AWARD: Approximation-aWAre Restore in Further Scaling DRAM

Xianwei Zhang, Youtao Zhang, Bruce Childers
Computer Science Department
University of Pittsburgh, PA, USA
{xianeizhang}@cs.pitt.edu

Jun Yang
Electrical and Computer Engineering Department
University of Pittsburgh, PA, USA
juy9@pitt.edu

ABSTRACT

DRAM further scaling becomes more and more challenging, making restore operation an serious issue in the near future. Fortunately, a wide range of modern applications are able to tolerate error or inexactness, providing a new dimension to mitigate the slow-restore issue. And thus, we can trade-off acceptable QoS loss in those applications to accelerate restore operations, and further to achieve performance and energy improvements. In this extended research abstract, we briefly explore DRAM restore-based approximate computing, and present a preliminary evaluation on impacts of quality-of-service (QoS) degradation and performance speedup. We show that restore-based approximate computing is a challenging work, and dedicated error correction/tolerance techniques are needed to balance QoS and performance.

Categories and Subject Descriptors

B.3.1 [**Memory Structures**]: Semiconductor Memories—*Dynamic Memory (DRAM)*; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids

Keywords

DRAM Scaling, Slow Restore, Approximate Computing

1. INTRODUCTION

This section presents brief introductions on DRAM scaling issues and approximate computing works.

1.1 DRAM Scaling

As the *de facto* standard memory, DRAM's great success is tremendously contributed by its continuous scaling to maintain growth on density, energy efficiency and bandwidth, etc. However, after decades' scaling, DRAM nowadays has entered into 20nm [23], which faces significant challenges, including more leaky cells [22], slow sensing and restoring operations, severer process variations [7] and disturbance [16].

To tackle the scaling issues, whereas significant researches have been put into retention and refresh [11, 18, 5], and sensing [17, 28, 27], no explorations have been performed on restoring until recently [15, 30, 31]. Among the prior arts, Kang *et al.* [15] first raised the slow restore issue; and, `ChunkRemap` [30] was later proposed to utilize chunk remapping to lower restore timings; Zhang *et al.* mitigated slow restore issue with early truncation designs[31]. All of those slow-restore solutions lie on hardware and architectural levels, which are completely unaware of the features of running programs.

1.2 Approximate Computing

Energy and power are increasing concerns in modern computer systems, and much energy is spent on guaranteeing correctness [25]. Nevertheless, many applications have intrinsic resilience to runtime errors [8, 25], providing good opportunities to explore energy-accuracy tradeoff. For instance, in domains like machine learning and computer vision, algorithms are usually heuristic based, and thus the final results are not 100% accurate. The algorithm can function much better, if the execution time and power consumption can be significantly brought down, while the output accuracy has a slight decrease.

Previous works on approximate computing performed explorations on both hardware [6, 19, 26, 12, 14, 13] and software [4, 25, 3]. Among them, there is significant research on efficient memory system, which is also the focus of our exploration. `Flikker` [19] refreshes approximate data at lower rates to save DRAM refresh energy, which was recently extended by [20] and [24]. Esmaeilzadeh *et al* [9] proposed to apply dual voltage to SRAM array to balance energy and accuracy; `Drowsy caches` [10] reduces the supply voltage to save power; and, papers [26] and [12] applied approximate computing to optimize PCM lifetime and density.

1.3 Contributions

By exploiting program features, we can find more efficient and innovative ways to mitigate the slow restore issue. And, instead of getting efficient approximate storage [20, 12], our work here applies approximate computing to improve restore operation, which is on the critical path and has runtime impacts on programs. Compared to refresh-based approximation designs [19, 20, 24], it is more challenging to control the QoS in restore scenario, but the expected improvements are also much higher because restore directly contributes to access latency. This research abstract presents our pioneer-

MEMSYS '16 October 03-06, 2016, Alexandria, VA, USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4305-3/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2989081.2989127

Table 1: Sampled `tWR` cumulative distribution in DRAM chips. (in Byte units)

`tWR`(ns)	≥ 35	≥ 30	≥ 25	≥ 20	≥ 15	≥ 14	≥ 13	≥ 12	≥ 11
perc(%)	$\approx$0	0.01	0.03	0.05	0.44	1.45	4.59	13.59	36.76

ing studies on approximation-aware restore, and outlines the directions to perform further explorations.

2. MOTIVATION AND DESIGN

Table 1 shows the sampled `tWR` [1] distribution. From the table, we can see that while the worst-case value can be as large as 35ns, only 0.44% fall beyond the 15ns specification value, and over 95% are below 13ns. Accordingly, if the slow-to-restore cells can be tolerated by the applications, then we can accelerate accesses by applying lower restore timings instead of the original worst-case ones, as illustrated by Figure 1.

From the figure we can see that whereas worst-case determined strategy gives '25'/'25'/'26' for the rows, lower values of '20'/'13'/'22' can be achieved by sacrificing the very-slow ones, and only several faults ('0') are injected. We also know that the fault locations cause errors with 50% probability, i.e., bit flip only happens when original bit is '1', and hence the impact of those faulty bits is very limited.

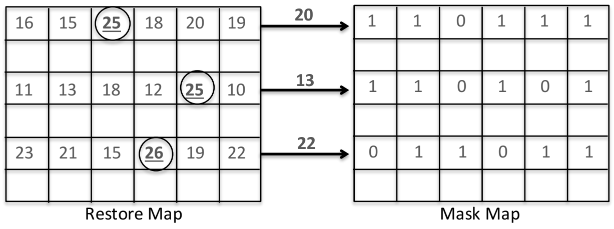

Figure 1: Generate mask map ('1' is precise, '0' is fault) by lowering row-level restore goals in original DRAM restore map (circled values are the worst timings, in nano-second, in the rows).

Nevertheless, the portion of slow-to-restore cells, as reported in Table 1, are much higher than that of leaky ones [18], and thus restore-based approximation are expected to have much larger impact than refresh-based ones [20, 24]. Moreover, differing form previous approximate proposals of using memory as storage [12, 20], approximated restore timings here impact the run-time computations, e.g., flips of sign and exponential bits in floating points may totally change the values and final outputs.

3. EXPERIMENTAL METHODOLOGY

The general goal of approximate computing is to achieve performance and energy improvements with acceptable quality-of-service (QoS) degradation. Accordingly, we adopt a two-phase methodology [21] to evaluate the proposed design. In first phase, a Pin-based simulator is used to instrument the beforehand annotated programs; during runtime, all loads and stores of integer and floating-point variables are captured, and the values of approximated ones are clobbered on-the-fly. For value change, the memory restore/mask map is accessed to inject faults into precise memory operands. Address mapping, allocation and data cache are also included in the Pintool, and thus we are capable to sweep over different allocation strategies and various restore goals to examine the quality of approximate results.

In the first phase, applications are executed to the end to output the final results; meanwhile, instructions are traced out as input to our second-phase simulation. In the second phase, conventional simulators are used to collect performance and energy values. Apparently, both phases should be involved into the evaluations, and hence we choose QoS and performance as the metrics. As for QoS metric, we compare the results of approximate execution against those of precise, and application-specific metric is used [25, 21]. For instance, QoS of **blackscholes**, a financial analysis application from PARSEC 3.0, is measured using the percentage of different prices between precise and approximate runs.

4. PRELIMINARY EVALUATION

Our preliminary evaluation was designed to study the effects of approximation-aware restore on application output quality (QoS). Benchmarks `kmeans` [29], `blackscholes` [29], `raytracer` [1] and `scimarks` [2] were evaluated, which comes from different domains, including machine learning, financial analysis and scientific computing, etc. And, the QoS metrics generally follow [29] and [25]. The configurations were simplified by setting a low restore goal for the whole memory system, and further random page mapping was adopted.

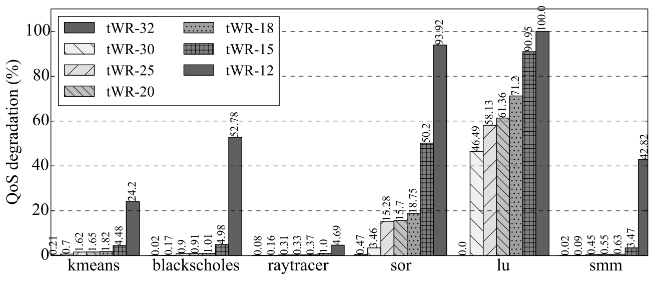

Figure 2: QoS of different restoring goals. For each benchmark, results are presented in `tWR` descending order. QoS loss values are labelled over each bar.

The collected QoS results are reported in Figure 2 by sweeping a series of restoring goals. Clearly, we can see that by decreasing restoring timings, QoS degradation gradually goes up, and the most serious degradation is reached at the lowest timing, i.e. tWR=12. Further, even quite high restore timings can lead to unacceptable QoS loss, e.g., `lu` suffers from 46.49% loss at tWR=30 and `sor` sees 15.28% loss at tWR=25. Nevertheless, low power consumption and high performance gains can only be effectively achieved at low re-

[1]`tWR` is *write recovery time*, which is the restore timing of write operation. The specification value has been kept at 15ns from DDR to DDR4 [30]. Values were collected following prior work [30].

store timings [30, 31]. As a result, it is a super challenging work to achieve both acceptable QoS and energy/performance improvements.

5. CONCLUSION AND FUTURE WORK

The preliminary results imply that it is challenging to strike a balance between performance/energy improvements and QoS loss. To achieve acceptable QoS, conservative restore approximation and dedicated correction techniques should be utilized to protect the important bits. In addition, higher level techniques like memory allocation and data remapping can be used to constraint the impacts of faulty locations.

We plan to leverage a combination of techniques to implement generalized approximation-aware restore, have detailed hardware and software designs, and perform the evaluations across a wider spectrum of applications.

6. REFERENCES

[1] Raytracer: http://www.planet-source-code.com/vb/scripts/raytracer. http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=5590&lngWId=2.

[2] SciMark2. http://math.nist.gov/scimark2/.

[3] S. Achour and M. C. Rinard. Approximate computation with outlier detection in Topaz. In *OOPSLA*, pages 711–730, 2015.

[4] W. Baek, T. M. Chilimbi, et al. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, pages 198–209, 2010.

[5] I. Bhati, Z. Chishti, et al. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *ISCA*, pages 235–246, 2015.

[6] L. N. Chakrapani, B. E. S. Akgul, et al. Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology. In *DATE*, pages 1110–1115, 2006.

[7] B. R. Childers, J. Yang, et al. Achieving yield, density and performance effective DRAM at extreme technology sizes. In *MEMSYS*, pages 78–84, 2015.

[8] M. d. Kruijf, S. Nomura, et al. Relax: an architectural framework for software recovery of hardware faults. In *ISCA*, pages 497–508, 2011.

[9] H. Esmaeilzadeh, A. Sampson, et al. Architecture support for disciplined approximate programming. In *ASPLOS*, pages 301–312, 2012.

[10] K. Flautner, N. S. Kim, et al. Drowsy caches: Simple techniques for reducing leakage power. In *ISCA*, pages 148–157, 2002.

[11] M. Ghosh and H. S. Lee. Smart refresh: An enahnced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *MICRO*, pages 134–145, 2007.

[12] Q. Guo, K. Strauss, et al. High-density image storage using approximate memory cells. In *ASPLOS*, 2016.

[13] M. Jung, D. M. Mathew, et al. Efficient reliability management in SoCs - an approximate DRAM perspective. In *ASPDAC*, 2016.

[14] M. Jung, E. Zulian, et al. Omitting refresh: A case study for commodity and Wide I/O DRAMs. In *MEMSYS*, 2015.

[15] U. Kang, H. s. Yu, et al. Co-architecting controllers and DRAM to enhance DRAM process scaling. In *The Memory Forum*, pages 1–4, 2014.

[16] Y. Kim, R. Daly, et al. Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. In *ISCA*, pages 361–372, 2014.

[17] D. Lee, Y. Kim, et al. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*, pages 615–626, 2013.

[18] J. Liu, B. Jaiyen, et al. RAIDR: Retention-aware intelligent DRAM refresh. In *ISCA*, pages 1–12, 2012.

[19] S. Liu, K. Pattabiraman, et al. Flikker: saving DRAM refresh-power through critical data partitioning. In *ASPLOS*, pages 213–224, 2011.

[20] J. Lucas, M. Alvarez-Mesa, et al. Sparkk: Quality-scalable approximate storage in DRAM. In *The Memory Forum*, pages 1–6, 2014.

[21] J. S. Miguel, M. Badr, et al. Load value approximation. In *MICRO*, pages 127–139, 2014.

[22] P. J. Nair, D.-H. Kim, et al. ArchShield: architectural framework for assisting DRAM scaling by tolerating high error rates. In *ISCA*, pages 72–83, 2013.

[23] J. M. Park, Y. S. Hwang, et al. 20nm DRAM: A new beginning of another revolution. In *IEDM*, pages 26.5.1–26.5.4, 2015.

[24] A. Raha, H. Jayakumar, et al. Quality-aware data allocation in approximate DRAM. In *CASES*, pages 89–98, 2015.

[25] A. Sampson, W. Dietl, et al. EnerJ: approximate data types for safe and general low-power computation. In *PLDI*, pages 164–174, 2011.

[26] A. Sampson, J. Nelson, et al. Approximate storage in solid-state memories. In *MICRO*, pages 25–36, 2013.

[27] W. Shin, J. Yang, et al. NUAT: A non-uniform access time memory controller. In *HPCA*, pages 464–475, 2014.

[28] Y. H. Son, O. Seongil, et al. Reducing memory access latency with asymmetric DRAM bank organizations. In *ISCA*, pages 380–391, 2013.

[29] A. Yazdanbakhsh, D. Mahajan, et al. AXBENCH. In *IEEE Design and Test, special issue on Computing in the Dark Silicon Era*, 2016.

[30] X. Zhang, Y. Zhang, et al. Exploiting DRAM restore time variations in deep sub-micron scaling. In *DATE*, pages 477–482, 2015.

[31] X. Zhang, Y. Zhang, et al. Restore truncation for performance improvement in future DRAM systems. In *HPCA*, pages 543–554, 2016.