ARMOR: Towards Restricted Approximation with A Worst-Case Guarantee

Sihang Liu
University of Virginia

Kevin Angstadt
University of Michigan

Mike Ferdman
Stony Brook University

Samira Khan
University of Virginia

Abstract

Approximate applications can tolerate failures in memory and provide a better performance and energy efficiency. However, DRAM failures are randomly distributed in the approximate memory, and therefore, can introduce a large variation in the output quality of approximate applications. Prior works utilize the statistical analysis to guarantee that the error in the output is limited to a narrow range with a high confidence. In this work, we demonstrate that the errors in the output can be significantly higher than the statistical guarantee in many cases, and therefore, statistical guarantee cannot reason about the worst-case error.

The goal of this work is to provide a guarantee on the worst-case error. We propose the notion of restricted approximation where the memory failure can only introduce a certain maximum error in the input, such that it is possible to estimate the worst-case error in the output by analyzing the program. We provide ARMOR, a software-hardware solution that provides a worst-case guarantee of the error in the output. With approximation, ARMOR reduces the energy by 12% and improves the performance by 18%. By restricting the error, ARMOR achieves 31 to 4×10^4 times reduction in error over the case where memory failures are randomly distributed.

1 Introduction

As the demand for big data analysis and computation grows exponentially, the memory capacity becomes one of the key limiters for these applications. Many of these memory intensive applications can intrinsically tolerate some error in the input data [1–7]. Leveraging this characteristic, a number of studies have advocated the use of *approximate memory*, where data that need not be precise are allocated in the memory that contain errors [3–5, 8–10].

A typical approach in approximate memory is lowering the memory refresh rate. Although this approach improves energy and performance by reducing the number of DRAM refresh operations, and mitigate the reliability of memory systems due to the scaling of DRAM cells, it inevitably incurs randomly placed memory failures, and therefore can drastically change the value of approximate data. Therefore, the use of approximate memory can potentially degrade the quality of the output to the point that the output is no longer acceptable to the users [5]. Recent studies have proposed various solutions to provide a statistical guarantee such that the deviation of results is within an acceptable range with a high confidence [11–16]. Unfortunately, statistical guarantee cannot reason about worst-case results. Figure 1 shows the root-mean-square-error (RMSE) of Sobel, an image processing application, across 300 runs on an approximate memory with a random error rate of 10^{-4}, where the worst-case quality degradation can be 72% worse than the average and 33% worse than the 90th percentile. The root cause of the undesirable outliers is that it is difficult to determine the extent to which the randomly-distributed failures can affect the output. For example, a 32-bit floating point variable stored on approximate memory can have a failure on any bit position. If the failure appears on one of the low-order bits, the relative error of this variable is low; whereas an failure in the exponent

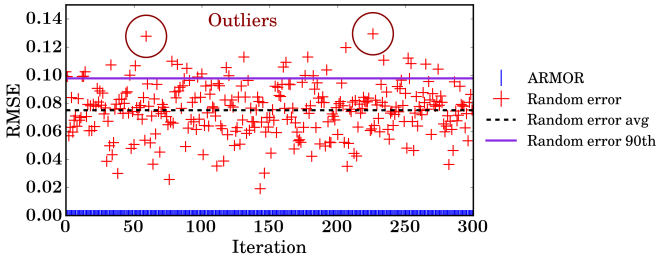

Figure 1. Errors across 300 runs of Sobel.

can make the relative error exponentially high, drastically affecting the output. Therefore, we observe that controlling the impact on the output becomes tractable if the positions of failing bits are constrained.

Based on this key observation, we develop ARMOR, a novel approximate memory system, that targets *error-controlled restricted approximation*. ARMOR first constrains failures in the approximate data by reshuffling the byte fetched from approximate memory that contains failures to the *low-order byte*. Then it provides a *worst-case guarantee* on the output error by analyzing the program, according to the limited error in approximate data. Although, in most cases, limiting failures to the low-order byte effectively constrains the error in the output, prior works on floating point arithmetic have point out that some cases [17–21], depending on the operation and value of operands, can lead to high errors. To ensure that error is still below the worst-case guarantee in these corner cases, we further develop an online error detection and re-execution mechanism that detects and mitigates these corner cases at runtime. With these techniques, ARMOR significantly reduces the error and provides a guarantee on the worst-case error. The blue marks in Figure 1 shows the RMSE of ARMOR, that is 7.4×10^4 lower than that from an unconstrained approximation.

To summarize, ARMOR provides an *error-controlled restricted approximation* with the following approaches: *(i)* Designs a memory system that constrains memory failures to the *low-order byte* of the mantissa in floating points variables. *(ii)* Statically analyzes how the error in approximate values propagates to the output to provide a *worst-case guarantee*. *(iii)* Handles the corner cases that can potentially violate this guarantee.

2 Challenges and Solutions

In this section, we discuss the major challenges in designing a *restricted approximate memory* and provide our solutions to solve those challenges.

Challenge 1: How to design a memory system that restricts the *maximum error* in approximate variables? ARMOR restricts the failing bits in the less significant bits so that the applications will not exhibit significant error in the data stored in approximate memory. We refer to the relative error in each approximate variable that occurs due to the memory failures as the *base error*. Based on the observation that a 64-bit word is the smallest unit of DRAM data transfer, ARMOR *reshuffles* the bytes in the memory controller during a read access to approximate memory, such that the byte with memory failure is positioned in the lowest-order

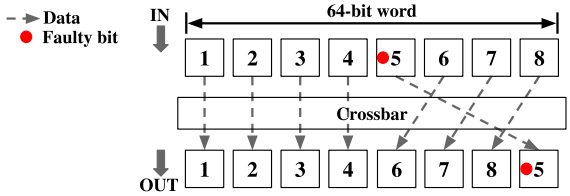

Figure 2. Reshuffle mechanism of ARMOR.

byte. It utilizes recent DRAM failure detection techniques to track the locations of failing bits in each data transfer from memory [10, 22–29]. Figure 2 demonstrates our reshuffling technique, where the byte with failure (#5) is reshuffled to the lowest-order. In this way, failing bits are always positioned in the lowest-order byte of 32-bit or 64-bit floating point variables. Using this method, ARMOR not only constrains the error, but also makes it possible to calculate the maximum relative error in floating point variables. Here we demonstrate how to analyze the *maximum base error* of a 32-bit float. The value of a 32-bit float in IEEE754 format [30] is represented as $Sign \cdot 2^{Exponent} \cdot Mantissa$, and the relative error between its precise and approximated value is:

$$relative\_error = \left| \frac{approx - precise}{precise} \right|. \quad (1)$$

As failures can only appear in mantissa bits, the exponent and sign bits in the approximate and precise floats are identical. Therefore, the relative error only depends on mantissas:

$$relative\_error = \left| \frac{mantissa_{approx} - mantissa_{precise}}{mantissa_{precise}} \right|. \quad (2)$$

To calculate the maximum $relative\_error$, we need to consider the case where $mantissa_{precise}$ (denominator) is of its minimum value and the difference between approximate and precise mantissa (nominator) is of its maximum value. According to the IEEE754 standard, a 32-bit float has 23 visible mantissa bits and 1 hidden leading mantissa bit which is always 1 in ordinary normalized mode. Therefore, the minimum value represented by the mantissa occurs when all 23 mantissa bits are 0 and the hidden bit is 1. The maximum difference in the precise and approximate mantissa occurs when all bits in the low-order byte contains memory failures. Hence, the maximum relative error in a floating point is:

$$max\_relative\_error = \frac{\sum_{n=1}^{8} 2^{-15-i}}{2^0} \approx \frac{3.04^{-5}}{1.0} = 3.04 \times 10^{-5}. \quad (3)$$

Challenge 2: How to analyze *maximum error* in the program output? Once ARMOR has constrained the base error in the input approximate data, we need to analyze how it affects the final output of the approximate programs. Depending on the operations applied on the inputs, the base error propagates differently through the program. For example, different operations, such as addition, multiplication, etc. that take operands with the same base error could produce results with different maximum errors. We refer to the the maximum error in the output that is dependent on the operations in the program as maximum *propagated error*. ARMOR analyzes the maximum propagated error to provide a worst-case guarantee on the output of the approximate program. The key idea is to assume that each approximate variable allocated in memory contains the *base error* and perform static analysis of the program to calculate the maximum propagated error in the output. Figure 3 shows a data-flow graph of a snip of code where all variables are approximate. Based on the dependencies between these approximate variables and the type of operations, ARMOR can calculate the propagated error of the final output e.

$$err_a, err_b, err_d = max\_base\_error$$
$$err_c = PropError(err_a, err_b)$$
$$err_e = PropError(err_c, err_d)$$

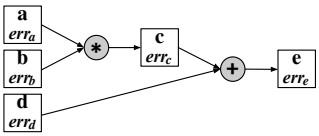

Figure 3. Example of error propagation.

Challenge 3: How to constrain *value-dependent* errors? So far, we have introduced two sources of errors in the approximate programs: *base error* caused by the memory failures in approximate data, and *propagated error* that depends on different operations the program is consisted of. They do not depend on the value of the operands, and can be statically analyzed using the maximum error in the source operands and the type of the operation. Unfortunately, some of the mathematical operations exhibit error that depends on the value of the operands [17–19]. We refer to such errors as *value-dependent error* and refer to the cases where the maximum value-dependent becomes exceptionally high as the *corner cases*. For example, subtracting two floating point variables that are very close can lead to a high error, as the the denominator in Equation 1 is close to zero. This phenomenon is known as catastrophic cancellation [20, 21]. Bounding the maximum error propagated from these operations is challenging as though the operators can be statically determined, the values of the operands can only be known at runtime. To solve this issue, we build upon the observation that these cases are fairly infrequent in the applications. In our evaluation, we find that only up to 0.34% of the inputs falls into these corner cases. To solve this issue, we propose a simple hardware-assisted corner case detector for these specific operations and determine whether the values fall into the corner cases. As the probability of occurrence is very low, ARMOR re-executes the corner cases with precise values, making sure the output is always bounded by the worst-case guarantee.

3 Results

Here we briefly summarize the evaluation of ARMOR using six approximate applications from AxBench [31]. First, we evaluate the benefit from low-refresh DRAM (256ms refresh interval) compared to a standard system using a higher refresh rate (64ms refresh interval). On average, ARMOR improves the energy by 18% and performance by 12% over the standard system. Second, we evaluate the quality of ARMOR with a baseline system using an approximated memory with randomly distributed errors (error rate is 10^{-4}). ARMOR achieves 30.9 to 4.3×10^4 times reduction in error compared to the baseline. The worst-case error guarantee provided by ARMOR is up to 3.93 to 5×10^{38} times lower than the maximum error in the baseline. The results shows that the output error is well controlled with ARMOR.

4 Conclusions

In this work, we show that a practical approximate system needs to restrict the error in the data such that the error in the output becomes predictable. Our work, ARMOR, is the first step to establishing a worst-case guarantee on the error in approximate computing. We believe that it will lead to future works that leverage error-controlled restricted approximation, paving the way for adopting approximate hardware in real systems.

References

[1] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh, "Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration," in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016.

[2] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[3] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[4] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[5] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011.

[6] A. Jain, P. Hill, S. Lin, M. Khan, M. E. Haque, M. A. Laurenzano, S. A. Mahlke, L. Tang, and J. Mars, "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation," in *49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[7] C. Rubio-GonzÃąlez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[8] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Approximate computing with partially unreliable dynamic random access memory - approximate DRAM," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016.

[9] C. Chou, P. J. Nair, and M. K. Qureshi, "Reducing refresh power in mobile devices with morphable ECC," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[10] M. K. Qureshi, D. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[11] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[12] M. Ringenburg, A. Sampson, I. Ackerman, L. Ceze, and D. Grossman, "Monitoring and debugging the quality of results in approximate programs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[13] J. Park, E. Amaro, D. Mahajan, B. Thwaites, and H. Esmaeilzadeh, "AxGames: Towards crowdsourcing quality target determination in approximate computing," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[14] V. Vassiliadis, J. Riehme, J. Deussen, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis, and U. Naumann, "Towards automatic significance analysis for approximate computing," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, 2016.

[15] P. Hill, M. Laurenzano, B. Zamirai, M. Samadi, S. Mahlke, J. Mars, and L. Tang, "Statistical error bounds for data parallel applications," in *2016 Workshop on Approximate Computing Across the Stack*, 2016.

[16] T. Moreau, A. Sampson, L. Ceze, and M. Oskin, "Approximating to the last bit," in *2016 Workshop on Approximate Computing Across the Stack*, 2016.

[17] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock, "Automatically improving accuracy for floating point expressions," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015.

[18] M. Tang, X. Zeng, K. Song, and J. Liu, "Error analysis on floating-point arithmetic in c programming language library functions," in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, 2013.

[19] R. Sedgewick and K. Wayne, "Introduction to programming in Java." https://introcs.cs.princeton.edu/java/91float/, 2014.

[20] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, pp. 5–48, Mar. 1991.

[21] J. W. Carr, III, "Error analysis in floating point arithmetic," *Commun. ACM*, vol. 2, pp. 10–15, May 1959.

[22] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware intelligent DRAM refresh," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012.

[23] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013.

[24] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, 2014.

[25] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An efficient system-level technique to detect data-dependent failures in DRAM," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.

[26] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013.

[27] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent dram failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[28] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, 2014.

[29] M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (REAPER): Enabling the mitigation of dram retention failures via profiling at aggressive conditions," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[30] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[31] A. Yazdanbakhsh, D. Mahajan, P. Lotfi-Kamran, and H. Esmaeilzadeh, "AXBENCH: A multi-platform benchmark suite for approximate computing," in *IEEE Design and Test, special issue on Computing in the Dark Silicon Era*, 2016.