Approximate Circuits
Approximate Circuits

Methodologies and CAD
Approximate computing has emerged as a new paradigm to reduce the resources (e.g., design area and power) required to realize digital systems at the expense of a negligible or small amount of reduction in quality-of-results or accuracy. This trade-off between resources and accuracy is specially relevant for a large class of data-rich applications such as machine learning and multimedia processing that offer inherent error resiliency. This chapter overviews the main technical themes in approximate circuit design methodologies. We elucidate various application domains that are most suitable for approximate circuits, and we then describe a number of error metrics that can capture the quality of results of the applications. We overview the main four technical themes of this book, which are (1) design of approximate arithmetic building blocks, such as adders, multipliers, and dividers, (2) circuit synthesis techniques for arbitrary logic circuits, (3) approximate accelerator design for a number of applications, including deep learning and video coding, and (4) approximate circuit techniques for general-purpose central-processing units and graphical processing units.

Introduction

With the emergence of more and more complex applications in domains like machine learning and multimedia processing, the overall computational workloads of the applications and their respective energy consumption are on the rise. Applications from the domains of Internet-of-Things (IoT) and cyber-physical systems (CPS) that involve huge amount of data analytics require a significant amount of computational and thereby energy and power resources. To overcome these escalating challenges, technology scaling has played a vital role in the past few decades; especially the accelerator-based computing has shown promising results for high energy efficiency. However, with the diminishing returns of technology scaling, alternative computing paradigms have to be considered for alleviating the
resource requirements of the applications and for providing near-optimal level of performance and energy efficiency.

Recent studies by several research groups, like TU-Wien [3] and Brown University [6], and industries, like IBM [10], Microsoft [1], and Intel [9], have shown that a large body of applications are inherently error resilient. This error resilience is usually due to one or more of the following factors (also shown in Fig. 1):

1. Perceptual limitations of the users, where a small error in the visual data is unnoticeable by the users because of their psycho-visual limitations.
2. Lack of a golden answer, where multiple outcomes are equally acceptable for a given input.
3. Resilience to input noise, where applications are designed to deal with noisy inputs and still produce acceptable results even in the presence of such noise.
4. Error masking and attenuation, where error in one stage of the application can be compensated/suppressed because of the application’s characteristics or by the negating error(s) in other stages.

Approximate computing (AC) is one such paradigm that leverages error resilience characteristics of applications for improving the overall resource efficiency of the systems. Traditional techniques, like power gating, dynamic voltage and frequency scaling (DVFS), and power-/energy-aware application mapping, which are popular and have been widely used for achieving significant efficiency gains, are not sufficient enough to meet the growing computing-efficiency demands and can only offer improvements to a limited extent. However, relaxing the bounds of precise computing can open new horizons for the designers by offering (design- and run-time) trade-offs that were never possible using conventional techniques. Figure 2 shows a comparison between traditional and emerging approximate computing-based HW/SW computing stacks.

This book is focused towards introducing the most prominent techniques proposed for employing approximations at the hardware level for achieving high gains in terms of area, power, energy, and performance efficiency. The book is composed of several parts that cover approximate circuits as well as the methodologies for
Fig. 2 Comparison of hardware and software stacks in traditional and approximate computing paradigms. The highlighted blocks in the stack illustrate the focus of the book (Adapted from [13].)

designing approximate hardware. More details related to parts are available in section “Approximate Circuits: Methodologies and CAD” of this chapter. This book also provides a brief overview of the applications and the application domains that can benefit from AC, see section “Application Domains” in this chapter; it also highlights few of the most commonly used error metrics for quantifying quality/efficiency of an application in the presence of approximations, see section “Quality-of-Result Error Metrics” of this chapter.

Application Domains

Approximate computing techniques benefit circuits and systems from application domains that have inherent error resilience as described in section “Introduction”. We describe in this section specific examples of these application domains.

1. Big Data Applications. Big data applications have inherent noise in their data collection process, where large volumes of data often include statistical anomalies. Furthermore, the collected data can be based on inputs from users which can be subjective and can include erroneous or unexplainable data points. Furthermore, big data processing often relies on machine learning techniques, where the results are not 100% accurate anyway.

2. Computer Graphics. Computer graphic algorithms require large computational resources and have strict frame rate requirements, especially for computer games. If the underlying hardware resources are not sufficient, often the rendering of scenes (e.g., shading or lighting) is compromised in accuracy to ensure that target frame rates are met.

3. Image/Signal Processing. Signal processing algorithms rely on inputs from discrete sensors, which introduce errors by their own nature compared to the original analog signals they are measuring. Thus, digital audio or video
data is inherently discrete, whether in floating or fixed-point representations. Furthermore, fixed-point format, which introduces a degree of discretization, is usually the method of choice for digital filtering in embedded systems.

4. **Computer Vision.** Computer vision algorithms seek to provide an understanding of digital images. These applications, for example, include object detection, pose estimation, and motion tracking. Many of the computer vision algorithms are not optimal and can only achieve their objectives within certain accuracy, and even human subjects can often misunderstand digital images.

5. **Deep Learning.** Deep learning algorithms rely on multi-layer artificial neural networks for processing input data to produce the required classification outcomes or labels for the input data. Neural networks are inherently tolerant to errors as a network can recover from hardware errors and “heal” itself by retraining, where a new set of weights can be used to compensate for the simplifications in the underlying hardware.

6. **Biometric security.** Biometric applications include fingerprinting, iris scanning, and 3D facial recognition. These applications are data rich, and they use intensive statistical machine learning, image processing, and computer vision techniques, which are naturally resilient to variations in the inputs (e.g., arising from the light conditions). More importantly, the differences in the biometric signatures among individuals are large enough, such that minor changes in the signature can be considered inconsequential.

**Quality-of-Result Error Metrics**

Given that the goal of approximate computing is to design circuits and systems with improved hardware resources at the expense of accuracy, designers need to be able to evaluate an approximate design against its original input design and compare the quality-of-results (QoR) according to some error metric as illustrated in Fig. 3. An error metric is often incorporated as a part of the objective function or constraints of an optimization formulation that is used to generate and evaluate approximate circuits. There are a large range of possible error metrics that can be considered.
depending on the type of the circuit and the target application. Examples of these error metrics include the following.

1. **Hamming distance.** The Hamming distance metric considers the number of output bit flips in the approximate circuit compared to the original circuit across all possible inputs. This generic metric is often useful; however, it can be insufficient for arithmetic circuits where different bits have different weights; i.e., a bit flip in the most significant bit of an adder’s output has much more impact than a bit flip in the least significant bit.

2. **Arithmetic difference.** In this metric the semantics of the output bits of the circuit are known; for instance, if an adder has 32 outputs, then these 32-bit outputs represent a number. Thus, we can directly compare the numerical values of the outputs of the arithmetic original circuit against its approximate version. The absolute value of the arithmetic difference can be also considered.

3. **Square error.** In this error metric, the numerical difference between the original circuit and approximate circuit is squared. If the square error metric is used in the objective function, then it helps guide the optimization method towards approximate designs that do not produce large errors, as these errors will be amplified by squaring, penalizing any design with large deviations.

4. **Application-specific metrics.** Depending on the application, more specific error metrics can be considered.

   (a) **Signal-to-noise ratio.** This error metric is typically used in signal processing applications, and it is defined as the ratio between the power of the output signal and the power of the background noise. The background noise here arises from the difference between the approximate and original circuit results. That is, the impact of the introduced approximations can be considered as a factor contributing to the noise in the outputs.

   (b) **Classification rate.** This error metric is popular in machine and deep learning applications, where the accuracy of the circuit or system is quantified based on the correctness of the classification. For instance, for an object detection application, the classification rate would be the ratio of the number of objects that have been recognized correctly in an image set divided by the number of objects in the image set.

   (c) **False positive and negative rates.** These two error metrics are helpful for security systems. For instance, the false positive rate is the ratio of the number of individuals who were authorized in mistake to the total number of authorized individuals, whereas the false negative rate is the ratio of the number of individuals who were denied in mistake to the total number of denied individuals.

   (d) **User experience.** This metric is subjective as it is based on experience of users, and it is often used in computer graphics and some computer vision systems. In this case, a sample of the users of the system experience both the accurate and approximate systems, and then respond to a series of questionnaire that assess the quality of the experience.
For each one of the aforementioned error metrics, one could be concerned about the sum of the errors, average case error, the average absolute error, the standard deviation of the error, the worst-case error, or the entire error distribution. The entire distribution of errors is often desirable in cases where the approximate circuit is composed with other approximate circuits, and the error distributions can help calculate the final error distribution of the circuit. Sengupta et al. discuss methods for error composition in circuits in Chap. 11.

The evaluation of error metrics can be done through either simulation or formal methods. For simulation, it is often the case there are testbenches that assess the output quality of the original circuit (e.g., classification rates) on widely used benchmarks. These same testbenches can be used to evaluate the approximate circuit and compare its outcomes against the original accurate circuit to quantify the error between them [11]. The downside of simulation-based techniques is they are unlikely to be exhaustive, especially if the input range of the circuit is large. With simulation-based methods, there is always the possibility that the average or worst-case errors calculated from the testbenches deviate from the real average or worst-case errors considering all possible input scenarios. If an absolute guarantee on the QoR is required, then formal methods can be used. Formal methods can rely on Boolean satisfiability solvers to provide a guarantee on the solution. However, scalability is often the main challenge for formal methods. A comprehensive overview of evaluation methods is provided by Sekanina et al. in Chap. 9.

**Approximate Circuits: Methodologies and CAD**

The material of this book is organized into four main parts that cover the scope of technical work done for approximate circuit design methodologies and design automation. Part I focuses on techniques for generating approximate arithmetic building blocks for circuits such as adders, subtractors, multipliers, and dividers. For general circuits, Part II discusses a number of approximate synthesis and error analysis techniques for arbitrary arithmetic and logic circuits, and for systems described at a higher level of abstraction (e.g., C language). When the application context of a circuit is known, as in the case of accelerator circuits, it is possible to achieve a larger range of approximations with improved accuracy. Part III discusses a number of approximate accelerator designs in the domains of computer vision, deep learning, and biometric security. In Part IV, a number of techniques are provided for designing approximate general-purpose CPUs and GPUs as well as accompanying software-based methods. We briefly overview the chapters in each of these parts.
Approximate Arithmetic Building Blocks

The first part of this book covers approximate building blocks that are used for constructing larger accelerators/architectures. The main focus of this part is on approximate arithmetic units like approximate adders, multipliers, and dividers, which are vital for any digital processing architecture.

Arithmetic circuits are the building blocks of almost all the applications that involve data processing. Applications that are more computationally intensive spend most of their resource budget in these modules. Therefore, in such cases, the energy/power or performance efficiency can be improved by optimizing/simplifying such modules. A lot of techniques have been proposed that make use of functional simplifications of the circuits for achieving significant efficiency gains. One such example is approximate full-adder units where the functionality of the units is approximated to reduce the overall power consumption of the multi-bit adders [4]. Similar types of approximations are also common for multiplier modules where the functionality of the building blocks of the larger multiplier units (like $2 \times 2$ multipliers and full-/half-adder units) is approximated to reduce resource consumption of the modules [8]. The organization of Part I is presented in Fig. 4, where two chapters are allocated to designs and methodologies for building approximate arithmetic units, two chapters are allocated to evaluation and design space exploration which analyzes the characteristics of various current approximate arithmetic circuits, and one chapter is allocated to error analysis which covers probabilistic analysis of state-of-the-art approximate adders and multipliers.

Fig. 4 Organization of Part I, highlighting key contributions of different chapters
In Chap. 1, Hanif et al. present generic configurable models for low-latency approximate adders. These models provide design-time support for selecting a suitable approximate adder configuration that offers a desired level of performance while minimally affecting the overall accuracy of the applications. The chapter also presents mathematical analysis to show that, given a latency constraint, a configuration which provides optimal quality-area trade-off can effortlessly be selected from the entire design space of low-latency adders, considering uniformly distributed inputs.

In Chap. 2, Hashemi and Reda present a unique methodology for designing approximate multipliers and dividers. The methodology is based upon dynamic selection of input bits, which are then used for performing the computation. The selection of bits is made by analyzing the leading zeros in the binary inputs. The designs developed using the proposed methodology have the feature to maintain an upper bound on the maximum possible error while providing dynamic accuracy, depending upon the input values. The methodology is highly scalable and offers a wide range of power and inaccuracy trade-off.

Chapter 3 by Rehman et al. presents a comprehensive methodology for exploring the entire design space of approximate recursive multipliers. The recursive multipliers are composed of elementary multiplier and adder modules, e.g., 2×2 multipliers and full-/half-adders and can be broken down into three major stages: (1) partial product generation stage; (2) accumulation stage; and (3) summation stage. To explore the entire design space of approximate multipliers, the work exploits approximations in all three stages of the multiplier architecture for identifying points that provide optimal trade-off between output quality and efficiency.

In Chap. 4, Jiang et al. present a classification of approximate arithmetic circuits including approximate adders, multipliers, and dividers. The chapter also presents a comparative study of the current approximate arithmetic modules using various quality metrics, such as error rate (ER), mean relative error distance (MRED), mean error distance (MED), and normalized mean error distance (NMED), and circuit characteristics, like power, area, delay, power delay product (PDP), and area delay product (ADP). The study provides insightful results about the trade-offs of different designs.

In Chap. 5, Mazahir et al. present probabilistic error analysis of approximate adders and multipliers. The presented analysis can be used to compute error probability mass function (PMF) of different approximate modules, which in turn can be used for commuting other error measures that are used for quantifying the quality of an approximate module. The analysis mainly covers approximate adders, which comprise sub-adder units, and recursive approximate multipliers with approximate partial products. The analysis is highly useful for efficiently computing error measures of different configurations of a component for a given application and input distribution. This enables the designer to explore the complete design space in a relatively smaller amount of time for a particular application.
Approximate Circuit Synthesis

The second part in the book addresses the general problem of synthesizing approximate circuits from an arbitrary input circuit. The organization of Part II is given in Fig. 5. The first three chapters describe approximate synthesis techniques for circuits described at the logic level. The fourth chapter describes a technique that can approximate circuits described at the arithmetic or logic levels. The fifth chapter raises the level of abstraction and provides approximate high-level synthesis techniques, while the last chapter provides a comprehensive review of error analysis and budgeting techniques.

In Chap. 6, Ranjan et al. discuss a good number of methods for generating approximate and quality configurable circuits. One of the discussed methods is SALSA [14]. SALSA first creates a difference circuit that consists of the original circuit, the approximate circuit, and a comparator that compares the results of the original circuit and the approximated circuit as illustrated earlier in Fig. 3. The key idea is that the observability don’t cares of the outputs of the approximate circuit, which are internal nodes in the difference circuit, can be used to simplify
the approximate circuit using regular logic synthesis techniques. This method has been extended in ASLAN [12] for sequential circuits to model errors arising over multiple cycles. ASLAN also uses a circuit block exploration method that identifies the impact of approximating the combinational blocks and then uses a gradient-descent approach to find good approximations for the entire circuit. The chapter also discusses techniques to configure the quality of results during runtime.

In Chap. 6, Hashemi et al. describe a new method for logic synthesis based on Boolean matrix factorization (BMF). The main idea of this technique is to first enumerate the truth table of a given multi-input, multi-output circuit. The output-side evaluations of the truth table can be considered as a Boolean matrix $M$. Using BMF, the matrix $M$ can be then factored into two matrices $BC$, such that $|M - BC|_2$ is minimized. The degree of factorization controls the size of the matrices $B$ and $C$, i.e., number of columns of $B$ and number of rows of $C$, which in turn controls the accuracy of the factorization. The compressor circuit whose truth table is given by the matrix $B$ is then synthesized, and the outputs of the compressor circuits are then ORed by a decompressor circuit according to the matrix $C$ to produce the final approximate outputs. To generalize to larger circuits, the chapter describes a method that can decompose a large circuit into smaller subcircuits, where BMF can be applied on each one individually, and a subcircuit exploration method is used to determine the best order of subcircuits for factorization.

Froehlich et al. describe in Chap. 8 the use of formal techniques such as binary decision diagrams (BDDs) and symbolic computer algebra (SCA) to generate single-output and multi-output approximate circuits, respectively. For a single-output circuit, the goal is to find the minimal function, where the output of this function differs in at most $e$ possible input combinations to the circuit; i.e., at most $e$ outputs are flipped. To identify this function, the idea is to first construct a new BDD, $F$, that enumerates every possible function whose output differs in at most $e$ bits. That is, $F$ represents all possible circuit approximations with at most $e$ output flips. For example, if the original single-output circuit has two inputs, then it has four output possibilities, one for each input combination. As a result, the BDD $F$ will have six variables: four variables indicate which output bits are flipped in the approximate circuit and two variables for the possible input combinations. A partial path in this BDD for the first four variables will lead to “0” if the output function of the approximate circuit has more than $e$ bit flips; otherwise, it will lead to a subgraph BDD, whose variables are the two original circuit inputs. This subgraph BDD represents the logic of the approximate circuit corresponding to a particular configuration of output bit flips as determined by the partial path. If we enumerate the paths of this subgraph BDD and compare it to the original circuit, we will find that no more than $e$ paths lead to different outcomes. For multi-output circuits, a heuristic is proposed where a gate is replaced by one of its inputs as long as the approximation error does not exceed a given bound. SCA techniques are used to compute the error metric in response to a potential replacement.

In Chap. 9, Sekanina et al. consider datapath circuits that are composed of basic arithmetic blocks (e.g., adders and multipliers) and logic blocks. The methodology encodes the exact circuit in a string-based representation as a “chromosome”
and then uses genetic algorithms to mutate the circuit to create approximate versions as long as the error is kept below target. These mutations can modify a block’s function, a block’s input connection or an output connection. The chapter also provides a comprehensive overview of efficient QoR evaluation techniques, including fast parallel simulation, BDD-based analysis, and error analysis using Boolean satisfiability solvers.

Raising the level of abstraction, Lee and Gerstlauer describe in Chap. 10 techniques for high-level synthesis from C to register-transfer level (RTL) under various design targets and QoR constraints. Models for energy, latency, and QoR are first developed and then later incorporated into an optimization formulation to find the Pareto-optimal designs. Based on profiling results, loop clustering is first performed, where a loop is split into a number of clusters, each with various approximation QoR targets and iteration counts. Then, for each cluster, an optimization formulation is solved to minimize the latency or energy subject to QoR constraints, where the decision variables determine what data operands and operations to approximate. Finally, operation scheduling and binding are performed to synthesize the RTL of the approximate designs.

Finally in Chap. 11, Sengupta et al. describe general techniques for error analysis and error budget optimization for approximate circuits. For error analysis, the idea is to first establish the probability mass function (PMF) of basic approximate building blocks (e.g., adders or multipliers) and then compose these PMFs according to the topology of the datapath to produce the PMF of the entire circuit. For error budget optimization, the key problem is to take the error budget for an entire circuit and allocate it among its components, while minimizing the resources used by the circuit. Test cases for optimizing JPEG hardware and FIR filters are considered.

**Approximate Accelerator Design**

Part III of the book presents approximate accelerators and techniques that can be employed for developing approximate accelerators.

Employing application-specific accelerators is one of the most effective ways for improving the efficiency of an application. However, in some cases, even the cost of accelerators is significantly higher than the available resource budget. Therefore, to meet the performance and budget constraints simultaneously, approximate computing can be adopted to alleviate the resource consumption at the cost of insignificant accuracy loss by exploiting error resilience of the applications. As mentioned in section “Application Domains” of this chapter, the main applications of AC include multimedia processing and machine learning. Therefore, Part III of the book is focused towards illustrating the use of AC in such applications. The organization of Part III is presented in Fig. 6, where one chapter is allocated to approximations in video coding, two chapters are allocated to DNNs, and one chapter is allocated to approximations in biometric security systems with a case study in iris recognition system.
Video coding is one of the most widely used applications for compressing the size of the videos. It is mainly composed of several stages that involve intra- and inter-prediction based encoding for achieving maximum compression ratios. In Chap. 12, Prabakaran et al. present a brief overview of the latest High Efficiency Video Coding (HEVC) standard followed by a thorough analysis of its computational complexity and energy consumption. Based on the analysis, the chapter highlights the importance of approximations in motion estimation and presents a full-system approach to realize an approximate architecture for energy-efficient motion estimation.

Deep learning is an area that is widely followed because of its state-of-the-art accuracy in many AI applications. However, the high computational complexity of these algorithms renders them unusable for many small-scale applications, for instance in IoT nodes and highly constrained CPS devices. In Chap. 13, Hanif et al. present hardware and software level approximations for DNNs. The chapter presents a methodology for analyzing the sensitivity of different parts of a neural network through error injection. The sensitivity analysis is then used as the basis for employing approximations at different parts and at different abstractions of a neural network. The chapter also highlights few of the key challenges involved in employing approximations in neural networks.

Along the same dimensions, in Chap. 14, Tann et al. present methods to devise light-weight approximate accelerators for DNN accelerations with minimal accuracy loss. The work analyzes the complete range of data precision methods, like fixed-point, dynamic fixed point, powers-of-two, and binary data, to reduce the overall hardware complexity of the accelerator. In conjunction, a novel training method has also been proposed in the chapter to compensate for the accuracy loss because of approximations. To further boost the accuracy, an ensemble processing
technique is discussed in the chapter that makes use of a group of lightweight neural networks for improving the accuracy beyond state of the art.

In Chap. 15, Choi and Venkataramani present approximation techniques that can be employed for alleviating the computational and resource demands of state-of-the-art neural networks—which are presumed to be one of the highly computational intensive applications of the current era. The chapter presents a holistic overview of the techniques (both for training and inference) that have shown significant results in conserving the energy and improving the performance efficiency of the networks. The chapter mainly summarizes the state of the art from three main subfields of optimizations in deep neural networks (DNNs), i.e., (1) pruning, (2) quantization, and (3) input-adaptive approximations.

In Chap. 17 Raha and Raghunathan highlight the significance of employing approximations in multiple subsystems of a computing system, rather than focusing on an individual subsystem, for achieving substantial energy benefits. Towards this end, the chapter presents a gradient descent-based approach for optimizing the quality-energy trade-off, which proved to be as efficient as exhaustive grid search. To illustrate the applicability of the proposed concept, the chapter makes use of an example of a smart camera system, capable of executing various computer vision and image processing applications, and illustrates how the different subsystems can be approximated synergistically for achieving optimal energy benefits.

In Chap. 16, Tann et al. present approximations for biometric security systems with a thorough case study on iris recognition application. The chapter explores the design space of entire hardware/software pipeline of the system and investigates the possibility of approximation at various abstraction levels in search of a highly efficient system. To explore the extended design space it presents a reinforcement learning technique with recurrent neural networks as learning agents. This application mainly highlights the potential of AC even in applications that require significant accuracy for operating. The results in the chapter illustrate that AC can achieve significant performance/efficiency improvements while meeting the standard industrial accuracy constraints.

**Approximate CPU and GPU Design**

Approximate computing techniques have been also advocated for general-purpose CPUs and GPUs. For CPUs/GPUs, one can change the underlying hardware to make it approximate by nature, and/or change the software application to make it more approximate. For hardware approximations, a major question is what are the best structures to apply approximate computing methods. Analysis of dynamic instruction counts for CPUs shows that, on average, loads and stores account for 34% of the instructions, followed by branches and comparisons at 36%, and ALU instructions at 19% [7]. Furthermore, based on the published data of Hammed et al. [5], Fig. 7 illustrates the power consumption breakdown and analysis of CPU and GPU processors using the same application (H.264 encoding). Relying on single-instruction multiple data (SIMD) ALU units, GPGPUs naturally consume
more power for the arithmetic vector units. Combining the analysis from power decomposition and dynamic instruction count breakdown, one can arrive to a prioritization of the potential targets for approximate computing in CPUs and GPUs. First, software techniques that simplify the underlying computations can yield large savings as they inherently reduce the number of instructions and required data, which reduces the usage of all CPU/GPU structures. Second, at the hardware levels, techniques that target loads and data transfer in general can lead to big power savings. Techniques that target the arithmetic units can lead to some savings with greater benefits for the GPGPU than the CPU. Accordingly, the selected chapters for this part reflect this prioritization. The organization of Part IV is given in Fig. 8, where two chapters are allocated for ALU approximations, two chapters are allocated for load approximations, and one chapter is allocated for software techniques.

Fig. 7  Power breakdown for a CPU and a GPU processor using H.264 application. Data based on Hammed et al. [5]. FU denotes functional units. RF denotes register file. Ctl denotes the control unit. Pipe denotes the pipeline registers. D$ denotes the data cache. IF denotes instruction fetch.
In Chap. 18, N.S. Kim and U.R. Karpuzcu propose designing approximate many-core processors using near-threshold voltage methods. Bringing the operating voltage near the threshold level leads to dramatic power savings, but also decent loss to performance. In compensation, one can increase the number of cores to leverage parallel processing, while still achieving large savings in power consumption. However, timing errors might still occur, which necessitates the use of error-resilient applications as discussed earlier in section “Application Domains”. Since timing errors should not impact the control structures of the processor, the chapter advocates decoupling the data and control portions of a CPU core, such that the control portion operates at safe voltage levels. An analysis of the overall power savings and performance gains is presented.

M. Imani and T.S. Rosing present in Chap. 19 a technique that approximates arithmetical logic units (ALUs) in both CPUs and general-purpose GPUs. Instead of replacing the ALU with approximate arithmetic components (e.g., using the techniques in Part I) or using aggressive voltage scaling techniques as in Chap. 18, the chapter advocates the use of associative memory to store precomputed results from the ALU. The memory is later looked up to identify results for input operands of the ALU even if they do not match up exactly to the stored computations. The authors show that replacing the ALU with associative memory look-ups can lead to decent energy savings, with reasonable accuracy. The chapter describes how to design such associative memory using memristor technology.

Given that data movement represents a decent portion of power consumption in a CPU as discussed earlier, N.E. Jerger and J.S. Miguel describe techniques in Chap. 20 to reduce the energy consumption of caches and the costs of load transfers by exploiting the error resilience of approximate computing applications. The first technique is based on a load value approximator that estimates the value of a cached memory location bypassing the need to look up the actual data value in the cache memory hierarchy. An efficient hardware design for the approximator is presented. A second technique for effective cache organization that exploits the similarity among data values to increase the effective cache capacity is presented. The idea is to achieve greater deduplication by exploring approximate value similarity across cache blocks for approximate applications. For instance, if two cache blocks are similar, then we can replace the values of one block with the other one, thereby freeing cache storage space. To realize this idea, hardware designs based on the Doppelgänger cache and Bucker cache are presented.

In case of load cache misses, the main memory has to be accessed in order to provide the required data values, which can take a large number of cycles and can cause stalls. Furthermore, memory bandwidth is often a limitation for accelerator-rich systems. In Chap. 21, A. Yazdanbakhash et al. offer techniques to mitigate both the long latencies and limited memory bandwidth using approximate computing techniques. The main idea is to skip main memory accesses for safe-to-approximate load operations and instead predict the value requested by the load, with no recovery mechanisms for mispredictions. That is, the load value predictions are roll free. Mispredictions are tolerated due to the inherent error resilience nature of approximate computing applications. By skipping memory accesses, the long
latency and demand for memory bandwidth are mitigated. For GPGPUs, a new multi-value prediction algorithm for single-instruction multiple-data (SIMD) load instructions is described. By exploiting the similarity among multiple data values accessed by adjacent GPGPU threads, one can avoid the overhead for a multi-value load prediction mechanism.

Finally, W.F. Wong et al. present in Chap. 22 a summary of high-level approximation and precision analysis techniques that are applicable at the software and compilation levels. Software techniques such as precision reduction, loop perforation, task skipping, and sloppy memorization are described. These techniques can be applied to any CPU or GPU to approximate the underlying computations while still executing on an accurate hardware substrate. For precision reduction, dynamic and static analysis and optimization techniques are described. Dynamic techniques for sensitivity analysis are described to identify the better variables to approximate from actual execution traces. Static dataflow analysis techniques are also presented to assess the approximability of variables. Based on the static analysis, precision tuning techniques for both fixed-point and floating-point variables are presented.

References


Contents

Part I  Approximate Arithmetic Circuit

1 Configurable Models and Design Space Exploration for Low-Latency Approximate Adders ........................................ 3
  Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique

2 Approximate Multipliers and Dividers Using Dynamic Bit Selection ................................................................. 25
  Soheil Hashemi and Sherief Reda

3 Heterogeneous Approximate Multipliers: Architectures and Design Methodologies ................................................. 45
  Semeen Rehman, Bharath Srinivas Prabakaran, Walaa El-Harouni, Muhammad Shafique, and Jörg Henkel

4 Approximate Arithmetic Circuits: Design and Evaluation .......... 67
  Honglan Jiang, Leibo Liu, Fabrizio Lombardi, and Jie Han

5 Probabilistic Error Analysis of Approximate Adders and Multipliers ................................................................. 99
  Sana Mazahir, Muhammad Kamran Ayub, Osman Hasan, and Muhammad Shafique

Part II  Approximate Circuit Synthesis

6 Automatic Synthesis Techniques for Approximate Circuits .......... 123
  Ashish Ranjan, Swagath Venkataramani, Shubham Jain,
  Younghoon Kim, Shankar Ganesh Ramasubramanian, Arnab Raha,
  Kaushik Roy, and Anand Raghunathan

7 Approximate Logic Synthesis Using Boolean Matrix Factorization . 141
  Soheil Hashemi, Hokchhay Tann, and Sherief Reda

8 Approximate Hardware Generation Using Formal Techniques ....... 155
  Saman Froehlich, Daniel Große, and Rolf Drechsler
9 Automated Search-Based Functional Approximation for Digital Circuits .......................................................... 175
Lukas Sekanina, Zdenek Vasicek, and Vojtech Mrazek

10 Approximate High-Level Synthesis of Custom Hardware ........... 205
Seogoo Lee and Andreas Gerstlauer

11 Error Analysis and Optimization in Approximate Arithmetic Circuits ......................................................... 225
Deepashree Sengupta, Jiang Hu, and Sachin S. Sapatnekar

Part III Application-Specific Approximate Accelerators and Systems

12 Approximate Multi-Accelerator Tiled Architecture for Energy-Efficient Motion Estimation ................................. 249
Bharath Srinivas Prabakaran, Walaa El-Harouni, Semeen Rehman, and Muhammad Shafique

13 Hardware–Software Approximations for Deep Neural Networks .... 269
Muhammad Abdullah Hanif, Muhammad Usama Javed, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique

14 Lightweight Deep Neural Network Accelerators Using Approximate SW/HW Techniques ........................................ 289
Hokchhay Tann, Soheil Hashemi, and Sherief Reda

15 Approximate Computing Techniques for Deep Neural Networks….. 307
Jungwook Choi and Swagath Venkataramani

16 Approximate Computing for Iris Recognition Systems ............ 311
Hokchhay Tann, Soheil Hashemi, Francesco Buttafuoco, and Sherief Reda

Arnab Raha and Vijay Raghunathan

Part IV Approximate Methods for CPUs and GPUs

18 Approximate Ultra-Low Voltage Many-Core Processor Design...... 371
Nam Sung Kim and Ulya R. Karpuzcu

19 Approximate CPU and GPU Design Using Emerging Memory Technologies .......................................................... 383
Mohsen Imani and Tajana S. Rosing

20 Approximate Cache Architectures ........................................ 399
Natalie Enright Jerger and Joshua San Miguel
21 Towards Breaking the Memory Bandwidth Wall Using Approximate Value Prediction ............................................ 417
Amir Yazdanbakhsh, Gennady Pekhimenko, Hadi Esmailzadeh, Onur Mutlu, and Todd C. Mowry

22 Compilation and Other Software Techniques Enabling Approximate Computing ............................................. 443
Weng-Fai Wong, Pooja Roy, Rajashi Ray, and Nhut-Minh Ho

Index ................................................................................................................. 465
Contributors

Muhammad Kamran Ayub  National University of Sciences and Technology, Islamabad, Pakistan
Francesco Buttafuoco  Polytechnic University of Turin, Turin, Italy
Jungwook Choi  IBM, Yorktown Heights, NY, USA
Rolf Drechsler  Group of Computer Architecture, University of Bremen, Bremen, Germany
Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
Walaa El-Harouni  Private Researcher
Hadi Esmaeilzadeh  UC San Diego, San Diego, CA, USA
Saman Froehlich  Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
Andreas Gerstlauer  UT Austin, Austin, TX, USA
Daniel Große  Group of Computer Architecture, University of Bremen, Bremen, Germany
Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
Rehan Hafiz  Information Technology University, Lahore, Pakistan
Jie Han  University of Alberta, Edmonton, AB, Canada
Muhammad Abdullah Hanif  TU Wien, Vienna, Austria
Osman Hasan  National University of Sciences and Technology, Islamabad, Pakistan
Soheil Hashemi  Brown University, Providence, RI, USA
Jörg Henkel  Karlsruhe Institute of Technology, Karlsruhe, Germany
Nhut-Minh Ho  National University of Singapore, Singapore, Singapore
Jiang Hu  Texas A&M, College Station, TX, USA
Mohsen Imani  UC San Diego, La Jolla, CA, USA
Shubham Jain  Purdue University, West Lafayette, IN, USA
Muhammad Usama Javed  Information Technology University, Lahore, Pakistan,
Natalie Enright Jerger  University of Toronto, Toronto, ON, Canada
Honglan Jiang  University of Alberta, Edmonton, AB, Canada
Ulya R. Karpuzcu  University of Minnesota, Minneapolis, MN, USA
Younghoon Kim  Purdue University, West Lafayette, IN, USA
Nam Sung Kim  University of Illinois Urbana Champaign, Urbana Champaign, IL, USA
Seogoo Lee  Cadence Design Systems, San Jose, CA, USA
Leibo Liu  Tsinghua University, Beijing, China
Fabrizio Lombardi  Northeastern University, Boston, MA, USA
Sana Mazahir  Georgia Tech, Atlanta, GA, USA
Joshua San Miguel  University of Wisconsin-Madison, Madison, WI, USA
Todd C. Mowry  Carnegie Mellon University, Pittsburgh, PA, USA
Vojtech Mrazek  Brno University of Technology, Brno, Czech Republic
Onur Mutlu  ETH Zurich, Zurich, Switzerland
Bharath Srinivas Prabakaran  TU Wien, Vienna, Austria
Gennady Pekhimenko  University of Toronto, Toronto, ON, Canada
Anand Raghunathan  Purdue University, West Lafayette, IN, USA
Vijay Raghunathan  Purdue University, West Lafayette, IN, USA
Arnab Raha  Intel Corporation, Santa Clara, CA, USA
Shankar Ganesh Ramasubramanian  Intel University, Hillsboro, OR, USA
Ashish Ranjan  Purdue University, West Lafayette, IN, USA
Rajashi Ray  National Institute of Technology, Meghalaya, India
Sherief Reda  Brown University, Providence, RI, USA
Semeen Rehman  TU Wien, Vienna, Austria
Tajana S. Rosing  UC San Diego, La Jolla, CA, USA
Pooja Roy  Intel Technology India Pvt Ltd, Bengaluru, India
Kaushik Roy  Purdue University, West Lafayette, IN, USA
Sachin S. Sapatnekar  University of Minnesota, Minneapolis, MN, USA
Muhammad Shafique  TU Wien, Vienna, Austria
Lukas Sekanina  Brno University of Technology, Brno, Czech Republic
Deepashree Sengupta  Synopsys Inc., Sunnyvale, CA, USA
Hokchhay Tann  Brown University, Providence, RI, USA
Zdenek Vasicek  Brno University, Brno, Czech Republic
Swagath Venkataramani  IBM, Yorktown Heights, NY, USA
Weng-Fai Wong  School of Computing National University of Singapore, Singapore
Amir Yazdanbakhsh  Georgia Tech, Atlanta, GA, USA