Improving Coverage and Reliability in Approximate Computing Using Application-Specific, Light-Weight Checks

Beayna Grigorian, Glenn Reinman
UCLA Computer Science Department
{bgrigori, reinman}@cs.ucla.edu

Abstract

Prior art in approximate computing has extensively studied computational resilience to errors. However, existing approaches often rely on static techniques, which potentially compromise ***coverage*** *and* ***reliability****. Our approach, on the other hand, decouples error analysis of the approximate accelerator from quality analysis of the overall application. We use high-level, application-specific metrics, or* ***Light-Weight Checks (LWCs)****, to gain coverage by exploiting imprecision tolerance at the application level. Unlike metrics that compare approximate solutions to exact ones, LWCs can be leveraged dynamically for error analysis and recovery, providing guarantees on worst-case application-level error. In our* ***platform-agnostic*** *approach, these light-weight metrics are integrated directly into the application, enabling compatibility with any approximate acceleration technique. Our results present a case study of dynamic error control for inverse kinematics. Using software-based neural acceleration with LWC support, we demonstrate improvements in coverage, reliability, and overall performance.*

1. Introduction

As a response to increasingly important concerns of power and energy consumption, along with lofty performance goals, computer architecture research has gravitated towards methodologies that provide inexact, yet acceptable solutions. Taking inspiration from techniques in fault tolerance, *approximate computing* [16] has evolved into a means for not only tolerating, but also embracing imprecision. An "approximate accelerator", or compute unit based on approximate hardware (e.g. low-precision functional unit) or software (e.g. algorithmic shortcuts), leverages imprecision tolerance (i.e. "slack" in computational accuracy) to improve performance and energy consumption.

Prior art has extensively studied computational resilience to errors in both hardware [37, 11, 1, 5, 12] and software [19, 28, 25, 29, 26]. These existing approaches often couple the overall application quality with the accuracy of the approximate accelerator. Though this allows for efficient quality analysis by way of offline, static techniques [20], *coverage* and *reliability* are potentially compromised.

Coverage is lost when cases that are statically determined to lead to unacceptably inaccurate solutions are exempted from approximation. This may occur, for instance, with inputs outside the training data range or with the use of otherwise inaccurate approximate accelerators. One approach to combat this loss in coverage is to exploit algorithmic and cognitive resilience [5], such as with high performance workloads from Recognition, Mining, and Synthesis (RMS) [9], potentially uncovering larger slack in accuracy. Related work includes methodologies that employ high-level, application-specific metrics for assessing output quality [1, 28, 22, 8]. These metrics, however, often determine quality degradation of approximate solutions by measuring against exact solutions, such as by finding the average difference between images produced by approximate and exact versions of an image processing application.

Dynamic *reliability* entails providing absolute guarantees for satisfying quality of service (QoS) constraints. With static quality analysis, guarantees on worst-case accuracy cannot be provided unless the space of possible inputs is exhaustively explored. As this is an inherent issue for approximation techniques, instead of providing worst-case guarantees and ensuring dynamic reliability, measures are often taken to statically mitigate low-quality results [13]. Despite its statistical unlikelihood, however, a low-quality result could still render an application's output meaningless. This would be unacceptable for circumstances involving strict QoS expectations. As such, we recognize a general need for mechanisms that allow dynamic error analysis and control (much like with circuit-level errors [11]). Moreover, since acceptable ranges of error could vary across different uses of an application, user-based specification of QoS constraints must also be featured. An example of this would be using an inverse kinematics application for robot control, which requires high precision for performing surgery, yet tolerates lower precision for moving large blocks.

With coverage and reliability in mind, we have designed an approach that decouples error analysis of the approximate accelerator from quality analysis of the overall application. Rather than relying on statically-constructed models of error distribution [20, 13, 7], we dynamically guarantee the worst-case error for an application, as well as gain additional coverage from leveraging slack that may not be evident at the level of the approximation unit.

For achieving dynamic error detection, it is unreasonable to compare an application's exact output to the approximate output, as the generation of the exact output would merely add overhead, defeating the purpose of the approximation. As a more high-level approach to quality analysis, we employ **Light-Weight Checks (LWCs)**. The key inspiration behind this approach is that while finding a solution may be complex,

Table 1: Examples of applications, algorithms, domains, and LWCs

Application	Sample Algorithm	Application Domains	Light-Weight Check (LWC)
Inverse Kinematics	Cyclic Coordinate Descent	Robotics, Graphics, Gaming	Forward Kinematics
State Estimation	Kalman Filter	Navigation, Signal Processing, Finance	Measurement Comparison
Physics-Based Simulation	Gilbert-Johnson-Keerthi	Gaming, Fluid Dynamics, Control Systems	Energy Conservation
Image Denoising	Total Variation Minimization	Computer Vision, Medical Imaging	Universal Image Quality Index

checking the quality of that solution could be simple. By definition, an LWC is a metric that is **light-weight** relative to the application, allowing it to be utilized dynamically instead of solely for the purpose of offline error analysis. This metric is also **application-specific**, yet **algorithm-independent**, meaning it is not necessarily bound by a specific implementation. Although LWCs are unique to each application, they may be fairly easy to determine for certain categories of applications (e.g. problems that could be solved using iterative refinement). Further descriptions and examples of LWCs are presented in Section 2.

Once an LWC has been established for a given application, the metric is integrated directly into the application, allowing for a **platform-agnostic** implementation that is compatible with any approximate acceleration technique. An interface also exists for specification of a user-defined QoS threshold based on the LWC. For each set of inputs to the application, the approximate accelerator is initially executed and the resulting outputs are tested using the LWC. In case of unacceptable quality loss, recovery is initiated by performing an exact computation. Otherwise, the approximation is accepted and the program moves onto the next set of inputs, allowing for performance gains and energy savings without loss of reliability.

The following are the overall benefits of our approach:

- ▶ Reliable, dynamic guarantees on user-specified QoS;
- ▶ Better coverage for acceptable approximations;
- ▶ Platform agnosticism using application-based metrics;
- ▶ Negligible overhead using light-weight error checking.

2. Methodology

2.1. Light-Weight Checking

Static error analysis, such as establishing confidence intervals [7] or finding average error across a range of training values [13], is commonly used to control the usage of approximate accelerators. In our approach, we advocate dynamic quality analysis and approximation control by way of LWCs. The key characteristics of LWCs are (1) application-specific quality assessment, and (2) light-weight evaluation relative to the application.

LWCs are not specific to any particular algorithm or implementation. They are only specific to the type of application. For instance, physics-based simulation could involve many variations on algorithms for collision detection and constraint solving. However, an LWC for analyzing simulated scenes would remain relevant across the different implementations.

By nature, these LWCs could be based on values accessed from within the application (i.e. inputs, approximated outputs, and intermediate values), or via external values (e.g. additional sensor-based inputs). In a mobile robot application, for instance, an approximate accelerator used for augmenting the navigation program may be provided with supplemental sensory feedback, such as rangefinder data. This feedback may or may not be directly used for planning high-level navigation solutions, but it could be helpful in warning the robot about possible nearby collisions, rendering it an ideal LWC.

Similar to existing methods for error control in approximate computing [1, 19], it is the responsibility of the user to define these light-weight quality metrics. However, while LWCs are application-specific, they may be found fairly easily for certain categories of applications. For instance, problems that could be solved using iterative refinement, such as inverse kinematics [35], may use the refinement criteria as an LWC. Likewise, an LWC for image quality assessment could be reused for various image processing applications, such as image enhancement, restoration, manipulation, etc.

Once an LWC has been established for a given application, in order to provide a platform-agnostic solution, the application code is modified to execute the following:

(1) Call approximate accelerator
(2) Evaluate LWC and determine QoS
(3) ***If QoS constraint is met***: Continue to next input
(4) ***Else***: Recover – reprocess input with exact computation

Note that the LWC is checked once for each input, and not multiple times, as in incremental refinement [25]. Also, overall quality of the application is assessed independently of the approximate accelerator, and recovery is dynamically initiated as necessary. While overall performance is undoubtedly affected by the performance and accuracy of the chosen approximate accelerator, it is also largely impacted by the specified QoS constraint and the nature of the inputs being processed.

2.2. Application Examples

Figure 1 contains examples of applications and their corresponding LWCs. Inverse kinematics based on the Cyclic Coordinate Descent method [35] is a well-known iterative algorithm for computing joint angles given target positions. This technique is commonly applied in robotics (e.g. robotic arm motion) and graphics (e.g. character animation). By adjusting joints in a series of steps, this iterative method continuously refines a solution until the end effector reaches the goal within a given tolerance bound. The error in the solution at each iteration is gauged using a quick forward kinematics computation. This forward kinematics check is inherently part of the iterative algorithm and is computationally simpler than inverse kinematics. It is therefore application-specific and light-weight, and serves as a valid LWC. Note that if the application entails hardware sensors for robotic motion, for example, sensory feedback would serve as an alternative LWC.

Another application that could benefit from LWC support is state estimation using the Kalman filter [21]. The Kalman filter is a recursive technique for state estimation of linear systems

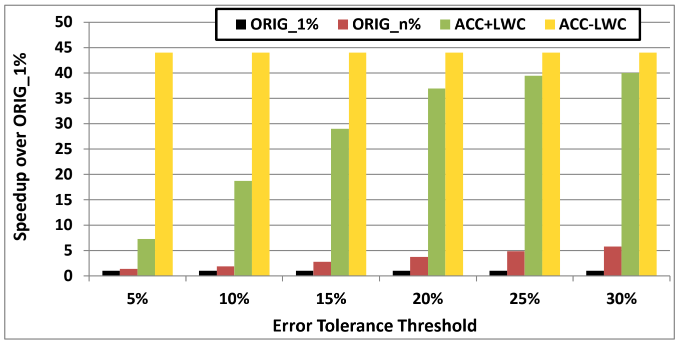

Figure 1: Performance comparison of approximation schemes

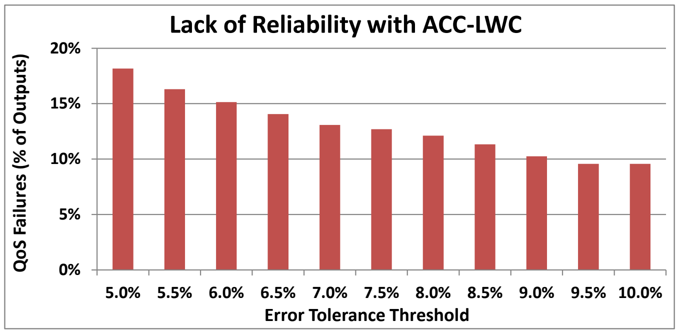

Figure 2: Reliability issues of approximation without LWC

with applications in object tracking, localization, multimodal data fusion, exchange rate modeling, voltage estimation, and much more. Algorithmically, it includes a "prediction" step followed by an "update" step for comparing the prediction to actual measurements (e.g. from sensory feedback) and using a gain matrix to adjust the state accordingly. Though the computation of the gain is expensive, the measurement comparison is light-weight and may be leveraged as an LWC.

With physics-based simulation, numerous algorithms could be used for object motion, collision detection, collision response, and constraint solving. However, regardless of those algorithms, the LWC metric would entail analysis of the simulated scene itself. Prior work in accelerating physics engines [37] has found energy conservation across scenes to be a useful metric for analyzing approximation error. Since solving for energy is considerably more light-weight than the entire simulation process, this metric could serve as an LWC.

Our final example is for image denoising (e.g. using total variation minimization [34]). Quality assessment for the outputs of this application could rely on an overarching image analysis metric, such as the Universal Image Quality Index (UIQI) [36]. Similar to the Peak Signal-to-Noise Ratio (PSNR), which is commonly used for application-specific quality analysis, UIQI is generally applicable to images. UIQI is also easy to compute dynamically because it may serve as a comparison metric based on input-output images without the need for an exact output. However, unlike traditional error summation methods, UIQI models image distortion as a combination of loss of correlation, luminance distortion, and contrast distortion, allowing it to provide more meaningful comparisons across different images. UIQI could therefore be leveraged as a general-purpose LWC for image processing applications.

3. Evaluation

3.1. Experimental Setup

To demonstrate the benefits of dynamic quality analysis using LWCs, we examine inverse kinematics for a 3-joint arm. In this case study, the application continuously receives target x-y coordinates as input (e.g. for continuous robotic motion or character animation). Though the application operates on a non-data-parallel input stream, the computation for each input could be accelerated using approximate computing techniques. Also, the **error tolerance threshold represents the maximum percentage of error the user is willing to accept for any given input**, where error is measured as the distance (relative to the total length of the arm) from the end effector to the target location.

Though our approach is compatible with other approximation techniques, for the purposes of this paper we exemplify benefits through software-based neural acceleration [13] (i.e. integrating a trained neural network directly into application code). To approximate inverse kinematics, a neural network (NN) is trained using 7500 input-output sets. Each set contains 2 inputs for the target x-y coordinates and 3 outputs for the joint angles; all values are expressed as floating point numbers. The NN has a total of 4 layers: 1 input layer with 2 neurons, 2 hidden layers of 8 neurons each, and 1 output layer with 3 neurons. For our performance evaluations, we use 1024 input-output sets, which are completely distinct from the training data. With respect to this evaluation data, the average error of our trained 8x8 NN is 4.1% with a standard deviation of 3.6%.

Our experiments are run on a 2GHz quad-core Intel Xeon E5405 processor. In our results, we compare the following schemes:

- **ORIG_1%**: Original benchmark with 1% set threshold.
- **ORIG_n%**: Original benchmark with user-specified adjustable threshold "n", which represents QoS as maximum error being tolerated.
- **ACC+LWC**: Benchmark integrated with NN and LWC, where the LWC is used to determine when to make use of exact computation (i.e. revert to *ORIG_n%* computation).
- **ACC-LWC**: Benchmark integrated with NN, but no LWC is used; recovery is therefore never initiated and the approximate solution is always employed, regardless of error.

Note that, as iterative inverse kinematics repeatedly refines its solution to match a given error threshold, the original benchmark could either be statically set to an "acceptable" threshold (i.e. *ORIG_1%*), or could have the threshold be adjustable based on user specification (i.e. *ORIG_n%*), which resembles a software-based implementation of the *incremental refinement* technique [25].

3.2. Experimental Results

3.2.1. Performance

Performance corresponding to each of the schemes described

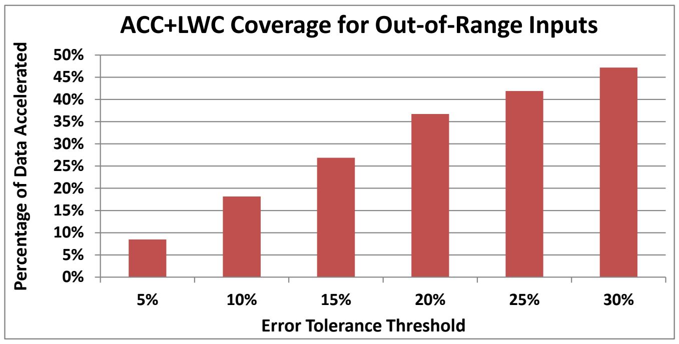

Figure 3: Amount of coverage for out-of-range inputs

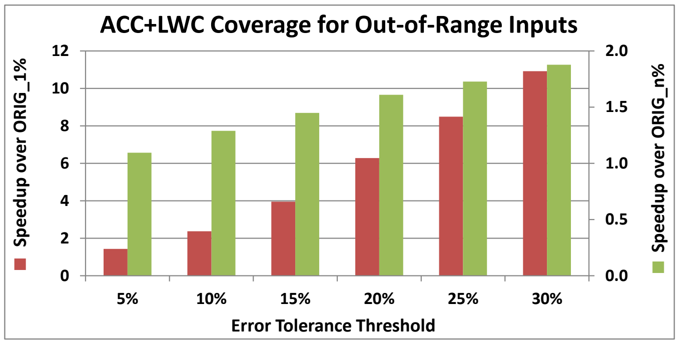

Figure 4: Speedup from coverage for out-of-range inputs

in Section 3.1 is represented in Figure 1. On average, compared to the original benchmark with a set threshold of 1% (*ORIG_1%*), we see performance gains of 3.4X for software-based incremental refinement (*ORIG_n%*), 28.6X for neural acceleration with LWC support (*ACC+LWC*), and 44X for neural acceleration without LWC support (*ACC-LWC*).

Approximation via software-based incremental refinement (*ORIG_n%*) and approximation with LWC support (*ACC+LWC*) are comparable techniques because they are both platform-agnostic approaches for dynamically ensuring QoS. Our results for inverse kinematics demonstrate an average improvement of 8.5X for *ACC+LWC* relative to *ORIG_n%*. Although the main source of the performance gain for *ACC+LWC* is the neural approximation, the efficiency of its dynamic error analysis is due to LWC support, which involves a single quick check for each approximation. The incremental refinement in *ORIG_n%*, on the other hand, loses efficiency as it satisfies QoS constraints by continuously checking its solution at each iteration, incurring unnecessary costs that render the light-weight nature of its checks ineffective.

In these result, *ACC-LWC* provides a notion of ideal gains due to its inability to provide guarantees on worst-case accuracy. From these trends, we see the performance of *ACC+LWC* rapidly approaching that of *ACC-LWC* for tolerance thresholds ranging from 5%-30%, thereby emphasizing the light-weight impact of our dynamic error management approach. The majority of the overhead preventing *ACC+LWC* from achieving ideal *ACC-LWC* performance is caused by recovery stages using exact computation. Though the need for recovery could be reduced by using more accurate approximation techniques, the higher accuracy would sacrifice performance in the accelerator. Excluding the overhead for recovery, the remaining overhead, which is for evaluating the LWC, is negligible. In this inverse kinematics benchmark, for instance, we observe LWC overhead of less than 1% on average. Though the overhead of an LWC will vary across different applications due to its application-specific nature, it will remain beneficial for dynamic error analysis so long as it is light-weight relative to the overall application.

3.2.2. Reliability

As previously described, *ACC-LWC* achieves large, idealistic performance gains due to a lack of LWC support and an inability to initiate dynamic recovery. When the output quality of an approximate accelerator is statically analyzed, the assumption is that for the majority of executions, the approximation unit will have errors near its average error value (i.e. within a standard deviation given a normal distribution). Based on this assumption, it would be acceptable to use an approximate accelerator for an application if its average error falls within a given tolerance threshold, assuming the inputs also fall within an acceptable range (e.g. the range of the training data). However, though statistically less likely, there would still be cases where the error exceeds the threshold, resulting in a QoS failure.

In Figure 2, we present the QoS failures that occur when an NN with 4.1% average error is used for accelerating inverse kinematics with tolerance thresholds of 5%-10%. On average, we see QoS failures in 13% of the outputs. Notably, for thresholds of 8%-10%, which are past a standard deviation of the average error, there are QoS failures in 10% of the outputs, highlighting the unreliability of approximation without LWC support. Furthermore, while the NN used in this example approximates the entire application, there may be cases where only a portion of the application is approximated. For instance, if the floating point operations are computed using an approximate floating point unit in hardware, this would result in approximation of select parts of the application. For cases such as this, QoS guarantees are even less reliable if based on the approximation error of the accelerator (e.g. the accuracy of the approximate floating point unit), as even small errors may accumulate and result in unacceptably large quality loss in the overall application. We therefore advocate analysis of QoS based on application-level quality metrics.

3.2.3. Coverage

Approximate accelerators are often unused when input data lies outside an acceptable range, such as the range of the training data for a neural accelerator [13]. This decision is based on the assumption that out-of-range inputs result in poor approximations. To demonstrate how this may be a costly oversight, we feed 1024 out-of-range inputs to our 8x8 NN, and reveal how 9%-47% of the data could be approximated acceptably for tolerance thresholds of 5%-30% (Figure 3), leading to average performance improvements of 5.6X over *ORIG_1%* and 1.6X over *ORIG_n%* (Figure 4).

Similarly, static techniques for error analysis (e.g. statistical models) disallow the use of an accelerator when its average

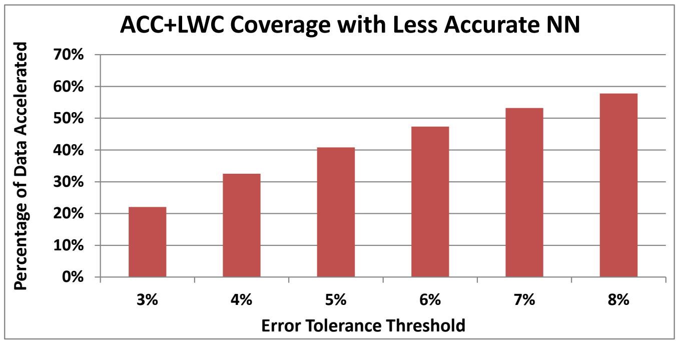

Figure 5: Amount of coverage with less accurate approx.

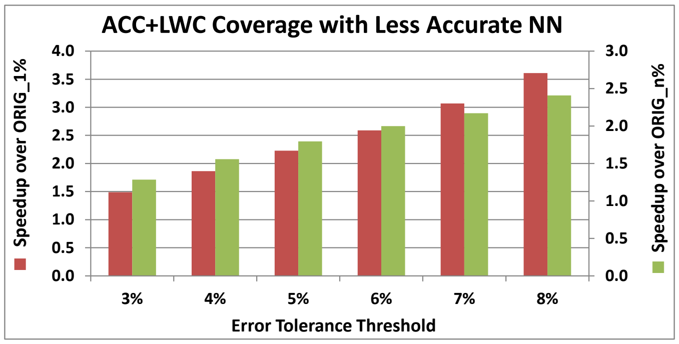

Figure 6: Speedup from coverage with less accurate approx.

error is above the tolerance threshold. Relying on the assumption that there would be too many cases of unacceptable quality loss, these static techniques lose coverage with portions of the data for which the low-accuracy accelerator could produce acceptable results. For instance, we train a smaller NN (same as the previous NN, only the 2 hidden layers each have 4 neurons) and find it has an average error of 8.9% (with a standard deviation of 11.7%) for the evaluation data. Although this error exceeds tolerance thresholds of 3%-8%, significant portions of the data (i.e. 22%-58%) may still be reliably accelerated (Figure 5), resulting in average performance gains of 2.5X over *ORIG_1%* and 1.9X over *ORIG_n%* (Figure 6). Furthermore, while these gains are with respect to an entirely software-based accelerator, efficient hardware-based approximate accelerators may be coupled with LWC support to yield even greater performance improvements.

4. Related Work

4.1. Approximate Computing

Approximate computing has been studied extensively for the purposes of improving performance, energy consumption, and resource utilization. There exist numerous hardware-based approaches, including stochastic or probabilistic technology [4, 24, 27], approximate circuitry for arithmetic [37, 20, 15] as well as general logic [23, 6, 32], architectures based on voltage scaling [11, 12, 18], and processing units for computing neural-network-based approximations [13]. In relation to these approaches, our methodology is software-based and platform-agnostic, allowing it to be interchangeably coupled with different approximate accelerators. We also make the distinction to decouple error analysis of the approximation unit from quality analysis of the overall application, demonstrating benefits in terms of coverage and reliability.

Prior art has also explored purely software-based approximate computing. Incremental refinement [25] and loop perforation [29], for instance, approximate iteratively-constructed solutions by reducing compute iterations. Other dynamic approaches include selective bit-width adaptation [26] and transformation of static configuration parameters into adjustable knobs [19]. Though our approach is similarly software-based, we employ high-level quality metrics and avoid unnecessary error checking. We also obviate the need for statically-constructed models of error distribution, allowing us to provide absolute guarantees on worst-case error.

In addition to pure hardware- or software-based solutions, approximate computing experts have also looked to hardware-software codesign. These systems [1, 5, 12, 8] typically include software support (e.g. new programming language and compiler) along with a series of architectural innovations (e.g. ISA extensions). Similar to these techniques, our software-based methodology could be combined with hardware-based approximate accelerators to form a synergistic hardware-software design. Unlike these approximate computing systems, however, we endorse a methodology based entirely on dynamic application-level error analysis, gaining additional coverage from leveraging slack that may not be evident at the level of the approximation unit.

4.2. Error Management

Language features, static analysis, and program logic may be used to control the impact of errors and ensure reliability during program execution [3, 14]. With EnerJ [28], for example, language support enables protection of specific values and compute regions, allowing hardware to readily perform approximate computation without being burdened by online error detection. Novelty detection [2], which enables recognition of out-of-range inputs using statistical estimations, may also be used to avoid potentially poor approximations. Likewise, correction mechanisms, such as those based on algorithmic noise tolerance [18, 31], allow for detection of errors during the course of the algorithm, albeit at an additional hardware cost. Error acceptance [17] also relates to our research, as it allows erroneous results to proceed so long as overall application output quality is not compromised. However, though the growth of error rates is controlled during program execution, there is no mechanism for dynamically leveraging the error information to initiate recovery for cases with unacceptable quality loss.

4.3. Error Analysis

Common metrics for error analysis include error rate (ER), error significance (ES), mean squared error (MSE), root mean squared error (RMSE), mean error distance (MED), and peak signal-to-noise ratio (PSNR) [16, 33]. Error could also be predicted using estimations of confidence intervals [7, 10] and error bounds [30]. Error prediction is an orthogonal approach compared to our methodology, and could be readily combined with error detection and recovery mechanisms to provide a more wholistic approach to imprecision tolerance.

Application-specific error analysis [22], on the other hand, is most relevant to our work. Approaches such as ERSA [5] and EnerJ [28] similarly conduct high-level error analysis using application-specific quality metrics. However, output quality degradation of approximate executions is most often measured with respect to precise executions (e.g. deviations in classification results), deeming the QoS metrics unsuitable for light-weight, dynamic error detection. These works nevertheless present thorough studies pertaining to application-level correctness, such as its impact on fault tolerance [22], thereby providing further support for the basis of our methodology.

5. Conclusion

In conclusion, we have presented a methodology for performing online error analysis and recovery based on LWCs, leveraging application-level imprecision tolerance to improve coverage and reliability. Platform-agnostic in nature, LWCs allow for an elegant solution to dynamic error control.

6. Acknowledgements

This work is supported by the NSF Graduate Research Fellowship Grant # DGE-0707424. It is also supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] Woongki Baek and Trishul M. Chilimbi. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *PLDI*, pages 198–209, 2010.

[2] C. M. Bishop. Novelty Detection and Neural Network Validation. *Vision, Image and Signal Processing*, 141(4):217–222, 1994.

[3] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware. In *OOPSLA*, pages 33–52, 2013.

[4] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee. Ultra-Efficient (Embedded) SOC Architectures Based on Probabilistic CMOS (PCMOS) Technology. In *DATE*, pages 1–6, 2006.

[5] Hyungmin Cho, L. Leem, and S Mitra. ERSA: Error Resilient System Architecture for Probabilistic Applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(4):546–558, 2012.

[6] M. R. Choudhury and K. Mohanram. Approximate Logic Circuits for Low Overhead, Non-Intrusive Concurrent Error Detection. In *DATE*, pages 903–908, 2008.

[7] G. Chryssolouris, M. Lee, and A. Ramsey. Confidence Interval Prediction for Neural Network Models. *IEEE Transactions on Neural Networks*, 7(1):229–232, 1996.

[8] Marc de Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. Relax: An Architectural Framework for Software Recovery of Hardware Faults. In *ISCA*, pages 497–508, 2010.

[9] Pradeep Dubey. A Platform 2015 Workload Model: Recognition, Mining and Synthesis Moves Computers to the Era of Tera. White paper, Intel Corporation, 2007.

[10] B. Efron and R. Tibshirani. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statistical Science*, 1(1):54–75, 1986.

[11] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO*, pages 7–18, 2003.

[12] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture Support for Disciplined Approximate Programming. In *ASPLOS*, pages 301–312, 2012.

[13] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural Acceleration for General-Purpose Approximate Programs. In *MICRO*, pages 449–460, 2012.

[14] Shuguang Feng, Shantanu Gupta, Amin Ansari, and Scott Mahlke. Shoestring: Probabilistic Soft Error Reliability on the Cheap. In *ASPLOS*, pages 385–396, 2010.

[15] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[16] J. Han and M. Orshansky. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. In *Proceedings of the 18th IEEE European Test Symposium*, pages 1–6, 2013.

[17] K. He, A. Gerstlauer, and M. Orshansky. Controlled Timing-Error Acceptance for Low Energy IDCT Design. In *DATE*, pages 1–6, 2011.

[18] R. Hegde and N. R. Shanbhag. Soft Digital Signal Processing. *IEEE Transactions on VLSI*, 9(6):813–823, 2001.

[19] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic Knobs for Responsive Power-Aware Computing. In *ASPLOS*, pages 199–212, 2011.

[20] Jiawei Huang, John Lach, and Gabriel Robins. A Methodology for Energy-Quality Tradeoff Using Imprecise Hardware. In *DAC*, pages 504–509, 2012.

[21] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82:35–45, 1960.

[22] X. Li and D. Yeung. Application-Level Correctness and its Impact on Fault Tolerance. In *HPCA*, pages 181–192, 2007.

[23] Shih-Lien Lu. Speeding Up Processing with Approximation Circuits. *Computer*, 37(3):67–73, 2004.

[24] S. Narayanan, J. Sartori, R. Kumar, and D. L. Jones. Scalable Stochastic Processors. In *DATE*, pages 335–338, 2010.

[25] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig. Approximate Signal Processing. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 15(1–2):177–200, 1997.

[26] J. Park, J. H. Choi, and K. Roy. Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy. *IEEE Transactions on VLSI*, 18(5):787–793, 2010.

[27] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[28] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *PLDI*, pages 164–174, 2011.

[29] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. In *ESEC/FSE*, pages 124–134, 2011.

[30] N. W. Townsend and L. Tarassenko. Estimations of Error Bounds for Neural-Network Function Approximators. *IEEE Transactions on Neural Networks*, 10(2):217–230, 1999.

[31] G. V. Varatkar and N. R. Shanbhag. Energy-Efficient Motion Estimation Using Error-Tolerance. In *ISLPED*, pages 113–118, 2006.

[32] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: Systematic Logic Synthesis of Approximate Circuits. In *DAC*, pages 796–801, 2012.

[33] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: Modeling and Analysis of Circuits for Approximate Computing. In *ICCAD*, pages 667–673, 2011.

[34] Luminita A. Vese and Stanley J. Osher. Image Denoising and Decomposition with Total Variation Minimization and Oscillatory Functions. *Journal of Mathematical Imaging and Vision*, 20(1–2):7–18, Jan 2004.

[35] Li-Chun Tommy Wang and Chih Cheng Chen. A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, Aug 1991.

[36] Z. Wang and A. C. Bovik. A Universal Image Quality Index. *IEEE Signal Processing Letters*, 9(3):81–84, 2002.

[37] Thomas Yeh, Petros Faloutsos, Milos Ercegovac, Sanjay Patel, and Glenn Reinman. The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration. In *MICRO*, pages 394–406, 2007.