Computing Approximately, and Efficiently

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

***Abstract*—Recent years have witnessed significant interest in the area of approximate computing. Much of this interest stems from the quest for new sources of computing efficiency in the face of diminishing benefits from technology scaling. We argue that trends in computing workloads will greatly increase the opportunities for approximate computing, describe the vision and key principles that have guided our work in this area, and outline a range of approximate computing techniques that we have developed at all layers of the computing stack, spanning circuits, architecture, and software.**

I. Introduction

The gap created due to diminishing benefits from technology scaling on the one hand, and projected growth in computing demand from future workloads on the other, leads to a need for new sources of computing efficiency. Increases in clock frequency and reductions in voltage with each technology generation have greatly slowed with the end of Dennard scaling [1], [2], giving birth to the era of multi-cores or performance through parallelism. However, this new scaling paradigm is not without its own challenges – most applications demonstrate saturating, if not diminishing, performance with increasing core counts, due to well-known bottlenecks such as serial computations, synchronization, global communication and off-chip memory bandwidth [3], [4]. Moreover, the inability to fit more cores within constrained power budgets threatens even this new scaling paradigm [5]. These challenges are pushing designers to look for alternate avenues to improve the capabilities of computing platforms [6].

Concurrent with the above developments, the nature of computing workloads has also changed profoundly and fundamentally across the computing spectrum. In data centers and the cloud, the demand for computing is driven by the need to organize, search through, analyze, and draw inferences from, exploding amounts of digital data [7]. In mobile devices and deeply embedded systems, the creation and consumption of richer media and the need to interact more naturally and intelligently with users and the environment are trends that drive much of the computing demand. There is a common pattern that emerges from both ends of 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 produce an acceptable user experience.

As a result, contemporary computing workloads exhibit significant *intrinsic application resilience*, or the ability to produce acceptable outputs despite some of their computations being performed in an approximate manner [8]. This resilience to approximations stems collectively from the following characteristics [9]: (i) The notion of a unique, golden result simply does not exist; instead, a range of answers are acceptable (*e.g.*, search, recommendation systems), (ii) Even the best algorithms fall well short of perfection, and users are conditioned to accept good-enough results (most recognition problems), (iii) Algorithms are designed to deal with significant noise in their input data, which propagates through their internal computations; this naturally endows them with a resilience to approximations in the computations themselves, and (iv) The use of computation patterns (such as aggregation or iterative-refinement) that lead to the effects of approximations being self-healed or attenuated. 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 resilience in a broad context.

This work was supported in part by the National Science Foundation under grants CNS-1423290, and CNS-1018621.

II. Approximate Computing

Approximate computing exploits the forgiving nature (intrinsic resilience) of applications to realize improvements in performance or energy efficiency at all layers of the computing stack.

It is important to recognize that in a broad sense, the roots of approximate computing can be traced back to several disciplines. Algorithm designers and software developers routinely make trade-offs between optimality of results and computational requirements through approximation, probabilistic, and heuristic algorithms [10], [11]. Image, audio and video compression algorithms exploit the fact that minor variations in content are imperceptible to users, and use this to greatly reduce storage and bandwidth requirements. In networking, the notion of best-effort packet delivery is embraced by protocols such as IP and UDP, which many applications use, eschewing the guaranteed packet delivery of protocols such as TCP [12]. Relaxed consistency models have arguably enabled much-improved scalability in modern unstructured databases [13]. These comparisons support the acceptability of approximate computing, but also raise a question. Are there opportunities beyond the examples listed above, *i.e.*, can the key principles be captured and applied to other aspects of computing system design?

To answer this question, we note that all of these techniques are invariably limited to the very top layers of the computing

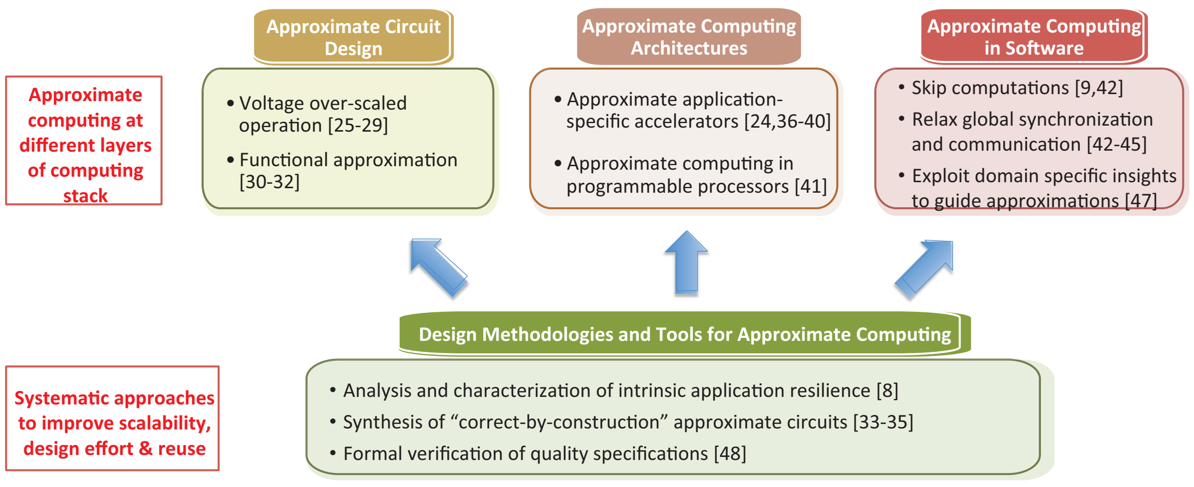
Approximate computing at different layers of computing stack
Approximate Circuit Design
• Voltage over-scaled operation [25-29]
• Functional approximation [30-32]
Approximate Computing Architectures
• Approximate application-specific accelerators [24,36-40]
• Approximate computing in programmable processors [41]
Approximate Computing in Software
• Skip computations [9,42]
• Relax global synchronization and communication [42-45]
• Exploit domain specific insights to guide approximations [47]
Systematic approaches to improve scalability, design effort & reuse
Design Methodologies and Tools for Approximate Computing
• Analysis and characterization of intrinsic application resilience [8]
• Synthesis of "correct-by-construction" approximate circuits [33-35]
• Formal verification of quality specifications [48]

Fig. 1. Overview of representative approximate computing techniques at various levels of the computing stack

stack. Most of the layers – programming languages, compilers, runtimes, operating systems, architecture, and circuits – are designed to adhere to the dogma that every computation must be executed with the same strict notion of correctness. 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.

There are a few specific application domains where the spirit of approximate computing has permeated down to some (but not all) layers of the stack. For example, coefficient and bitwidth optimization are commonly used in the design of digital filters and signal processing systems [14], [15] to reduce computational requirements. Imprecise tasks, whose computation times can be reduced when needed at the cost of lower quality of results, have been proposed to provide scheduling flexibility in real-time systems [16]. However, even in these cases, several layers of the stack are still designed to preserve strict correctness or equivalence.

In summary, 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 rigid and strict notion of correctness. Approximate computing is about realizing that this is not only unnecessary but also imposes significant inefficiency.

Over the past decade, researchers have come to recognize that approximate computing can benefit a broader range of application domains, and be applied at all layers of the computing stack. Consequently, approximate computing has evolved into a vibrant research area that 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. In the rest of the paper, we present an overview of our efforts to establish a comprehensive framework for the design of approximate computing systems, and the techniques that we have developed at various layers of the computing stack.

III. Key Principles of Approximate Computing

We first outline the key principles that have guided our research in approximate computing.

- First and foremost, intrinsic resilience does not mean that *any* result is acceptable. Therefore, it is important to have a clear definition of what constitutes acceptable quality of results, and methods to ensure that it is maintained when approximate computing techniques are used. In this context, it is important to note that, while quality metrics do vary across applications (classification accuracy, clustering quality, visual quality of images or video, relevance of search results, *etc.*), the abstractions and methodology used to specify and validate quality may still be general.
- Most applications demonstrate a large variance in the tolerance of their constituent computations to approximations. Some computations such as those that involve pointer arithmetic or affect control flow may not be approximated without catastrophic effects like exceptions or program crashes. Even among the computations that may be approximated, the extent to which they can be approximated, or the benefits that can accrue, vary greatly. Therefore, it is necessary to distinguish between sensitive and resilient computations, or between more significant and less significant computations, and be selective in the approximating them.
- Approximate computing should yield disproportionate benefits, or large improvements in performance or energy with little or no impact on output quality. Therefore, bottleneck operations such as global synchronization and communication in software, or critical paths in hardware, are often effective targets for approximation. These sources of disproportionate benefit may come from var-

ious layers of the computing stack, hence a cross-layer approach to approximate computing is desirable.

- Resilience to approximations depends on the application's context and the data processed. For example, a machine learning algorithm used in a health-critical medical diagnosis application may have much more stringent quality constraints than the same algorithm used in a product recommendation system. Since hardware or software implementations are often re-used across different application contexts, any approximations used need to scalable, so that they can be modulated according to the opportunity. From a different perspective, the goal of approximate computing should be to design systems that provide a favorable quality *vs.* performance or energy tradeoff, rather than design systems that are optimized for a fixed quality.

IV. Approximate Computing Techniques

We next provide a brief overview of representative approximate computing techniques that we have developed at various layers of the computing stack (Figure 1).

Circuits. 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]. 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].

Architecture. At the architecture level, significance-driven computation [36] and scalable effort hardware design [24], [37]–[40] can lead to highly energy efficient and specialized hardware 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. 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 [41]. 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 [41].

Software. 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 [42]–[45]. Identifying computations for approximation that have lower impact on the quality of results is a key challenge in approximate computing. One approach to address this challenge is to use programming templates that enable the programmer to naturally specify resilient or less significant computations [42], [43], [45]. Alternatively, a profiling-based resilience characterization framework [8] can automatically identify resilient computations, quantitatively characterize how approximations to these computations impact the application output, and assess the potential of various approximate computing techniques.

Domain-specific insights can be utilized to further improve the efficacy of approximate computing techniques. 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 [46]. 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 [47] 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.

Methodology and tools. To be adopted in practice, approximate computing 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 with in-built models of approximate computing techniques 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 [48]. Finally, synthesis tools have been developed that can generate "correct by construction" approximate circuits from arbitrary (RTL) specifications and quality constraints [33]–[35].

Cross-layer optimization. It has been shown that, in order to fully exploit the potential of approximate computing, it is desirable to adopt a cross-layer approach wherein approximate 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].

V. Conclusion

The intrinsic resilience exhibited by applications from many prevalent domains has led to widespread interest in approximate computing as a new approach to optimize computing platforms. Although considerable ground has been covered in the area of approximate computing, much yet remains to be explored and many challenges need to be addressed. A consistent methodology for specification and validation of quality across layers of the stack is essential. The techniques proposed at various levels of abstraction need to be synthesized into a cohesive, cross-layer framework. Achieving these objectives without unacceptably increasing design effort or verification complexity is critical to the broader 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, Debabrata Mohapatra, Jiayuan Meng, Jongsun Park, Ashish Ranjan, and Rangharajan Venkatesan.

References

[1] 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.

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

[3] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. Spring Joint Computer Conference*, pages 483–485, 1967.

[4] D. E. Culler, A. Gupta, and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., 1st edition, 1997.

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

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

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

[44] Surendra Byna, Jiayuan Meng, Anand Raghunathan, Srimat Chakradhar, and Srihari Cadambi. Best-effort semantic document search on gpus. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 86–93, New York, NY, USA, 2010. ACM.

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

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

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

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