

Kari BME ösztöndíj
2018/19. tanév tavaszi félév

Publikáció és előadás adatok adatlap

Hazai: []

Nemzetközi: [x]

Elfogadó igazolás dátuma: 2018.12.10.

Megjelenés/előadás dátuma: 2019.01.17.

Megjelenés/előadás helye: Institut für Technik der
Informationsverarbeitung, Karlsruhe, Németország

Pontos címe: A Review of Approximate Computing methods for a Universal
Approximate Hardware Accelerator

Szerzők: Reizinger Patrik

DOI:-

Publikáció elektronikus elérhetősége:-

Társszerzők aláírása:

Kelt: 2019.02.05.

.....
Aláírás



Patrik Reizinger <rpatrik1996@gmail.com>

WG: Thema Seminarvortrag

Harbaum, Tanja (ITIV) <tanja.harbaum@kit.edu>
Címzett: Patrik Reizinger <rpatrik1996@gmail.com>

2018. december 10. 11:23

Hallo Patrik,

es klappt leider erst im Januar, aber wir können vorher ja schon einmal alles fertig machen!

Viele Grüße,
Tanja

-----Ursprüngliche Nachricht-----
Von: Hirzler, Birgitta (ITIV)
Gesendet: Montag, 10. Dezember 2018 11:22
An: Harbaum, Tanja (ITIV) <tanja.harbaum@kit.edu>
Betreff: AW: Thema Seminarvortrag

Hallo Tanja,

hiermit bestätige ich dir den Termin für den Vortrag von Herrn Reizinger:

Donnerstag, 17.01.18, um 13:30 Uhr, Raum 326

Gruß
Birgit

-----Ursprüngliche Nachricht-----
Von: Harbaum, Tanja (ITIV) <tanja.harbaum@kit.edu>
Gesendet: Montag, 10. Dezember 2018 11:19
An: Hirzler, Birgitta (ITIV) <birgitta.hirzler@kit.edu>
Betreff: Thema Seminarvortrag

Hallo Birgit,

das Thema ist "A Review of Approximate Computing methods for a Universal Approximate Hardware Accelerator".

Viele Grüße,
Tanja

Karlsruher Institut für Technologie (KIT)
Institut für Technik der Informationsverarbeitung (ITIV)

Dipl.-Inform. Tanja Harbaum
Wissenschaftliche Mitarbeiterin

Engesserstr. 5
Gebäude 30.10
76131 Karlsruhe

Telefon: +49 721 608- 47169
Fax: +49 721 608-42511
E-Mail: harbaum@kit.edu

A bekeretezett rész tartalmazza a visszaigazolást a január 17-én tartandó előadásomról.

Web: www.itiv.kit.edu

KIT – Die Forschungsuniversität in der Helmholtz-Gemeinschaft
Das KIT ist seit 2010 als familiengerechte Hochschule zertifiziert.

A Review of Approximate Computing methods for a Universal Approximate Hardware Accelerator

Patrik Reizinger

Institute for Information Processing Technologies

Karlsruhe Institute of Technology

Karlsruhe, Germany

rpatrik1996@gmail.com

Abstract—The advancement of the fourth industrial revolution are generating an extensive amount of data but current computing paradigms cannot be scaled in a feasible way to keep pace with this tendency. Fortunately, approximate computing provides an alternative approach by sacrificing accuracy for performance within reasonable bounds, giving the designer the power of choice. This paper intends to provide a detailed overview of the state-of-the-art methods focusing mainly on the hardware level. Based on that an approach is proposed which may be worth considering for heterogeneous multi-core environments.

Index Terms—Approximate computing, error resilience, quality management, survey

I. INTRODUCTION

Approximate computing, i.e. the computing paradigm of preferring reasonable errors to precise code execution, is not a new paradigm - consider e.g. the finite resolution of floating point data types -, but it became more important in recent years due to the popularity gain of data-intensive methods. The need for approximate computing is depicted on Figure 1 from the point of view of Artificial Intelligence (AI), one of the most data-intensive fields of applied computing. It shows an exponential tendency with a doubling rate of required computational power about each 3.5 months (significantly less compared to the 18 months formulated in Moore's Law for the number of transistors) concerning the largest AI models [1].

Current research has focused i.a. on the identification of error resilient operation, on almost all abstraction levels. Solutions span from silicon-level to compiler extensions, for the sake of trading off performance, power consumption and non-recurring engineering (NRE) costs (Figure 2). The preferred factor is determined based on the constraints of the specific task.

This is often the case for recognition, mining and synthesis (RMS) applications - mainly based on the fact that no golden answer exists. The reasons why approximate computing is applicable for several types of real-life problems are diverse: properties of the input, transfer or output stages can be exploited for the expected results [2]. Due to the fact that available data is in the majority of cases noisy or redundant, the task itself becomes inherently error-resilient. The effect is the same while relying on the statistical nature of the processing algorithmic pipeline or the limitations of the output stage, e.g. the limited perception ability of humans concerning a signal of visual or acoustic nature.

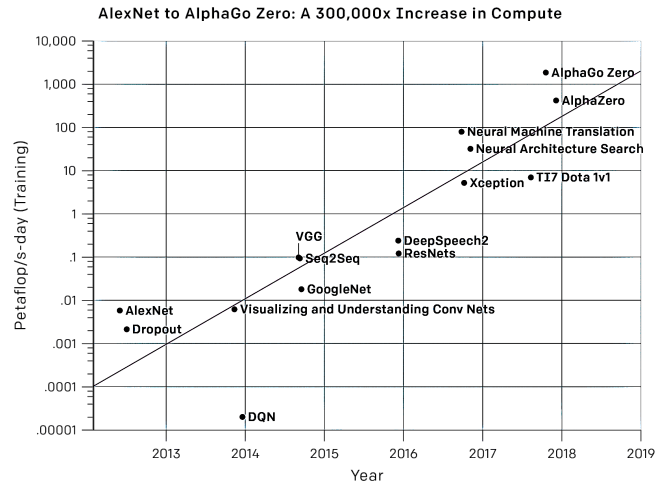


Fig. 1. The exponential need for more computational power in AI [1]

Finding the optimal approximate method is rather challenging for the definition of "accurate", "acceptable", or generally that of a quality measure is always task-specific. Thus, the identification of the proper metric is at least as important as the approximate method itself.

The paper gives an overview of recent advancements in the domain of approximate computing and is organized as follows: Section II summarizes the state-of-the-art of research, focusing mainly on hardware-based solutions, but also mentioning approximate computing paradigms for software, and not least emphasizing the significance of co-design. Section III investigates the potential of approximate computing for multi-core systems. Section IV briefly summarizes the contributions of this paper.

II. REVIEW OF CURRENT METHODS

This section intends to review the most significant and promising approaches of approximate computing. The solutions are organized into two subsections based on whether they can be categorized either as hardware or software, although it should be emphasized that the co-design of hardware and software can increase efficiency, such as constructing a compiler tool can only fully utilize its power if there is a proper

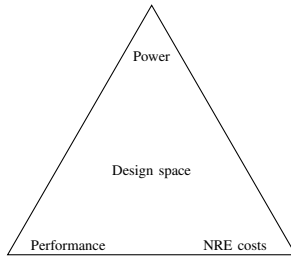


Fig. 2. The contradictory aspects of optimization

architecture designed for that specific group of problems. A review of approximate computing methods, from different aspects, is given also in [3]–[5].

A. Software

The intention of providing off-the-self software tools for the scientific and industrial community is mainly due to its flexible and rapid applicability, thus these solutions represent a proper way of decreasing also NRE costs, while requiring only adequate design choices and sometimes a modified IDE, thus the review of them is of high significance.

1) *Code profiling*: The software approach of approximate computing methods often involves profiling steps - either as separate analysis runs of the code or with the help of compiler directives - which are responsible for determining the eligibility of approximation. Such a lightweight quality management system is Rumba [6], which can identify and correct large errors in specific tasks. Rumba consists of two parts: error detection and recovery - it utilizes not only offline techniques, but the concept developed in [6] also proposes a hardware unit, thus opting for hardware-software co-design. The main assumption of the model is that a small number of errors are responsible for the majority of the quality loss, thus identifying and correcting them - with a minimal overhead - can result in significant performance gains. In the case of a required precise re-run, Rumba applies an analysis to identify code regions which can be executed without side effects.

Another approach is examined in [7], the proposed framework called Green builds a probabilistic Quality of Service (QoS) model of the program to carry out approximate routines based on it. With a proper subsystem, it monitors the energy-consumption and the effect on quality for specific loops and functions, based on which the framework can decide to use one of the approximations provided by the programmer. This approach gives full control to the user with enabling multiple approximation versions of a function/loop. While being an advantage, it is also a drawback regarding design time.

Mixing precise and approximate data in the pipeline may help the designer save computational power, but also imposes the danger of accidentally merging approximate subresults into the precise data-flow. With splitting approximate and precise data-flow EnerJ [8] provides a safe approach - also including a bridge between both paradigms, the so-called endorsement construct. EnerJ uses type annotations, which

are exploited by the compiler to achieve significant savings regarding computational power.

A systematic overview of the characterization of error-resilience in RMS applications can help us understand the properties of the available data, thus it can help making better design choices. The ARC (Application Resilience Characterization) framework proposed in [2] argues for this approach. ARC uses a two-step algorithm, first unconstrained random errors are injected, the result of which is the set of potentially resilient kernels. After that, the characterization step decides whether an approximation technique can be utilized or the precise variant should be insisted on. An important remark in [2] is the evaluation of error-resilience with different levels of granularity: the conclusion is that error-sensitivity may vary significantly given different granularity levels.

2) *Choosing the precision of number representation*: One of the most simple and trivial methods is to choose different representations for error-resilient and error-prone records. Besides choosing between single and double precision there is also the possibility to apply fixed point arithmetic for specific variables, the main advantage is the exploitation of the speed-up of integer operations. For noisy data - a common phenomenon in RMS applications - this design choice may bring a significant performance gain for a little price. In Deep Learning it is also quite common to exploit the advantages of mixed precision representations [9] - which rely on current GPU architectures [10] enabling the use of floating point numbers with different amount of bits, thus software and hardware are hard to distinguish.

3) *Loop perforation*: Loop perforation [11] is a framework which analyzes loops in the code to separate critical and tunable loops to explore the design parameter space of the latter. Critical loops cannot be the target of loop perforation due to the high probability of causing significant quality degradation, memory errors etc. Several global patterns suitable for loop perforation are identified and probabilistic guarantees given for its success. Considering RMS applications, loop perforation can be thought to be similar in a way of using minibatch samples, i.e. not calculating statistics over the whole dataset but sampling it.

4) *Easing synchronization barriers*: Parallel program execution often needs the synchronization of different threads to avoid race conditions and erroneous results, but increasing the parallelism of a program also makes the synchronization overhead more significant. Current research shows that strategies exist for easing this overhead, and thus boosting program performance in means of execution time, without significant loss of quality. An approximate compiler called ACCEPT is proposed in [3] which also provides a more detailed overview of the topic.

5) *Expressing probabilistic behaviour in code*: Probabilistic programming languages [12] have emerged to fulfil the requirements of Bayesian machine learning, for which inference is of the same importance as automatic differentiation is for Deep Learning. Probabilistic languages, such as Pyro [13] are widespread in Artificial Intelligence, which - being a

method of black-box modelling from an engineering point of view - can utilize the probabilistic approach.

B. Hardware

The hardware approach of approximate computing - although generally being less flexible than its software counterpart - can be accountable for substantial performance boost by reducing the overhead introduced by the abstraction layers of software tools.

1) *Near-Threshold Voltage Programming*: Near-Threshold Voltage Programming (NTC) [14] exploits the fact that reducing the operating voltage (V_{dd}) to the level of the threshold voltage (V_{th}), can result in significant energy savings, because

$$P \sim fU^2, \quad (1)$$

thus reducing the operating voltage results in a quadratic improvement regarding the performance budget, for the main contributing factor to the energy consumption of today's electric circuits is the re- and discharging of CMOS capacitances which imply the quadratic relationship mentioned above. The main drawbacks of the NTC approach are the increased sensitivity for parameter variations, functional failure and obviously performance loss, not to mention the increased operation times due to larger delays. The longer time needed for signal propagation will also limit the performance gains, because due to the non-linear relation between voltage and delay, signal propagation will be deteriorated in a significant way, so that lower voltage levels cannot compensate. Nevertheless, given the performance budget chips are intended to operate from, the increased number of cores cannot be utilized perfectly with traditional voltage scaling techniques, while NTC enables the use of a higher ratio of cores, so systems can experience a performance gain.

To overcome that losses, frameworks such as Accordion [15] were proposed, which can schedule tasks depending on their error-resilience for near-threshold or traditional computing units. Accordion has two scaling modes, one for the case if the problem size changes with the number of cores (soft scaling) and one if it does not (strong scaling). Ensuring the best trade-off, Accordion schedules data-intensive - and thus, in general, error-resilient - tasks on approximate cores, while control instructions are executed strictly on reliable ones.

2) *Approximate storage*: Current memory systems are designed for accurately storing data, although this restriction can be loosened in several cases - as an analogy to lossy data compression. The reduction of the refresh rate of DRAM cells makes the stored bits more error-prone, on the other hand not re-charging the capacitances reduces the power budget. This approach can also be exploited from a manufacturing point of view, in the case of Multi-level Cells (MLC) smaller guard bands can be used [3], thus increasing the cell density (see Figure 3) - while increasing the probability that neighbouring cells influence/modify each other. The main point is to identify which parts of the data can be stored in approximate cells, thus maximizing energy savings while maintaining an acceptable performance level. To demonstrate the relevance of

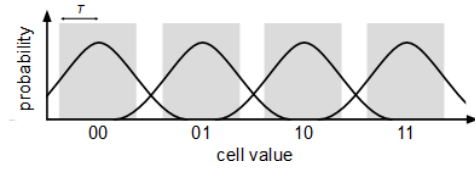


Fig. 3. Overlapping guard bands for approximate storage cells [3]

this approach, refer to [16], which proposes an image storage framework for approximate data storage hardware.

3) *Approximate arithmetic components*: Approximating computational results on the lowest, i.e. silicon, level can reduce the software overhead if bigger units and whole programs can be executed approximately or with providing hardware components which approximate code can be delegated to. [17] reviews different approaches how adders and multipliers can be approximated efficiently. One approach is to predict/manipulate the carry propagation chain, thus providing speed-up for the hardware operation. The fact that the lower bits generally have a higher noise level enables the approximation without significant loss of accuracy. [18] also states that approximate multipliers are rather scarce in the literature, although, as [17] concurs, approximate multipliers built from approximate adders may not be considered as the best design approach.

Memristors are a rather new element for electronic design. The main advantage of these elements is that due to their highly non-linear characteristics, memristors can be used simultaneously for data processing and storage - the so called computation-with-memory approach. Thus memristors are suitable for analog signal processing, nevertheless digital applications can also exploit the advantages, e.g. [19] proposed an approximate matrix multiplier built by means of memristors.

4) *Instruction set architecture*: Quality programmable processors, proposed in [20], represent a solution which targets the ISA itself with encoding quality requirements in it. With QUORA [20] a quality programmable vector processor is proposed, which also utilizes a quality programmable microarchitecture mapping the quality requirements included in the instructions to energy savings. To achieve that, the design of QUORA includes three types of processing units: approximate, mixed and accurate. A Quality Control Unit is responsible for dispatching the instructions to the appropriate processing units.

5) *Hardware synthesis*: To maximally utilize the advantages of approximate hardware structures, proper synthesis tools are required which meet the different requirements in approximate hardware design, of course the statistical modelling of approximate hardware should complement synthesis frameworks - such as in [21] - to evaluate the quality of the results. [21] proposes a statistical quality modelling framework (based on dataflow graphs) also incorporating input statistics, which uses energy as the optimization target.

[22] proposes a Symbolic Computer Algebra-based method to generate netlists given an optimality criterion, e.g. the

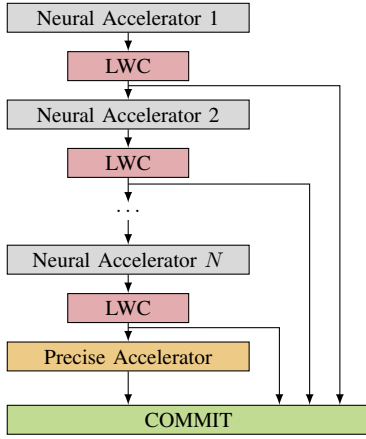


Fig. 4. The chain of neural accelerators in BRAINIAC [26]

number of gates (area) or the lengths of the critical path (delay). The main idea of the algorithm is to replace gates with one of their inputs as a means of approximation.

[23] proposes an extension for the hardware description language Verilog, which provides a tool for automating the design and reuse of approximate hardware. Besides being an aide during the design process of a given IP core, the proposed framework also incorporates the necessary abstraction tools for the ease of reuse, e.g. as in the case of EnerJ [8] the language constructs for bridging do.

6) *Neuromorphic computing*: Based on the recent boost of attention regarding Artificial Intelligence and neural networks, it seems reasonable that dedicated hardware approaches emerged for efficient neural network implementations. The framework AxTrain [24] is a hardware oriented framework for neural networks, which optimizes both for robustness and accuracy. I.e. parameter variations of approximate hardware imply a not so significant performance degradation as in the case of classical optimization techniques in machine learning. The difference is, that AxTrain also incorporates the neighbourhood of the (local) minimum during the optimization process, so small errors due to approximate hardware do not introduce significant effects.

Reservoir computing, which utilizes a large pool of sparsely connected, randomly initialized neurons, has been brought into a new perspective with the memristor-based implementation in [25]. Due to the fact that in the case of reservoir computing the reservoir is not trained, only the output layer, significant training costs can be saved.

BRAINIAC [26] is a hierarchical neuromorphic framework suitable for approximate computational tasks, even in ASICs. It uses a chain of neural accelerators as shown in Figure 4. After each level of accelerators a quality check is carried out (for the detailed description of Light-Weight Checks (LWC) refer to [27]). If the specified quality requirement is not met, the data is forwarded to the next level, otherwise it is accepted and committed. BRAINIAC utilizes three types of neural accelerators which represent different advantages. The first type with fixed weights and connections obviously trades

off flexibility for speed, the second one prefers flexibility by making both parameter sets (i.e. connections and weights) configurable. The third architecture merges the advantages of the first two to provide a compromise: it is faster than the fully flexible due to fixed connections and more configurable than the first because the weights can be modified.

III. APPROXIMATE COMPUTING IN MULTI-CORE ENVIRONMENTS

Multi-core architectures are dominating the last era of processing units due to the fact that the increase of frequency has become rather challenging from a manufacturing point of view, not to mention the higher power consumption of the chips. With smaller transistors the number of cores on a single chip has increased significantly in recent years, despite the fact that multi-core environments require more sophisticated task-scheduling, load-balancing algorithms to avoid the cores being idle. It is also clear that distributed computing imposes synchronization difficulties, thus further overhead.

The engagement towards multi-core architectures is clear for both manufacturers and customers, so approximate computing models should also be transferred to/developed for this computing paradigm. The challenges are sometimes rather hard to overcome even in the case of homogeneous processing units, nevertheless the - practically logical, and also feasible - case of heterogeneous processors is even more challenging. Providing task specific accelerators, optimized units is a clear advantage, which comes at the price of more diverse energy, computational, temporal etc. characteristics. The specialized accelerators in heterogeneous multi-core systems, supplemented with the speed-up/energy savings of approximate techniques provide clearly a vast opportunity for computing. The exploitation of the synergy between both topics can be found e.g. in [28], where a scheduling framework is introduced which utilizes approximate computing methods for better results.

E.g. consider a specific real-time application, with the help of approximate accelerators the perceived quality can be improved if resources will be scarce, e.g. in the case of multimedia applications, instead of skipping a frame when the processing units do not have the capability to process a frame accurately, an approximation may be still sufficient.

To realize such a solution, it is important to have a sophisticated resource-management unit, which keeps track of the workload of both precise and approximate cores. It is also a relevant research topic to determine a threshold when is it feasible to switch between levels of precision, because the re-allocation of a new core and additional data transfer may introduce an overhead, which in some situations makes the precise version faster.

Figure 4 also sheds light to a setup being worth considering: if a given task can be decomposed into orthogonal subtasks t_i , so that every increment improves the QoS, i.e.

$$QoS(\cup_{i=1}^{N-1} t_i) \leq QoS(\cup_{i=1}^N t_i), \quad (2)$$

where $QoS()$ defines the quality metric, $\cup$ is used to compose the subtasks t_i (i.e. to take the union of them), then a

hierarchical structure like in BRAINIAC can be implemented efficiently. Consider an RMS-application with batch-like execution, i.e. massive amount of data is fetched in every iteration. If a QoS metric for multiple levels is specified, it is clear to see that given a Gaussian quality distribution of the subresults the higher levels of approximation accelerators will not have the same load as lower ones. Hence organizing the accelerators into e.g. a tree-like structure can decrease the number of allocated accelerators, thus increasing efficiency.

The question of selecting the tasks which can be executed in an approximate manner can be either determined in software, which means a further burden for the programmer - also assuming that the language and compiler extensions are designed and developed to fulfil this properties. The other way is to determine this ability based on specified error criteria at runtime. Of course, the latter is more convenient for the programmer, but implies other challenges. Assume that a specific computing kernel is launched in both approximate and precise accelerators, which is required to carry out runtime checks, thus the workload and the power-consumption of the architecture is at least temporarily increased. If we take into consideration the inherent error-resilience of most RMS applications, then organizing the comparisons themselves iteratively, e.g. into a tree-like structure (i.e. units carrying out more precise checks are less), e.g. with a Markov-model-based state machine - as used for branch prediction, thus providing adaptivity - then there is a possibility to reduce the number of quality checks while experiencing speed-up and energy savings.

IV. SUMMARY

In this paper, the different aspects of approximate computing were covered, from silicon level hardware to high-level software tools, also emphasising the importance of hardware-software co-design. The potential of approximate computing methods were also analyzed from the point of view of heterogeneous multi-core architectures.

REFERENCES

- [1] D. Amodi and D. Hernandez, "AI and Compute," 2018.
- [2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, no. i, p. 1, 2013.
- [3] A. Sampson, *Hardware and Software for Approximate Computing*. PhD thesis, 2015.
- [4] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited - Cross-layer approximate computing: from logic to architectures," *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, pp. 1–6, 2016.
- [5] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate Computing: A Survey," *IEEE Design and Test*, vol. 33, no. 1, pp. 8–22, 2016.
- [6] S. Mahlke, "Rumba : An Online Quality Management System for Approximate Computing," pp. 554–566.
- [7] W. Visser, J. Geldenhuys, and M. B. Dwyer, "Green," *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, p. 1, 2012.
- [8] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-Power Computation," *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pp. 164–174, 2011.
- [9] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," pp. 1–12, 2017.
- [10] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance & precision," *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*, no. August, pp. 522–531, 2018.
- [11] S. Sidiropoulos and H. Hoffmann, "Managing Performance vs. Accuracy Trade-offs With Loop Perforation," in *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2011.
- [12] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood, "An Introduction to Probabilistic Programming," 2018.
- [13] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.
- [14] R. G. Dreslinski, M. Wiecekowsky, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.
- [15] U. R. Karpuzcu and I. Akturk, "Accordion : Toward Soft Near-Threshold Voltage Computing ," 2011.
- [16] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar, "High-Density Image Storage Using Approximate Memory Cells," *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '16*, pp. 413–426, 2016.
- [17] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm For Energy-Efficient Design," *18th IEEE European Test Symposium (ETS) Approximate*, pp. 278–282, 2013.
- [18] J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*, p. 504, 2012.
- [19] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Memristor-based approximate matrix multiplier," *Analog Integrated Circuits and Signal Processing*, vol. 93, no. 2, pp. 363–373, 2017.
- [20] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46*, no. i, pp. 1–12, 2013.
- [21] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer, "Statistical quality modeling of approximate hardware," *Proceedings - International Symposium on Quality Electronic Design, ISQED*, vol. 2016-May, pp. 163–168, 2016.
- [22] S. Froehlich, D. Große, and R. Drechsler, "Approximate hardware generation using symbolic computer algebra employing gröbner basis," *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, vol. 2018-Janua, no. March, pp. 889–892, 2018.
- [23] A. Yazdanbakhsh, B. Thwaites, J. Park, and H. Esmailzadeh, "Methodical Approximate Hardware Design and Reuse." 2014.
- [24] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang, "AxTrain: Hardware-Oriented Neural Network Training for Approximate Inference," 2018.
- [25] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nature Communications*, vol. 8, no. 1, pp. 1–10, 2017.
- [26] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing," *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, pp. 615–626, 2015.
- [27] B. Grigorian and G. Reinman, "Dynamically adaptive and reliable approximate computing using light-weight error analysis," *Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2014*, pp. 248–255, 2014.
- [28] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures," *20th Asia and South Pacific Design Automation Conference, ASP-DAC 2015*, pp. 618–623, 2015.