Approximate Computing and the Quest for Computing Efficiency *

Swagath Venkataramani†, Srimat T. Chakradhar‡, Kaushik Roy† and Anand Raghunathan†
† School of Electrical and Computer Engineering, Purdue University
‡ Systems Architecture Department, NEC Laboratories America
†{venkata0,kaushik,raghunathan}@purdue.edu, ‡chak@nec-labs.com

Invited

ABSTRACT

Diminishing benefits from technology scaling have pushed designers to look for new sources of computing efficiency. Multi-cores and heterogeneous accelerator-based architectures are a by-product of this quest to obtain improvements in the performance of computing platforms at similar or lower power budgets. In light of the need for new innovations to sustain these improvements, we discuss approximate computing, a field that has attracted considerable interest over the last decade. While the core principles of approximate computing — computing efficiently by producing results that are good enough or of sufficient quality — are not new and are shared by many fields from algorithm design to networks and distributed systems, recent efforts have seen a percolation of these principles to all layers of the computing stack, including circuits, architecture, and software. Approximate computing techniques have also evolved from ad hoc and application-specific to more broadly applicable, supported by systematic design methodologies. Finally, the emergence of workloads such as recognition, mining, search, data analytics, inference and vision are greatly increasing the opportunities for approximate computing. We describe the vision and key principles that have guided our work in this area, and outline a holistic cross-layer framework for approximate computing.

1. INTRODUCTION

Designers of integrated circuits and computing platforms are increasingly squeezed between diminishing benefits from technology scaling and increased demands from computing workloads. The end of classical or Dennard scaling [3, 4] has led to the slowing of clock frequency increases and voltage reductions with each technology generation, driving the move to the multi-core era. However, the inability to fit more cores within constrained power budgets has been projected to threaten even this new scaling paradigm [5]. As a result, designers are being pushed to seek new sources of computing efficiency [6].

Alongside the above trends, the nature of workloads that drive computing demand has also changed fundamentally across the computing spectrum, from mobile devices to the cloud. In data centers and the cloud, the demand for computing is driven by the need to manage — organize, search and draw inferences from — exploding amounts of digital data [7]. In mobile and deeply embedded devices, richer media and the need to recognize and interact more naturally and intelligently with users and the environment drive much of the computing demand. There is a common pattern that emerges from across the spectrum. These applications are largely not about calculating a precise numerical answer; instead, "correctness" is defined as producing results that are good enough, or of sufficient quality, to achieve an acceptable user experience.

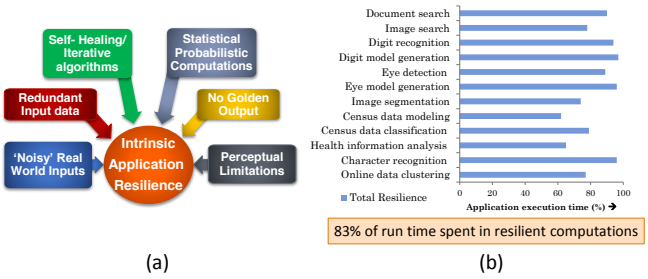

Figure 1: Intrinsic application resilience

Not surprisingly, these workloads exhibit *intrinsic application resilience* to approximations, or an ability to produce acceptable outputs even when some of their computations are performed in an approximate manner [8, 9]. As illustrated in Figure 1, intrinsic application resilience arises due to the following factors:

- A unique, golden result does not exist and a range of answers are equally acceptable (common examples include search and recommendation systems).
- Even when a golden answer exists, the best known algorithms may not be guaranteed to find it, and hence, users are conditioned to accept imperfect, but good-enough results (most machine learning applications fall into this category).
- Applications are often designed to deal with noisy input data. Noise at the inputs naturally propagates to the intermediate results; qualitatively, approximations have the same effect. In other words, robustness to

*This work was supported in part by the National Science Foundation under grants CNS-1423290, and CNS-1018621. This paper is part of the DAC 2015 special session "Dark Silicon: No Way Out", along with [1, 2]

DAC'15, June 07–11, 2015, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2744769.2744904.

noisy inputs also endows applications with robustness to approximations in computations.

- Applications frequently use computation patterns such as aggregation or iterative-refinement, which have the property of attenuating or healing the effects of approximations.

Recent studies have quantitatively established the high degree of intrinsic resilience in many applications. For example, our analysis of a benchmark suite of 12 recognition, mining and search applications shows that on average, 83% of the runtime is spent in computations that can tolerate at least some degree of approximation [8]. Therefore, there is significant potential to leverage intrinsic application resilience in a broad context.

2. APPROXIMATE COMPUTING

Approximate computing broadly refers to techniques that exploit the intrinsic resilience of applications to realize improvements in efficiency at all layers of the computing stack. Some of the key principles underlying approximate computing can be traced back to several well-established disciplines. For example, approximation, probabilistic, and heuristic algorithms [10, 11] embody a trade-off between optimality of results and computational complexity. Compression algorithms for images, audio and video exploit the perceptual limitations of users who consume their output to greatly reduce the network bandwidth and storage required for these media. Networking protocols such as IP and UDP embrace best-effort packet delivery, and many applications use these protocols, eschewing the overheads associated with guaranteed packet delivery [12]. Modern information storage and retrieval systems have embraced relaxed consistency models in order to achieve scalability [13]. *Can the principles embodied in the above examples can be captured and applied to other aspects of computing system design, and across all layers of the computing stack?*

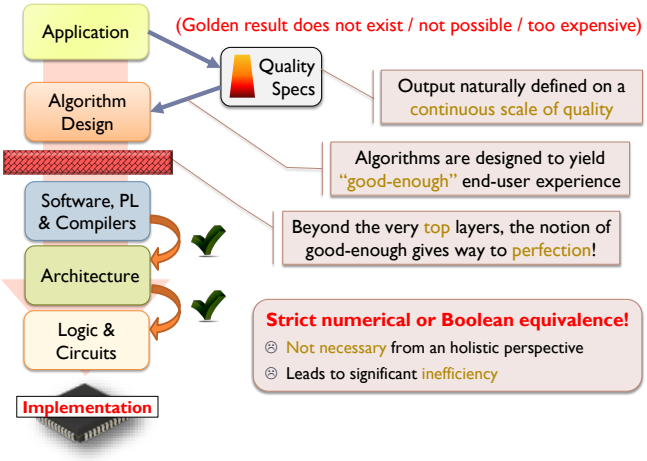

Figure 2: Most layers of the computing stack are designed with a strict notion of correctness

We note that all the examples mentioned above are invariably limited to the very top layers of the computing stack. Most of the layers – programming languages, compilers, runtimes, operating systems, architecture, and circuits – are still designed to adhere to a strict notion of correctness. As algorithms and applications are translated into software or hardware implementations, there is typically a point in the process where a notion of acceptable or good-enough results gives way to a strict notion of correctness, as illustrated in Figure 2. Once even an approximate algorithm is specified as a program, its execution must preserve the semantics of the programming language. Once an algorithm is converted into a hardware specification, strict Boolean equivalence is preserved as it is refined across the levels of design abstraction. Approximate computing is about realizing that this rigid notion of correctness is unnecessary and imposes significant inefficiency.

It bears mentioning that in a few application domains, the spirit of approximate computing has permeated down to some (but not all) layers of the stack. For example, coefficient and bit-width optimization are well-known techniques used in the design of efficient digital filters [14, 15]. In the field of real-time systems, imprecise tasks, whose computation times can be reduced when needed at the cost of lower quality of results, have been proposed to improve the ability to meet deadlines [16]. Even in these notable cases, several layers of the stack are still designed to preserve strict correctness or equivalence.

Over the past decade, approximate computing has attracted the attention of researchers from many different communities, including programming languages, compilers, architecture, circuits, and design automation [17–23]. Our objective here is not to perform a comprehensive review of the approximate computing literature. Instead, we present an overview of our efforts to establish an integrated framework for the design of approximate computing systems, and the techniques that we have developed at various layers of the computing stack.

3. KEY PRINCIPLES OF APPROXIMATE COMPUTING

We believe the design of approximate computing platforms should be guided by the following principles:

- *Measurable notion of quality.* Since both intrinsic resilience and approximate computing arise from the notion of acceptable quality of results, it is important to have a clear, measurable definition of what constitutes acceptable quality. In addition, it is critical to develop methods to ensure that acceptable quality is maintained when approximate computing techniques are used. Broadly speaking, quality specification and verification remains an open challenge. It is important to note that quality metrics do vary across applications (recognition or classification accuracy, relevance of search results, visual quality of images or video, *etc.*). However, the abstractions and methodology used to specify and validate quality should still be general, and to the extent possible re-use tools and concepts from functional verification.

- *Significance-driven.* Not all computations in an application — even the most forgiving ones — may be subject to approximations. Computations that involve pointer arithmetic or affect control flow may lead to catastrophic effects when approximated. Even among the computations that may be approximated, the ex-

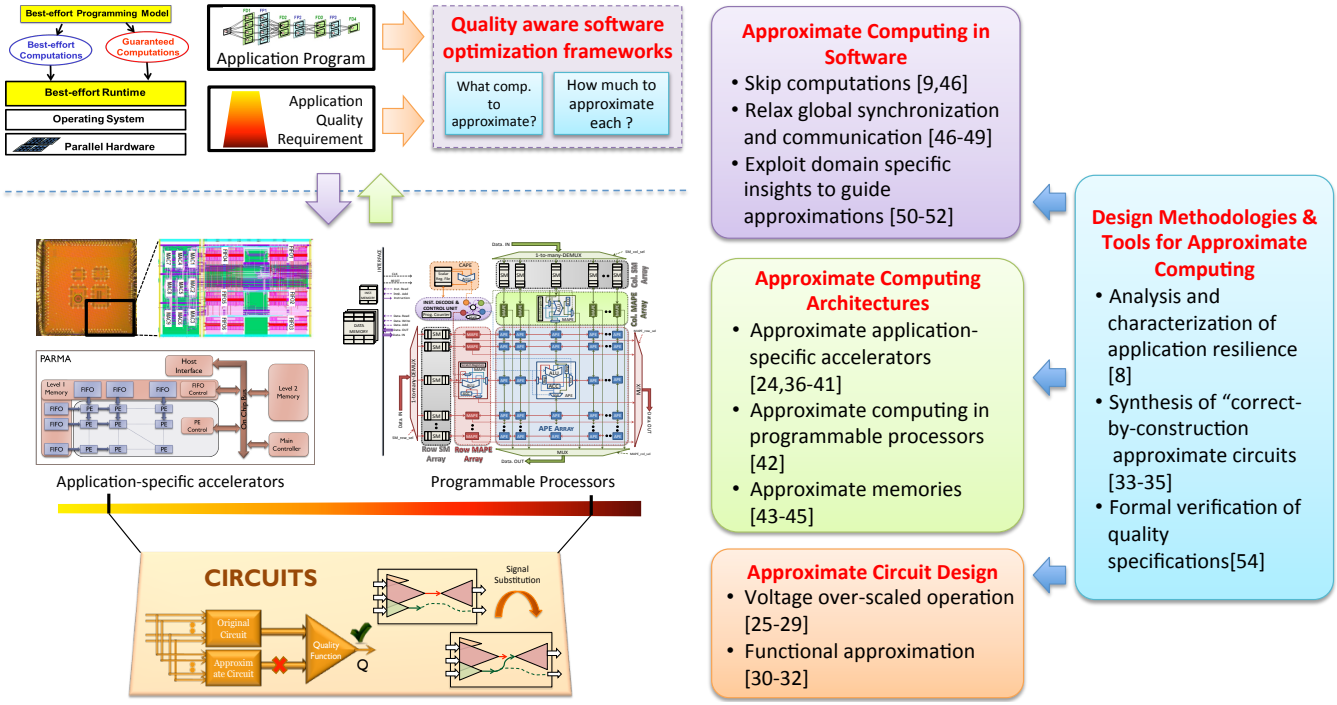

Figure 3: An integrated framework for approximate computing and representative approximate computing techniques at various levels of the computing stack

tent to which they impact application quality when approximated varies greatly. Therefore, it is important to adopt a "significance-driven" approach, *i.e.*, separate resilient and sensitive computations, and approximate resilient computations based on how significantly they impact quality.

- *Disproportionate benefits.* Approximate computing should result in disproportionate benefits, *i.e.*, large improvements in efficiency for little to no impact on quality. To achieve this, it is often beneficial to target bottleneck operations such as global synchronization and communication in software, or critical paths in hardware, for approximation. We note that these sources of disproportionate benefit are inherently spread across various layers of the computing stack. Therefore, a cross-layer approach to approximate computing is more likely to yield a superior overall tradeoff.

- *Quality-configurable.* Resilience to approximations is not a static property of an application, and may depend on both the input data processed and the context in which the outputs are used. For example, a machine learning algorithm when used in a health-critical medical diagnosis application may have much more stringent quality constraints than when used in a product recommendation system. Similarly, resilience often manifests at scale, *i.e.*, when the input data set is large and more likely to contain redundancy. Since hardware or software components are often re-used across applications, approximate computing techniques need to be quality-configurable so that they can be modulated according to the available opportunity. Therefore, the objective should be to design approximate computing platforms that provide the best quality *vs.* efficiency tradeoff across a range of output qualities, rather optimize them for a fixed quality.

4. APPROXIMATE COMPUTING: AN INTEGRATED FRAMEWORK

To realize the vision outlined in the previous sections, we present an integrated framework for approximate computing. As shown in Figure 3, the proposed framework comprises of three key components, which collectively involve developing approximate computing techniques at various layers of the computing stack. First, at the circuit level, we develop techniques to design approximate circuits that are highly efficient in performance, area and power. To enable generality and scalability, we embody the fundamental approximate circuit design principles into synthesis frameworks that can automatically generate "correct-by-construction" approximate versions for any given circuit, and any desired accuracy bound specified at the circuit outputs. Next, at the architecture level, we utilize approximate circuits to build approximate computing architectures that yield a favorable trade-off between efficiency and application output quality. Towards this end, we consider a spectrum of architectural choices from application-specific accelerators to programmable processors, and investigate how approximate computing can be best leveraged in their context. Finally, at the software level, we develop methodologies to systematically identify resilient computations within an application and map them to approximate computing platforms. We further investigate software approximation techniques that enable faster run-times for programs by skipping computations as well as improving parallel scalability by eliminating bottlenecks such as

global synchronization and communication. We next provide a brief overview of representative approximate computing techniques at each layer of the computing stack to illustrate our framework.

4.1 Approximate Circuits

Approximate circuits are the basic hardware building blocks of approximate computing platforms. Approximate computing can be achieved at the circuit level using two broad approaches. The first approach is to design circuits that operate under overscaled conditions (timing/voltage), leading to timing-induced errors. The key challenge in this approach is to mitigate the natural tendency of overscaling to impact the most significant bits, resulting in an unacceptable loss in quality [24]. Techniques to alleviate the impact of timing errors and obtain a more graceful degradation in quality were proposed in [25–29]. Alternatively, functionally approximate circuits can be designed that deviate from the golden specification but contain fewer transistors or gates [30–35]. Functionally approximate circuits have lower hardware complexity (switched capacitance, leakage, and critical path) compared to accurate implementations while evaluating the required function within a desired quality or error bound.

For instance, the number of transistors and internal node capacitances in a conventional mirror adder can be substantially reduced, resulting in an approximate full adder whose truth table deviates from the correct one for a few inputs [32]. Such approximate building blocks can be selectively utilized to construct energy efficient hardware implementations of various applications [32, 35].

4.2 Approximate Computing Architectures

At the architecture level, the goal is to utilize approximate circuits to design computing architectures that provide a favorable trade-off between efficiency and quality. Architectural design techniques for approximate computing can be investigated in the context of a spectrum of architectures, ranging from application-specific accelerators to fully programmable processors. In the case of specialized hardware accelerators, significance-driven computation [36] and scalable effort hardware design [24, 37–41] can lead to highly energy efficient implementations. The key principles are to ensure that computations are approximated based on their significance in determining output quality, and to design hardware to be scalable-effort, *i.e.*, to expose knobs that regulate quality-efficiency trade-offs, so that the hardware can be operated at any desired quality point. Measurements from an approximate recognition and mining accelerator chip implemented in 65nm CMOS technology yielded 2X-20X improvements in energy for very little reduction in output quality [41].

In the domain of programmable processors, approximate computing may be realized through quality programmable processors, wherein the notion of quality is codified into the natural hardware/software interface, *viz.* the instruction set [42]. Software expresses its resilience to approximations through instruction fields that specify expectations on the accuracy of the instruction results. Hardware guarantees that the instruction-level quality specifications are met, while leveraging the flexibility that they engender to derive energy savings. As a first embodiment, a quality-programmable vector processor was designed that demonstrated significant energy improvements for applications from the recognition, mining, and search domains [42].

Approximate computing can also be leveraged to improve the energy consumption of memories [43–45]. For example, a hybrid memory array composed of 8T SRAM cells for storing sensitive data *vs.* 6T SRAM cells for storing resilient ones, allows for more aggresive voltage scaling, resulting in over 32% power savings in the context of video applications [43]. In the context of spintronic memories, various mechanisms at the bit-cell level can be used to create tradeoffs between energy and the probability of errors during read and write operations [45]. These mechanisms can in turn be utilized to design quality-configurable arrays that offer the ability to modulate both the probability and magnitude of errors in return for energy savings [45].

4.3 Approximate Computing in Software

Approximate computing can reduce the run-time of programs by selectively skipping computations [9] or by relaxing synchronization and reducing communication between threads to enable better parallel scalability on multi-cores, GPUs, and clusters [46–49]. A key challenge in approximate computing is the identification of computations for approximation that have lower impact on the quality of results. One approach to address this challenge is to provide the programmer with abstractions to naturally specify resilient or less significant computations [46, 47, 49]. Alternatively, a profiling-based framework [8] can automatically identify resilient computations and quantitatively evaluate how approximations to these computations impact the application output.

Domain-specific insights can be utilized to further improve the efficacy of approximate computing techniques [50–52]. For example, large-scale neural networks (also called deep learning networks) have become popular due to their state-of-the-art performance on a wide range of machine learning problems [53]. One of the key challenges with deep learning networks is their high computational complexity. Approximate computing can be used to improve the efficiency of deep learning networks [50] by adapting backpropagation to identify neurons that contribute less significantly to the network's accuracy, approximating these neurons (*e.g.*, by using lower precision or simpler activation functions), and incrementally re-training the network to mitigate the impact of approximations on output quality.

4.4 Methodology and Tools

A key requirement for the adoption of approximate computing in practice is that it should not require a significant increase in design effort. This mandates the development of clean abstractions and supporting design methodologies and tools. As mentioned above, profiling tools can determine parts of an application that are amenable to approximate computing, and identify specific approximate computing techniques that are most suitable to these parts [8]. Formal verification tools can be extended to verify quality properties or approximate equivalence of hardware and software implementations [54]. Finally, synthesis tools can generate "correct by construction" approximate circuits from arbitrary specifications and quality constraints [33–35].

4.5 Cross-layer optimization

Fully exploiting the potential of approximate computing requires the use of a cross-layer approach, wherein approxi-

mate computing techniques are employed at multiple levels of design abstraction. For instance, in the context of a hardware accelerator for recognition and mining, an additional 1.4X-2X energy savings is achieved when approximations are introduced together across the algorithm, architecture and circuit levels, as compared to the case where approximations are restricted to any single level [24].

5. CONCLUSION

The gap created by diminishing benefits from technology scaling on the one hand, and projected growth in computing demand on the other, has led to a quest for new sources of efficiency in computing. Approximate computing offers promise in this regard as a new dimension to optimize computing platforms. While many advances have been made in the area of approximate computing, much remains to be explored and many challenges need to be addressed. These include the development of a methodology to specify and validate quality, the integration of techniques proposed at various layers of the stack into a cohesive, cross-layer framework, extending approximate computing beyond the processor sub-system into storage and I/O, and an evaluation of the benefits of approximate computing in real end systems. Addressing these challenges is critical to the eventual adoption of approximate computing.

Acknowledgment. We thank our collaborators whose insights helped shape our views on approximate computing, and who have contributed significantly to the body of work summarized here: Nilanjan Banerjee, Vinay Chippa, Reza Farivar, Vaibhav Gupta, Georgios Karakonstatis, Younghoon Kim, Debabrata Mohapatra, Jiayuan Meng, Jongsun Park, Ashish Ranjan, and Rangharajan Venkatesan.

6. REFERENCES

[1] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. New trends in dark silicon. In *Proc. DAC*, 2015.

[2] Yuan Xie, Hsiangyun Cheng, Jia Zhan, Jishen Zhao, Jack Sampson, and Mary Jane Irwin. Core vs. uncore: Which part of silicon is darker? In *Proc. DAC*, 2015.

[3] R. H. Dennard et. al. Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[4] M. Bohr. A 30 year retrospective on Dennard's MOSFET scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, Winter 2007.

[5] H. Esmaeilzadeh et. al. Dark silicon and the end of multicore scaling. In *Proc. ISCA*, pages 365–376, 2011.

[6] M. Hill and C. Kozyrakis. Advancing computer systems without technology progress. In *Outbrief of DARPA/ISAT Workshop (http://www.cs.wisc.edu/~markhill/papers/isat2012_ACSWTP.pdf)*, March 2012.

[7] Y. K. Chen et. al. Convergence of recognition, mining, and synthesis workloads and its implications. *Proc. IEEE*, 96(5):790–807, May 2008.

[8] V. K. Chippa et. al. Analysis and characterization of inherent application resilience for approximate computing. In *Proc. DAC*, 2013.

[9] S. Chakradhar et. al. Best-effort computing: Re-thinking parallel software and hardware. In *Proc. DAC*, 2010.

[10] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

[11] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[12] L. L. Peterson and B. S. Davie. *Computer Networks, Fifth Edition: A Systems Approach*. Morgan Kaufmann Publishers Inc., 2011.

[13] W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, January 2009.

[14] B. Liu. Effect of finite word length on the accuracy of digital filters–a review. *IEEE Transactions on Circuit Theory*, 18(6):670–677, Nov 1971.

[15] V. Madisetti. *Digital Signal Processing Handbook 2nd Edition*. CRC Press, 2008.

[16] J. W-S. Liu et. al. Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94, Jan 1994.

[17] S. H. Nawab et. al. Approximate signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 15(1-2):177–200, 1997.

[18] R. Hegde et. al. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. ISLPED*, pages 30–35, 1999.

[19] K. Palem et. al. Ten years of building broken chips: The physics and engineering of inexact computing. *ACM Trans. on Embedded Computing Systems*, 12(2s):1–23, 2013.

[20] H. Esmaeilzadeh et. al. Architecture support for disciplined approximate programming. In *Proc. ASPLOS*, pages 301–312, 2012.

[21] S. Sidiroglou-Douskos et. al. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. ACM SIGSOFT symposium*, pages 124–134, 2011.

[22] S. Narayanan et. al. Scalable stochastic processors. In *Proc. DATE*, pages 335–338, 2010.

[23] J. Han et. al. Approximate computing: An emerging paradigm for energy-efficient design. In *Proc. ETS*, pages 1–6, 2013.

[24] V. K. Chippa et. al. Scalable effort hardware design. *IEEE Trans. on VLSI Systems*, pages 2004–2016, Sept 2014.

[25] N. Banerjee et. al. Process variation tolerant low power DCT architecture. In *Proc. DATE*, pages 1–6, April 2007.

[26] N. Banerjee et. al. A process variation aware low power synthesis methodology for fixed-point FIR filters. In *Proc. ISLPED*, pages 147–152, 2007.

[27] G. Karakonstantis et. al. Design methodology to trade off power, output quality and error resiliency: Application to color interpolation filtering. In *Proc. ICCAD*, pages 199–204, 2007.

[28] D. Mohapatra et. al. Design of voltage-scalable meta-functions for approximate computing. In *Proc. DATE*, 2011.

[29] S. Ramasubramanian et. al. Relax-and-retime: A methodology for energy-efficient recovery based design. In *Proc. DAC*, 2013.

[30] Jong-Sun Park. *Low Complexity Digital Signal Processing System Design Techniques*. PhD thesis, West Lafayette, IN, USA, 2005. AAI3198154.

[31] G. Karakonstantis and K. Roy. An optimal algorithm for low power multiplierless fir filter design using Chebychev criterion. In *Proc. ICASSP*, volume 2, pages II–49–II–52, April 2007.

[32] V. Gupta et. al. IMPACT: imprecise adders for low-power approximate computing. In *Proc. ISLPED*, pages 409–414, 2011.

[33] S. Venkataramani et. al. SALSA: systematic logic synthesis of approximate circuits. In *Proc. DAC*, 2012.

[34] S. Venkataramani et. al. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Proc. DATE*, pages 1367–1372, 2013.

[35] A. Ranjan et. al. ASLAN: synthesis of approximate sequential circuits. In *Proc. DATE*, 2014.

[36] D. Mohapatra et. al. Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator. In *Proc. ISLPED*, pages 195–200, 2009.

[37] V. K. Chippa et. al. Approximate computing: An integrated hardware approach. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, pages 111–117, 2013.

[38] V. K. Chippa et. al. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Proc. DAC*, pages 555 –560, 2010.

[39] V. K. Chippa et. al. Dynamic effort scaling: Managing the quality-efficiency tradeoff. In *Proc. DAC*, pages 603–608, 2011.

[40] V. K. Chippa, S. Venkataramani, K. Roy, and A. Raghunathan. StoRM: A stochastic recognition and mining processor. In *Proc. ISLPED*, pages 39–44, 2014.

[41] V. K. Chippa et. al. Energy-efficient recognition and mining processor using scalable effort design. In *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, pages 1–4, Sept 2013.

[42] S. Venkataramani et. al. Quality programmable vector processors for approximate computing. In *Proc. MICRO*, 2013.

[43] Ik Joon Chang, D. Mohapatra, and K. Roy. A priority-based 6t/8t hybrid sram architecture for aggressive voltage scaling in video applications. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(2):101–112, Feb 2011.

[44] Georgios Karakonstantis, Debabrata Mohapatra, and Kaushik

Roy. Logic and memory design based on unequal error protection for voltage-scalable, robust and adaptive dsp systems. *Journal of Signal Processing Systems*, 68(3):415–431, 2012.

[45] A. Ranjan et. al. Approximate storage for energy efficient spintronic memories. In *Proc. DAC*, 2015.

[46] J. Meng et. al. Best-effort parallel execution framework for recognition and mining applications. In *Proc. IPDPS*, 2009.

[47] J. Meng et. al. Exploiting the forgiving nature of applications for scalable parallel execution. In *Proc. IPDPS*, April 2010.

[48] Surendra Byna, Jiayuan Meng, Anand Raghunathan, Srimat Chakradhar, and Srihari Cadambi. Best-effort semantic document search on gpus. In *Proc. GPGPU Workshop*, pages 86–93, 2010.

[49] R. Farivar et. al. PIC: Partitioned iterative convergence for clusters. In *Proc. CLUSTER*, pages 391–401, Sept 2012.

[50] S. Venkataramani et. al. AxNN: energy-efficient neuromorphic systems using approximate computing. In *Proc. ISLPED*, pages 27–32, 2014.

[51] A. Raha et. al. Quality configurable reduce-and-rank kernals for energy-efficient approximate computing. In *Proc. DATE*, March 2015.

[52] S. Venkataramani et. al. Scalable-effort classifiers for energy-efficient machine learning. In *Proc. DAC*, June 2015.

[53] J. Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[54] R. Venkatesan et. al. MACACO: modeling and analysis of circuits for approximate computing. In *Proc. ICCAD*, pages 667–673, 2011.