations because the mantissas of each operand are multiplied together. Discarding one of the input mantissas in favour of the second one is both faster and more energy efficient. To ensure that the final result is accurate, two adjustments are offered. Because utilising a single mantissa directly might lead to significant error rates for some multipliers, we propose two adjustments.
Adaptive operand selection identifies and eliminates the mantissa with the lowest error. Minimizing errors in each operation will allow us to run more computations on our CFPU[3] while keeping our current output error.
It is possible to anticipate the mistake by looking at the first N bits of the rejected mantissa.

2) Shift and Add –
There will be an approximation mode if the mantissa discarding does not yield results with an error below the user-specified threshold. We use the discarded mantissa to find the first '1' bit. Shifting the non-discarded mantissa and adding it to itself generates the new mantissa for the result value. While this method necessitates a greater investment of time and effort, the end results are more precise. Errors in the second stage are at least 50% lower than those in the first stage for the same set of input operands. In the event that neither of these approaches is successful, our system has the option of allocating the inputs that result in the highest output error to the CFPU.
The effectiveness of the suggested approach on AMD Southern Island GPUs is assessed by substituting the CFPU for the conventional FPU. When compared to an unmodified GPU, CFPU approximation improves EDP by 3.5, while ensuring less than 10% average relative error for OpenCL applications. When you add a second stage, you get a rise in EDP precision from 3.0 to 4. Previous cutting-edge multipliers have a lower mistake rate and a higher energy-delay product than the planned CFPU, which is being developed.

CFPU's impact on various machine learning approaches is also examined. Improved performance and reduced output inaccuracies are necessary as these algorithms become more widely used. Stochasticity allows them to accept some mistake in their output. We employ K-Nearest Neighbor (KNN), Back Propagation (BP), and K-means to test our design[4]. Back Unlike K-means and K-nearest neighbour, neural network weights are trained using propagation, whereas these algorithms rely on dense linear algebraic computations in data mining applications It can save up to 2.4 watts of energy and boost machine learning algorithms by 2.0 times while maintaining an average relative error less than 1% when compared to a non-updated GPU architecture. These benchmarks utilise 50% less energy and 40% less time when performed on a two-stage CFPU, compared to the previously proposed one-stage CFPU system.

2. RELATED WORK

APPROXIMATE FPU MULTIPLIER

Floating point units (FPUs) are more expensive and need more energy than integer processing units due to the more complicated storage method used for floating point numbers. Multimedia streaming and neural networks are examples of current applications where multiplication-based computations are wasteful. There is a higher concentration of multiply and multiply-add (muladd) operations than adds in the Sobel picture filter program. In 45nm, floating point addition only uses 0.9 pJ of energy, according to Horowitz and colleagues. Because a floating point multiplication consumes just 20% more power than an integer multiplication, fixed point operations save a substantial amount of power[5]. Sobel's floating-point multiply and muladd operations use roughly 90 percent of the ALU's energy, making them prime candidates for energy optimization.

Floating point multiplication efficiency can be improved by using a two-stage multiplier. Early levels increase multiplication by directly reusing an input mantissa in a subsequent output. The second stage shifts and adds the retained mantissa to itself in an effort to further reduce error.

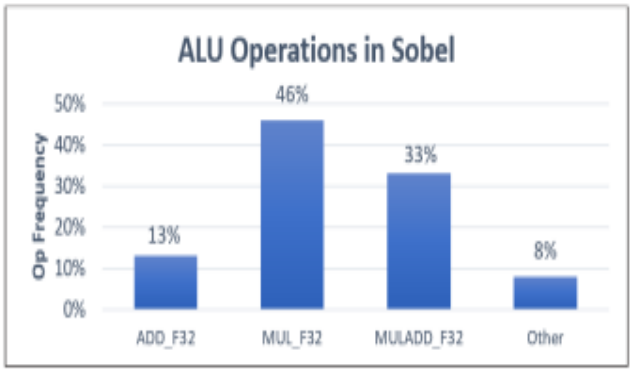

Fig1. ALU operation breakdown for the Sobel application

IEEE 754 FLOATING POINT MULTIPLY

The sign bit, the exponent, and the fractional value are all separate elements of a floating point number. When using IEEE 754 floating point notation, bits 31 to 24 are reserved for exponent values, while remaining bits are reserved for fractional values, which are referred to as mantissa[6].

It's clear to see that the exponent bits are utilized to represent the power of two. The mantissa bits must be multiplied by 2exp to get a value between 0 and 1.

C = A B = C's sign bit may be obtained by XORing the sign bits of both A and B. Exponentiation terms are joined in step two. The final mantissa is determined by multiplying the two mantissa values together. The mantissa is always between 1 and 4, due to the fact that the multiplication uses a mantissa whose values vary from 1 to 2. The exponent is incremented by one and the output mantissa is split by two if the output mantissa is bigger than 2.

CFPU USAGE

CFPU is an unknown quantity to the application at this time. User inputs Errormax, the maximum error that can occur during a multiply operation, before the process begins. A more accurate approximation mode or precise hardware will be used if the error is less than or equal to this number, according to CFPU.

The flow diagram of the suggested design is depicted in the figure. Selecting a precise output is the goal of adaptive selection. The MUX between inputs A and B copies one mantissa from A and discards it. The computation is deemed complete if a predetermined value is not exceeded against the first N bits of the discarded mantissa. Shift and add are used in the second stage if the threshold is exceeded. Using the accurate FPU hardware becomes necessary if the second stage error still exceeds Errormax. The following sections go into deeper depth on the alterations.

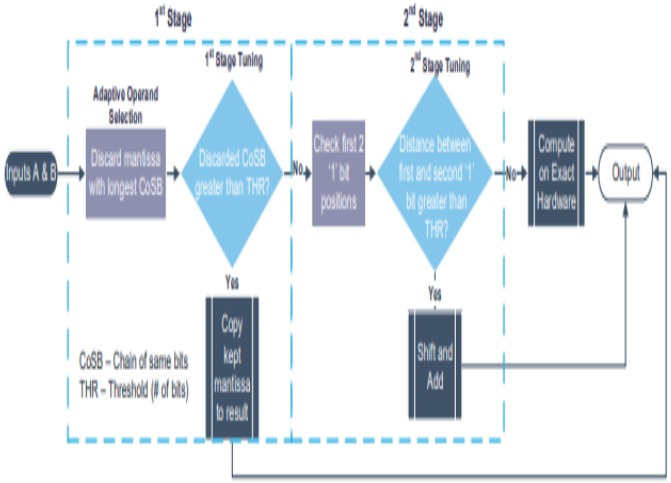

Fig2. CFPU operating flow including 1st and 2nd stage approximation

RMAC OVERVIEW

Sign, exponent, and fractional values in IEEE 754 32-bit floating point notation are encoded as 32-bit binary numbers (A32,...A1). The sign bit is the first bit in floating point notation (A32). In binary numbers, the exponent is represented by a range of eight bits, which are -126 to 127 in length. (A31,...,A24). Between one and two, this value is known as the mantissa. Known as the mantissa, these 23 bits represent the mantissa (A23,...A1). FIGURE shows a floating-point multiplication of A and B, the input operands (a). Exponents and mantissas of A and B are multiplied after the XOR of their sign bits. The mantissa multiplication process must be sped up because it is the most expensive procedure[7].

Exact and approximation multiplication are both possible with the RMAC floating point multiplier, which is shown below as a solution. The input values are multiplied exactly when IEEE-754 32-bit precision is applied. Mantissa addition has been substituted for mantissa multiplication in our suggested architecture, as seen in Figure (b). The extra carry bit is also applied to the exponent in the event of an overflow. Adding and shifting the operation between partial products makes this a near approximation to fixed-point multiplication. When the mantissa values are normalized to a range of 1-2, a close approximation to mantissa multiplication can be obtained by adding the shift. This is already done in floating point format.

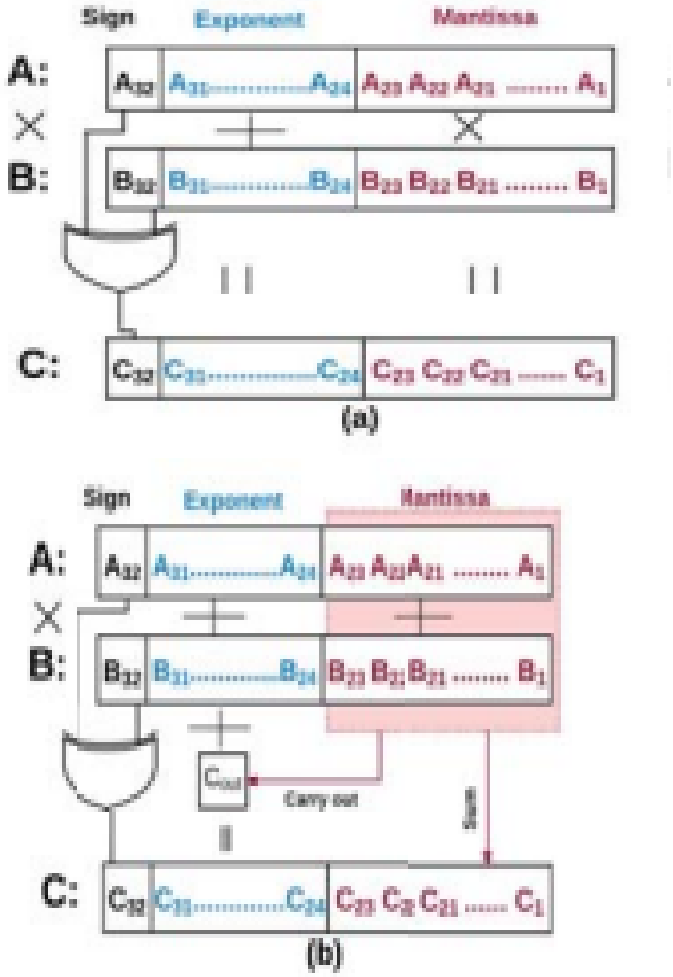

Fig 3: The multiplication between A and B floating point values in (a) IEEE-745 32 bit standard and (b) the proposed RMAC

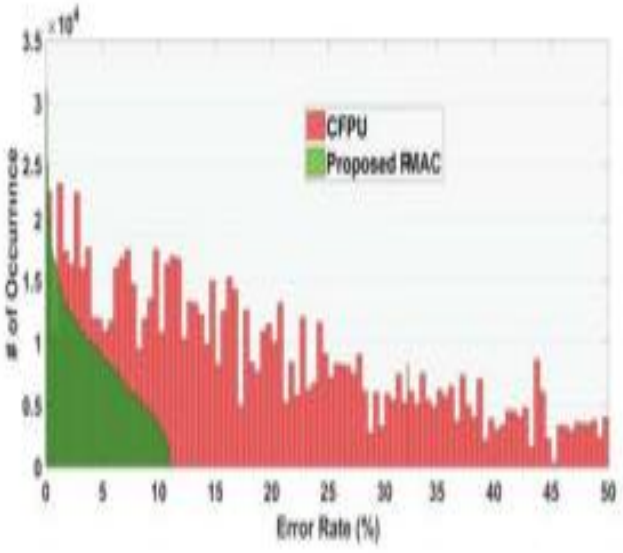

Fig 4: Histogram of the error distribution of the CFPU and proposed RMAC

Comparing the proposed error distribution of RMAC to that of CFPU, the most recent approximate FPU, involves multiplying 1,000,000 random floating point values together. It shows the error histograms for the two architectures. In comparison to RMAC, CFPU[8] error rates can approach 50%, which is nearly five times higher. According to the data, RMAC suffers a quality loss of up to 11.1%.

It is impossible to predict the RMAC's error rate if it does not have a suitable tuning mechanism. Correct answer is 50 for the floating point multiplication of A=5 and B=10. There is a 4% mistake due to the RMAC approximation in this case, which is set at 48. For A and B, the approximate product is 128 and 144, respectively, with an error of 11.1%. Users can choose how much inaccuracy they're willing to put up with because the RMAC's error rate can be lowered through a tuning method. The only option available when both input operands are less than one is approximation mode.

Accuracy Tuning

An approximation's sensitivity changes depending on how it is used in the real world. Programs' approximation behavior can vary widely even within the same one. When used with general-purpose processors, such as the CPU and GPU, RMAC needs to be flexible and data-dependent by design. This means RMAC should have the ability to modify the level of precision based on the application that is running. For an approximation, it is possible to analyze its error rate in order to see if it meets the appropriate level of precision. Multiplication can be re-started in exact mode if the target error rate is not met. Any A or any B might be multiplied by any other to produce a tuning approach for IEEE 754's 8, 16, and 32-bit floating point notation. There are distinct RMAC approximation patterns when the initial bit of the mantissa is altered. The RMAC approximation's sensitivity to input operands is shown in a 32-bit evaluation in Figure 3. The rate of error is directly proportional to the number of 1s and 0s in a row.

The diagram depicts the tuning process. The leading mantissa bits of both input operands are first detected by RMAC. The RMAC can change error rates based on A23 and B23 values in one of the following ways:

Case 1.

RMAC's error rate is affected by the amount of successive 0s in mantissa C when A23 and B23 are both 1. (shown in Figure a). RMAC uses the number of consecutive 0s in the answer's mantissa to adjust the accuracy level. As the necessary level of precision rises, so does the number of successive zeros. Mantissa error rates range from 11.1% when the answer contains only 0s to 5.53% when the mantissa begins with a 0 and concludes with a 1.

Case 2.

As illustrated in the first picture, if both A23 and B23 are zero, the number of consecutive one-digit mantissas determines the right result. Figure b shows the RMAC error rate if the mantissa C comprises N consecutive 1s, as illustrated. Assuming that the answer is given with a single 1 as a mantissa, the greatest mistake is 11.10 percent, and the minimum error is 7.43 percent.

Case 3.

C23 is taken into account when A23 and B23 differ in value. The mantissa of the responses determines whether or not C23 is valid. Figure c shows RMAC's error rate when C comprises N continuous 1s or 0s. " In this situation, the maximum and minimum errors are both feasible due to the fact that C23 might be either 1 or 0. On average, the mantissa C has an inaccuracy of 11.10 percent when all 1s are present, but only 6.66 percent when an initial 1 is present, followed by a zero. A leading zero followed by a 1 in the first place of the mantissa C results in the lowest inaccuracy of 4.76 percent when C23 is zero[9].

Approximation accuracy improves when the number of consecutive 1s or 0s decreases. By multiplying two floating numbers, the tuning operation is depicted in the figure (50 and -25). There are a total of one zeros in C23 to C1, which is represented by A23 and B23, which are both one in the design. This approximation yields an error rate of 7.84 percent because mantissa C has two consecutive 0s, which corresponds to the results in Figure a. Using varying N values, the RMAC can be fine-tuned to achieve a desired level of accuracy. For example, if N=1 necessitates greater precision, the application will execute in precise mode. This scenario will be conducted in approximation mode if N is set to 5, for example.

The RMAC error distribution is shown in the figure when N different tuning bits are utilized. If no tuning bits are used, the maximum RMAC error rate is displayed in the figure. In the case of N = 3 tuning bits, the error rate is lowered to 10.7%. When N is dropped to two, the mistake rate rises to 9.3%, and when N is reduced to one, the error rate drops to 6.8%.

3. EXPERIMENTAL RESULTS

EXPERIMENTAL SETUP

The Radeon HD 7970 processor uses a GPU Southern Island design to properly implement the RMAC. Three primary floating point units in the GPU architecture have been added to Multi2sim's cycle-accurate CPU-GPU simulator to emulate RMAC. Synopsys Design Compiler is used to synthesis all of the FPUs, and the Prime Time tool suite is used to optimize power consumption. The energy consumption and execution time of the proposed RMAC[11] were modelled using the HSPICE simulator in 45-nm technology. Several GPU-based apps are utilized to verify the effectiveness of the proposed RMAC at the application level, as demonstrated below:

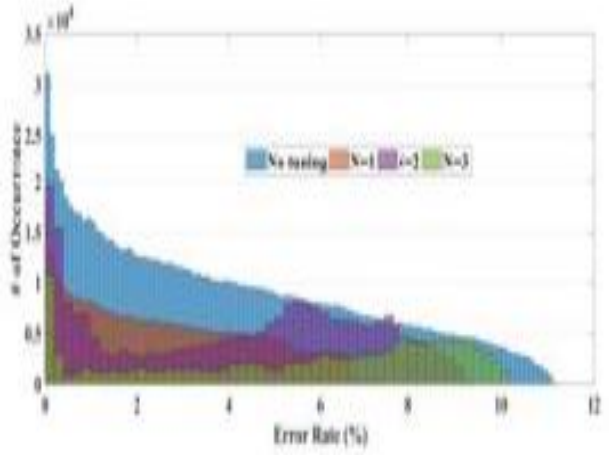

Fig 5: RMAC error distribution on different tuning bits.

Table: Comparison of the proposed RMAC with state of the art approximate multipliers.

	Kulkarni[18]	ESSM8 [17]	DRUM6 [16]	CFPU3 [21]	RMAC
Tunable	No	No	No	Yes	Yes
Max Error	22.2%	11.1%	6.3%	6.3%	6.3%
Energy (pJ)	2.9	0.58	0.55	0.27	0.15
Execution (ns)	3.5	2.1	1.9	1.6	0.52

Table: CFPU and RMAC Hit rate and EDP improvement while multiplying 1 million randomly generated numbers.

Error Rate		1%	2%	4%	6%	10%	12%
CFPU[21]	*Hit Rate*	3.4%	7.5%	14.7%	19.4%	31.1%	37.2%
	EDP Improv.	1.03 x	1.07 x	1.16 x	1.22 x	1.41 x	1.54 x
RMAC	*Hit Rate*	25.1%	42.7%	71.2%	93.7%	99.9%	100%
	EDP Improv.	1.29 x	1.63 x	2.84 x	6.80 x	11.07 x	11.31 x

Open CL:

Sobel, Roberts, HwtHaar1D, and Binomial Option were selected using the AMD APP SDK v2.5. We use Caltech 101a as our dataset for image processing applications, while a random generator is used to produce the dataset for other applications. For these applications, we devised an average relative error metric to measure quality.

Rodinia:

For our tests, we use Rodinia 3.1's benchmark set, which includes K-means, back propagation, Lud, and K-nearest neighbor (KNN). In neural networks, Back Propagation is used to train weights, whereas in data mining, K-means and K-nearest neighbor are in use. How frequently classification or clustering points are erroneously classed or clustered is a measure of quality loss in Rodinia applications (K-means and KNN).

Neural Network:

The effectiveness of RMAC is measured using three different NN applications. Open CL, a widely accepted programming language for heterogeneous computing, is used to implement NNs in practice. This resulted in a 95 percent accuracy using 10-batch-size stochastic gradient descent and 0.1-momentum learning rates. It has been set to "Rectified Linear Unit" in order to limit the number of activation functions to six. The "Softmax" function is applied to the output layer.. The quality loss for neural networks is defined as the difference in classification accuracy between programs executed on precise and approximation hardware. The following are some of the applications that have been tested:

Hand Writing Recognition (MNIST):

The MNIST dataset, a well-known source for machine learning, contains images of handwritten digits. There are ten digits (0, 1, 2, 3, 4, 5, 6, 7, or 9) to choose from when it comes to the picture input number.

Activity Recognition (HAR):
Linear and angular velocity measurements are used to detect human activity at 50Hz.

Voice Recognition (ISOLET):
ISOLET is a 150-speaker voice collection. Each of the English alphabet's 26 letters can be allocated to a different sound effect in this challenge.

COMPARISON
It is compared to other top approximation multipliers, such as ESSM, Kulkarni, DRUM, and CFPU, on the basis of energy efficiency and execution time. In varied designs, the only approximation multipliers having runtime adjustment capabilities are the CFPU and the RMAC. It is possible to control the level of approximation without requiring any input data through a variety of multipliers. Assuming that all designs have the same maximum error rate, the table below demonstrates the energy efficiency and execution time of various multipliers at their optimal values Due to the data-dependent nature of CFPU and the proposed RMAC, we examined their average energy and execution time using 1 million randomly produced data points. At least 2.1 more energy efficiency and 1.7 more execution speed may be achieved when the error rate is equal to one (N = 1). Due to the RMAC's capacity to approximate a greater percentage of values than the CFPU, it is more efficient than the CFPU.

They are both programmable, thus the CFPU and RMAC that have been shown are tested simultaneously. Table 2 illustrates the CFPU's hit rate and EDP improvement, as well as the proposed RMAC multiplication, after executing 1 million random values. Results are shown when the total number of test data points falls below a predefined error rate. The average time each multiplier spends in a mode that approximates the total number of multiplications is called the hit rate. For floating point multipliers, the EDP improvement is large. In spite of the fact that RMAC's computational quality is comparable to CFPU's, it has a far better success rate and more efficiency than CFPU. CFPU and RMAC, for example, have attained rates of 19.5 and 93.7 percent, respectively, and EDP increases of 1.22 and 6.8 percent above standard FPU, respectively.. The efficiency gap between CFPU and RMAC rises as the mistake rate is raised. As-is, the RMAC promises a low error rate of fewer than 11%. No tuning mechanism is required for RMAC to work with an error rate of 11.1 percent, demonstrating that RMAC may be utilized without tuning. However, approximation can result in up to a 50% increase in CFPU error rates.

Efficiency in Application Level
An energy usage and execution time graph depicting several programs operating on updated GPUs, which are standardized to GPUs with standard FPUs, may be seen here. The great majority of workloads can be served well by GPUs even without RMAC modification or optimization, according to our research. Rodinia applications appear to be more resistant to the RMAC approximation because of the stochastic and approximate nature of several Rodinia benchmarks. Approximate RMAC units, such as digital filters, have a bigger impact on quality than other RMAC units. We found that RMAC-enabled GPUs can save between 3.0 and 2.9% more energy while delivering 2.3% and 1.9% more performance on popular OpenCL and Rodinia workloads, with a quality loss of no more than 7.4 percent, in our experiments.

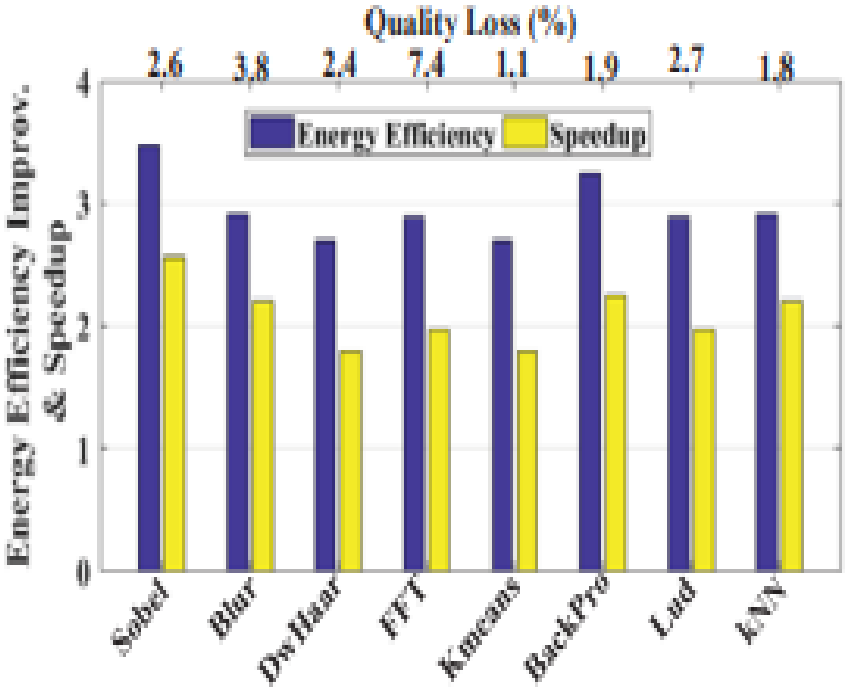

Fig 6: It's possible to enhance energy efficiency, speed up, and reduce quality by using newer GPUs in many applications (RMAC with no tuning).

EFFICIENCY-ACCURACY TRADE-OFF

Our new GPU should be able to run a wide variety of applications in order to maintain the GPU's universality. A wide range of applications will require RMAC to be able to change its precision. Different sorts of tuning bits can be used for a number of purposes, as shown in Figure 8. The RMAC's normalized energy and execution time are shown on the y-axis, while the number of tuning bits is shown on the x-axis. Different RMAC setups, according to our findings, deliver the same level of service quality to apps. Machine learning applications like k-means may achieve a quality loss of 1% with no tuning strategy at all, whereas a Sobel application requires four tuning bits to ensure a loss of quality of less than 1%.

The chart indicates that each application loses a different degree of quality, despite the fact that each application has distinct amounts of configuration and energy deferred product improvement. When compared to standard deterministic OpenCL benchmarks, Rodinia benchmarks, which use lower approximation levels, suffer a quality loss of less than 1%. (lower tuning bits). An increase in energy efficiency of 2.1 percent and a performance boost of 1.7 percent above regular FPU-based GPUs has been demonstrated using updated GPUs (a 3.6 EDP improvement). It consumes less energy and takes less time to complete, resulting in an EDP improvement of 5.2%. (less than 2 percent quality loss).

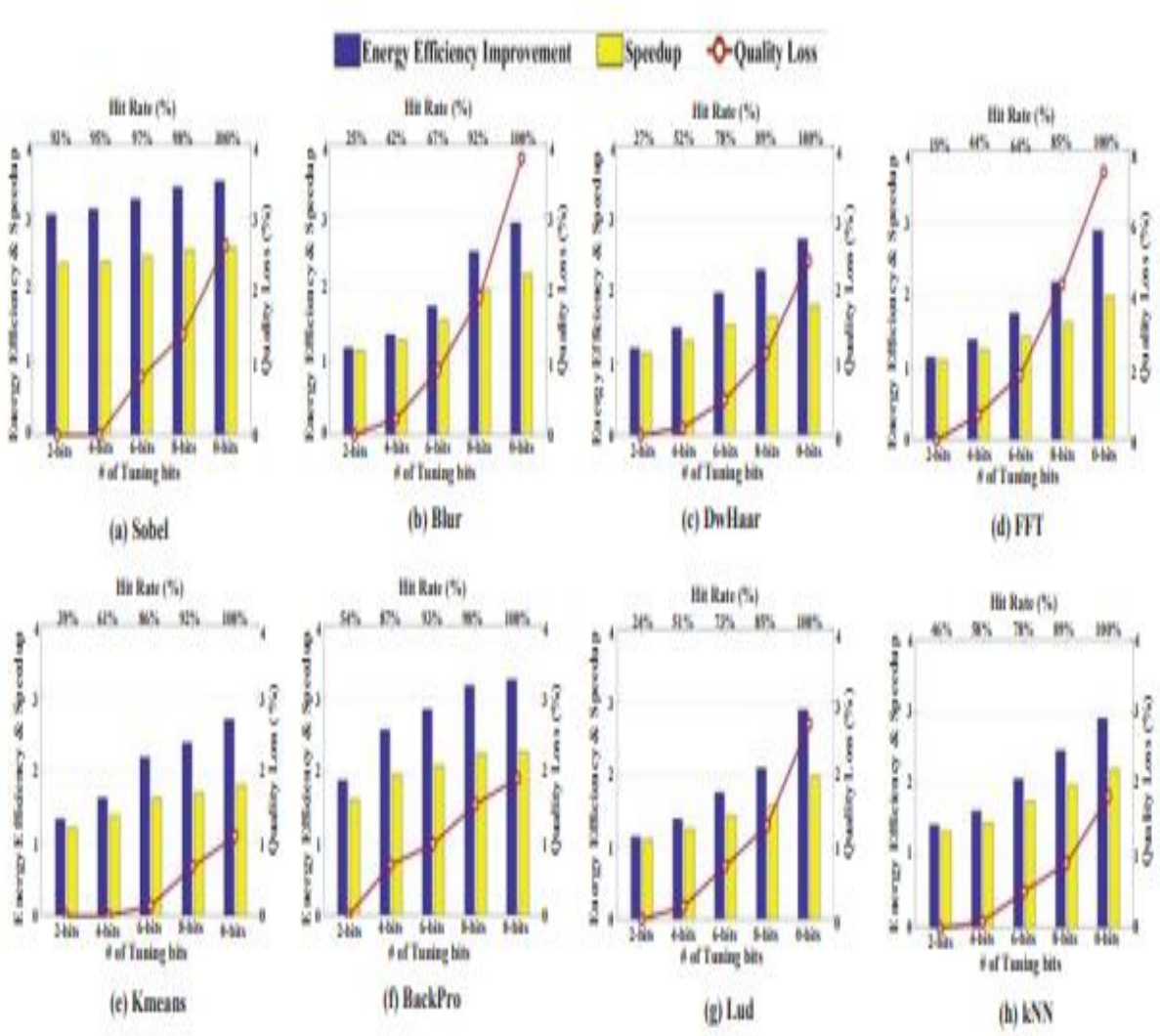

Fig 7: Power consumption and execution time can vary depending on the RMAC configuration. There are N tuning bits on the x-axis, and the left y-axis shows normalization energy and execution while the right y-axis shows accuracy.

Table: The application setting and the product's energy delay can both be optimized to reduce different types of computation quality losses.

Quality Loss		Sobel	Blur	DwHaar	FFT	K-means	BackPro	Lud	kNN	Average
0%	*N Tuning*	4-bits	2-bits	2-bits	-bits 2-bits	4-bits	2-bits	2-bits	2-bits	**2.47**×
	EDP Improv.	7.37×	1.38×	1.36×	1.25×	2.22×	2.99×	1.25×	1.91×	
1%	*N Tuning*	6-bits	6-bits	6-bits	4-bits	8-bits	4-bits	6-bits	8-bits	**3.98**×
	EDP Improv.	7.94×	2.81×	3.01×	1.71×	4.02×	4.97×	2.50×	4.87×	
2%	*N Tuning*	8-bits	8-bits	8-bits	6-bits	0-bits	0-bits	8-bits	0-bits	**5.23**×
	EDP Improv.	8.59×	5.08×	3.76×	2.50×	4.86×	7.31×	3.31×	6.43×	

4. CONCLUSION

A variable floating point multiplier that consumes much less energy and performs significantly better is described in this article. The proposed approximate multiplication can be fine-tuned by precisely processing an unknown proportion of input. Floating point is used to implement the required approximation architecture in our notion. Using current FPUs instead of CFPUs reduces power usage by 77% and improves energy delay quality by 4.8%, according to our testing. While keeping the same degree of precision, the CFPU reduces energy delay by 2.4 compared to conventional approximation multipliers.

REFERENCES

[1]. J. Gantz and D. Reinsel, "Extracting value from chaos," IDC iview, vol. 1142, no. 2011, pp. 1–12, 2011.

[2]. L. Atzori et al., "The internet of things: A survey," Computer networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[3]. N. Umapathi, G.L. 2020. Design and Implementation of Low Power 16x16 Multiplier using Dadda Algorithm and Optimized Full Adder. International Journal of Advanced Science and Technology. 29, 3 (Feb. 2020), 918 - 926.

[4]. M. K. Tavana, A. Kulkarni, A. Rahimi, T. Mohsenin, and H. Homayoun, "Energy-efficient mapping of biomedical applications on domainspecific accelerator under process variation," in Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on, pp. 275–278, IEEE, 2014.

[5]. Fehske et al., "The global footprint of mobile communications: The ecological and economic perspective," IEEE Communications Magazine, vol. 49, no. 8, 2011.

[6]. F. Xia et al., "Internet of things," International Journal of Communication Systems, vol. 25, no. 9, p. 1101, 2012.

[7]. Murali Krishna G., Karthick G., Umapathi N. (2021) Design of Dynamic Comparator for Low-Power and High-Speed Applications. In: Kumar A., Mozar S. (eds) ICCCE 2020. Lecture Notes in Electrical Engineering, vol 698. Springer, Singapore. https://doi.org/10.1007/978-981-15-7961-5_110

[8]. J. Gubbi et al., "Internet of things (iot): A vision, architectural elements, and future directions," Future generation computer systems, vol. 29, no. 7, pp. 1645–1660, 2013.

[9]. N. Khoshavi, X. Chen, J. Wang, and R. F. DeMara, "Read-tuned stt-ram and edram cache hierarchies for throughput and energy enhancement," arXiv preprint arXiv:1607.08086, 2016.

[10]. J. Han et al., "Approximate computing: An emerging paradigm for energyefficient design," in IEEE ETS, pp. 1–6, IEEE, 2013.

[11]. N. Umapathi, G. M. Krishna and L. Srinivas, "A Comprehensive Survey on Distinctive Implementations of Carry Select Adder," 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), 2021, pp. 1-5, doi: 10.1109/ICNTE51185.2021.9487718.

[12]. Sekanina, L. Introduction to approximate computing: Embedded tutorial. In Proceedings of the 2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits

& Systems (DDECS), Kosice, Slovakia, 20–22 April 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.

[13]. Liang, J.; Han, J.; Lombardi, F. New metrics for the reliability of approximate and probabilistic adders. IEEE Trans. Comput. 2012, 62, 1760–1771.

[14]. Yin, P.; Wang, C.; Liu, W.; Lombardi, F. Design and Performance Evaluation of Approximate Floating-Point Multipliers. In Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, USA, 11–13 July 2016; pp. 296–301.

[15]. Srinivas, L., & Umapathi, N. (2022, May). New realization of low area and high-performance Wallace tree multipliers using booth recoding unit. In AIP Conference Proceedings (Vol. 2393, No. 1, p. 020221). AIP Publishing LLC.