ApproxQA: A Unified Quality Assurance Framework for Approximate Computing

Ting Wang, Qian Zhang and Qiang Xu

CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {twang,qzhang,qxu}@cse.cuhk.edu.hk

***Abstract*—Approximate computing, being able to trade off computation quality and computational effort (e.g., energy) by exploiting the inherent error-resilience of emerging applications (e.g., recognition and mining), has garnered significant attention recently. No doubt to say, quality assurance is indispensable for satisfactory user experience with approximate computing, but this issue has remained largely unexplored in the literature. In this work, we propose a novel framework namely ApproxQA to tackle this problem, in which approximation mode tuning and rollback recovery are considered in a unified manner. To be specific, ApproxQA resorts to a two-level controller, in which the high-level approximation controller tunes approximation modes at a coarse-grained scale based on Q-learning while the low-level rollback controller judiciously determines whether to perform rollback recovery at a fine-grained scale based on the target quality requirement. Experimental results on various benchmark applications demonstrate that it significantly outperforms existing solutions in terms of energy efficiency with quality assurance.**

I. Introduction

Energy efficiency has become a first-class design objective for both warehouse scale data centers and mobile devices. Considering that a large amount of emerging applications (e.g., Recognition, Mining and Synthesis (RMS)) are error resilient in nature, approximate computing [1], where computation quality is traded off for computational effort, is regarded as one of the most promising energy-efficient design techniques.

As the computation quality requirement of an application may vary significantly at runtime, it is preferable to tune approximation modes (different quality-effort tradeoff levels) accordingly to satisfy quality requirements and save unnecessary computational effort. On the other hand, the output quality is highly dependent on input patterns, and it is difficult, if not impossible, to select an appropriate approximation mode that can always satisfy the target quality requirement. Therefore, rollback recovery is indispensable when quality violations occur [2]. To the best of our knowledge, however, none of existing works simultaneously take both approximation mode tuning and rollback recovery into consideration, which could cause significant energy waste and/or unsatisfactory user experience with quality violations.

Generally speaking, an effective dynamic quality management system should first select an appropriate approximation mode whenever possible, and perform rollback recovery to correct occasional quality violations when necessary. If frequent rollback recoveries were performed, which indicates the current approximation mode is too aggressive (less accurate), a more accurate approximation mode (or exact computation) should be chosen for the following computations. On the other hand, if the output quality is much better than the target quality, it is preferable to select a more aggressive approximation mode for the following computations.

Based on the above observations, in this paper, we propose a unified quality assurance framework for approximate computing, namely *ApproxQA*, in which a two-level controller is designed. The low-level *rollback controller* checks whether there are quality violations and judiciously determines whether to rollback or not at a fine-grained scale. The high-level *approximation controller* decides which approximation mode to use at a coarse-grained scale based on Q-learning, by observing the state of the system (e.g., how many rollback recoveries are performed). With a carefully-chosen reward function used in Q-learning, ApproxQA facilitates to choose appropriate approximation modes according to runtime variations with optimized energy efficiency.

The rest of the paper is organized as follows. In Section II, we discuss our motivation and previous arts related to our work. Section III describes our proposed quality assurance framework in detail. Experimental results are presented in Section IV. Section V concludes the paper.

II. Related Works and Motivation

Some research efforts have been dedicated to quality assurance with approximate computing in the literature [5–9]. In [5], a quality control method for iterative methods is proposed. SAGE [6] and Green [7] monitor online output quality periodically and select an appropriate approximation mode accordingly. Recently, IRA [8] controls how to apply program approximation based on canary inputs that can capture the intrinsic properties of the full input. However, none of the above works consider rollback recovery. There may exist large occasional errors from time to time, which would significantly degrade user experience [2].

Grigorian et al. [9] detects quality violations with application specific quality checkers and performs rollback recov-

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

ery for satisfying the target quality requirement. Rumba [2] achieves quality assurance by tuning the threshold for rollback recovery. However, they both do not consider choosing an appropriate approximation mode. For inappropriate approximation, too aggressive mode (overly approximate) would result in frequent rollback recoveries causing significant performance and energy overheads, and too conservative (overly accurate) mode would compromise the energy efficiency by not fully utilizing the potential of approximation.

Motivated by the above, in this paper, we present a unified quality assurance framework for approximate computing, called ApproxQA. To the best of our knowledge, it is the first *holistic* work that explicitly considers both approximation mode tuning and rollback recovery, as detailed in the following sections.

III. ApproxQA Framework

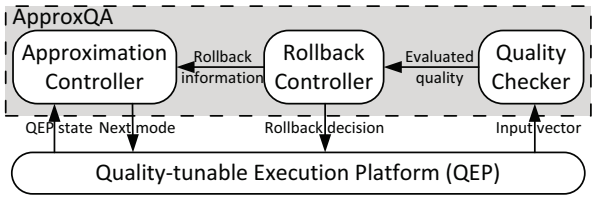

Fig. 1. ApproxQA diagram.

Our quality assurance framework, namely ApproxQA, is shown in Figure 1. Given a Quality-tunable Execution Platform (QEP) that can execute RMS applications with different accuracy levels, ApproxQA decides whether to perform rollback recovery and what computation mode to take. The QEP can be implemented with any existing hardware/software level reconfigurable approximation techniques with different mode transition costs. If the approximation is achieved with hardware techniques (e.g., reconfigurable adders in [3]), the transition between approximation modes may take nonnegligible time. As a result, changing approximation modes too frequently may diminish the energy/performance advantage of approximate computing. However, if the approximation is achieved with accuracy-tunable programs (e.g., expensive function approximation in [7]), such transition cost may become much lower, which makes it possible to tune the approximation mode frequently.

Our ApproxQA framework consists of three components: the *approximation controller*, *rollback controller* and *quality checker*, respectively. The quality checker receives *input vectors* (detailed in the next section) from the QEP and evaluates the current computation quality. Our rollback controller and approximation controller work together as a two-level controller. The high-level approximation controller operates at a coarse-grained scale to tune computation modes and the low-level rollback controller performs at a fine-grained scale to decide whether a rollback recovery is needed or not.

As computing patterns amenable for approximation (e.g., map and reduce) are usually data parallel in nature [2], the approximation controller and rollback controller can operate at intervals with different numbers of input patterns. The rollback controller determines whether to perform rollback recovery for a single input pattern (e.g., pixel in an image) based on the evaluated quality from the quality checker and target quality requirement. We re-execute the original input pattern with the fully accurate mode when rollback recovery is issued. The approximation controller determines what approximation mode to use with a long interval, for example, N input patterns. It sets checkpoints every N input patterns. At each checkpoint, it observes the state of the QEP and the rollback information from the rollback controller, and then it chooses an appropriate approximation mode for the following computations based on Q-learning technique [10].

A. Quality Checker

Quality checkers are used to check computation errors caused by applied approximation and output the current approximation error. As different input patterns and approximation modes may lead to different approximation errors, the input of quality checkers should contain both of these information.

Many existing quality checker designs (e.g., [2, 11, 12]) can be used in ApproxQA, as long as they have the same input/output interfaces as the above definition. For instance, if we utilize linear model based quality checker as proposed in [2], the approximation error is computed as the linear function of input patterns and the approximation mode.

B. Rollback Controller

To ensure satisfactory user experience, ApproxQA issues rollback recovery when the approximation error of the current input pattern is greater than a pre-determined error threshold. Therefore, the error threshold plays a vital role to guarantee the output quality. However, due to the complex behavior of applications and the infinity of possible input patterns, it is difficult, if not impossible, to provide formal guarantees on the output quality. Thus, ApproxQA provides statistical quality assurance that the probability of large approximation errors is bounded.

The objective of this phase is to find an appropriate error threshold (th) for rollback recovery so that we can guarantee the output quality statistically. We assume the actual approximation error for the current input pattern is x, the error given by quality checker is y ($y > 0$), the application-specific quality evaluation metric is $q(x)$ (such as the absolute relative error, direction error and update error given in [5]), and the user defined quality loss constraint is C_{max}, the overall problem can be expressed as:

$$P_r(q(x) > C_{max}) < \epsilon. \tag{1}$$

This equation represents that the probability of large quality loss is bounded by ϵ. Based on the law of total probability, we can obtain

$$\begin{aligned} P_r(q(x) > C_{max}) = P_r(q(x) > C_{max}, y \leq th) + \\ P_r(q(x) > C_{max}, y > th). \end{aligned} \tag{2}$$

Considering rollback recovery, $P_r(q(x) > C_{max}, y > th) = 0$. Then

$$P_r(q(x) > C_{max}, y \leq th) = \iint_D p(x,y)d\sigma, \quad (3)$$

where $D \in \{(x,y)|q(x) > C_{max}, y \leq th\}$ and $p(x,y)$ is the joint probability density function of x and y that can automatically consider the inaccuracy of quality checkers.

We can achieve the density function based on the histogram or kernel method. Then the threshold for rollback recovery is obtained by choosing the maximum th that can satisfy Equation 1.

C. Approximation Controller

The basic idea of Q-learning is to determine what action to take based on the current system state in order to maximize the expected reward [10]. Our Q-learning based approximation controller sets checkpoints every N input patterns, during which rollback recovery may have been performed many times. At each checkpoint, the approximation controller checks system states (e.g., states of the QEP and the rollback controller) and chooses an approximation mode for the following computations. Next, we discuss the key gradients in Q-learning for our approximation controller.

State Space: In our approximation controller, system states are composed of the states of the QEP and the rollback controller only. We do not consider the state of the system workload, as it is unknown a priori. For example, if the QEP has S_e different states and the rollback controller contains S_r different states, then the system can have $S = S_e \times S_r$ states in total. The state of the QEP is characterized by which approximation mode is used. That is to say, the platform can work at S_e approximation modes (fully accurate mode included).

The state of the rollback controller can be characterized by the number of rollback recovery performed since the last checkpoint. However, due to its large number, we plan to discretize the number of rollback recovery. Usually, a large number of rollback recovery is rare. If it is unlikely to perform more than $R\%$ rollback recovery, we discretize the $0\% - R\%$ rollback recovery into n equal levels as n states, and treat more than $R\%$ rollback recovery as the $n+1$ state.

Action Space: Our approximation controller decides what approximation mode to take for the next N input patterns. Therefore, the action space of the approximation controller is the set containing all approximation modes (including the fully accurate mode) for the QEP.

Reward Function: The purpose of the approximation controller in ApproxQA is to achieve maximum energy savings under the target quality requirement. The reward function of our Q-learning model should reflect this purpose. As the target quality requirement is ensured by the rollback controller, the approximation controller only needs to maximize energy savings. Therefore, the less energy consumed, the more reward. Based on this definition, we define the reward function as follows:

$$r(s_t, a_t) = 1/E(s_t, a_t), \quad (4)$$

where $E(s_t, a_t)$ is the energy consumption of all system components including the QEP and ApproxQA in state s_t by taking the action a_t.

Intuitively, in the learning period, if too aggressive approximation mode was chosen, it would lead to many rollback recoveries, which consume significant energy. Therefore, less reward is received. On the other hand, if too accurate approximation mode was selected, it also wastes energy and receives less reward as less accurate approximation mode is enough. Less reward leads to less probability of selecting the too aggressive/accurate modes next time. As learning proceeds, our approximation controller will prefer to select the approximation mode with better reward or less energy consumption.

IV. Experimental Results

A. Experimental Setup

To evaluate ApproxQA, we assume that the QEP consists of a CPU and a Neural Processing Unit (NPU) [4] based approximate accelerator. Approximation modes with different accuracy levels and energy consumptions are achieved with different neural network topologies. As the NPU does not provide an accurate mode, the CPU is used when the fully accurate execution is needed. In our experiments, we have 12 approximation modes in total. In our experiment, we simulate the execution of benchmark applications on QEP with *gem5* simulator. The energy consumption of the system is obtained with McPAT simulator. The whole system is modeled in 32nm technology node. The reported energy includes that consumed by all system components, such as NPU, CPU and ApproxQA. Table I shows the specifications of benchmark applications used in our simulation.

B. Quality Assurance

ApproxQA ensures that the probability of large errors or quality violations is bounded even when the underlying quality checkers are not reliable. Figure 2 shows the final quality obtained for different benchmark applications when targeting 5% quality violation bound ($\epsilon = 5\%$). We compare results of ApproxQA with the so-called *Baseline* in which they assume the underlying quality checker is perfect. We choose an error threshold for rollback recovery based on an ideal quality checker for Baseline. In our experiments, we use neural network based quality checker, as we can adjust the topology of the network to control the checker accuracy.

The results in Figure 2 show that ApproxQA can always ensure the quality requirement no matter what the checker accuracy is. This is because our rollback controller can automatically consider the inaccuracy of quality checkers. For example, if the quality checker is not accurate, our rollback controller may choose a slightly smaller threshold for rollback than that for a perfect quality checker to fix more outputs so that the quality requirement is satisfied. However, without considering the inaccuracy of quality checkers, the output quality obtained with the Baseline is poor. Take *blackscholes* as an example, when the accuracy of quality checker is 81%,

Benchmark	Domain	Train Data	Test Data	Error Evaluation Metric
blackscholes	Financial	1M options	10M options	Absolute Error
fft	Signal Procession	5M fp numbers	100M fp numbers	Absolute Relative Error
inversek2j	Robotics	2M (x, y) points	50M (x, y) points	Absolute Relative Error
sobel	Image Processing	256x256 pixel images	512x512 pixel images	Absolute Relative Error
hotspot	Temperature Simulation	1M power traces	20M power traces	Absolute Error

TABLE I
BENCHMARK APPLICATION SPECIFICATIONS IN THE SIMULATION.

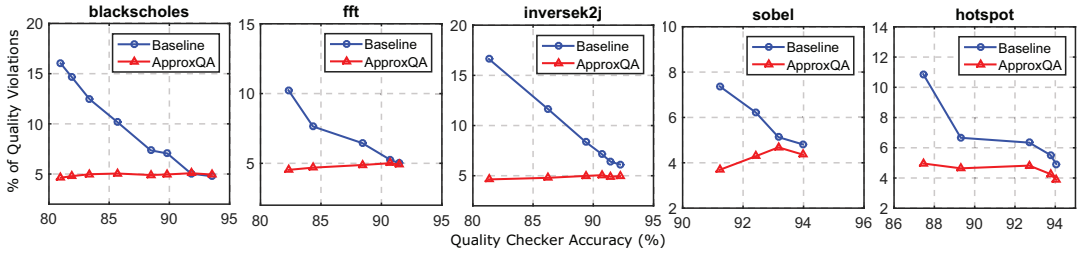

Fig. 2. The final quality when targeting 5% quality violation bound ($\epsilon = 5\%$).

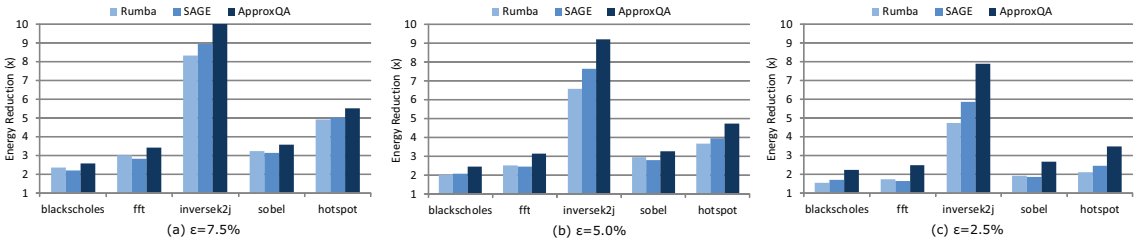

Fig. 3. Energy reduction when targeting different quality violation bounds, ApproxQA achieves 5.03x, 4.55x and 3.75x energy reduction on average.

there are about 16% quality violations while users can only accept 5%. The is due to the false negatives resulting from inaccurate quality checkers that miss many essential rollback recoveries.

C. Energy Benefits

Figure 3 shows the energy consumption with three quality management methods when targeting different violation bounds, where we regard the energy consumption of applications executed on CPU as the baseline. Our proposed ApproxQA can obtain better energy savings when compared with Rumba and SAGE. To be specific, ApproxQA can achieve 5.03x, 4.55x and 3.75x energy reduction on average compared to the baseline when targeting 7.5%, 5.0% and 2.5% quality violation bounds, while Rumba can only obtain 4.36x, 3.55x and 2.40x, respectively. Without tuning approximation modes at runtime, Rumba causes significant energy waste on the NPU with overly conservative approximation mode or on rollback recovery with overly aggressive approximation mode. On the other hand, even though SAGE saves energy on quality checking and rollback recovery, it still achieves less energy reduction. This can be explained as follows: 1) SAGE tunes approximation modes based on a greedy tree algorithm, which may not achieve satisfactory expected benefits over the long term; 2) Without rollback recovery, SAGE tends to choose an overly accurate approximation mode in order to satisfy the target quality requirement.

V. CONCLUSION

In this paper, we propose a unified quality assurance framework for approximate computing, namely ApproxQA, which takes both approximation mode tuning and rollback recovery into consideration with the help of a two-level controller. When compared with state-of-the-art quality management techniques, ApproxQA can achieve significant better energy efficiency and provide stochastic quality guarantee even when the underlying quality checker is not reliable.

ACKNOWLEDGMENT

This project was supported in part by the Hong Kong SAR Research Grants Council (RGC) under General Research Fund No. CUHK418112, in part by National Natural Science Foundation of China (NSFC) under Grant No. 61432017 and No. 61532017.

REFERENCES

[1] Q. Xu, N. S. Kim, and T. Mytkowicz, "Approximate computing: A survey," in *IEEE Design & Test*, vol 33, pp. 8–22, Feb. 2016.
[2] D. S. Khudia, et al., "Rumba: an online quality management system for approximate computing," in *International Symposium on Computer Architecture (ISCA)*, 2015.
[3] R. Ye, et al., "On reconfiguration-oriented approximate adder design and its application," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 48–54, 2013.
[4] H. Esmaeilzadeh, et al., "Neural acceleration for general-purpose approximate programs," in *International Symposium on Microarchitecture (MICRO)*, 2012.
[5] Q. Zhang, et al., "Approxit: An approximate computing framework for iterative methods," in *Design Automation Conference (DAC)*, pp. 1–6, 2014.
[6] M. Samadi, et al., "Sage: Self-tuning approximation for graphics engines," in *International Symposium on Microarchitecture (MICRO)*, pp. 13–24, 2013.
[7] W. Baek and T. M. Chilimbi, "Green: a framework for supporting energy-conscious programming using controlled approximation," in *ACM Sigplan Notices*, 2010.
[8] M. A. Laurenzano, et al., "Input responsiveness: using canary inputs to dynamically steer approximation," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2016.
[9] B. Grigorian, et al., "Brainiac: Bringing reliable accuracy into neurally-implemented approximate computing," in *High Performance Computer Architecture (HPCA)*, 2015.
[10] S. Marsland, "Machine learning: An algorithmic perspective," 2009.
[11] D. Mahajan, et al., "Prediction-based quality control for approximate accelerators," *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2015.
[12] T. Wang, et al., "On effective and efficient quality management for approximate computing," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2016.