Approximate Computing

An Annotated Bibliography

This is an annotated bibliography on the topic of *approximate computing*. It's a living document meant to exhaustively catalog everything we know about approximation along with the earlier work that influenced it. It's also a collaborative, open-source project: to contribute, see its home on GitHub.

Contents

1. Overview

Here's the definition of *approximate computing* that this document uses:

> Approximate computing is the idea that computer systems can let applications trade off *accuracy* for *efficiency*. It includes any technique where the system intentionally exposes *incorrectness* to the application layer in return for conserving some resource.

That definition is clearly broad enough to include many ideas that have existed since the beginning of (computational) time. Floating-point numbers, for example, approximate real-number arithmetic to save space and time over arbitrary-precision numerical representation. This document focuses on the

study of approximate computing *in general* and system-level techniques that apply this theory to create new trade-offs.

There are two main research directions in approximate computing, corresponding to the two main sections in this annotated bibliography. This first is on *techniques* for approximation: specific strategies for exploiting resilience in applications for efficiency gains. The second is on *programming* approximate systems: assuming that approximation techniques exist, a host of new programmability problems arise.

2. Approximation Techniques

This section enumerates *techniques* for implementing approximation. There are three main categories: approximation in *computer architecture* (i.e., computation and storage hardware), approximation in *software* via program transformation, and approximation elsewhere (e.g., networks).

2.1. In Architecture

This section deals with hardware-oriented approximation techniques. We categorize the techniques according to the hardware component they affect.

2.1.1. Functional Units

One straightforward strategy for approximation in floating-point units is to dynamically adapt mantissa width [133, 149].

A paper by Alvarez et al. proposes *fuzzy memoization* for FPUs [4]. The idea is to store previously-computed results, as in ordinary memoization, but also to provide a "match" even when inputs are merely *close* to other, previously-seen inputs. (Fuzzy memoization comes up elsewhere in approximate computing too.)

To facilitate *voltage overscaling* techniques for approximation, some work designs functional units that are more resilient to timing errors than traditional, precise designs [49]. Other work extends this graceful voltage–error scaling to coarser computational blocks [91].

TODO. The above paragraph must be missing other work on voltage-overscaling-tolerant units.

Alternative number representations work in tandem with relaxed functional units to bound the numerical error that can result from bit flips [125].

A body of VLSI work has designed approximate adders, which are allowed to yield incorrect results for some minority of input combinations [45, 46, 54, 57, 84, 118, 140, 143, 148, 154]. Liang et al. propose metrics for evaluating these adders [72].

TODO. Should we break down the adder work into finer categories? Also, there is now more work multipliers that deserves its own paragraph

2.1.2. Memory

Several categories of work apply approximation to memory technologies. The general idea is to spend less energy on retaining or accessing data; in return, there is a small probability that bits will flip in the memory.

SRAM structures spend significant static power on retaining data, so they represent another opportunity for fidelity trade-offs [21, 63, 120].

Similarly, DRAM structures can reduce the power spent on refresh cycles where bit flips are allowed [74, 77].

In persistent memories where storage cells can wear out, approximate systems can reduce the number of bits they flip to lengthen the useful device lifetime [41]. Similarly, low-power writes to memories like flash can exploit its probabilistic properties while hiding them from software [73, 108, 134].

TODO. [112]

Spintronic memories exhibit similarly favorable trade-offs between access cost and error [101].

2.1.3. Circuit Design

A broad category of work has proposed general techniques for making quality trade-offs when synthesizing and optimizing general hardware circuits [9, 12, 83, 97, 100, 136, 137, 147]. Other tools focus on analyzing approximate circuit designs [135, 139].

Near-threshold voltage domains also present a new opportunity for embracing unpredictable circuit operation [56].

Kahng et al. propose to place and route processor designs with paths that do not exhibit a "cliff" where voltage scaling causes catastrophic failures [53]. The original idea there was for so-called better-than-worst-case (BTWC) designs such as Razor [35], not approximate computing, but the connection to voltage-overscaling architectures such as Truffle [37] is clear.

TODO. Need more citations on voltage overscaling here.

2.1.4. Relaxed Fault Tolerance

As a dual to adding errors in some circuits, some researchers have explored differential fault protection in the face of universally unreliable circuits. As process sizes continue to shrink, it is likely that reliable transistors will become the minority; redundancy and checking will be necessary to provide reliable operation [68]. Circuit design techniques have been proposed that reduce the cost of redundancy by providing it selectively for certain instructions in a CPU [128], certain blocks in a DSP [5, 47, 55], or to components of a GPU [95]. Other work has used criticality information to selectively allocate software-level error detection and correction resources [32, 59, 119].

2.1.5. Microarchitecture

Microarchitectural mechanisms can exploit different opportunities from circuit-level techniques. Specifically, "soft coherence" relaxes intercore communication [76], and load value approximation [85, 132] approximates numerical values instead of fetching them from main memory on cache misses.

Recent work has proposed system organizations that apply approximation at a coarser grain. One set of techniques uses external monitoring to allow errors even in processor control logic [151, 152]. Other approaches compose separate processing units with different levels of reliability [65]. Duwe [33] proposes runtime coalescing of approximate and precise computations to reduce the overhead of switching between modes. Other work allocates approximation among the lanes of a SIMD unit [1]. In all cases, the gains from approximation can be larger than for lower-level techniques that affect individual operations. As the granularity principle from outlines, techniques like these that approximate entire computations, including control flow, have the greatest efficiency potential.

2.1.6. Stochastic Computing

Stochastic computing is an alternative computational model where values are represented using probabilities [8, 20, 27, 79, 93, 94, 141]. For example, a wire could carry a random sequence of bits, where the wire's value corresponds to the probability that a given bit is a 1. Multiplication can be implemented in this model using a single *and* gate, so simple circuits can be low-power and area-efficient. A persistent challenge in stochastic circuits, however, is that reading and output value requires a number of bits that is exponential in the value's magnitude. Relaxing this constraint represents an opportunity for an time–accuracy trade-off.

2.2. In Software

Aside from hardware-level accuracy trade-offs, there are opportunities for adapting *algorithms* to execute with varying precision. Algorithmic quality–complexity trade-offs are not new, but recent work has proposed tools for *automatically* transforming programs to take advantage of them. Transformations include removing portions of a program's dynamic execution (termed *code perforation*) [121], unsound parallelization of serial programs [86], eliminating synchronization in parallel programs [82, 89, 102, 103], identifying and adjusting parameters that control output quality Hoffmann et al. [51], randomizing deterministic programs [88, 155], dynamically choosing between different programmer-provided implementations of the same specification [6, 7, 10, 39, 138, 142], and replacing subcomputations with invocations of a trained neural network [36].

Some work on algorithmic approximation targets specific hardware: notably, general-purpose GPUs [44, 109, 110, 114]. In a GPU setting, approximation strategies benefit most by optimizing for memory bandwidth and control divergence.

Recently, a research direction has developed in *automated program repair* and other approaches to heuristically patching software according to programmer-specified criteria. These techniques are typically approximate in that they abandon a traditional compiler's goal of perfectly preserving the original program's semantics. Notably, Schulte et al. [116] propose to use program evolution to optimize for energy.

Precimonious [107] addresses the problem of choosing appropriate floating-point widths, which amount to a trade-off between numerical accuracy and space or operation cost. Similarly, STOKE's floating-point extension [115] synthesizes new versions of floating-point functions from scratch to meet different accuracy requirements with optimal efficiency.

Neural acceleration is a recent technique that treats code as a black box and transforms it into a neural network [25, 36, 80, 129]. It is, at its core, an algorithmic transformation, but it integrates tightly with hardware support: a digital accelerator Esmaeilzadeh et al. [36], analog circuits [124], FPGAs [92], GPUs [44], or, recently, new analog substrates using resistive memory [67] or memristors [75].

2.3. In Other Systems

While architecture optimizations and program transformations dominate the field of proposed exploitations of approximate software, some recent work has explored the same trade-off in other components of computer systems.

Network communication, with its reliance on imperfect underlying channels, exhibits opportunities for fidelity trade-offs [52, 78, 117, 126]. Notably, SoftCast [52] transmits images and video by making the signal magnitude directly proportional to pixel luminance. BlinkDB, a recent instance of research on *approximate query answering*, is a database system that can respond to queries that include a required accuracy band on their output [2]. Uncertain\<T> [13] and Lax [127] propose to expose the probabilistic behavior of sensors to programs. In a distributed system or a supercomputer, approximation techniques can eschew redundancy and recovery for efficiency [50].

3. Programming with Approximation

This work tends to assume an existing, domain-specific notion of "quality" for each application. As the principle in suggests, these quality metrics need careful consideration: one quality metric is not necessarily just as good as another. Recent work has proposed guidelines for rigorous quality measurement [3].

3.1. Approximate Languages

Recently, language constructs that express and constrain approximation have become a focus in the programming-languages research community. Relax de Kruijf et al. [32] is a language with ISA support for tolerating architectural faults in

software. Rely Carbin et al. [18] uses specifications that relate the reliability of the input to an approximate region of code to its outputs.

A related set of recent approximate-programming tools attempt to *adapt* a program to meet accuracy demands while using as few resources as possible. Chisel Misailovic et al. [90] is an extension to Rely that searches for the subset of operations in a program that can safely be made approximate. ExpAX [38] finds safe-to-approximate operations automatically and uses a metaheuristic to find which subset of them to actually approximate.

Some other programming systems that focus on energy efficiency include approximation ideas: Eon [123] is a language for long-running embedded systems that can drop tasks when energy resources are low, and the Energy Types language [29] incorporates a variety of strategies for expressing energy requirements.

3.2. Programmer Tools

Aside from programming languages, separate programmer tools can help analyze and control the effects of approximation.

A quality-of-service profiler helps programmers identify parts of programs that may be good targets for approximation techniques [87]. Conversely, debugging tools can identify components where approximation is too aggressive [104]. Some verification tools and proof systems help the programmer prove relationships between the original program and a candidate relaxed version [15–17, 144].

As an alternative to statically bounding errors, dynamic techniques can monitor quality degradation at run time. The critical challenge for these techniques is balancing detection accuracy with the added cost, which takes away from the efficiency advantages of approximation. Some work has suggested that programmers can provide domain-specific checks on output quality [43, 104]. Recent work has explored automatic generation of error detectors [58]. A variety of techniques propose mechanisms for run-time or profiling feedback to adapt approximation parameters [7, 10, 51, 153].

3.3. Probabilistic Languages

One specific research direction, *probabilistic programming languages*, focuses on expressing statistical models, especially for machine learning [11, 19, 42, 60, 61, 96, 113, 145]. The goal is to enable efficient statistical inference over arbitrary models written in the probabilistic programming language.

Earlier work examines the semantics of probabilistic behavior in more traditional programming models [62]. Similarly, the probability monad captures a variable's discrete probability distribution in functional programs [99]. Statistical model checking tools can analyze programs to prove statistical properties [64, 66]. Recently, Bornholt et al. [13] proposed a construct for explicitly representing probability distributions in a mainstream programming language.

3.4. Robustness Analysis

As the studies in Section [sec:related:studies] repeatedly find, error tolerance varies greatly in existing software, both within and between programs. Independent of approximate computing, programming-languages researchers have sought to identify and enhance error resilience properties.

SJava analyzes programs to prove that errors only temporarily disrupt the execution path of a program [34]. Program smoothing [22–24] and *robustification* [122] both find continuous, mathematical functions that resemble the input–output behavior of numerical programs. Auto-tuning approaches can help empirically identify error-resilient components [105]. Finally, Cong and Gururaj describe a technique for automatically distinguishing between critical and non-critical instructions for the purpose of selective fault tolerance [30].

3.5. Application Tolerance Studies

This category of proto-approximate-computing work focuses on analyzing applications to measure their resilience to error. These papers typically assume a particular model of error—often hardware-inspired, such as random bit flips in memory—and execute programs under simulation, measuring crashes and output-quality degradation. To measure output quality, these studies typically define a straightforward metric for each application, such as PSNR for media outputs.

TODO. Summarize the papers in the next paragraph.

Three papers by Li and Yeung in 2006–08 [69–71]; other papers that need summaries [26, 48, 81, 106, 130]. A 2009 study in SELSE, de Kruijf and Sankaralingam [31], precedes the authors' later work on software-directed fault recovery [32].

One category of studies focuses on specific application domains. Wong and Horowitz identify resilience specifically in probabilistic-inference applications [146]. Fang et al. [40] address video applications, and Yeh et al. [150] address physical simulation for animation. Other studies have focused on integrated circuit designs rather than software applications [14, 28].

LLFI is a tool based on LLVM for performing this kind of simulation by injecting errors at the compiler-IR level [131].

Some of these studies conclude that there is a useful distinction between critical and non-critical program points, typically instructions [48, 130, 131]. This conclusion is borne out in later work on systems that exploit this distinction [74, 111].

3.6. Security

Recent work in security has exploited patterns in these variability-based errors in DRAM to deanonymize users [98].

References

[1] S. Abdallah, A. Chehab, A. Kayssi, and I.H. Elhajj. TABSH: Tag-based stochastic hardware. In *International Conference on Energy Aware Computing Systems & Applications (ICEAC)*, 2013.

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *ACM European Conference on Computer Systems (EuroSys)*, 2013.

[3] Ismail Akturk, Karen Khatamifard, and Ulya R. Karpuzcu. On quantification of accuracy loss in approximate computing. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2015.

[4] Carlos Alvarez, Jesus Corbal, and Mateo Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54 (7), 2005.

[5] Rajeevan Amirtharajah and Anantha P Chandrakasan. A micropower programmable DSP using approximate signal processing based on distributed arithmetic. *IEEE Journal of Solid-State Circuits*, 39 (2): 337–347, 2004.

[6] Jason Ansel, Cy P. Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman P. Amarasinghe. PetaBricks: a language and compiler for algorithmic choice. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2009.

[7] Jason Ansel, Yee Lok Wong, Cy P. Chan, Marek Olszewski, Alan Edelman, and Saman P. Amarasinghe. Language and compiler support for auto-tuning variable-accuracy algorithms. In *International Symposium on Code Generation and Optimization (CGO)*, 2011.

[8] Gary Anthes. Inexact design: beyond fault-tolerance. *Communications of the ACM*, 56 (4): 18–20, April 2013.

[9] Lingamneni Avinash, Christian C. Enz, Jean-Luc Nagel, Krishna V. Palem, and Christian Piguet. Energy parsimonious circuit design through probabilistic pruning. In *Design, Automation and Test in Europe (DATE)*, 2011.

[10] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2010.

[11] Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio Russo. Deriving probability density functions from probabilistic functional programs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2013.

[12] David Boland and George A. Constantinides. A scalable approach for automated precision analysis. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2012.

[13] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. Uncertain<T>: A first-order type for uncertain data. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[14] Melvin A. Breuer. Multi-media applications and imprecise computation. In *Euromicro Conference on Digital System Design (DSD)*, 2005.

[15] Michael Carbin and Martin Rinard. (Relative) safety properties for relaxed approximate programs. In *Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[16] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. Proving acceptability properties of relaxed nondeterministic approximate programs. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2012.

[17] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. Verified integrity properties for safe approximate program transformations. In *ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM)*, 2013a.

[18] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2013b.

[19] Arun T. Chaganty, Aditya V. Nori, and Sriram K. Rajamani. Efficiently sampling probabilistic programs via program analysis. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

[20] Lakshmi N. Chakrapani, Bilge E. S. Akgul, Suresh Cheemalavagu, Pinar Korkmaz, Krishna V. Palem, and Balasubramanian Seshasayee. Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology. In *Design, Automation and Test in Europe (DATE)*, 2006.

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

[22] Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2010.

[23] Swarat Chaudhuri and Armando Solar-Lezama. Smoothing a program soundly and robustly. In *International Conference on Computer Aided Verification (CAV)*, 2011.

[24] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. Proving programs robust. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 2011.

[25] Tianshi Chen, Yunji Chen, Marc Duranton, Qi Guo, Atif Hashmi, Mikko H. Lipasti, Andrew Nere, Shi Qiu, Michèle Sebag, and Olivier Temam. BenchNN: On the broad potential application scope of hardware neural network accelerators. In *IEEE International Symposium on Workload Characterization (IISWC)*, 2012.

[26] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conference (DAC)*, 2013.

[27] Vinay K. Chippa, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. StoRM: A stochastic recognition and mining processor. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[28] V.K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S.T. Chakradhar. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Design Automation Conference (DAC)*, 2010.

[29] Michael Cohen, Haitao Steve Zhu, Emgin Ezgi Senem, and Yu David Liu. Energy types. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2012.

[30] Jason Cong and Karthik Gururaj. Assuring application-level correctness against soft errors. In *IEEE–ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

[31] M. de Kruijf and K. Sankaralingam. Exploring the synergy of emerging workloads and silicon reliability trends. In *Workshop on Silicon Errors in Logic: System Effects (SELSE)*, 2009.

[32] Marc de Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. Relax: an architectural framework for software recovery of hardware faults. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[33] Henry Duwe. Exploiting application level error resilience via deferred execution. Master's thesis, University of Illinois at Urbana-Champaign, 2013.

[34] Yong hun Eom and Brian Demsky. Self-stabilizing Java. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2012.

[35] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2003.

[36] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012a.

[37] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012b.

[38] Hadi Esmaeilzadeh, Kangqi Ni, and Mayur Naik. Expectation-oriented framework for automating approximate programming. Technical Report GT-CS-13-07, Georgia Institute of Technology, 2013. URL http://hdl.handle.net/1853/49755.

[39] Shuangde Fang, Zidong Du, Yuntan Fang, Yuanjie Huang, Yang Chen, Lieven Eeckhout, Olivier Temam, Huawei Li, Yunji Chen, and Chengyong Wu. Performance portability across heterogeneous SoCs using a generalized library-based approach. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11 (2): 21:1–21:25, June 2014.

[40] Yuntan Fang, Huawei Li, and Xiaowei Li. A fault criticality evaluation framework of digital systems for error tolerant video applications. In *Asian Test Symposium (ATS)*, 2011.

[41] Yuntan Fang, Huawei Li, and Xiaowei Li. SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write. In *Asian Test Symposium (ATS)*, 2012.

[42] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[43] Beayna Grigorian and Glenn Reinman. Dynamically adaptive and reliable approximate computing using light-weight error analysis. In *NASA–ESA Conference On Adaptive Hardware And Systems (AHS)*, 2014a.

[44] Beayna Grigorian and Glenn Reinman. Accelerating divergent applications on SIMD architectures using neural networks. In *IEEE International Conference on Computer Design*, 2014b.

[45] V. Gupta, D. Mohapatra, Sang Phill Park, A. Raghunathan, and K. Roy. IMPACT: Imprecise adders for low-power approximate computing. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2011.

[46] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 32 (1): 124–137, Jan 2013.

[47] Rajamohana Hegde and Naresh R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 1999.

[48] Andreas Heinig, Vincent John Mooney, Florian Schmoll, Peter Marwedel, Krishna V. Palem, and Michael Engel. Classification-based improvement of application robustness and quality of service in probabilistic computer systems. In *International Conference on Architecture of Computing Systems (ARCS)*, 2012.

[49] Caglar Hizli. Energy aware probabilistic arithmetics. Master's thesis, Eindhoven University of Technology, 2013.

[50] Chen-Han Ho, M. de Kruijf, K. Sankaralingam, B. Rountree, M. Schulz, and B.R. De Supinski. Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing. In *IEEE International Conference on Parallel Processing (ICPP)*, 2012.

[51] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin C. Rinard. Dynamic knobs for responsive power-aware computing. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[52] Szymon Jakubczak and Dina Katabi. SoftCast: clean-slate scalable wireless video. In *Workshop on Wireless of the Students, by the Students, for the Students (S3)*, 2010.

[53] A.B. Kahng, Seokhyeong Kang, R. Kumar, and J. Sartori. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2010.

[54] Andrew B. Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC)*, 2012.

[55] 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, September 2012.

[56] Ulya R. Karpuzcu, Ismail Akturk, and Nam Sung Kim. Accordion: Toward soft near-threshold voltage computing. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2014.

[57] Zvi M. Kedem, Vincent J. Mooney, Kirthi Krishna Muntimadugu, and Krishna V. Palem. An approach to energy-error tradeoffs in approximate ripple carry adders. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2011.

[58] Daya S. Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. Rumba: An online quality management system for approximate computing. In *International Symposium on Computer Architecture (ISCA)*, 2015.

[59] Daya Shanker Khudia and Scott Mahlke. Harnessing soft computations for low-budget fault tolerance. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[60] Oleg Kiselyov and Chung-Chieh Shan. Embedded probabilistic programming. In *IFIP Working Conference on Domain-Specific Languages (DSL)*, 2009.

[61] Daphne Koller, David McAllester, and Avi Pfeffer. Effective Bayesian inference for stochastic programs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1997.

[62] D. Kozen. Semantics of probabilistic programs. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 101–114, Oct 1979.

[63] Animesh Kumar, Jan Rabaey, and Kannan Ramchandran. SRAM supply voltage scaling: A reliability perspective. In *International Symposium on Quality Electronic Design (ISQED)*, 2009.

[64] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification (CAV)*, 2011.

[65] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A. Jacobson, and Subhasish Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *Design, Automation and Test in Europe (DATE)*, 2010.

[66] A. Legay and B. Delahaye. Statistical model checking: A brief overview. *Quantitative Models: Expressiveness and Analysis*, 2010.

[67] Boxun Li, Peng Gu, Yi Shan, Yu Wang, Yiran Chen, and Huazhong Yang. RRAM-based analog approximate computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2015.

[68] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[69] Xuanhua Li and Donald Yeung. Exploiting soft computing for increased fault tolerance. In *Workshop on Architectural Support for Gigascale Integration (ASGI)*, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.2997.

[70] Xuanhua Li and Donald Yeung. Application-level correctness and its impact on fault tolerance. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2007. URL http://dx.doi.org/10.1109/HPCA.2007.346196.

[71] Xuanhua Li and Donald Yeung. Exploiting application-level correctness for low-cost fault tolerance. *Journal of Instruction-Level Parallelism*, 2008. URL http://www.jilp.org/vol10/v10paper10.pdf.

[72] Jinghang Liang, Jie Han, and Fabrizio Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 99, 2012.

[73] Ren-Shuo Liu, Chia-Lin Yang, and Wei Wu. Optimizing NAND flash-based SSDs via retention relaxation. In *USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[74] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: Saving refresh-power in mobile devices through critical data partitioning. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[75] Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, and Jianhua Yang. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In *Design Automation Conference (DAC)*, 2015.

[76] G. Long, F. T. Chong, D. Franklin, J. Gilbert, and D. Fan. Soft coherence: Preliminary experiments with error-tolerant memory consistency in numerical applications. In *Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2009.

[77] Jan Lucas, Mauricio Alvarez Mesa, Michael Andersch, and Ben Juurlink. Sparkk: Quality-scalable approximate storage in dram. In *The Memory Forum*, 2014.

[78] Chong Luo, Jun Sun, and Feng Wu. Compressive network coding for approximate sensor data gathering. In *IEEE Global Communications Conference (GLOBECOM)*, 2011.

[79] Vikash K. Mansinghka, Eric M. Jonas, and Joshua B. Tenenbaum. Stochastic digital circuits for probabilistic inference. Technical Report MIT-CSAIL-TR-2008-069, MIT, 2008.

[80] Lawrence McAfee and Kunle Olukotun. EMEURO: A framework for generating multi-purpose accelerators via deep learning. In *International Symposium on Code Generation and Optimization (CGO)*, 2015.

[81] Jiayuan Meng, Srimat Chakradhar, and Anand Raghunathan. Best-effort parallel execution framework for recognition and mining applications. In *IEEE International Parallel & Distributed Processing Symposium*, 2009.

[82] Jiayuan Meng, Anand Raghunathan, Srimat Chakradhar, and Surendra Byna. Exploiting the forgiving nature of applications for scalable parallel execution. In *IEEE International Parallel & Distributed Processing Symposium*, 2010.

[83] Jin Miao. *Modeling and synthesis of approximate digital circuits.* PhD thesis, The University of Texas at Austin, 2014.

[84] Jin Miao, Ku He, Andreas Gerstlauer, and Michael Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *IEEE–ACM International Conference on Computer-Aided Design (ICCAD)*, 2012.

[85] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. Load value approximation. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[86] Sasa Misailovic, Deokhwan Kim, and Martin Rinard. Parallelizing sequential programs with statistical accuracy tests. Technical Report MIT-CSAIL-TR-2010-038, MIT, August 2010a.

[87] Sasa Misailovic, Stelios Sidiroglou, Hank Hoffman, and Martin Rinard. Quality of service profiling. In *International Conference on Software Engineering (ICSE)*, 2010b.

[88] Sasa Misailovic, Daniel M. Roy, and Martin C. Rinard. Probabilistically accurate program transformations. In *International Static Analysis Symposium (SAS)*, 2011.

[89] Sasa Misailovic, Stelios Sidiroglou, and Martin Rinard. Dancing with uncertainty. In *Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[90] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2014.

[91] Debabrata Mohapatra, Vinay K Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation and Test in Europe (DATE)*, 2011.

[92] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. SNNAP: Approximate computing on programmable SoCs via neural acceleration. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2015.

[93] Sriram Narayanan, John Sartori, Rakesh Kumar, and Douglas L. Jones. Scalable stochastic processors. In *Design, Automation and Test in Europe (DATE)*, 2010.

[94] Krishna Palem and Avinash Lingamneni. What to do about the end of Moore's law, probably! In *Design Automation Conference (DAC)*, 2012.

[95] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. Precision-aware soft error protection for GPUs. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2014.

[96] Avi Pfeffer. A general importance sampling algorithm for probabilistic programs. Technical Report TR-12-07, Harvard University, 2007.

[97] A. Rahimi, A. Marongiu, R.K. Gupta, and L. Benini. A variability-aware OpenMP environment for efficient execution of accuracy-configurable computation on shared-FPU processor clusters. In *IEEE–ACM–IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[98] Amir Rahmati, Matthew Hicks, Daniel E. Holcomb, and Kevin Fu. Probable cause: The deanonymizing effects of approximate DRAM. In *International Symposium on Computer Architecture (ISCA)*, 2015.

[99] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, 2002.

[100] Ashish Ranjan, Arnab Raha, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. ASLAN: Synthesis of approximate sequential circuits. In *Design, Automation and Test in Europe (DATE)*, 2014.

[101] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. Approximate storage for energy efficient spintronic memories. In *Design Automation Conference (DAC)*, 2015.

[102] Benjamin Recht, Christopher Re, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Conference on Neural Information Processing Systems (NIPS)*, 2011.

[103] Lakshminarayanan Renganarayanan, Vijayalakshmi Srinivasan, Ravi Nair, and Daniel Prener. Programming with relaxed synchronization. In *Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[104] Michael F. Ringenburg, Adrian Sampson, Isaac Ackerman, Luis Ceze, and Dan Grossman. Monitoring and debugging the quality of results in approximate programs. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[105] Pooja Roy, Rajarshi Ray, Chundong Wang, and Weng Fai Wong. ASAC: Automatic sensitivity analysis for approximate computing. In *ACM SIGPLAN–SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES)*, 2014.

[106] Sourya Roy, Tyler Clemons, S M Faisal, Ke Liu, Nikos Hardavellas, and Srinivasan Parthasarathy. Elastic fidelity: Trading-off computational accuracy for energy reduction. Technical Report NWU-EECS-11-02, Northwestern University, 2011.

[107] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. Precimonious: Tuning assistant for floating-point precision. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[108] Mastooreh Salajegheh, Yue Wang, Kevin Fu, Anxiao Jiang, and Erik Learned-Miller. Exploiting half-wits: Smarter storage for low-power devices. In *USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[109] Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. Sage: Self-tuning approximation for graphics engines. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013.

[110] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[111] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: approximate data types for safe and general low-power computation. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[112] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013.

[113] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2013.

[114] John Sartori and Rakesh Kumar. Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[115] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic optimization of floating-point programs with tunable precision. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2014.

[116] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. Post-compiler software optimization for reducing energy. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[117] Sayandeep Sen, Syed Gilani, Shreesha Srinath, Stephen Schmitt, and Suman Banerjee. Design and implementation of an "approximate" communication system for wireless media applications. In *ACM SIGCOMM*, 2010.

[118] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC)*, 2015.

[119] Q. Shi, H. Hoffmann, and O. Khan. A HW-SW multicore architecture to tradeoff program accuracy and resilience overheads. *Computer Architecture Letters*, 2014.

[120] Majid Shoushtari, Abbas BanaiyanMofrad, and Nikil Dutt. Exploiting partially-forgetful memories for approximate computing. *IEEE Embedded Systems Letters*, 7 (1): 19–22, March 2015.

[121] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin C. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 2011.

[122] J. Sloan, D. Kesler, R. Kumar, and A. Rahimi. A numerical optimization-based methodology for application robustification: Transforming applications for error tolerance. In *IEEE–IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010.

[123] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: A language and runtime system for perpetual systems. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[124] Renée St. Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-purpose code acceleration with limited-precision analog computation. In *International Symposium on Computer Architecture (ISCA)*, 2014.

[125] Phillip Stanley-Marbell. Encoding efficiency of digital number representations under deviation constraints. In *Information Theory Workshop (ITW)*, 2009.

[126] Phillip Stanley-Marbell and Diana Marculescu. A programming model and language implementation for concurrent failureprone hardware. In *Workshop on Programming Models for Ubiquitous Parallelism (PMUP)*, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.121.9864.

[127] Phillip Stanley-Marbell and Martin Rinard. Lax: Driver interfaces for approximate sensor device access. In *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.

[128] Ayswarya Sundaram, Ameen Aakel, Derek Lockhart, Darshan Thaker, and Diana Franklin. Efficient fault tolerance in multi-media applications through selective instruction replication. In *Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies*, 2008.

[129] Olivier Temam. A defect-tolerant accelerator for emerging high-performance applications. In *International Symposium on Computer Architecture (ISCA)*, 2012.

[130] Darshan D. Thaker, Diana Franklin, John Oliver, Susmit Biswas, Derek Lockhart, Tzvetan S. Metodi, and Frederic T. Chong. Characterization of error-tolerant applications when protecting control data. In *IEEE International Symposium on Workload Characterization (IISWC)*, 2006.

[131] Anna Thomas and Karthik Pattabiraman. Llfi: An intermediate code level fault injector for soft computing applications. In *Workshop on Silicon Errors in Logic: System Effects (SELSE)*, 2013.

[132] Bradley Thwaites, Gennady Pekhimenko, Amir Yazdanbakhsh, Jongse Park, Girish Mururu, Hadi Esmaeilzadeh, Onur Mutlu, and Todd Mowry. Rollback-free value prediction with approximate loads. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014.

[133] Jonathan Ying Fai Tong, David Nagle, and Rob. A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8 (3), 2000.

[134] Hung-Wei Tseng, Laura M. Grupp, and Steven Swanson. Underpowering NAND flash: Profits and perils. In *Design Automation Conference (DAC)*, 2013.

[135] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy. b-HiVE: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units. In *Design Automation Conference (DAC)*.

[136] Swagath Venkataramani, Amit Sabne, Vivek Kozhikkottu, Kaushik Roy, and Anand Raghunathan. SALSA: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC)*, 2012.

[137] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation and Test in Europe (DATE)*, 2013.

[138] Swagath Venkataramani, Anand Raghunathan, Jie Liu, and Mohammed Shoaib. Scalable-effort classifiers for energy-efficient machine learning. In *Design Automation Conference (DAC)*, 2015.

[139] Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. MACACO: Modeling and analysis of circuits for approximate computing. In *IEEE–ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

[140] Ajay K. Verma, Philip Brisk, and Paolo Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Design, Automation and Test in Europe (DATE)*, 2008.

[141] Benjamin Vigoda, David Reynolds, Jeffrey Bernstein, Theophane Weber, and Bill Bradley. Low power logic for statistical inference. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2010.

[142] Lucas Wanner and Mani Srivastava. ViRUS: Virtual function replacement under stress. In *USENIX Workshop on Power-Aware Computing and Systems (HotPower)*, 2014.

[143] M. Weber, M. Putic, Hang Zhang, J. Lach, and Jiawei Huang. Balancing adder for error tolerant applications. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013.

[144] Edwin Westbrook and Swarat Chaudhuri. A semantics for approximate program transformations. Technical Report Preprint: `arXiv:1304.5531`, 2013.

[145] David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[146] Vicky Wong and Mark Horowitz. Soft error resilience of probabilistic inference applications. In *Workshop on Silicon Errors in Logic: System Effects (SELSE)*, 2006.

[147] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, Jongse Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan. Axilog: Language support for approximate hardware design. In *Design, Automation and Test in Europe (DATE)*, 2015.

[148] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. On reconfiguration-oriented approximate adder design and its application. In *IEEE–ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.

[149] Thomas Y. Yeh, Petros Faloutsos, Milos Ercegovac, Sanjay J. Patel, and Glen Reinman. The art of deception: Adaptive precision reduction for area efficient physics acceleration. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007.

[150] Thomas Y. Yeh, Glenn Reinman, Sanjay J. Patel, and Petros Faloutsos. Fool me twice: Exploring and exploiting error tolerance in physics-based animation. *ACM Transactions on Graphics*, 29 (1), December 2009.

[151] Yavuz Yetim, Margaret Martonosi, and Sharad Malik. Extracting useful computation from error-prone processors for streaming applications. In *Design, Automation and Test in Europe (DATE)*, 2013.

[152] Yavuz Yetim, Sharad Malik, and Margaret Martonosi. CommGuard: Mitigating communication errors in error-prone parallel execution. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[153] Qian Zhang, Feng Yuan, Rong Ye, and Qiang Xu. ApproxIt: An approximate computing framework for iterative methods. In *Design Automation Conference (DAC)*, 2014.

[154] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *International Symposium on Integrated Circuits (ISIC)*, 2009.

[155] Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A. Kelner, and Martin C. Rinard. Randomized accuracy-aware program transformations for efficient approximate computations. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, 2012.