ACOCO: Adaptive Coding for Approximate Computing on Faulty Memories

Chu-Hsiang Huang
Department of Electrical Engineering, University of California, Los Angeles
seanhuang0522@ucla.edu

Yao Li
Akamai Technologies, Inc.
yli@akamai.com

Lara Dolecek
Department of Electrical Engineering, University of California, Los Angeles
dolecek@ee.ucla.edu

Abstract

With scaling of process technologies and increase in process variations, embedded memories will be inherently unreliable. Approximate computing is a new class of techniques that relax the accuracy requirement of computing systems. In this work, we present the *A*daptive *Co*ding for approximate *Co*mputing (ACOCO) framework, which provides us with a theory-guided design methodology to develop adaptive codes for different computations on the data read from faulty memories. In ACOCO, we first compress the data by introducing distortion via a designed source encoder. We then add redundant bits via a designed channel encoder in order to protect this distorted data against memory errors; with a proper design of source/channel encoder pairs we are able to protect the data against memory errors without additional memory overhead. We develop an adaptive code (consisting of the source/channel encoder pair) for several applications, including machine learning algorithms and iterative inference/decoding under ACOCO, and demonstrate that the adaptive code successfully protects against memory errors while the designed data compression component has a negligible effect on residual error rate.

Keywords Approximate computing, fault-tolerant computing, faulty memory, iterative decoders, error-correcting code.

1. Introduction

Nanometer CMOS technologies are susceptible to a variety of reliability issues. Popular approaches to enhancing hardware reliability, such as overdesigning and guardbanding, incur large power consumption and are inefficient for current digital systems. A promising alternative is to develop algorithmic-based robust systems tolerant of hardware errors [8].

Memory failure rates are increasing due to the impact of shrinking dimensions, high integration densities, lower operating voltages, etc. [2, 7]. Error correction coding was introduced to mitigate the effects of memory cell errors by redundant bits stored in additional memory cells [4, 11, 14].

Approximate computing has attracted significant interest in recent years for its capability to trade computation accuracy for data processing throughput or energy efficiency [13, 15, 16]. Several previous efforts have achieved promising results by exploring approximate computing both in software and in hardware [3]. However, there is still a need for a systematic methodology that will offer designers an ability to analytically evaluate the distortion (introduced by relaxing the accuracy requirement) and to develop theory-guided approximate computing techniques.

In this work, we present the *A*daptive *Co*ding for approximate *Co*mputing (ACOCO) framework [10]. The ACOCO framework provides a theory-guided design methodology to develop adaptive codes for different types of systems subject to memory errors. For systems that have inherent tolerance to distortion in data representation, the ACOCO framework uses cleverly-designed adaptive codes to restrict the distortion in data presentation brought about by faulty data storage (memory) to the range that least impacts the system performance. We first introduce the binary symmetric channel (BSC) memory error model, and lay out the ACOCO framework. The adaptive coding scheme first defines a (lossy) source code, and uses the memory cells saved via source coding to store the redundant bits produced by a channel code whose code rate is equal to the ratio of the length (in bits) of the source encoder output (x_{se} in Fig. 1) to the length (in bits) of the original input data (x_s in Fig. 1). Therefore, unlike the previous error-correcting codes which require additional memory cells (e.g., [4, 11, 14]), the adaptive codes developed under ACOCO have coding rate 1, i.e., no additional (redundant) memory cells are required. Although we introduce some distortion in the source encoder, the channel codes are able to correct possibly harmful memory errors (e.g., the errors in the sign bit). In this abstract, we report our results for three representative applications: max-product (MP) inference algorithm, naïve Bayesian classifier (NBC) and an iterative decoder. MP algorithm and NBC are extensively used in machine learning systems, and iterative decoders are ubiquitous in modern data communication and storage systems.

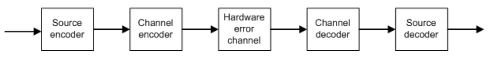

Figure 1: A system diagram of ACOCO framework.

2. System Model and The ACOCO Framework

In this work, we use the bit-flipping model for faulty memories [1, 6, 9]. We model the effect of errors in each memory cell as passing a binary input through a BSC with cross-over probability ρ. To be more specific, suppose a bit is stored in a memory cell. The value retrieved at read time differs from the original value with probability ρ, $0 < \rho < 1$. The logic gates performing the computations (computation units) are assumed to be noise-free.

Inspired by approximate computing techniques, we propose the ACOCO framework. Consider each computation unit input (from the memory) as a source message. We first encode the source message (x_s) using a (lossy) source encoder so as to represent the message with fewer bits (x_{se}). The source encoder aims to minimize a suitably quantified difference in the output of the system with noise-free memory and the system subject to memory errors but protected by the proposed adaptive codes; we design different source encoders for systems with different (output) error characteristics. We then append redundant bits using a channel encoder; its output is x_{ce}. The output x_{ch} of the hardware error channel represents the binary representation read from memory cells. A channel decoder followed by a source decoder decodes x_{ch}, and the output x_o is the input to the computation units performing operations on

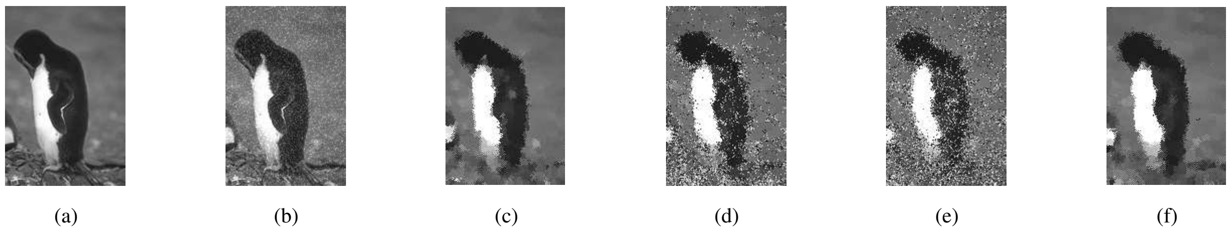

Figure 2: Image denoising via MP: (a) original image, (b) contaminated image, (c) recovered image by noise-free MP, (d) recovered image by noisy MP without ECC, (e) recovered image by noisy MP with QC, (f) recovered image by noisy MP with AD.

the data. The system block diagram for ACOCO is shown in Fig. 1. Note that we choose the number of bits reduced during the source coding stage to be equal to the number of redundant bits introduced during the channel coding stage. Hence, jointly, our adaptive code is of rate 1.

The authors in [6] proposed a coding scheme following a similar methodology as the one proposed here, but in their approach the least significant bits (LSBs) in the source encoder are simply and non-adaptively discarded. We compare the performance of the systems implementing proposed adaptive codes (referred to as AD) with the systems implementing the code in [6] (referred to as QC) and the nominal systems (without any codes, referred to as NC), and demonstrate the advantage of our theory-guided code design in the following section.

3. Representative Results

We first consider the MP algorithm. In this example, we apply the MP algorithm to image denoising and demonstrate the advantages of the adaptive code developed under ACOCO. We recover the original image from the noise-contaminated image by running MP to obtain the most likely value of every pixel based on the contaminated observations. We use the “penguin” image and Potts model [5]. The original image and the contaminated image are shown in Fig. 2a and Fig. 2b. The MP messages are stored in the faulty memories with bit-flipping rate $\rho = 2.5 \times 10^{-3}$. We use the following adaptive code. When the magnitude of x_s is large, we discard the four LSBs and protect the MSBs by the channel code. When the magnitude of x_s is small, however, we keep the LSBs, use a single bit to indicate that the magnitude of x_s is small, and discard the four MSBs. We choose $(7, 4)$ Hamming code as our channel code.

Comparing Figs. 2e, 2d, and 2f, we observe that for QC and NC, a large portion of the pixels are determined incorrectly and the recovered image is very different from the image recovered using noise-free MP (Fig. 2c). On the other hand, for AD, the recovered image closely resembles the image recovered by noise-free MP.

Next, we consider the NBC system subject to memory errors. Since NBC has similar error characteristics as the MP algorithm, we use the same adaptive code as the one used in MP. We show the average classification error rates (over 1000 experiments) in Fig. 3. We observe that the increase in bit-flipping rate ρ has very little effect on the classification error rates under QC and AD, while the classification error rate under NC increases a lot as ρ increases. We also find that the classification error rate under QC is much larger than under AD due to (non-adaptively) discarding the LSBs. AD is very close to the noise-free case (less than 0.5% difference) in the entire ρ region we considered.

Finally, we consider a popular iterative decoder: min-sum decoder, subject to memory errors. We use a $(3, 6)$-regular LDPC code with code length 2640 [12]. The communication channel noise is modeled as a zero-mean Gaussian random variable with variance σ^2. The memory bit-flipping probability is $\rho = 5 \times 10^{-4}$. The maximum number of iteration is 30. We design the following adaptive code for the min-sum decoder. When the MSB is 1, the source encoder discards the two LSBs, and uses the $(3, 1)$ repetition code to protect the sign bit. When the MSB is 0, both source and channel encoder become an identity function.

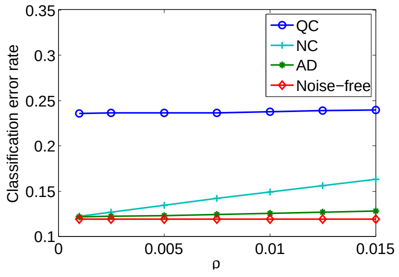

Figure 3: Naïve Bayes classifiers comparison.

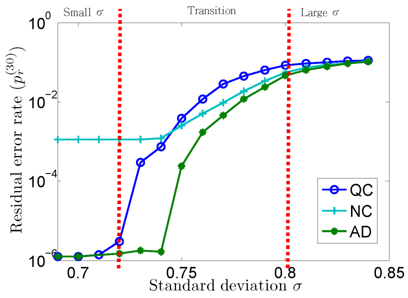

Figure 4: Comparison of residual error rates.

We compare the residual error rates under different σ’s (Fig. 4). We observe that for small σ, the residual error rates under AD (and QC) are much lower than NC. In the transition region between small and large σ, AD still achieves a lower residual error rate than QC and NC.

From the above three applications, we conclude that some memory errors are harmful to the system, hence we see large performance degradation under NC. Although QC can correct memory errors, the detrimental effect of simply discarding the LSBs is potentially large and the output can be significantly deteriorated. On the other hand, by applying the proposed adaptive code developed under ACOCO, we successfully correct most of the harmful memory errors as well as keep the effects of the distortion introduced in the source encoder small.

4. Conclusion and Future Work

In this work, we presented the ACOCO framework for the design of adaptive codes that improve the performance of computation systems in the presence of faulty memory cells. ACOCO achieves its goal by shifting resources to protect critical components of the system and relaxing accuracy requirements on less critical components. We demonstrated the effectiveness of ACOCO on three representative applications: the MP algorithm, NBC, and an iterative decoder. In all the three applications, our codes successfully correct harmful memory errors, while the distortion introduced by the source encoder has negligible effects on the system performance. Due to lack of space, the theoretical analysis is omitted from this abstract, please see our companion paper [10].

References

[1] A. Balatsoukas-Stimming and A. Burg. Density evolution for min-sum decoding of LDPC codes under unreliable message storage. *IEEE Commun. Lett.*, 18(5):849–852, May 2014.

[2] A. Chandrakasan et al. Technologies for ultradynamic voltage scaling. *Proceedings of the IEEE*, 98(2):191–214, Feb 2010.

[3] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *IEEE/ACM DAC*, page 113, 2013.

[4] Y. Emre and C. Chakrabarti. Memory error compensation techniques for JPEG2000. In *IEEE SIPS*, 2010.

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *Int. J. Comput. Vision*, 70(1):41–54, Oct. 2006.

[6] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto. 13.8 a 32kb SRAM for error-free and error-tolerant applications with dynamic energy-quality management in 28nm CMOS. In *IEEE ISSCC*, 2014.

[7] S. Ganapathy, G. Karakonstantis, R. Canal, and A. P. Burg. Variability-aware design space exploration of embedded memories. In *IEEE IEEEI*, 2014.

[8] P. Gupta et al. Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 32(1):8–23, Jan. 2013.

[9] C.-H. Huang, Y. Li, and L. Dolecek. Belief propagation algorithms on noisy hardware. *IEEE Trans. Commun.*, 63(1):11–24, Jan 2015.

[10] C.-H. Huang, Y. Li, and L. Dolecek. ACOCO: Adaptive coding for approximate computing on faulty memories. *IEEE Trans. Commun.*, 2015. submitted.

[11] M. Jayarani and M. Jagadeeswari. A novel fault detection and correction technique for memory applications. In *ICCCI*, 2013.

[12] D. MacKay. Encyclopedia of sparse graph codes. URL `http://www.inference.phy.cam.ac.uk/mackay/codes/data.html`.

[13] A. K. Mishra, R. Barik, and S. Paul. iACT: A software-hardware framework for understanding the scope of approximate computing. In *WACAS*, 2014.

[14] D. Rossi, N. Timoncini, M. Spica, and C. Metra. Error correcting code analysis for cache memory high reliability and performance. In *DATE*, 2011.

[15] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Sage: Self-tuning approximation for graphics engines. In *IEEE/ACM MICRO*, pages 13–24, 2013.

[16] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: Modeling and analysis of circuits for approximate computing. In *ICCAD*, pages 667–673, 2011.