SECO: A Scalable Accuracy Approximate Exponential Function Via Cross-Layer Optimization

Di Wu, Tianen Chen, Chienfu Chen, Oghenefego Ahia, Joshua San Miguel, Mikko Lipasti, and Younghyun Kim
Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, Wisconsin 53706
{dwu94, tianen.chen, cchen452, ahia, jsanmiguel, younghyun.kim}@wisc.edu, mikko@engr.wisc.edu

***Abstract*—From signal processing to emerging deep neural networks, a range of applications exhibit intrinsic error resilience. For such applications, approximate computing opens up new possibilities for energy-efficient computing by producing slightly inaccurate results using greatly simplified hardware. Adopting this approach, a variety of basic arithmetic units, such as adders and multipliers, have been effectively redesigned to generate approximate results for many error-resilient applications.**

In this work, we propose *SECO*, an approximate exponential function unit (EFU). Exponentiation is a key operation in many signal processing applications and more importantly in spiking neuron models, but its energy-efficient implementation has been inadequately explored. We also introduce a cross-layer design method for SECO to optimize the energy-accuracy trade-off. At the algorithm level, SECO offers runtime scaling between energy efficiency and accuracy based on approximate Taylor expansion, where the error is minimized by optimizing parameters using discrete gradient descent at design time. At the circuit level, our error analysis method efficiently explores the design space to select the energy-accuracy-optimal approximate multiplier at design time. In tandem, the cross-layer design and runtime optimization method are able to generate energy-efficient and accurate approximate EFU designs that are up to 99.7% accurate at a power consumption of 3.73 pJ per exponential operation. SECO is also evaluated on the adaptive exponential integrate-and-fire neuron model, yielding only 0.002% timing error and 0.067% value error compared to the precise neuron model.

***Index Terms*—Dynamic Accuracy Scaling, Cross-Layer Approximation, Error Modeling, Exponential Function**

I. Introduction

Energy efficiency is one of the most crucial design constraints of modern computing systems, which dictates performance, lifetime, form factor, and cost. *Approximate computing* is an emerging computing paradigm to improve energy efficiency by expanding the scope of the design space to where the accuracy constraints of computations can be relaxed. For emerging application domains that are intrinsically resilient to errors, approximate computing opens up new possibilities for more energy-efficient designs at the cost of minor or negligible accuracy loss.

The key to the optimal design of approximate computing systems is to maximally exploit the trade-off between energy efficiency and output quality to meet application-specific requirements. Researchers have sought to develop various energy-efficient approximate hardware blocks with different energy-accuracy trade-offs, such as approximate adders [1], multipliers [2], and dividers [3]. They have proven more accurate and/or more energy-efficient than simply truncating precise arithmetic units. Recently, with an emergence of more complex hardware accelerators, high-level approximate hardware to perform sophisticated non-linear arithmetic operations, such as exponentiation, is garnering more interest. Typically, exponentiation is not supported by a hardware arithmetic unit in microprocessors but implemented in a math software library using multiplication. However, the recent emergence of distributed signal processing systems and embedded neural networks has led to the demand for a fast yet energy-efficient exponential function unit (EFU).

Existing approaches to implementing hardware EFUs are typically based on the Taylor series, the definition of limit, Newton's method, base conversion, etc. [4], and the hardware is designed to produce the most precise results allowed by the numerical representation. However, for error-resilient applications, performing exponentiation with a precise EFU wastes time and energy in producing overexact results; thus an approximate EFU would be a more efficient solution to such applications. Adopting an approximate EFU involves two major design challenges: i) An application's accuracy requirement and energy efficiency requirement are input-dependent and time-varying [5]; in order to maximally exploit the energy-accuracy trade-offs, an approximate EFU should support *dynamic energy-accuracy scaling*. ii) The approximate EFU itself should be significantly more energy-efficient and accurate than simply truncating precise EFUs.

In this paper, we introduce an approximate EFU named *SECO (Scalable Accuracy Approximate Exponential Function via Cross-Layer Optimization)* to address the aforementioned challenges. The contributions are as follows:

- We propose *SECO*, an approximate EFU design based on the *incremental approximation* of exponentiation using the Taylor series. By adjusting the number of added Taylor terms, the energy efficiency and accuracy can be traded off against each other at runtime.
- We present a cross-layer optimization framework that takes both *algorithm-level* (approximate parameters for the Taylor series) and *circuit-level* (approximate multiplier selection) design factors into account in order to maximize both energy efficiency and accuracy for a given input distribution.
- We evaluate the energy efficiency and accuracy of SECO for various input distributions. We also present an application of SECO to the adaptive exponential (AdEx) integrate-and-fire neuron model.

978-1-7281-2954-9/19/$31.00 ©2019 IEEE

II. Background

A. Approximate Arithmetic

Various approximate arithmetic units have been introduced to improve the energy efficiency of arithmetic operations in error-resilient domains. Approximate addition [1], [6] and multiplication [2], [7] have received the most attention since they are heavily used in many applications. Approximate dividers have been introduced more recently for energy-efficient signal processing [8], [9]. These approximate arithmetic units can be categorized as circuit-level approximation techniques [10], [11] that use simplified hardware to generate similar logic outputs. A more recently proposed approximate divider is based on the Taylor series approximation for runtime energy-accuracy scaling [3], but it does not optimize its selection of the underlying multiplier and approximate algorithm.

B. Hardware EFUs

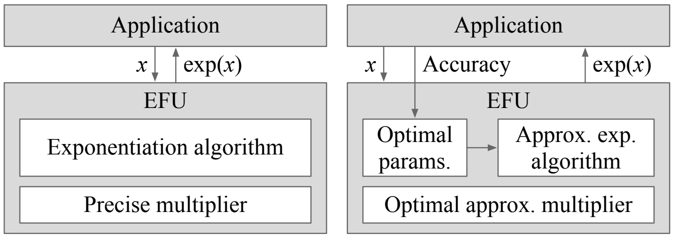
Natural exponentiation is computationally expensive. Researchers have proposed various hardware EFUs to accelerate exponentiation-heavy applications. Hardware-efficient floating point EFUs have been proposed for applications that demand high-precision computing [12], but they are far from energy-accuracy-optimal for error-resilient applications. Another design proposes to replace the multiplications of integer part and upper fractional part with two lookup tables (LUTs) to perform low-width multiplications for lower fractional part [13]. It is designed to produce accurate results with less than one LSB (least significant bit) absolute error for all input operands and thus requires large area for multiple multipliers, registers, and LUTs. For more error-forgiving and hardware-constrained systems, approximate EFUs have been proposed. Taylor series-based approximation of exponentiation is a widely used method, which is adopted in [14]. However, this EFU is designed to accumulate a fixed number of Taylor terms determined at design time, and the energy-accuracy trade-offs cannot be adjusted at runtime. To the best of our knowledge, there has been no prior work on approximate hardware EFUs with runtime energy-accuracy reconfiguration.

III. Design of SECO

The insight behind SECO is that *dynamic energy-accuracy scaling of the exponential function* can be achieved efficiently using *incremental Taylor series approximation.* The proposed approximate EFU is based on the Taylor series centered at $x = 0$ for the exponential function $\exp(x)$, written as

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \tag{1}$$

where the operand $x \in [0, 1)$. Since it is not practical nor feasible to sum an infinite number of Taylor terms, a good approximation is to sum the first $N + 1$ terms, where N is the maximum number of non-constant Taylor terms. For typical applications, N does not need to be a large number. For example, $N = 6$ yields less than 1 LSB error for 15-bit exponentiation [14].



(a) Conventional EFU (b) Proposed EFU

Fig. 1: Proposed approximate EFU in comparison to the conventional EFU.

Unlike prior approximate exponentiation schemes [13], [14] where N is fixed at design time, we propose a flexible hardware architecture that can adjust N at runtime. This enables dynamic energy-accuracy scaling, which is the key to energy-accuracy optimality. We achieve this via *incremental approximation* where the Taylor terms in (1) are added incrementally at each cycle, from low order to high order. When high accuracy is demanded at the cost of high energy consumption and longer latency, more terms are added over more cycles; when low accuracy can be tolerated for reducing energy consumption and latency, the accumulation is terminated sooner. The number of cycles can be dynamically reconfigured by the application to optimize energy efficiency and accuracy based on its requirements. Fig. 1 shows a comparison between conventional and the proposed EFU design. Note that the optimal runtime reconfiguration (i.e., finding the optimal N) is out of the scope of this paper, and we focus on how we efficiently facilitate the reconfigurability. We overcome two key challenges.

Challenge 1: Expensive arithmetic. The most power-hungry and area-demanding operations in the finite Taylor series are the numerous multiplication and division operations. We propose a power- and area-efficient approximation of (1) via *joint optimization of multiplication skipping and approximate division*. Specifically, each Taylor term is approximated by

$$s_n \cdot \frac{x^n}{n!} \approx s_n \cdot \frac{x^{p_n}}{2^{q_n}}, \tag{2}$$

where, for $n = 0, 1, \ldots, N$,

$$p_n = \begin{cases} p_{n-1} + 1 & \text{if multiplication is not skipped,} \\ p_{n-1} & \text{if multiplication is skipped,} \end{cases} \tag{3}$$

and q_n denotes the offset of right shifting, with $s_n \in \{-1, 1\}$ denoting the sign of Taylor terms. Multiplying x is performed selectively, rather than at every cycle, to reduce dynamic energy consumption, followed by the approximate division via right shifting. The sign s_n of each term is determined based on the sign of the accumulated error up to $(n-1)^{th}$ Taylor term. As presented in Section VI-A, the proposed approximation can skip up to 3 multiplications out of 7, saving dynamic energy significantly. The joint optimization of the three sequences $\{p_n\}$, $\{q_n\}$ and $\{s_n\}$ will be discussed in Section IV.

Challenge 2: Non-uniform error. The approximation error of (1) is not uniform over the input range $[0, 1)$, since the approximation is centered at $x = 0$. The approximation is more accurate close to $x = 0$, but less accurate close to

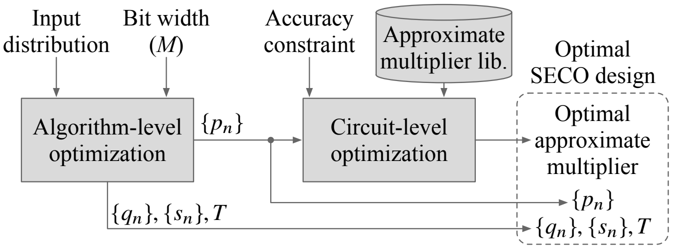


Fig. 2: Cross-layer optimization flow.

$x = 1$. Therefore, to minimize the overall error, *double-sided expansion* is adopted. Given a threshold T, at runtime, if $0 \leq x < T$, the Taylor series is expanded at $x = 0$; otherwise at $x = 1$. The optimal threshold T that minimizes error is determined at design time according to the input distribution and is one of the four target variables for the optimization algorithm, discussed in Section IV.

IV. Cross-Layer Optimization of SECO

In this section, we describe our optimization of the energy-accuracy trade-off at design-time via a cross-layer optimization flow. The optimization flow consists of two sequential optimization phases as shown in Fig. 2, namely the *algorithm-level optimization* and the *circuit-level optimization*. The algorithm-level optimization reduces the area and power at a high level while maintaining accuracy by deriving the best schemes for multiplication skipping, approximate division and double-sided expansion described in Section III for given application-specific design parameters, i.e., the input distribution and bit width. The circuit-level optimization targets further power reduction by selecting the most energy-efficient approximate multiplier from a given library,[1] under relaxed accuracy constraints specified by the designer.

Phase 1: Algorithm-level optimization. The algorithm-level optimization phase takes two factors into account, i.e., the input distribution and bit width. Prior approximate arithmetic hardware designs assume uniformly distributed input data, which often contrasts to practical scenarios. To design optimal hardware, application-specific (potentially non-uniform) input distribution should be considered. Furthermore, the bit width contributes significantly to final accuracy due to the truncation, and must be considered when designing approximate functions for varying accuracy demand.

For an input x, the approximate exponentiation result $\widetilde{\exp}(x)$ and the relative error $\epsilon(x)$ are defined as follows:

$$\begin{aligned} \widetilde{\exp}(x) &= \sum_{n=0}^{N} s_n \cdot \frac{x^{p_n}}{2^{q_n}}, \\ \epsilon(x) &= \frac{\widetilde{\exp}(x)}{\exp(x)} - 1. \end{aligned} \tag{4}$$

To capture the actual cumulative approximation error for a specific input distribution, we consider both the frequency and significance of errors and use weighted mean relative error (WMRE) as the metric. For a given input distribution and input bit width of M, WMRE for total 2^M discrete inputs is defined as

Algorithm 1 Algorithm-level optimization

```
 1: procedure MINIMIZE $WMRE$
 2:     for each triplet of $\{N_p, N_q, T\}$ do
 3:         Initialize $\{p_n\}$, $\{q_n\}$, $\{s_n\}$
 4:         while $n \leq N$ do
 5:             for $p_{n+1}$, $q_{n+1}$, $s_{n+1}$ do
 6:                 Calculate $WMRE_{n+1}$
 7:                 if $WMRE_{n+1}$ is minimized then
 8:                     Store $p_{n+1}$, $q_{n+1}$, $s_{n+1}$
 9:                 end if
10:             end for
11:             $\{p_{n+1}\} \leftarrow \{\{p_n\}, p_{n+1}\}$
12:             $\{q_{n+1}\} \leftarrow \{\{q_n\}, q_{n+1}\}$
13:             $\{s_{n+1}\} \leftarrow \{\{s_n\}, s_{n+1}\}$
14:             $n \leftarrow n + 1$
15:         end while
16:         if $WMRE$ is minimized then
17:             Store $\{p_n\}$, $\{q_n\}$, $\{s_n\}$, $T$
18:         end if
19:     end for
20:     return $\{p_n\}$, $\{q_n\}$, $\{s_n\}$, $T$
21: end procedure
```

$$WMRE = \sum_{m=0}^{2^M - 1} P_m \cdot |\epsilon(x_m)|, \tag{5}$$

where $x_m = m/2^M$, and P_m denotes the probability of $x = x_m$. Based on this, accuracy is defined as $(1 - \text{WRME}) \times 100$ (%).

The algorithm-level optimization is to optimize the four design parameters defined in Section III that minimize WMRE. A simplified discrete gradient descent algorithm is devised to solve the problem, as described in Algorithm 1. More specifically, the algorithm finds the order in Taylor term to trigger multiplication skipping and approximate division, defined as N_p and N_q respectively, the approximation direction indicator of error compensation $\{s_n\}$, and the double-sided expansion threshold T. We initialize the algorithm with multiple starting points to increase the possibility of convergence at a better local minimum (Line 3). Starting from each initial point, we find the adjacent point with minimum WMRE, and select its p_{n+1}, q_{n+1} and s_{n+1} as the best parameter in multiplication and right shifting for the $(n+1)^{th}$ Taylor term (Line 8). Each surrounding point has its p_{n+1} and q_{n+1} to be equal to or greater by 1 than p_n and q_n, according to the updating rule. Also, we assume that each surrounding candidate point always has a unit distance from the n^{th} point due to the discreteness of the design space, as such the point with minimum WMRE is the direction with the largest gradient, and merged to current $\{p_n\}$, $\{q_n\}$ and $\{s_n\}$ for updating the subsequent points (Lines 11–13). This process lasts until we has chosen all p_n, q_n and s_n for each updating rule. Finally, we choose the best p_n, q_n, s_n and T as the selected parameters among all updating rules (Line 16).

[1] We use the EvoApproxLib approximate multiplier library [15].

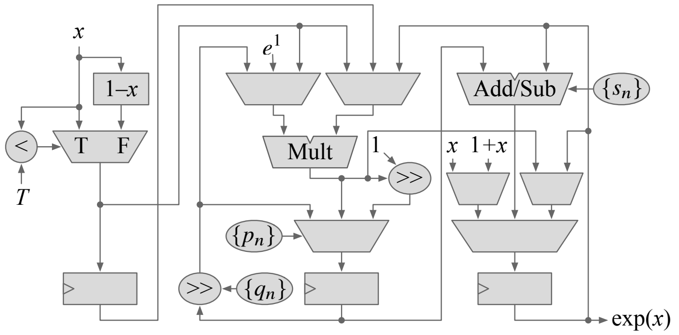


Fig. 3: SECO hardware architecture

Phase 2: Circuit-level optimization. The objective of the circuit-level optimization phase that follows the algorithm-level optimization is to find the optimal approximate multiplier to ensure the best power performance under a given WMRE constraint. We first build an analytic error model that can predict the output WMRE for a given approximate multiplier in the library, which is defined as

$$\overline{WMRE} = \sum_{m=0}^{2^M-1} P_m \cdot \left(\frac{\overline{\exp}(x_m)}{\exp(x_m)} - 1 \right), \tag{6}$$

where $\overline{\exp}(x_m)$ is the predicted output with the approximate multiplier, given by

$$\overline{\exp}(x_m) = \sum_{n=0}^{N} \left(Tr\left(s_n \cdot \frac{{x_m}^{p_n}}{2^{q_n}}, M \right) \cdot (1 + MRE)^{p_n - 1} \right), \tag{7}$$

where $Tr(*, M)$ is M-bit truncation, and MRE is the mean relative error of the approximate multiplier. Two sources of error are considered in the model: the quantization error due to fixed point representation, and the multiplication error due to the approximate multiplier. Note that one of the output of the algorithm-level optimization p_n is taken into account in this model.

Using the error model that can predict the average exponentiation error from average multiplication error, we can eliminate efforts for massive simulation and simply select the energy-accuracy-optimal multiplier from the library. Specifically, the most energy-efficient approximate multiplier that meets the given accuracy requirement is selected. However, the error model can be also used for finding most accurate multiplier that meets a power budget or one that minimizes power-error product as a single objective function.

V. Hardware Architecture

Fig. 3 shows the simplified hardware architecture of SECO. The implementation is composed of three parts: i) double-sided expansion, ii) Taylor term approximation, and iii) accumulation. The double-sided expansion part takes an input and decide whether to Taylor-expand it around 0 or 1. To reduce area, a simplified comparator is used to compare the input against the threshold T by evaluating only the upper few bits of the input. The Taylor term approximation part is composed of two components: an approximate multiplier with a bypass path and a shifter. The approximate multiplier is the outcome of the circuit-level optimization discussed in Section IV. At each cycle, multiplication is bypassed if $p_n = p_{n-1}$ and dynamic energy consumption is reduced. Similarly, the output of the multiplier is right-shifted by $q_n - q_{n-1}$ at each cycle. Finally, the accumulation part accumulates the output of the multiplier to generate the exponentiation result.

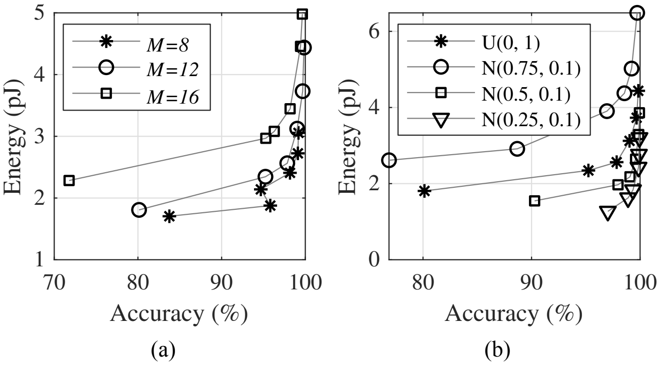


Fig. 4: Design-time optimization of energy-accuracy trade-off (a) for varying M and (b) for varying input distribution.

Note that multi-cycle operation that facilitates energy-accuracy scalability also enables a low hardware cost implementation. Unlike prior hardware EFUs composed of multiple multipliers and adders, SECO consists of one multiplier, one adder, and several peripheral control logic blocks. The resultant low hardware cost makes it suitable for extremely resource-constrained systems.

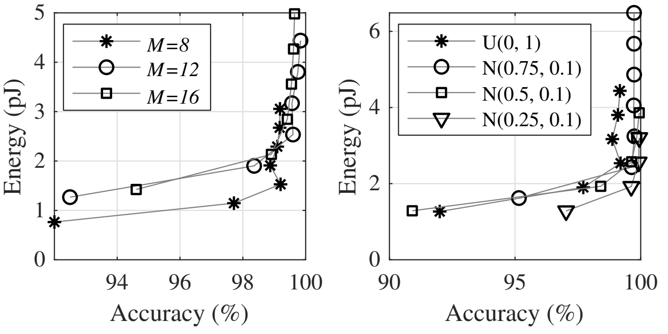
VI. Evaluation

In this section, we present the evaluation results for SECO. We first show the design-time optimization and runtime scaling of energy efficiency and accuracy. Next, SECO is applied to the AdEx neuron as a case study. Finally, we depict the detailed implementation results for the proposed EFU.

A. Energy-Accuracy Optimal Design

We first demonstrate the design-time energy-accuracy trade-off of SECO acheived by the proposed cross-layer optimization. We validate that the optimization algorithm derives energy-accuracy-optimal designs for varying input distribution, bit width, and accuracy constraint. The EvoApproxLib [15] is used as the approximate multiplier library.

Fig. 4(a) shows the energy consumption per operation and accuracy for three different bit widths: M = 8, 12 and 16. The accuracy loss is constrained to 0.5%, 1%, 2%, 4%, 8% and 16%, and uniform input distribution U(0,1) is assumed for all cases. For each case, as described in Section IV, an optimal approximate multiplier and optimal $\{p_n\}$, $\{q_n\}$, $\{s_n\}$ and T are selected by the cross-layer optimization. Note that each mark corresponds to the optimal design for each combination of input distribution and bit width, and the marks on the same curve may be different since they are optimized for different accuracy constraint. Note that the outlier with accuracy drop for M = 8 attributes to the discreteness of



(a) (b)

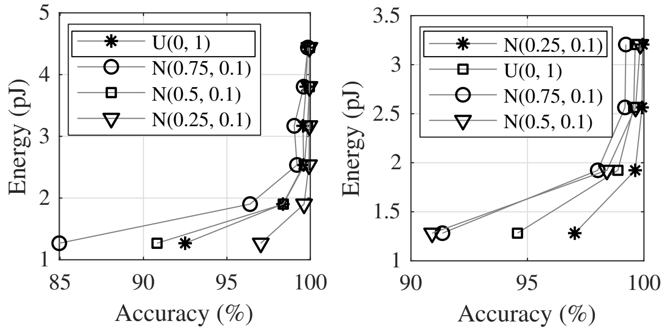
Fig. 5: Dynamic energy-accuracy scaling (a) for varying *M* and (b) for varying input distribution.

TABLE I: Design optimization result for $M = 12$.

Input dist.	Optimal parameters			
U(0, 1)	$\{p_n\}$	0, 1, 2, 3, 4, 5, 5		
	$\{s_n \cdot q_n\}$	0, 0, 1, 2, -3, 4, 5		
	T	0.875	Multiplier	mul12u_2QN
N(0.75, 0.1)	$\{p_n\}$	0, 1, 2, 3, 4, 4, 4, 4		
	$\{s_n \cdot q_n\}$	0, 0, 1, 3, -4, 5, -6, 7		
	T	0.375	Multiplier	mul12u_2PM
N(0.5, 0.1)	$\{p_n\}$	0, 1, 2, 3, 4, 5		
	$\{s_n \cdot q_n\}$	0, 0, 1, 2, -3, 4		
	T	0.75	Multiplier	mul12u_2DH
N(0.25, 0.1)	$\{p_n\}$	0, 1, 2, 3, 4		
	$\{s_n \cdot q_n\}$	0, 0, 1, 2, -3		
	T	0.75	Multiplier	mul12u_2DH

the library. Fig. 4(b) shows the energy-accuracy trade-off for four different input distributions: uniform distribution and three Gaussian distributions. The three Gaussian distributions have the same standard deviation $\sigma = 0.1$, but different means, $\mu \in \{0.75, 0.5, 0.25\}$. The same accuracy constraints are used, and the bit width M is fixed to 12. As shown in the figures, the cross-layer optimization algorithm successfully derives optimal designs for various input distributions and bit widths.

Next, we demonstrate the dynamic energy-accuracy scaling of SECO at runtime. Fig. 5(a) shows the energy-accuracy trade-off of three designs, each of which is optimized for different bit widths. Fig. 5(b) shows the same, and each design is optimized for different input distributions. Note that, unlike Fig. 4, each curve corresponds to the optimal design for each combination of input distribution and bit width, and the marks on the same curve denote different energy-accuracy trade-offs that can be achieved by adjusting latency. Though our algorithm optimization guarantees maximum accuracy at the final cycle, in Fig. 5, some SECO instances may have maximum accuracy at a certain immediate cycle due to unpredictable approximate multipliers. Post simulations can further suggest whether to terminate computing earlier before the final cycle. Table I shows the optimal approximate multiplier and the values of $\{p_n\}$, $\{q_n\}$, $\{s_n\}$ and T chosen by the cross-layer optimization for the designs in Fig. 5(b). As we configure SECO to maximum 8 terms, SECO for N(0.75, 0.1) skips maximum 3 multiplications out of 7. As shown in the figures, SECO provides a wide spectrum of runtime energy-accuracy trade-off that can exploited by applications.



(a) (b)

Fig. 6: Input variation affecting dynamic energy-accuracy scaling of SECO optimized (a) for uniform distribution U(0,1) and (b) for Gaussian distribution N(0.25,0.1).

Finally, Fig. 6 shows how input distribution influences dynamic energy-accuracy scaling. Fig. 6(a) is optimized for U(0,1), and Fig. 6(b) is optimized for N(0.25, 0.1). As shown in the figures, the input-awareness is crucial in achieving application-specific energy-accuracy optimally.

B. Case Study: Application to AdEx Neuron Model

We implement a key component in brain simulation, the AdEx neuron model [16], with SECO, and evaluate its accuracy scaling behavior. The AdEx neuron gradually increases the membrane potential and fires a spike after membrane potential crosses a threshold, followed by a reset. Its membrane potential $V(t)$ is modeled by following differential equations upon an injected current $I(t)$:

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L \cdot \Delta_T \cdot \exp(\frac{V - V_T}{\Delta_T}) + I - w, \quad (8)$$

$$\tau_w \frac{dw}{dt} = a(V - E_L) - w, \quad (9)$$

where w is the adaptation current, C is total capacitance, g_L is total leak conductance, E_L is effective rest potential, Δ_T is threshold slope factor, V_T is effective threshold potential, a is conductance, τ_w is time constant. Both of the equations rely on following reset equation:

$$\text{if } V > 0 \text{ then} \begin{cases} V \rightarrow V_r, \\ w \rightarrow w_r = w + b, \end{cases} \quad (10)$$

where V_r is reset potential, and b is spike triggered adaptation.

For evaluation, we focus on the firing responses with respect to membrane potential $V(t)$. As the baseline, four AdEx neuron firing patterns in [16] are implemented using full-precision floating point exponentiation. By varying the parameters in (8), (9) and (10), we achieve four distinguishable neuron responses with unique spiking behaviors in Fig. 7. We use two error accuracy metrics from [17] to compare against the baseline. The first metric depicts the timing error of our model:

$$ERRt = \left|\frac{\Delta t_p - \Delta t_o}{\Delta t_o}\right| \times 100, \quad (11)$$

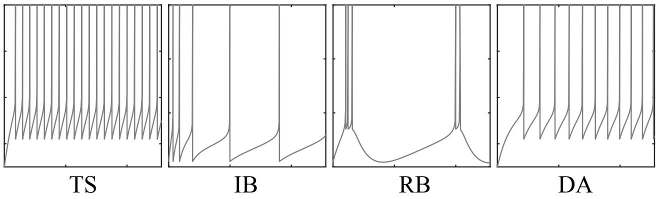
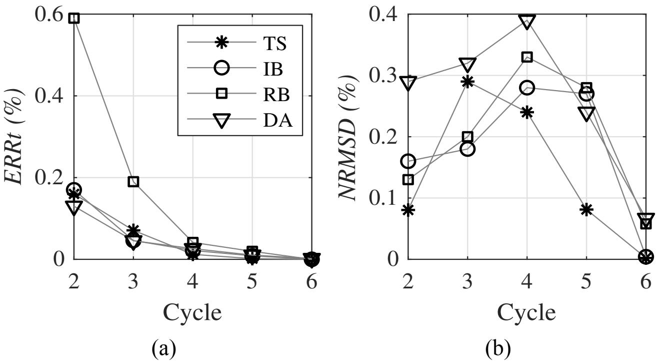


Fig. 7: Four neuron responses used in experiment. TS: Tonic spiking, IB: Initial burst, RB: Regular bursting, DA: Delayed accelerating [16].



(a) (b)

Fig. 8: (a) ERRt and (b) NRMSD for four neuron responses for varying cycle for SECO.

where Δt_p and Δt_o are the time intervals between two adjacent spikes in the proposed and original neurons, respectively. Secondly, the value error, normalized root mean square deviation (NRMSD), is calculated as

$$NRMSD = \sqrt{\frac{\sum_{i=1}^{n}(vp(i) - vo(i))^2}{n}} \cdot \frac{1}{v_{max} - v_{min}}, \qquad (12)$$

where v_{max} and v_{min} are the maximum and minimum spike responses of the original neuron, vp and vo are the of proposed and original responses, respectively, synchronized at each spike, and n is the sample count for the same spike.

The two errors of our neuron are shown in Fig. 8. The timing error ERRt monotonically decreases as we increase the latency of SECO, while the potential value error NRMSD finally drop after an initial increase, proving the accuracy scaling capability.

C. Hardware Evaluation

SECO is synthesized with Synopsys Design Compiler, and the implementation result is listed in Table II. Each SECO instance corresponds to the best SECO implementation for a given accuracy constraint. Relaxing the accuracy constraint decreases both area and power, in which the largest drop occurs from 99.7% to 99.1%, i.e., with negligible accuracy loss. Note that in some cases, as the computation can be terminated earlier with no accuracy loss, SECO may show even higher energy efficiency.

TABLE II: Implementation results for various accuracy constraints (TSMC 45 nm) and comparison to [14] (STM 65 nm).

Design	**Accuracy const.** (%)	**Latency** (ns)	**Area** (μm^2)	**Power** (mW)	**Energy** (pJ)
SECO	99.7	17.5	1,118	0.223	3.73
	99.1	20	611	0.136	2.72
	98.2	20	517	0.120	2.41
	95.2	20	378	0.094	1.88
	83.8	20	328	0.085	1.70
[14]	99.997	100	20,700	0.959	95.9

VII. Conclusion

In this paper, we present SECO, an approximate exponential function unit that supports dynamic energy-accuracy reconfiguration. We propose a low-cost approximation scheme for the Taylor expansion of exponentiation and an input-aware hardware optimization algorithm. The energy-accuracy reconfigurability of SECO is demonstrated for various input bit widths and input distributions. We also evaluate an application of SECO to the adaptive exponential integrate-and-fire neuron model. The cross-layer optimization framework in SECO is able to be further generalized to other non-linear designs.

Acknowledgements

This work was supported in part by the Wisconsin Alumni Research Foundation and NSF under awards CNS-1845469, CCF-1628384, and CCF-1813434, and AFRL award FA9550-18-1-0166.

References

[1] V. Gupta *et al.*, “IMPACT: Imprecise adders for low-power approximate computing,” in *ISLPED*, 2011.
[2] C. Liu *et al.*, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” in *DATE*, 2014.
[3] S. Behroozi *et al.*, “SAADI: A scalable accuracy approximate divider for dynamic energy-quality scaling,” in *ASP-DAC*, 2019.
[4] C. Chang *et al.*, “A division-free algorithm for fixed-point power exponential function in embedded system,” in *ICOT*, 2013.
[5] A. Raha *et al.*, “Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system,” in *DAC*, 2017.
[6] A. B. Kahng *et al.*, “Accuracy-configurable adder for approximate arithmetic designs,” in *DAC*, 2012.
[7] S. Hashemi *et al.*, “DRUM: A dynamic range unbiased multiplier for approximate applications,” in *ICCAD*, 2015.
[8] H. Jiang *et al.*, “Adaptive approximation in arithmetic circuits: A low-power unsigned divider design,” in *DATE*, March 2018.
[9] M. Vaeztourshizi *et al.*, “An energy-efficient, yet highly-accurate, approximate non-iterative divider,” in *ISLPED*, 2018.
[10] S. Venkataramani *et al.*, “Salsa: Systematic logic synthesis of approximate circuits,” in *DAC*, 2012.
[11] J. Schlachter *et al.*, “Design and applications of approximate circuits by gate-level pruning,” *IEEE TVLSI*, 2017.
[12] M. Langhammer *et al.*, “Single precision logarithm and exponential architectures for hard floating-point enabled FPGAs,” *IEEE Transactions on Computers*, 2017.
[13] J. Partzsch *et al.*, “A fixed point exponential function accelerator for a neuromorphic many-core system,” in *ISCAS*, 2017.
[14] P. Nilsson *et al.*, “Hardware implementation of the exponential function using taylor series,” in *NORCHIP*, 2014.
[15] V. Mrazek *et al.*, “EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *DATE*, 2017.
[16] R. Naud *et al.*, “Firing patterns in the adaptive exponential integrate-and-fire model,” *Biological Cybernetics*, 2008.
[17] M. Heidarpour *et al.*, “A CORDIC based digital hardware for adaptive exponential integrate and fire neuron,” *IEEE Trans. on Circuits and Systems I*, 2016.