

A Retrospective and Prospective View of Approximate Computing

By **WEIQIANG LIU**¹

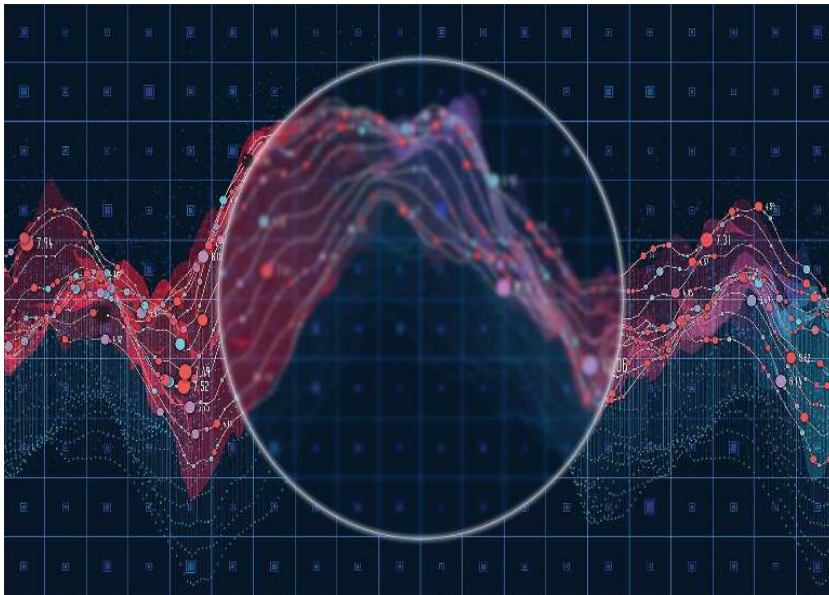
College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China

FABRIZIO LOMBARDI

Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA

MICHAEL SHULTE

Advanced Micro Devices Inc., Austin, TX 78735 USA



Computing systems are conventionally designed to operate as accurately as possible. However, this trend faces severe technology challenges, such as power consumption, circuit reliability, and high performance. For nearly half a century, performance and power consumption of computing systems have been consistently improved by relying mostly on technology scaling. As per Dennard's scaling, the size of a transistor has been considerably shrunk and the supply voltage has been reduced over the years, such that circuits operate at higher frequencies but nearly at the same power dissipation level. However, as Dennard's scaling tends toward an end, it is difficult to further improve performance under the same

power constraints. Power consumption has been a major concern, and it is now an industry-wide problem of critical importance. In addition to power, reliability deteriorates when the feature size of complementary metal-oxide-semiconductor (CMOS) technology is reduced below 7 nm, because parameter variations and faults at advanced nanoscales become difficult to control and prevent. Thus, to ensure the complete accuracy of signals, logic values, devices, and interconnects, manufacturing and verification costs will increase significantly.

There are a number of pervasive computing applications (such as machine learning, pattern recognition, digital signal processing, communication, data mining/analytics, robotics, and multimedia), which are inherently error-tolerant or error-resilient, i.e., in general, they require acceptable results rather than fully exact results. For example, in image processing, if the pixels are computed close to their exact values,

Digital Object Identifier 10.1109/JPROC.2020.2975695

0018-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See <https://www.ieee.org/publications/rights/index.html> for more information.

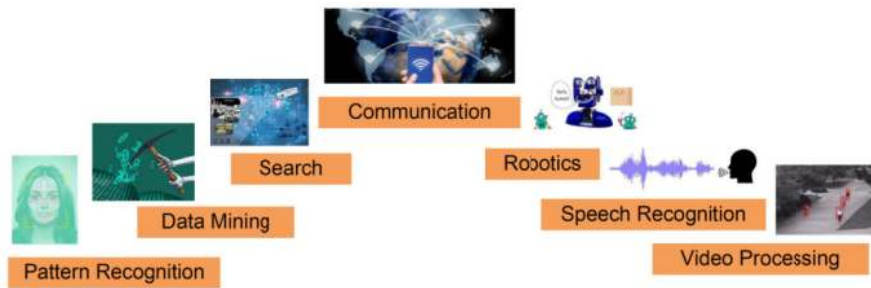


Fig. 1. Error-tolerant applications amenable to approximate computing.

may still correctly perceive an approximately processed image correctly. This is possible because usually there is either noise in data or redundancy in the algorithms, while humans have perceptual limitations.

Approximate computing has been proposed for highly energy-efficient systems targeting the abovementioned emerging error-tolerant applications (Fig. 1); approximate computing consists of approximately (inexactly) processing data to save power and achieve high performance, while results remain at an acceptable level for subsequent use. Approximation is not a new

concept in computing, and it has been extensively used in floating-point arithmetic units, or in the fixed-width fast Fourier transform (FFT). As data are usually of continuous nature, its conversion into quantized digital form inherently introduces an approximation; however, in the past, approximation has been normally considered for avoiding errors. Current research efforts allow the approximation error, and this can be considered as an additional design dimension that could be fully exploited to achieve significant performance improvements while reducing power consumption (Fig. 2).

Approximate computing is considered as one of few paradigms that could reduce power consumption by order of magnitude, and leading companies (such as IBM, Google, Intel, and ARM) are involved in experimental research and implementing commercial products and services based on approximate computing. For example, the tensor processing units (TPUs) by Google use reduced precision, a common approximate computing technique to reduce its power consumption [1]. IBM is applying approximate computing for on-chip artificial intelligence (AI) acceleration, while ARM is also

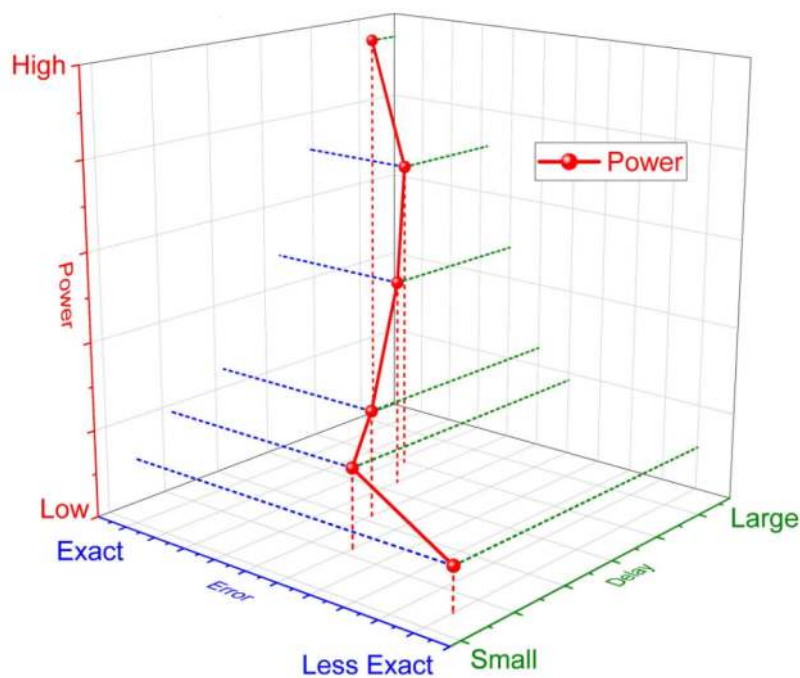


Fig. 2. Three-dimensional design space of approximate computing.

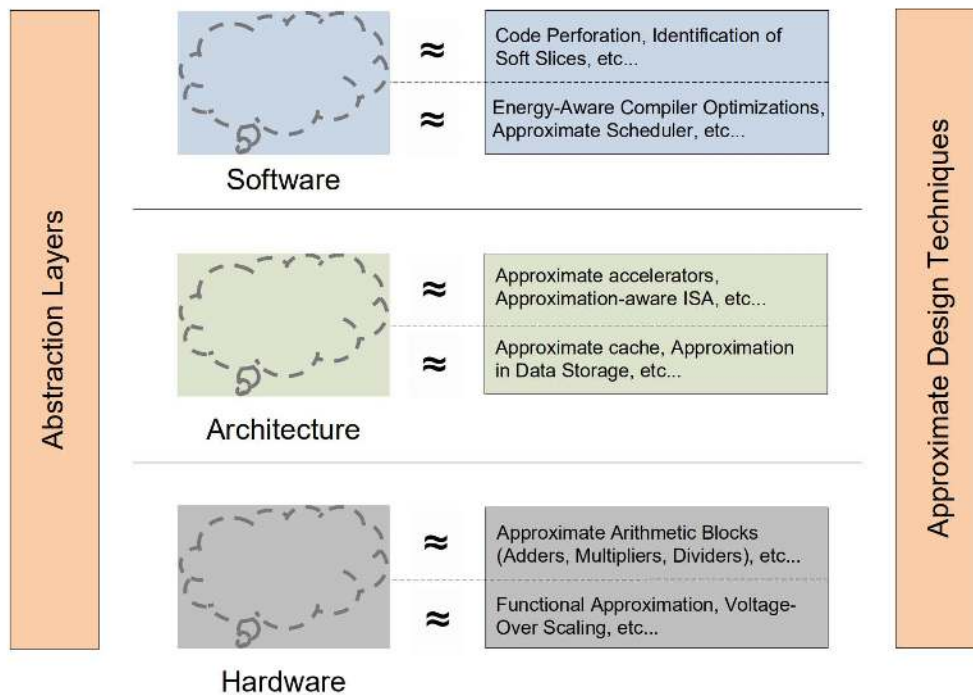


Fig. 3. Approximate techniques applied at different levels.

considering the design of a low-power approximate processor.

Approximate computing accomplishes a reduction in power consumption and higher performance by carefully controlling the introduced errors based on error analysis, to ensure that the measurable quality of error tolerance is still fully met in an application. Generally, the error analysis involves the least significant parts of the hardware or algorithm; then, errors are selectively introduced by these least significant parts of a design to achieve the desired benefits as well as different tradeoffs.

I. APPROXIMATE COMPUTING TECHNIQUES

Approximate computing can be applied at different levels, from devices to systems including hardware (devices, circuits, and architectures), software, algorithms, and programming languages. A classification of approximate computing techniques is summarized in Fig. 3. A few approximate computing tools have also been developed for automated design and evaluation; various

approximate computing techniques rely on general design principles such as significance guided design. Some can only be used at specific levels or applications.

A. Approximate Hardware

Due to their widespread utilization, approximate arithmetic circuits have been extensively studied based on the general principle of significance guided design (fewer resources are provided for insignificant parts with lower complexity). Under this principle, both logic reduction/pruning and voltage scaling for probabilistic CMOS have been applied. Probabilistic CMOS can reduce energy consumption by allocating higher supply voltages to critical circuits/chips to ensure the accuracy of the most significant bits, while carefully reducing the supply voltages for the least important bits (that have a smaller impact on the accuracy of the result). For example, a probabilistic adder has been designed based on a conventional adder with various supply voltages depending on the degree of importance [2]. However, this technique has a very high implementation

cost due to the complex control for the supply voltages. Therefore, most approximate arithmetic circuits are designed based on logic reduction and the pruning method as simple schemes to control and implement.

Both speculative and transistor-level approximate full adders have been studied. Speculative adders can be generally grouped into two types: nonsegmented and segmented adders, which generally reduce the carry delay, but do not save significant power. In general, approximate full adders have not only low power dissipation but also low accuracy. Approximate multipliers have been designed based on approximation at different components [3], including approximate partial product generation, such as approximate Booth encoding, the approximate compressor, and the approximate partial product tree structure. Approximate operand-based multipliers have also been studied, including the dynamic range operand and logarithmic representation-based designs [4]. Also, more complex arithmetic circuits (such as an approximate divider, approximate CORDIC, and

approximate FFT) have been studied based on the logic reduction method.

Memory is another important part of a computing system. Approximate storage has shown to be an efficient option when reducing power consumption. Based on significance-driven criteria, higher order bits are stored in more reliable memory cells (as critical), and different operational modes are employed. These techniques include reducing the refresh rate for DRAM [5] for noncritical data, reducing the supply voltage in SRAM, and fast inexact read/write operations for nonvolatile memories (NVMs) for a longer lifetime, for example, lowering the read/write endurance or reducing the number of writes [6]. Approximate storage can also relax requirements for error correction codes (ECCs). Emerging NVMs including resistive, phase-change, and magnetic random access memories (RAMs)-based approximate vector-matrix multiplication are also attractive because computation is embedded in memory [7].

At a higher hardware level, approximate architectures have been studied for accelerating specific workloads. A quality-programmable extended instruction set architecture (ISA) has been proposed as processor [8]. A floating-point unit (with reduced precision), or fixed-point unit, can be chosen carefully by a graphics processing unit (GPU) architecture to save power [9]. The branch divergence in single instruction multiple data (SIMD) architectures can be limited, or avoided, by introducing approximation at the cost of a small quality loss [10]; an approximation can be used to estimate the load values in a cache and avoid a miss latency. Other techniques include memorization approaches to reuse with similar functions or inputs [11] and memory access skipping [12].

B. Approximate Software/Algorithms

At software and algorithm levels, one of the most effective approximate techniques is precision scaling. By reducing the bit-width of

an operand (so rather than using a unified bit-width), both computation and storage resources can be saved; for example, the width of data in computation can be reduced to 8–16 bits (or even a smaller width) from the original 32 or 64 bits for significant performance and energy improvements. Perforation and skipping can also be used in the iterations of a loop to reduce computation time and save power [13]. It has been shown that the so-called hot loops in error-tolerant applications can be perforated by up to 50% with a similar reduction in execution time, while still producing acceptable results [14].

At a higher level, a program can even skip some tasks and memory accesses in multicore architectures [15]. Data sampling can also be applied to process a portion of the data, and multiple approximate program versions can be provided with different accuracy levels for runtime optimization. Synchronization can also be relaxed for higher performance [14]. In parallel applications, the execution time can be reduced by half if relaxed synchronization mechanisms are allowed.

C. Tools for Approximate Computing

Tools have been developed to assess the different tradeoffs possible in approximate computing. For hardware design, specific approximation approaches are usually limited to a design; therefore, if its parameters are changed, then the approximations and related approach must be accordingly tailored. Therefore, an automated approximation is required for general designs. Systematic logic synthesis methods and tools for approximate circuits (both combinational and sequential) have been proposed by defining a metric constraint to establish an approximate synthesis problem [16]. Automation at the software layer has also been introduced based on a self-tuning approximation for graphics engines. This takes advantage of a GPU microarchitecture and operates based on approximated

kernels [12]. Compiler and programming languages have been proposed to support approximate computing in which type-qualifiers can be utilized to specify approximate data and a portion of a program [17]. The notation to indicate the approximate part using appropriate syntax and semantics has also been provided, such that approximate data can then be stored in approximate memory (such as DRAM) at a reduced refresh rate and processed by inexact arithmetic units.

Metrics for evaluating approximate designs based on error distance have been proposed and widely used when comparing approximate circuits [18]. Open-source libraries of approximate arithmetic circuits are available as both synthesizable hardware description language (HDL) files and C/MATLAB files [19]. Benchmarks for evaluation of various approximate techniques have been developed for comparison purposes; such a benchmark is AxBench [20], which includes a diverse set of applications and supports multiple platforms and various data sets.

II. CURRENT CHALLENGES

Although extensive research is underway in the area of approximate computing, there are still several significant challenges. The main challenge is centered on the analysis and management of the error introduced in an approximate computing system, as well as testing, reliability, and security issues as affected domains.

A. Error Analysis and Management

An error analysis is based on the metrics related to specific applications. Thus, a theory for general error analysis is still not available. Due to the large diversity of approximate computing techniques, errors are introduced both deterministically and nondeterministically. A deterministic error can be managed and controlled within a bounded range; however, for a nondeterministic error, the error

analysis and control are more challenging. Currently, error analysis of an approximate computing system is mostly domain-specific and based on simulation; in most cases, the acceptable amount of errors is found based on an iterative process. Some frameworks have been proposed for this type of error analysis using significance criteria, and in some cases, results from approximate designs are even better than those from conventional designs. As an alternative management technique, compensation has also been proposed to reduce the negative effects of errors, because both exact and approximate results may be required for a wider range of applications of approximate computing.

B. Cross-Layer Codesign

Current efforts are mostly focused on a flat/single layer in which approximation is restricted to modularity. To fully exploit the potential and benefits of approximate computing, codesign of hardware and software/algorithm by combining multiple approximate computing techniques for various applications is a challenging task. Few approaches in which an approximate algorithm can be run on approximate hardware have been proposed; in these cases, the basic principle of operation is that the error introduced from the hardware is bounded, and therefore can be tolerated by the software/algorithm (which has been modified to consider the error characteristics of the approximate hardware). Different applications require different codesign methodologies; therefore, the error analysis discussed in Section III-A can be used to guide the cross-layer design for better results.

C. Verification and Test

An approximate computing system requires verification and test to make sure that acceptable results are generated. This is more challenging

than usual because current electronic design automation (EDA) tools do not support an approximate design. Verification and test methodologies should be re-examined by considering fault classification and test pattern generation with respect to approximate designs and errors. Only so-called critical faults that affect accuracy in a very pronounced manner are then tested with appropriate patterns. Therefore, new testing procedures are needed to bridge approximation to traditional fault coverage.

D. Reliability and Security

Due to the introduced errors, the reliability of an approximate computing system must be closely examined under environmental variations and aging, as these phenomena may have a strong correlation for approximate computing compared to an exact system. For example, voltage overscaling is an immediate challenge for reliability at the circuit level, and it has strong implications on performance.

Furthermore, security threats are a challenge for approximate computing systems, because it is difficult to distinguish the output errors if they contain a malicious modification (such as a hardware/software Trojan). Approximate computing can be applied to security services such as video surveillance, security cameras, and smart grid; therefore, if approximate computing primitives have vulnerability, then the entire system will be under threat. At the same time, new opportunities for hardware security will also be provided by approximate computing (e.g., inexact hardware has been used for hiding security information and authentication for resource-constrained Internet-of-Things (IoT) devices).

III. OUTLOOK

Although approximate computing is not yet at a fully mature stage, it is very promising for future energy-efficient and high-performance

error-tolerant applications. Current approximate computing techniques have been used in several AI-based applications and services by Google and IBM and there is also an increasing trend in EDA and the software engineering community to support approximate computing designs.

In the coming decade, it is anticipated that approximate computing techniques will likely be employed in energy-efficient systems for applications such as AI and signal processing; for example, IBM fabricated an AI accelerator chip to achieve multi-tera operations per second (TOPS) performance by applying multiple approximate techniques (including hardware, architecture, algorithms, and programs) [21]. Furthermore, new approximate computing applications will be exploited. For instance, approximate computing can also provide new approaches for hardware security and implementation of cryptocurrency mining and lattice-based postquantum cryptography [22]. It is believed that more applications will utilize approximate computing due to power limitations. Therefore, it is likely that there will be a growing demand for approximate computing with diverse contributions, from the hardware designer, system developer, test engineer, and research and teaching community, to enable such computing as a mainstream paradigm.

IV. CONCLUSION

Approximate computing has shown great potential for emerging applications. Current research efforts have been pursued to address all aspects of approximate computation, even though a formal design flow has not yet been fully established. An ever-growing number of applications are possible. Significant innovation and research is needed to enable approximate computing as a practical mainstream computing paradigm with the direct involvement of industry. ■

REFERENCES

- [1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.
- [2] K. Palem and A. Lingamneni, "Ten years of building broken chips: The physics and engineering of inexact computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–23, May 2013.
- [3] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 1–34, Aug. 2017.
- [4] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 9, pp. 2856–2868, Sep. 2018.
- [5] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate DRAM," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1172–1187, Jul. 2017.
- [6] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, Mar. 2016.
- [7] Y. Kim, M. Imani, and T. Rosing, "ORCHARD: Visual object recognition accelerator based on approximate in-memory processing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 25–32.
- [8] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2013, pp. 1–12.
- [9] C.-C. Hsiao, S.-L. Chu, and C.-Y. Chen, "Energy-aware hybrid precision selection framework for mobile GPUs," *Comput. Graph.*, vol. 37, no. 5, pp. 431–444, Aug. 2013.
- [10] B. Grigorian and G. Reinman, "Accelerating divergent applications on SIMD architectures using neural networks," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 1, pp. 1–23, Mar. 2015.
- [11] S. Sinha and W. Zhang, "Low-power FPGA design using memoization-based approximate computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 8, pp. 2665–2678, Aug. 2016.
- [12] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "SAGE: Self-tuning approximation for graphics engines," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 2013, pp. 13–24.
- [13] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (SIGSOFT/FSE)*, 2011, pp. 124–134.
- [14] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, Art. no. 120.
- [15] L. Gouri, R. Bianchini, S. Nagarakatte, and T. Nguyen, "ApproxHadoop: Bringing approximations to MapReduce frameworks," in *Proc. 20th Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, 2015, pp. 383–397.
- [16] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 796–801.
- [17] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, Apr. 2011, pp. 85–96.
- [18] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Sep. 2013.
- [19] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based approximate multipliers," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.
- [20] A. Yazdanbakhsh, D. Mahajan, H. Esmailzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Des. Test.*, vol. 34, no. 2, pp. 60–68, Apr. 2017.
- [21] B. Fleischer *et al.*, "A scalable multi-TeraOPS deep learning processor core for AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 35–36.
- [22] W. Liu, C. Gu, G. Qu, and M. O'Neill, "Approximate computing and its application to hardware security," in *Cyber Physical Systems Security*, C. Koc, Ed. New York, NY, USA: Springer-Verlag, 2018, pp. 43–67.