On Effective and Efficient Quality Management for Approximate Computing

Ting Wang[†], Qian Zhang[†], Nam Sung Kim[§] 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

[§]University of Illinois, Urbana-Champaign
Email: {nskim}@illinois.edu

ABSTRACT

Approximate computing, where computation quality is traded off for better performance and/or energy savings, has gained significant tractions from both academia and industry. With approximate computing, we expect to obtain acceptable results, but how do we make sure the quality of the final results are acceptable? This challenging problem remains largely unexplored. In this paper, we propose an effective and efficient quality management framework to achieve controlled quality-efficiency tradeoffs. To be specific, at the offline stage, our solution automatically selects an appropriate approximator configuration considering rollback recovery for large occasional errors with minimum cost under the target quality requirement. Then during the online execution, our framework judiciously determines when and how to rollback, which is achieved with cost-effective yet accurate quality predictors that synergistically combine the outputs of several basic light-weight predictors. Experimental results demonstrate that our proposed solution can achieve 11% to 23% energy savings compared to existing solutions under the target quality requirement.

CCS Concepts

•**Hardware** → **Power and energy;**

Keywords

Approximate Computing; Quality Management

1. INTRODUCTION

Despite the advances in semiconductor technologies and circuit design techniques, the overall energy consumption of computer systems is still rapidly growing at an alarming rate in order to process an ever-increasing amount of information. Considering that a large amount of emerging applications (e.g., Recognition, Mining and Synthesis (RMS)) are inherently error-resilient, approximate computing [1], where computation quality is traded off for better performance and/or energy gains, has become one of the most promising energy-efficient design techniques for such kind of workloads.

Even though error-resilient RMS applications can accept some inaccuracy, it is still essential to be able to control approximation errors in a disciplined manner to ensure satisfactory user experience. With approximate computing, the quality of the output is highly dependent on the input patterns, and it is very difficult, if not impossible, to select an appropriate approximator configuration that can always satisfy the target quality requirement. Rollback recovery is therefore needed when the quality violation occurs [2, 3].

Grigorian *et al.* [2] utilizes the application specific light-weight checkers to detect quality violations and then re-computes with more accurate approximator configurations (or the fully accurate one) until the target quality requirement is satisfied. However, since the proposed solution always starts with the most inaccurate approximator configuration, frequent rollback recoveries may occur, causing significant performance and energy overheads. On the other hand, if an overly accurate approximator configuration is selected, the potential for energy efficiency of approximate computing is not fully utilized. Rumba [3] resorts to the light-weight quality predictors to detect large approximation errors and conducts rollback recovery with accurate re-execution when the predicted error is larger than a threshold. Three light-weight quality predictors are presented, which are based on the linear model, the decision tree and the moving average, respectively. These simple predictors, however, may suffer from low prediction accuracy when the application behavior is complicated. For inaccurate predictions, false positives would result in unnecessary rollback recoveries that damage the benefit of approximate computing while false negatives would degrade the computation quality as essential rollback recoveries are missed.

Generally speaking, two problems remained to be solved for the quality management of approximate computing: 1) Quality checkers need to be designed for efficiently detecting the quality violations; 2) considering rollback recovery for quality violations, approximator configurations should be carefully determined. To tackle the above problems, in this paper, we propose a novel quality management framework for approximate computing. To be specific, at the offline stage, our framework automatically determines an appropriate approximator configuration considering rollback recovery for large errors with minimum cost based on the target quality requirement. Then, at the online stage, we conduct rollback recovery based on our proposed cost-effective yet accurate quali-

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA

DOI: http://dx.doi.org/10.1145/2934583.2934608

ty predictors, which are achieved by synergistically combining the outputs of several basic light-weight predictors through the *majority voting*, *boosting* and *stacking* techniques. While our quality predictors consume slightly more energy when compared to the ones in [3], the prediction accuracy is considerably better and the overall energy-efficiency of the proposed solution is much higher than the existing solutions.

The rest of the paper is organized as follows. In Section 2, we discuss our motivation and related works. Section 3 gives an overview of our framework. Next, We introduce our framework in detail in Section 4 and Section 5. Experimental results are presented in Section 6. At last, Section 7 concludes the paper.

2. RELATED WORKS AND MOTIVATION

Many software and hardware techniques for approximate computing have been proposed to exploit the error-resilience capabilities of RMS applications. At the software layer, approximation techniques include the programming language level code transformations [4] and the algorithm level relaxations [5, 6, 7]. At the hardware layer, many approximate arithmetic unit designs (e.g., [8, 9, 10]) and approximate accelerators [11, 12] were proposed. While significant energy gains were shown with slight quality losses, none of them can achieve controlled quality-energy tradeoff and there may exist large occasional errors from time to time. These large occasional errors can significantly degrade the user experience [3].

To mitigate the above problem, SAGE [5] and Green [13] employ the sampling technique for quality checking. That is, they compute an accurate result once every N invocations and compare it against the approximate one. If their difference is larger than a pre-determined threshold, they would use more accurate approximator configurations (if any) for the following computations. While effective in many cases, since it is unrealistic to check every invocation, the quality for those unchecked invocations cannot be ensured. In addition, due to the lack of recovery support, previous quality violations cannot be compensated. Therefore, rollback recovery is indispensable when quality violations [2, 3] occur. More importantly, rollback recovery in fact provides the opportunity to further improve the energy-efficiency of approximate computing. For example, Rumba [3] has shown that only a small fraction (less than 20%) of output elements see large approximation errors. As rollback recovery can fix such errors, we can employ more aggressive approximator configuration (with less computing effort) than that without rollback recovery, and achieve better energy-efficiency under the same target quality constraint.

Unlike the checkpointing and rollback recovery mechanism widely used in the fault-tolerant computing domain, in approximate computing, usually we cannot exactly tell whether a quality violation occurs or not, simply because it would violate the very purpose of the approximation (by comparing with accurate results). Consequently, we have to rely on light-weight quality checkers to decide whether to rollback or not and the efficiency of quality checking has a significant impact on the benefits of approximate computing.

Generally speaking, a good quality checker should be accurate and light-weight in nature. This is because the output quality is highly input-dependent and every output element should be monitored. Quality checkers with high energy overheads would kill the benefits of approximation. There were some quality checkers proposed in the literature. The ones presented in [2] are rather application-specific and cannot be generalized for other applications. Recently, Rumba [3] resorts to prediction-based methods for quality checking. The three proposed quality predictors are based on the linear model, the decision tree and the exponential moving average (EMA), respectively. While they are shown to be quite effective for determining when to rollback in their experimental results, we argue that these quality predictors may suffer from low prediction accuracy when the application behavior is complicated and we detail the reasons in the following.

Obviously, the linear model-based quality predictor is applicable for those applications where the *decision boundary*[1] for rollback or not is linear. The prediction accuracy for non-linear problems, which cover the majority of real applications, hence cannot be guaranteed. The decision tree can be used to form complex *decision boundary* when the depth of the tree is high. However, considering the energy overhead of the quality predictor, we have to restrict the depth of the tree (in Rumba [3], the tree depth is at most 7), which limits its capability. The EMA-based quality predictor estimates approximation errors by observing the difference between the current output and the moving average of existing ones, under the assumption of the similarity between adjacent outputs. However, depending on the applications, adjacent elements may differ significantly, and then the prediction accuracy would be quite low.

Motivated by the above, in this paper, we propose to synergistically combine several basic quality predictors to achieve the light-weightiness and the high prediction accuracy simultaneously. The key design principle lies in the fact that each individual basic predictor may make wrong quality predictions for some instances but the likelihood for a strategic combination to be wrong is dramatically reduced. To enforce the synergistic combination, for each basic quality predictors used in our system, we train them with different input patterns such that the correlations among them are low, and then combine them with different mechanisms.

3. FRAMEWORK OVERVIEW

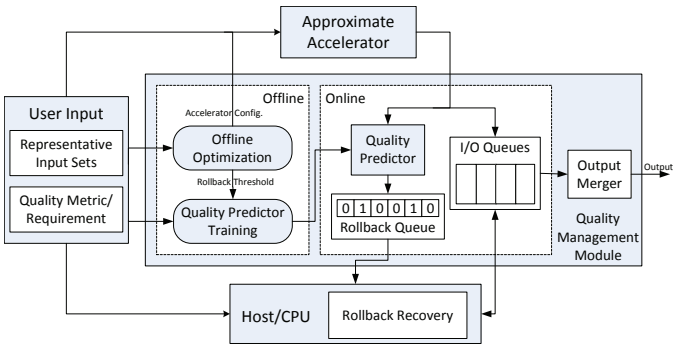

Figure 1: The proposed quality management framework.

Our quality management framework for approximate computing, as shown in Fig. 1, targets at the host-accelerator system, which is commonly used in today's high-performance computing systems. The host is often a CPU while the accelerator can be any kind of co-processors, e.g., GPUs or the approximate accelerators presented in [11, 12]. For the ease of explanation, we assume that an approximate accelerator is used and it serves as the approximator in the system[2]. The dedicated quality management module in Fig. 1 determines the approximator/accelerator configurations, controls the quality prediction and determines whether to rollback or not. The CPU conducts rollback recovery with exact re-executions

[1]Decision boundary is a hyperplane that partitions the underlying vector space.

[2]Note that, however, our framework is applicable to other platforms as well. If the approximation is not achieved with the approximate hardware units, users could provide tunable programs as the approximators. For example, when GPUs are used as the accelerators, users can employ software approximation techniques such as the loop perforation and function approximations [13].

of the original programs to correct quality violations when the rollback operations are triggered.

The overall quality management framework consists of both offline and online operations. At the offline stage, users need to provide representative input datasets similar to those used in the application test suite, the target quality requirement and the application-specific quality evaluation metric. Based on these information, our offline optimization step will determine the optimal accelerator configuration and the corresponding error threshold for rollback recovery with minimum energy cost. After that, we will train the quality predictors using the obtained rollback threshold.

We augment the approximate accelerator with an error predictor module for detecting quality violations. At the online phase, our quality predictors operate simultaneously with the approximate accelerator. The rollback decisions issued by quality predictors are put in a FIFO *rollback queue* in Fig. 1 ('1' for rollback, and '0' otherwise). These bits are sent to the host sequentially and then the CPU fixes quality violations by re-executing the original program accurately. The outputs of the approximate accelerator and the CPU are put in the I/O queues. Finally, the output merger module will choose the result for the final output either from the approximate accelerator or the CPU based on the bits stored in the rollback queue.

Needless to say, the key challenges for effective and efficient quality management include: i) how to do offline optimization to determine the approximator configuration and the corresponding rollback threshold and ii) how to design effective and efficient quality predictors. These two parts are detailed in the following.

4. OFFLINE OPTIMIZATION

As error resilient RMS applications are usually data-parallel in nature (e.g., image processing), whose final quality loss is often measured by the mean error of all output elements (e.g., pixels in the image), the output elements with larger errors contribute more to the final quality loss. As a result, rollback recoveries for the largest error elements achieve the best quality remedy. We define that the rollback recovery is issued when the approximation error of the current output element is greater than a pre-defined error threshold. As shown in Section 2, with rollback recovery, we can employ more aggressive approximator configuration than that without rollback recovery to achieve better energy efficiency under the same target quality constraint. However, we cannot relax the computations too aggressively, otherwise, frequent rollback recoveries are required. We try to choose the optimal approximator configuration and the corresponding error threshold for rollback recovery so that the total execution cost is minimized.

We assume that the representative input set I contains k input elements in total (e.g., the input image contains k pixels). The resilient program can be executed with n approximator configurations $AC = \{A_1, A_2, \cdots, A_n\}$. The users provide the target quality requirement Q and the quality evaluation function $q(E_{A_i})$, where E_{A_i} is an array containing approximation errors of k elements executed with the approximator configuration A_i. The total cost C_i is the sum of the approximator A_i execution, the quality prediction and the rollback recovery costs.

Our offline optimization works as follows: we iterate over all the approximator configurations in AC, for each A_i, if the initial quality $q(E_{A_i})$ can meet the quality requirement Q, then no rollback recovery is required. The corresponding error threshold $T_i = \infty$ and the total cost C_i is only the approximator execution cost. Otherwise, we need to fix a certain number of output elements for meeting the quality requirement. We fix the approximation errors in E_{A_i} from the largest to the smallest by setting the errors to be zero, until the quality requirement is satisfied, that is $q(E_{A_i}) \geq Q$. Then the error threshold T_i corresponding to the approximator configuration A_i is the largest approximation error remained in E_{A_i}, as the elements with the approximation errors larger than it need to be fixed for meeting the quality requirement. After sweeping over all the approximator configurations in AC, we choose the approximator configuration A_j and the corresponding error threshold T_j with the minimum total cost C_j.

5. QUALITY PREDICTOR DESIGNS

Before we discuss the design of quality predictors, we first define the input-output interfaces of the basic predictors. As they are used to predict the approximation errors for rollback recovery, the output of the basic predictor can be the approximation error of the output element or one bit indicating rollback or not (e.g., the approximation error is larger/smaller than the error threshold). As the approximation error of the element comes from the applied approximation during the program execution, the inputs of the basic predictors can be intermediate results before and/or after the approximation. In the system shown in Fig. 1, the approximate accelerator contributes to the final approximation error, so the inputs of the basic predictors can be the *input*, *output* and *intermediate* results of the approximate accelerator. As our quality predictors are the combinations of several basic quality predictors, the inputs of our quality predictors are the inputs of all the basic quality predictors. The outputs of our quality predictors can also be the approximation errors or the rollback decisions according to the combination mechanism. If the outputs of the quality predictors are the approximation errors, then an extra step of comparing them with the error threshold is needed for determining rollback or not.

5.1 Basic Quality Predictors

As discussed in Section 2, quality predictors should be designed both accurate and light-weight in nature. Therefore, we cannot use traditional models (e.g., decision trees and neural networks) due to their high energy overhead. Instead, we try to combine the outputs of several light-weight basic predictors (e.g., the linear model-based predictor) for both accuracy and low overhead. The two basic quality predictors are detailed in the following.

5.1.1 Linear Model-based Basic Predictor

This model computes a linear function of its input as the predicted value. Assume there are n input data, denoted by $\mathbf{x}$, then the predicted approximation error for the element is:

$$e = w_n \times x_n + w_{n-1} \times x_{n-1} + \cdots + w_1 \times x_1 + c,$$

in which the parameters $\mathbf{w}$ and c are determined by offline training.

5.1.2 Table-based Basic Predictor

A table-based light-weight basic quality predictor stores the contents (e.g., a single bit value for rollback decision or several bits denoting the approximation error) in a table indexed by a hash over the input bits. The key points for designing such a hash function for the predictor include: i) it should utilize all the bits of the input; ii) it can be implemented in the hardware efficiently. Based on these considerations, we plan to resort to Linear Feedback Shift Register (LFSR) for hashing the input bits. LFSR is a shift register whose input bit is the linear function of its previous state, in which exclusive-or (XOR) is the most commonly used linear function. We generate input hash with LFSR-based Toeplitz hash proposed in [14], in which the input bits of the predictor control the accumulations of the LFSR states, as shown in Fig. 2. If the input bit is 1, the corresponding state of LFSR is accumulated in the accumulator

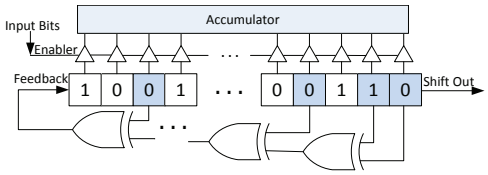

Figure 2: LFSR for generating the input hash.

register, otherwise not. After all the input bits are shifted, the value in the accumulator register is the corresponding input hash.

After obtaining the input hash, we can store the table contents of the corresponding hash in the offline stage of our quality management framework. If there are several approximation errors corresponding to one hash value, we choose to store either the largest or the average based on their prediction accuracy. If one bit for rollback decision is stored, the approximation error of the corresponding input hash is compared with the error threshold obtained in Section 4. If the approximation error is greater than the error threshold, the table content indexed by the corresponding input hash will be stored with 1, otherwise 0.

5.2 Predictor Combination

As shown in Section 2, basic quality predictors may suffer from low prediction accuracy when the application behavior is complicated. In this section, we describe three mechanisms to combine the outputs of basic predictors to enhance the prediction accuracy.

- **Voting-based Combination:** Voting-based mechanism outputs one bit for rollback decision by taking the majority of the outputs of all basic predictors. In this mechanism, we assume we have the odd number of basic predictors, whose outputs are one single-bit representing rollback or not. In this combination, at least three basic predictors are needed. The hardware implementation of the voters can be found in [15].

- **Boosting-based Combination:** This method comes from the ensemble learning [16], where several weak learners (predictors) are ensembled to achieve better learning capability. Boosting is a process of turning a weak predictor (less accurate basic predictor) into a strong predictor (more accurate predictor). We assume the outputs of basic predictors are rollback recovery decisions. This mechanism works by iteratively adding basic predictors to the final strong predictor. At the beginning, each element in the training dataset has the same weight, that is $1/n$, assuming n elements in total. After adding a new basic predictor, the element in the training dataset is re-weighted, in which the erroneous predicted elements gain weight and the successful predicted elements lose weight. After re-weighting, future basic predictors will focus more on the elements that are predicted erroneously by previous basic predictors. The outputs of the basic predictors are combined into a weighted sum upon which a sign function is applied to determine the rollback decision. The weights of the basic predictors are decided based on their prediction accuracy. In our framework, we use the linear models as the basic predictors. We continue adding basic predictors until the energy overhead exceeds the benefits.

- **Stacking-based Combination:** Stacking involves training a learning algorithm, or a meta-level predictor, to combine the predictions of several basic predictors. In stacking, the outputs of the basic predictors on a validation dataset are aggregated and combined with the known rollback decisions to create a meta-level training dataset. For example, given a validation dataset with n elements $\boldsymbol{X} = \{\boldsymbol{x_i} | i = 1, 2, \cdots, n\}$ and L basic predictors whose outputs on the element $\boldsymbol{x_i}$ are $\boldsymbol{y^i} = \{\boldsymbol{y^i_j} | j = 1, 2, \cdots, L\}$. The known rollback decision on $\boldsymbol{x_i}$ is d_i, so the meta-level training dataset is $\boldsymbol{M} = \{(\boldsymbol{y^i}, d_i) | i = 1, 2, \cdots, n\}$. Then another predictor, called the meta-level predictor, is trained on the meta-level dataset. At the test phase, the rollback decision is the output of the meta-level predictor which regards the outputs of all basic predictors as its inputs. In our framework, we choose the single-layer logistic regression as our meta-level predictor. After obtaining the model parameters, the logistic regression can be implemented with a small look up table.

Boosting- and Stacking-based combinations succeed under the theory of the ensemble learning [17]. In order to obtain the high accuracy of combinations, we need to make basic quality predictors work in a synergistic manner. To be specific, we need to train the basic quality predictors with different patterns to reduce the correlations among them such that they make wrong predictions on different instances. And then a strategic combination of them can significantly reduce the likelihood for wrong predictions. Our Boosting-based combination can automatically satisfy this principle by re-weighting elements in the training datasets. To ensure this principle for Voting- and Stacking-based combination mechanisms, we can choose different types of basic predictors and train them with different datasets.

For a combination mechanism, we can choose different numbers and types of basic quality predictors for combination. However, it is impossible for us to choose a specific number and type of basic quality predictors that can give satisfactory prediction accuracy all the time. When using our quality predictor design methods, users need to try different numbers and types of basic quality predictors on a validation dataset and choose the one with the best accuracy. Due to the energy overhead, the total number of basic predictors is usually restricted, for example, no more than 5, the cost for exploring and choosing the best number and type of basic predictors is not a big deal.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

Similar to Rumba [3], we evaluate our proposed framework with Neural Processing Unit (NPU) from [11], in which a neural network is selected and trained to mimic the execution of a region of computation-intensive code. And the different network topologies represent the approximator configurations with different accuracy levels.

Energy Modeling: We use *gem5* [18] simulator to measure the performance of our quality control system. The micro-architecture configurations of the x86_64 CPU and NPU parameters for *gem5* simulation are the same as those shown in [3], and the energy consumption is estimated by McPAT [19]. Quality predictors are implemented with Verilog and synthesized with Synopsys Design Compiler. Their energy consumptions and running time are achieved with Synopsys PrimeTime. The whole system is modeled in 32nm technology node.

Benchmarks: We select 5 benchmark applications from [11], as summarized in Table 1. The application domain, train data, test data and the application-specific quality loss metrics are listed. The initial topology of the NPU and the initial quality without rollback recovery are also given in the table. The quality loss is directly related to the output quality. For example, the 5% quality loss represents the 95% output quality. We estimate the accuracy of the predictors based on the initial NPU topology, in which 9->8->1 represents that the NPU accelerator has 9 inputs, 1 hidden layer

Benchmark	Domain	Train Data	Test Data	Quality Loss Metric	Topology	Initial Quality
blackscholes	Financial	4K inputs	16K inputs	Avg. Relative Error	6->4->2->1	90.58%
fft	Signal Procession	4K fp numbers	32K fp numbers	Avg. Relative Error	1->2->2->2	87.02%
inversek2j	Robotics	50K (x, y) points	500K (x, y) points	Avg. Relative Error	2->2->2->2	90.03%
jmeint	2D Gaming	10K 3D triangles	100K 3D triangles	Miss Rate	18->16->16->2	76.56%
sobel	Image Processing	256x256 pixel image	256x256 pixel image	Image Diff	9->8->1	83.79%

Table 1: Benchmark information, accelerator topology and initial quality without rollback recovery.

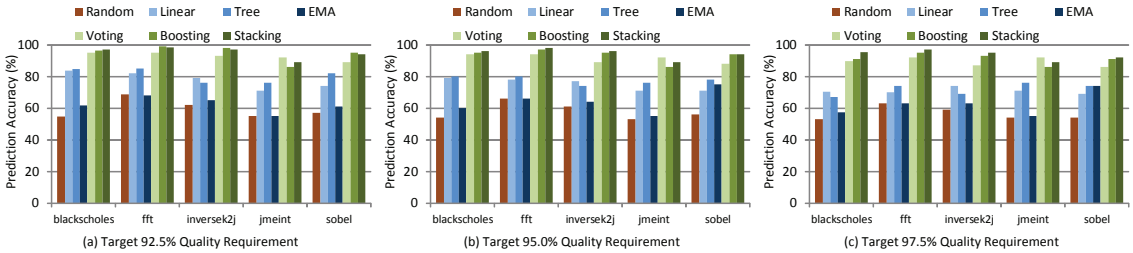

Figure 3: The prediction accuracy compared to the *Ideal* predictor when targeting different quality requirements.

with 8 neurons and 1 output. The NPU topology can be changed to obtain different computation accuracy levels. Usually, the computation accuracy can be improved by increasing the complexity of the NPU topology, e.g., increasing the number of hidden layers or neurons, but requiring more computation time and energy.

6.2 Prediction Accuracy

6.2.1 *Predictor Setup*

We use the topology listed in Table 1 to evaluate the prediction accuracy of different predictors, which should issue rollback recoveries for the elements with approximation errors larger than the error threshold and no rollback for other elements. The error threshold is obtained with the method described in Section 4, in which we restrict the approximator configuration as the NPU topology in Table 1. In order to evaluate the prediction accuracy of quality predictors, we compare the outputs of our predictors against an *Ideal* predictor which has the oracle knowledge about the approximation errors of all output elements. Therefore, it knows exactly which output element needs rollback recovery. The prediction accuracy of the other quality predictors is the percentage of the same prediction results compared with the *Ideal* predictor.

We combine the linear- and table-based basic predictors with three combination mechanisms shown in Section 5.2 to obtain three quality predictors, named as *Voting*, *Boosting* and *Stacking*. Their prediction results are presented in Fig. 3 when targeting for the 92.5%, 95.0% and 97.5% quality requirements, respectively. After exploring different numbers and types of the basic quality predictors, we choose the one with the best prediction accuracy on a validation dataset (a subset of the quality predictors training data) in our experiments. We also present the prediction accuracy of quality predictors proposed in [3], named as *Linear*, *Tree* and *EMA*, respectively. A *Random* predictor fixes a given percentage of the output elements randomly.

6.2.2 *Prediction Results*

According to the results shown in Fig. 3, we have the following observations. First of all, our predictors can achieve better prediction accuracy than the ones proposed in [3] and *Random* under different quality requirements. Take the *fft* benchmark as an example, as its output elements with high approximation errors have similar properties and can be easily separated with the ones with low approximation errors, our three quality predictors can obtain as high as 95%, 99% and 98% prediction accuracy, respectively, when targeting for 92.5% quality requirement. However, *Linear*, *Tree* and *EMA* can only obtain the prediction accuracy of 82%, 85% and 68%, respectively. And the prediction accuracy of *Random* is 69%. Secondly, when we increase the acceptable quality from 92.5% to 97.5%, all predictors' prediction accuracy levels decrease. The possible explanation is that as the quality requirement increases, more output elements are required to rollback. The rollback decision boundary becomes more complex and difficult to determine. However, our *Stacking* predictor is more robust than the other predictors, as it relies on a more sophisticated learning process to combine the basic predictors.

6.3 Energy Consumption

Our framework targets the selection of the approximator configuration considering rollback recovery and the quality predictor designs. In order to evaluate its energy efficiency, we use the CPU execution for benchmark applications as the baseline for comparison. The system energy includes the energy consumption of the CPU, the approximator and quality predictors. As the NPU topology space is large, we restrict that the neural network has at most 2 hidden layers and each hidden layer has at most 32 neurons.

6.3.1 *Impacts of the Approximator Configuration*

We show the normalized energy consumption with respect to the baseline (no approximation) in Fig. 4 when targeting different quality requirements, named as *NPU+Rollback*. We compare it with the method only tuning approximator configurations and the one only tuning error threshold for rollback, named as *NPU* and *Rumba* [3], respectively. NPU selects the topology with the minimum energy consumption that can satisfy the quality requirement. Rumba fixes the minimum number of output elements for meeting the quality requirement without changing the NPU topology. It utilizes the same topology as our method under the 95% quality requirement.

According to the results, our method, NPU+Rollback, achieves the best energy efficiency except *jmeint*. This is because we co-optimize the approximator configurations and the error threshold for rollback recovery, which is certainly superior to only optimizing one of them. NPU usually needs the approximator configuration with the higher accuracy level (e.g., more complex neural network topology) to satisfy the quality requirement, as no rollback recovery is utilized. Even though no quality prediction is needed, more complex network topology usually consumes much more energy. For example, when targeting the 92.5% quality requirement of the *fft* application, NPU consumes 37% energy of the baseline, while our method only needs 23% energy. Moreover, without rollback recovery, NPU cannot always satisfy the quality requirement. For instance, the final quality of *jmeint* is not acceptable, even though NPU achieves more energy efficiency. We cannot find a NPU topology to meet the target quality requirement, which indicates rollback recovery is indispensable for guaranteeing the quality requirement. For Rumba, when the quality requirement increases from 92.5% to

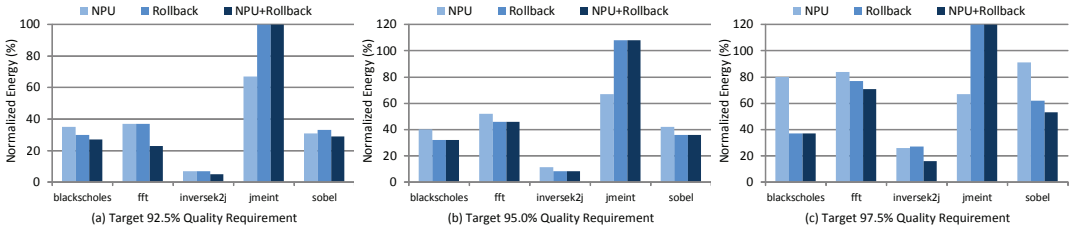

Figure 4: The normalized energy when targeting different quality requirements.

97.5%, it needs to fix more output elements. As rollback recovery with the CPU is expensive, Rumba consumes much more energy.

6.3.2 *Impacts of the Quality Predictors*

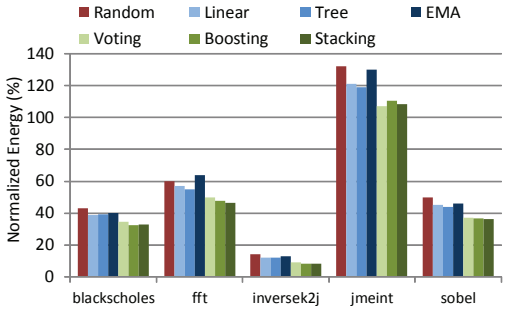

Figure 5: The normalized energy with different predictors.

In this section, we compare the impacts of different quality predictors on the system energy efficiency.

Fig. 5 shows the energy consumption of benchmark applications with respect to the baseline by applying different quality predictors when targeting the 95.0% quality requirement. We find that our predictors always obtain better energy efficiency than predictors from [3] and random rollback recovery (*Random* in the figure). Again, take the *fft* as an example, by performing rollback recovery randomly, the system consumes 60% energy with respect to the baseline. Predictors from [3] need 57%, 55% and 64% energy, respectively. However, by applying our predictors, the system only consumes 50%, 47% and 46% energy, respectively. This is because the prediction accuracy of predictors from [3] and *Random* is not satisfactory, which will produce many false positives and introduce many unnecessary rollback recoveries. As rollback recovery with the CPU is expensive in terms of energy, even though our predictors consume slightly more energy, we still achieve better energy efficiency by avoiding many unnecessary expensive rollback recoveries. However, we do not achieve energy gains from the *jmeint* benchmark. As NPU-based approximator is not good at approximating this application, many rollback recoveries are needed for ensuring the target quality requirement.

To conclude, considering both the impacts of approximator configurations and quality predictors, our framework can achieve 11% to 23% energy savings compared to existing solutions. Moreover, according to our experiments, the initial quality without rollback recovery of the selected NPU topology is slightly lower than the target quality requirement. So we can reach a conclusion that the most energy efficient way of applying approximate techniques is to select an approximator configuration whose quality is slightly worse than the target quality requirement, and then obtains the quality requirement by performing rollback recovery to fix a certain number of output elements.

7. CONCLUSIONS

In this paper, we propose an effective and efficient quality management framework for approximate computing, in which we synergistically combine the outputs of several basic light-weight predictors to obtain more accurate predictions. Our solution can achieve 11% to 23% more energy gains under the given target quality requirement, when compared to the state-of-the-art techniques.

8. 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, in part by NSFC/RGC Joint Research Scheme under Grant No. N_CUHK444/12, and in part by National Science Foundation (CCF-1600896/0953603).

9. 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] B. Grigorian, N. Farahpour, and G. Reinman "Brainiac: Bringing reliable accuracy into neurally-implemented approximate computing," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 615–626, 2015.

[3] D. S. Khudia, et al., "Rumba: an online quality management system for approximate computing," in *International Symposium on Computer Architecture (ISCA)*, pp. 554–566, 2015.

[4] S. Sidiroglou, et al., "Managing performance vs. accuracy trade-offs with loop perforation," in *ACM SIGSOFT symposium and European conference on Foundations of software engineering*, pp. 124–134, 2011.

[5] M. Samadi, et al., "Sage: Self-tuning approximation for graphics engines," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 13–24, 2013.

[6] H. Hoffmann, et al., "Using code perforation to improve performance, reduce energy consumption, and respond to failures," *Computer Science and Artificial Intelligence Laboratory Technical Report*, 2009.

[7] Q. Zhang, et al., "Approxit: An approximate computing framework for iterative methods," in *ACM Design Automation Conference (DAC)*, pp. 1–6, 2014.

[8] 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.

[9] N. S. Kim, et al., "Multiplier supporting accuracy and energy trade-offs for recognition applications," *Electronics Letters*, pp. 512–514, 2014.

[10] Q. Zhang and et al., "ApproxANN: an approximate computing framework for artificial neural network," in *IEEE/ACM Design, Automation & Test in Europe Conference (DATE)*, pp. 701–706, 2015.

[11] H. Esmaeilzadeh, et al., "Neural acceleration for general-purpose approximate programs," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 449–460, 2012.

[12] S. Venkataramani, et al., "Quality programmable vector processors for approximate computing," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2013.

[13] W. Baek and T. M. Chilimbi, "Green: a framework for supporting energy-conscious programming using controlled approximation," in *ACM Sigplan Notices*, vol. 45, pp. 198–209, 2010.

[14] P. Deepthi and P. Sathidevi, "Design, implementation and analysis of hardware efficient stream ciphers using lfsr based hash functions," *Computers & Security*, vol. 28, no. 3, pp. 229–241, 2009.

[15] B. Parhami, "Voting networks," *Reliability, IEEE Transactions on*, vol. 40, no. 3, pp. 380–394, 1991.

[16] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*, pp. 1–15, Springer, 2000.

[17] Z.-H. Zhou, "Ensemble learning," in *Encyclopedia of Biometrics*, pp. 270–273, Springer, 2009.

[18] N. Binkert, et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[19] S. Li, et al., "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *International Symposium on Microarchitecture(MICRO)*, pp. 469–480, 2009.