ApproxLUT: A Novel Approximate Lookup Table-Based Accelerator

Ye Tian, Ting Wang, Qian Zhang and Qiang Xu

Department of Computer Science & Engineering, The Chinese University of Hong Kong
Shenzhen Research Institute, The Chinese University of Hong Kong
{tianye, twang, qzhang, qxu}@cse.cuhk.edu.hk

***Abstract*—Computing with memory, which stores function responses of some input patterns into lookup tables offline and retrieves their values when encountering similar patterns (instead of performing online calculation), is a promising energy-efficient computing technique. No doubt to say, with a given lookup table size, the efficiency of this technique depends on which function responses are stored and how they are organized. In this paper, we propose a novel adaptive approximate lookup table based accelerator, wherein we store function responses in a hierarchical manner with increasing fine-grained granularity and accuracy. In addition, the proposed accelerator provides light-weight compensation on output results at different precision levels according to input patterns and output quality requirements. Moreover, our accelerator conducts adaptive lookup table search by exploiting input locality. Experimental results on various computation kernels show significant energy savings of the proposed accelerator over prior solutions.**

***Index Terms*—approximate computing, lookup table, energy efficiency**

I. Introduction

For complex functions that appear frequently in a program, computing with memory [1], which stores function responses of some input patterns in a lookup table (LUT) offline and retrieves their values without conducting online computations with processors, is able to achieve considerable energy savings and performance improvements. For example, involving hundreds of instructions, the commonly used function $\sin x$ has an energy consumption of about 100nJ, while energy for read operation of a 512kB SRAM at 32nm technology is only about 100pJ [2].

Another promising energy-efficient computing technique is approximate computing [3, 4], which leverages the intrinsic error resilience characteristics of emerging applications and trades off between computation quality and computational effort. This error resilience can be attributed to the following factors. Firstly, many big data applications often process large amounts of noisy and redundant data drawn from real world; Secondly, there is usually no specific "golden" output value but rather many "acceptable" outputs, and users are accustomed to any result of acceptable quality; Finally, algorithms and computation processes employed to find solution in these applications can be stochastic in nature or resort to error-resilient methods.

Recently, many researches on associative computing using content addressable memory (CAM) [5, 6] explore the combination of computing with memory and approximate computing techniques in order to achieve even larger energy efficiency gains. The proposed CAM enables low voltage operation and has ultra-low energy consumption. Instead of storing all possible patterns, CAM prestores only a subset of patterns and realizes approximation by inexact matching, i.e. finding a similar pattern within some distance to output an approximate result. With significantly reduced table size, the lookup table becomes easily implemented. Also, searching a table with much fewer entries can obtain remarkable energy savings and performance gains.

Even for error-resilient applications, there would be certain quality requirements on the final outputs, and the error constraints during computation can vary significantly at runtime [7]. Existing methods usually prestore patterns with the highest frequency and search for the nearest pattern in the LUT based on hamming distance when a new input pattern comes. They do not estimate or control the output errors, or adjust precision for varying online quality requirements. Therefore, the output quality is not guaranteed, and the application error resilience capability is not used to its full potential.

Motivated by the above, in this paper, we propose an adaptive lookup table based accelerator for approximate computing, named *ApproxLUT* and make the following contributions.

1) ApproxLUT is an effective approximate accelerator for commonly-used computation-intensive functions, wherein the lookup table is constructed offline considering both the features of target function and the application error resilience capability. ApproxLUT controls computation quality with worst case error or error expectation. When the approximate results stored in the LUT cannot satisfy the quality requirement, the function will be calculated in fully accurate mode. More importantly, the lookup table itself is hierarchical, consisting of subtables with different granularity and stores not only output but also parameters for compensation. With such design, ApproxLUT can provide light-weight compensation and output results of multi-precision levels according to online inputs and requirements.
2) We propose an online locality-based adjustment strategy for table search. In many applications, if a particular input operand for a certain function appears at a particular time, then it is likely that the same input or

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

its nearby inputs will appear in the near future. Based on such "input locality", ApproxLUT adjusts priority of subtables online to gain efficiency when searching the whole table.

The remainder of this paper is organized as follows. Section 2 presents related works on associative computing. Section 3 details our proposed adaptive lookup table based accelerator – ApproxLUT. In Section 4, experimental results show energy saving and quality loss of ApproxLUT when applying to various functions. Finally, Section 5 concludes this paper.

II. Related Works

Associative computing based on lookup table utilizes outputs similarity between neighboring inputs to enable computation reuse. Associative memories can be implemented on either software or hardware. When the address of a storage location for data can be obtained by some mapping algorithms applied to the data contents, hash-coding can implement a fast search by pure programming with conventional computer memories [8]. [9] proposes a family of lookup schemes which compactly encodes large tables and fits the associative memory into overall memory architecture with low overhead.

Associative memory on hardware uses content-addressable memory (CAM) to store input operands and resistive RAM (ReRAM) to store corresponding output results. [5, 6, 10] store only a subset of patterns and match an input pattern with prestored ones by applying an approximate search. Rather than exactly searching for the input pattern in the lookup table, these works find a similar one within some distance for approximate computing. [5] stores patterns of highest frequency, and uses voltage overscaling (VOS) to deliberately relax the searching criteria to approximately find stored patterns within a 1 or 2 bit Hamming distance of the search pattern. As significant bits may have much greater weight, it can result in large error when "important" bits of input are different from the stored operand. [10] considers the weight of each bit and finds the pattern that has the nearest distance by designing a multi-stage CAM. However, as how far is the closest distance cannot be predicted, the output error cannot be estimated and bounded.

III. Methodology

For a given application, at offline stage, we firstly select appropriate functions to accelerate considering both error resilience and computation intensity, and then apply ApproxLUT to the target functions.

The proposed framework of ApproxLUT is presented in Figure 1, which consists of three components, i.e., control unit, lookup table and arithmetic logic unit. The lookup table has two levels, and the second-level table is composed of a set of subtables with finer granularity compared to the first-level table. The number of levels of LUT determines how many precision levels ApproxLUT has. In this paper, we only discuss 2-level LUT, but it could be extended to more levels based on the amount of online quality variations and lookup overhead. For online calculation, given input operand and error bound, the accelerator computes approximately with the table and outputs an acceptable result. As mentioned earlier, the lookup table can be stored in SRAM, non-volatile memory, associate memristive memory, or other kinds of memory. And control unit can be implemented at either software or hardware level.

In this section, we first briefly introduce how to select proper target functions. Next, we discuss how to construct the lookup table at offline stage and perform online approximate computation.

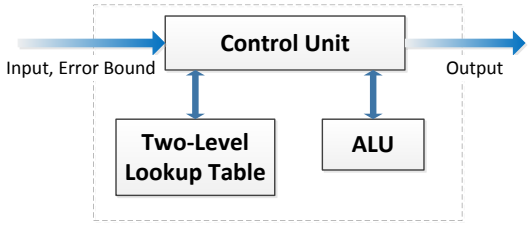

Fig. 1. Framework of Lookup Table Based Accelerator for Approximate Computing

A. Function Selection

Appropriate target function should be chosen for approximation before building the accelerator. Most applications and algorithms can be further decomposed into computational kernels with different error resilience capabilities. Some kernels are error-resilient, as small errors have little impact on the final output quality. While other kernels are less error-resilient or even error-sensitive, which are not suitable for approximation. The chosen function must be in error-resilient kernels in the target application. To obtain as much saving as possible, the function should be computation-intensive and frequently used.

Given any input, lookup table based accelerator searches for the pattern having the shortest distance from the input. To control the error and make valid approximation, target function must have similar outputs with similar inputs. Many functions have this property, for example, continuous function that satisfies the following Weierstrass and Jordan definitions. Given a function $f(x)$ and any element c of the domain of definition I, for every $\epsilon > 0$, there exists a $\delta > 0$ such that for all $x \in I$,

$$|x - c| < \delta \Rightarrow |f(x) - f(c)| < \epsilon.$$

Therefore, assuming that x_i is the nearest stored pattern from x, output error $|f(x) - f(x_i)|$ is upper bounded.

A large set of functions and applications satisfy the above properties and can be applied to ApproxLUT, some of which are listed in Table I.

B. Offline Lookup Table Construction

In this section, we will explain how to construct lookup table of the accelerator offline for certain function. There are mainly three steps:

- Decide the formula and coefficients for light-weight compensation;

TABLE I
COMPUTATION-INTENSIVE FUNCTIONS AND CORRESPONDING APPLICATIONS

Function	Application
Trigonometric Functions: $\cos x, \tan x, \arcsin x$	Geometry
Exponential Function: e^x Logarithmic Function: $\ln x$	Financial
Error Function: $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	Statistics
Riciandenoise 3D: $f(r) = \frac{r(2.38944+r(0.950037+r))}{4.65314+r(2.57541+r(1.48937+r))}$	Medical

- Select data points to store in the table and calculate the error of LUT;
- Construct the lookup table level by level.

1) Formula and Coefficients for Compensation: Given function $f(x)$, assuming that x_i is the nearest stored pattern in the lookup table from input x, error of output is $f(x) - f(x_i)$. To make light-weight compensation, we calculate error approximately with

$$f(x) - f(x_i) \approx ef(x, x_i, c_{i1}, c_{i2}, ...),$$

where ef is the approximate expression for error and $c_{i1}, c_{i2}, ...$ are corresponding coefficients at x_i. Formula and coefficients of the function ef can depend on the original function $f(x)$, domain of interest I, the nearest pattern x_i and so on. For example, when we target function $\log x$ on $[1, 2]$, ef can be defined as $2(x - x_i)/(x + x_i)$ [1].

One straightforward realization of ef for continuous derivable function is using Taylor expansion. The Taylor series of a function $f(x)$ that is infinitely differentiable at value x_0 is the power series:

$$f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ...$$

The tangent line approximation of $f(x)$ for x near x_0 is called the first degree Taylor Polynomial of $f(x)$ and is

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0).$$

And the second degree Taylor Polynomial is

$$f(x) \approx f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2.$$

Therefore, when using first order approximation,

$$ef(x, x_i, c_{i1}) = c_{i1}(x - x_i), c_{i1} = f'(x_i).$$

For second order approximation,

$$ef(x, x_i, c_{i1}, c_{i2}) = c_{i1}(x - x_i) + c_{i2}(x - x_i)^2,$$
$$c_{i1} = f'(x_i), c_{i2} = \frac{1}{2}f''(x_i).$$

2) Data Point Selection: Memory space of the lookup table is limited and when there is more stored patterns, both search and read operations have more energy consumption and longer execution time. For example, in 45nm technology, 32-row associate memristive memory (AMM) consumes about twice energy compared to that of 8-row AMM for one search and read operation [5].

Tradeoff between Error and Size of Table. Undoubtedly larger lookup table with more information has higher precision and there is tradeoff between error and size of table. According to which one is the main restrictive factor, either the number of data points to be selected is specified, or a maximum error is defined implying least number of data points required for approximation. [11] proposed methods for selecting a specified number of data points and selecting data points based on an error tolerance, which we apply to ApproxLUT.

Error Criteria for Lookup Table. Suppose $f(x)$ is the accurate output with input x, $f_o(x)$ is the output value of ApproxLUT, i.e.

$$f_o(x) = f(x_i) + ef(x, x_i, c_i),$$

where x_i is the nearest stored pattern, and ef and c_i are formula and coefficients for compensation. There are many kinds of error criteria we can use for the lookup table based on requirements of the application, and two of them are as follows.

- **Worst Case Error.** Worst case error is the largest error among all points, i.e.
$$\text{error} = \max_{x \in I} |f_o(x) - f(x)|.$$
- **Error Expectation.** Error expectation is the expectation of error for all points, i.e.
$$\text{error} = \sum_{x \in I} |f_o(x) - f(x)| p(x),$$
where $p(x)$ is the probability distribution of input data x.

In fact, error expectation is the mean error when we assume equal probability of all possible operands. With prestored data points, we calculate errors for all possible data points and then define precision with the error criteria for online error bounding.

3) Level by Level Table Construction: After data point selection, as shown in Figure 2, for every chosen pattern x_i, the table stores input operand x_i, output $y_i = f(x_i)$, and compensation coefficients c_i. The first-level table consists of N patterns within the interval between x_1 and x_N in ascending order, i.e. $x_1 < x_2 < ... < x_N$. The second-level table is constructed based on first-level table. When there is N patterns in first level, the second level has $N - 1$ subtables. Construction process is the same and interval of these subtables are respectively x_1 to x_2, x_2 to x_3, ..., x_{N-1} to x_N. Therefore the second level is composed of a set of subtables with finer granularity compared to first level.

Suppose $f_o(x)$ is the output value with input x. There are four approximation modes using this two-level table.

Mode 1 (2nd w/ c): Second Level with Compensation.

$$f_o(x) = f(x_{ij}) + ef(x, x_{ij}, c_{ij}),$$

where x_{ij} is the jth pattern in the ith second-level subtable and is the nearest pattern from x in the whole lookup table,

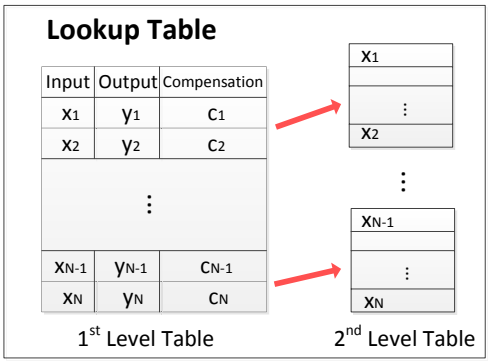

Fig. 2. Two-Level Lookup Table of the Accelerator

and c_{ij} is the compensation coefficient of x_{ij}.

Mode 2 (1st w/ c): First Level with Compensation.

$$f_o(x) = f(x_i) + ef(x, x_i, c_i),$$

where x_i is the ith and the nearest pattern from x in the first-level table, and c_i is the compensation coefficient of x_i.

Mode 3 (2nd w/o c): Second Level without Compensation.

$$f_o(x) = f(x_{ij}).$$

Mode 4 (1st w/o c): First Level without Compensation.

$$f_o(x) = f(x_i).$$

After construction of the whole table, precisions of ApproxLUT can be calculated for all approximation modes.

C. Online Approximate Computing with ApproxLUT

1) Approximation Mode Selection: It is unnecessary to perform fully accurate computation for error-resilient algorithm. On one hand, lower precision does not always lead to functional error. On the other hand, as many applications are inherently error-resilient, certain amount of functional errors are acceptable, which can sometimes be recovered and have no influence on the final output quality. Such error resilience capability is always varying online and decides precision requirement for current computation. Given online error bound, ApproxLUT compares it with errors characterized offline and selects appropriate approximation mode.

2) Computation with Lookup Table: With input x and error bound, the accelerator outputs acceptable result y following the steps shown in Figure 3. Errors of four approximation modes are denoted by e_1, e_2, e_3 and e_4 and we assume that $e_1 < e_2 < e_3 < e_4$, which means e_1 corresponds to the most accurate mode. If highest accuracy level cannot satisfy the bound, i.e. Error Bound $< e_1$, the accelerator calculates the function directly without the table. If Error Bound $\geq e_1$, control unit chooses appropriate approximation mode according to error bound. Then control unit searches only the first level or both levels of the lookup table and obtains the nearest pattern x', corresponding output y', compensation parameters denoted by CP and a hit flag. The hit flag is used to show whether the exact input pattern is stored in the table, i.e whether $x = x'$ is true. If this flag is true or if y' without compensation satisfies the precision requirement, y' is output directly. Otherwise the accelerator adds compensation with CPs to y' and outputs the final result $y = y' + ef(x, x', \text{CPs})$.

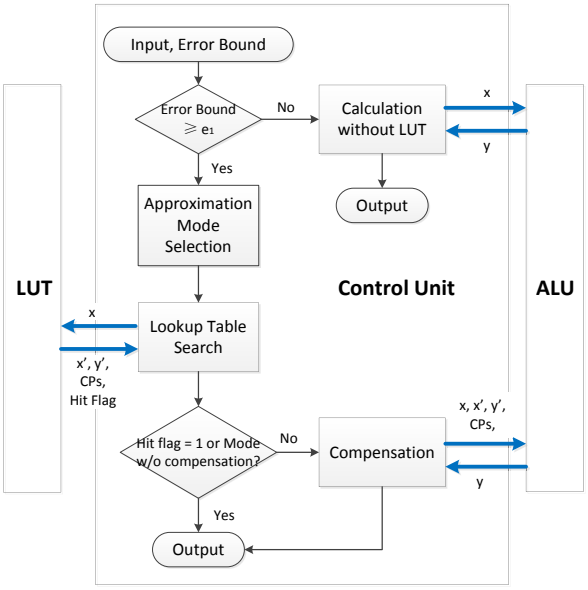

Fig. 3. Calculation Flow of the LUT Based Accelerator

3) Table Priority Adjustment: In many applications, input operands of a function may show the phenomena that if a particular input appears at a particular time, then it is likely that the same input or its nearby inputs will appear in the near future, and we call this phenomena "input locality".

For example, in the discrete cosine transform used for image impression, the n real numbers $x_0, ..., x_{n-1}$ are transformed into the n real numbers $X_0, ..., X_{n-1}$. DCT-II is the most commonly used form using the formula:

$$X_k = \sum_{i=0}^{n-1} x_i \cos[\frac{\pi}{n}(i + \frac{1}{2}k)], \text{ where } k = 0, ..., n-1.$$

Function $\cos x$ is frequently used and two successive calculations $\cos[\frac{\pi}{N}(i+\frac{1}{2}k)]$ and $\cos[\frac{\pi}{N}(i+1+\frac{1}{2}k)]$ have very similar input operands as n and k are fixed.

In this case, our accelerator for cos function has a high probability to find the nearest patterns of these two successive inputs in the same subtable. To utilize input locality, when the accelerator keeps reading from the same subtable for several successive computations, this subtable is put in a higher priority for later table search.

IV. Experimental Results

A. Experimental Setup

In the experiment, to prove the efficacy of compensation and multi-level table, we first compare energy consumption and precision between multi-level table with compensation parameters and traditional lookup table. We then apply ApproxLUT to various functions and evaluate its effectiveness.

For lookup table construction, we choose first degree Taylor polynomial for light weight compensation, i.e. $ef(x, x_i, c_i) = c_i(x - x_i)$, where the compensation parameter $c_i = f'(x_i)$.

Based on such compensation mechanism, methodology proposed in [11] is employed for data point selection. We then calculate errors (error expectation) and energy savings of four approximation modes of ApproxLUT compared to fully accurate computation. Errors of four approximation modes are denoted by e_1, e_2, e_3 and e_4 and we assume that $e_1 < e_2 < e_3 < e_4$, which means precision of mode 1 to mode 4 is decreasing and mode 1 corresponds to the most accurate approximation mode.

We use associate memristive memory with 32-bit word size for both input and output operands to store the lookup table. Control unit of the accelerator is implemented at software level. As the overhead of approximation mode selection is ignorable compared with table operation (search and read) and calculation for compensation, we do not take it into account for energy consumption of ApproxLUT.

To estimate power of associative memristive memory, we use TCAM power model proposed in [12] and obtain this model from the website [13]. For energy consumption per operation of basic operations ($+, \times$ for compensation) and target functions, we simulate benchmark applications on x86_64 CPU with gem5 simulator. The simulated statistics (e.g., how many instructions executed) and micro-architecture configurations are transformed to the power simulator McPAT, where the energy consumption of the system is estimated.

B. Comparison with Traditional LUT

1) Single-Level Table with Compensation vs. Single-Level Table without Compensation: For functions listed in Table I, we calculate the error of a 32-row table with compensation (32 patterns and their compensation parameters are stored). Then we find the number of rows for table without compensation which has similar precision. Number of rows in LUT (i.e. number of patterns required) and corresponding energy consumption are compared between these two LUTs having the same quality loss.

As shown in Fig. 4, table without compensation needs much more patterns, e.g., for $\cos$ function, traditional table needs 3200 rows to obtain the precision of 0.0004 while table with compensation only needs 32 rows. Energy consumption shown in the figure is the relative consumption compared to exact function computation. When we use traditional table for $\cos$, energy for table search and read is 2.5 times, while table with compensation (including calculation for compensation) achieves energy saving of about 50%. For the error function erf, compensation with gradient is not that effective, and the size of traditional table is only 420, therefore the energy consumption of traditional table is a little bit less than that of table with compensation.

2) Single-Level Table vs. Multi-Level Table: To compare single-level table and multi-level table (both are without compensation), we apply these two LUTs having the same size (256 rows in total) for function $\cos x$ to discrete cosine transform (DCT) application. To show the quality degradation, we then use accurate inverse discrete cosine transform (iDCT) and calculate peak signal-to-noise ratio (PSNR) of approximate

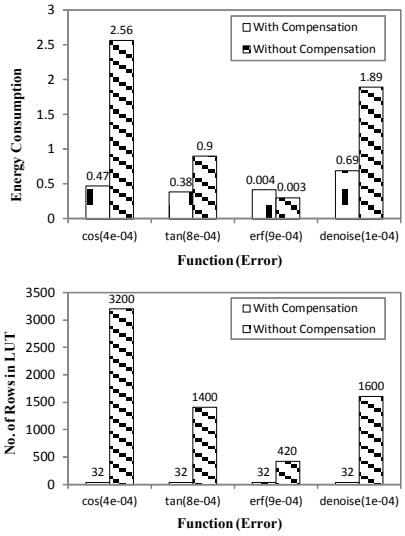

Fig. 4. Comparison of Number of Rows and Energy Consumption between Lookup Tables with and without Compensation

and accurate results. As shown in Fig. 5, multi-level table has multi-level precision (29dB and 18dB). The most accurate mode can achieve similar quality with single-level table (29dB vs. 31dB), while obtaining much more energy saving (95.7% vs. 73.9%, denoted by E_s in the figure) for calculating $\cos$.

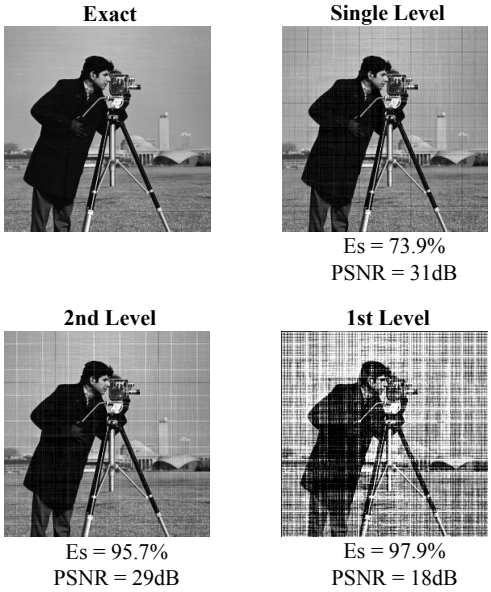

Fig. 5. Comparison of Energy Saving and Quality Loss between Single-Level and Multiple-Level Lookup Tables for DCT

C. ApproxLUT for Various Functions

We apply ApproxLUT to functions listed in Table I. The lookup table has 256 rows in total, wherein the first level has 16 rows and the second level has 240 rows (15 subtables, each of which has 16 rows). We estimate energy saving (denoted by E_s) compared to original computation of the function and error

TABLE II
ENERGY SAVING AND ERROR EXPECTATION OF APPROXLUT FOR VARIOUS FUNCTIONS

Approximation Mode		$\cos x$ $[0, \frac{\pi}{2}], [0, 1]$	$\tan x$ $[0, \frac{2\pi}{5}], [0, 3.08]$	e^x $[0, 3], [0, 20.09]$	$\ln x$ $[1, 10], [0, 1.61]$	$\text{erf}(x)$ $[0, 3], [0, 1]$	Denoise $[0, 3], [0, 0.81]$
1 (2nd, w/ c)	error	1.03e-05	1.65e-05	0.0008	7.13e-06	4.46e-05	2.51e-06
	E_s	49.85%	59.55%	-2.76%	-9.98%	99.56%	26.18%
2 (1st, w/ c)	error	0.0015	0.0033	0.1534	0.0016	0.0027	0.0006
	E_s	55.41%	64.03%	8.64%	2.21%	99.61%	34.37%
3 (2nd, w/o c)	error	0.0043	0.0045	0.0556	0.0022	0.0016	0.0010
	E_s	88.87%	91.03%	77.20%	75.60%	99.90%	83.62%
4 (1st, w/o c)	error	0.0414	0.0559	0.7912	0.0333	0.0177	0.0143
	E_s	94.44%	95.51%	88.60%	87.80%	99.95%	91.81%

expectation (denoted by Error) for four approximation modes, and the results are shown in Table II. The first row of the table lists all functions for evaluation with their definition domains and value domains (e.g. for $\cos x$, the definition domain is $[0, \frac{\pi}{2}]$ and the corresponding value domain is $[0, 1]$). Precision is decreasing from mode 1 to mode 4. Mode 1 searches second-level table and adds compensation (denoted by 2nd, w/ c in the table), mode 2 is first-level table with compensation, mode 3 is second-level table without compensation and mode 4 is first-level table without compensation.

Energy savings of mode 1 and 2 (e.g. 59.6% and 64.0% for $\tan x$), and mode 3 and 4 (91.0% and 95.5% for $\tan x$) are close, while mode 3 and 4 have much more energy savings than mode 1 and 2. This is because mode 1 and 2 conduct $+$ and $\times$ operations for compensation, which are more energy-consuming than table search. For function e^x and $\ln x$, energy savings are negative for mode 1, which means mode 1 consumes more energy than fully accurate mode. Energy consumptions of these two functions are inherently small and are comparable to that of $+$ and $\times$. Therefore approximation modes with compensation are not suitable for them. For $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ with a much more complex computation, ApproxLUT has the best energy savings above 99.5% while error of mode 1 is only 4e-05, which means energy consumption for original computation is about 200 times of that for ApproxLUT.

V. CONCLUSION

This paper proposes an adaptive lookup table based accelerator named ApproxLUT, which can be generally applied to computation-intensive functions in various platforms. ApproxLUT stores selected function responses in a hierarchical lookup table with light-weight compensation capability, which is able to output results at different precision levels according to online application quality requirements. At runtime, ApproxLUT adjusts table search strategy by exploring input locality to achieve high energy-efficiency. Experimental results show that the energy savings of the proposed solution outperforms state-of-the-art techniques considerably.

VI. ACKNOWLEDGEMENT

This work is supported in part by Project #61432017 and Project #61532017 by National Natural Science Foundation of China (NSFC), and in part by Huawei Technologies Co. Ltd.

REFERENCES

[1] Ping Tak Peter Tang. Table-lookup algorithms for elementary functions and their error analysis. In *IEEE Symposium on Computer Arithmetic*, pages 232–236, 1991.

[2] Jason Cong, Milos Ercegovac, Muhuan Huang, Sen Li, and Bingjun Xiao. Energy-efficient computing using adaptive table lookup based on nonvolatile memories. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 280–285. IEEE, 2013.

[3] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM, 2013.

[4] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2013.

[5] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K Gupta. Approximate associative memristive memory for energy-efficient gpus. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1497–1502. IEEE, 2015.

[6] Mohsen Imani, Abbas Rahimi, and Tajana S Rosing. Resistive configurable associative memory for approximate computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 1327–1332. IEEE, 2016.

[7] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. ACM, 2013.

[8] Teuvo Kohonen. *Associative memory: A system-theoretical approach*, volume 17. Springer Science & Business Media, 2012.

[9] Will Eatherton, George Varghese, and Zubin Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *ACM SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

[10] Mohsen Imani, Yan Cheng, and Tajana Rosing. Processing acceleration with resistive memory-based computation. In *Proceedings of the Second International Symposium on Memory Systems*, pages 208–210. ACM, 2016.

[11] Bernd Hamann and Jiann-Liang Chen. Data point selection for piecewise linear curve approximation. *Computer Aided Geometric Design*, 11(3):289–301, 1994.

[12] Banit Agrawal and Timothy Sherwood. Modeling tcam power for next generation network devices. In *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*, pages 120–129. IEEE, 2006.

[13] Ternary cam (tcam) power and delay modeling. http://www.cs.ucsb.edu/~arch/mem-model/.