ARCHITECTING MEMORY SYSTEMS UPON HIGHLY SCALED ERROR-PRONE MEMORY TECHNOLOGIES

By

Hao Wang

A Dissertation Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: ELECTRICAL ENGINEERING

Examining Committee:

____
Tong Zhang, Dissertation Adviser

____
Yannick L. Le Coz, Member

____
Gary J. Saulnier, Member

____
Christopher D. Carothers, Member

Rensselaer Polytechnic Institute
Troy, New York

May 2017
(For Graduation August 2017)

© Copyright 2017
by
Hao Wang
All Rights Reserved

CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

ACKNOWLEDGEMENTS x

ABSTRACT xi

1. Introduction 1
   - 1.1 DRAM basics 1
     - 1.1.1 DRAM memory Architecture 1
     - 1.1.2 High throughput 3D memory 2
   - 1.2 STT-RAM Memory Basics 3
   - 1.3 Thesis Outline 5

2. Summary of Motivations 7
   - 2.1 Scaling Challenge of Memory Industry 7
     - 2.1.1 System-aided DRAM Scaling 7
     - 2.1.2 System-aided STT-RAM Scaling 10
   - 2.2 Increasing demand of in-memory database 12

3. System-Aided Scaling of Memory and Data-dependent Error-tolerance 13
   - 3.1 Exploratory Study on System-aided DRAM Scaling 13
     - 3.1.1 Weak Cell Rate Estimation 13
     - 3.1.2 Reducing Redundancy Overhead through Data-dependent Error-tolerance 15
       - 3.1.2.1 Basic Concept 15
       - 3.1.2.2 Evaluation Results 16
   - 3.2 Optimizing the Use of STT-RAM in SSDs through Data-Dependent Error-Tolerance 20
     - 3.2.1 Basics of STT-RAM 20
     - 3.2.2 Replacing DRAM in SSDs 20
     - 3.2.3 Proposed Design Strategy 21
       - 3.2.3.1 Rationale and Key Ideas 21
       - 3.2.3.2 Practical Realization 24

3.2.4 Evaluations . . . 25
3.2.4.1 STT-RAM Modeling . . . 25
3.2.4.2 Effective Storage Capacity . . . 26
3.2.4.3 ECC Decoding Power Consumption . . . 28

4. Improving 3D DRAM Fault Tolerance through Weak Cell Aware Error Correction 32
4.1 Challenge in 3D memory . . . 32
4.2 Background . . . 35
4.2.1 3D DRAM Structure . . . 35
4.2.2 Related Work . . . 36
4.3 Proposed Design Solution . . . 37
4.3.1 Key Observations . . . 37
4.3.2 Formulation of the Design Solution . . . 38
4.3.3 Decoding Failure Probability Estimation . . . 42
4.3.4 Practical Implementation . . . 44
4.4 Evaluations . . . 47
4.4.1 Fault Tolerance Strength Analysis . . . 47
4.4.2 Hardware Overhead Estimation . . . 51
4.4.2.1 Virtual Repair Module . . . 52
4.4.2.2 Address Look-up Module . . . 54
4.4.2.3 BCH Coding Engines . . . 54
4.4.3 Read Latency Overhead . . . 56
4.4.4 System performance Overhead . . . 58
4.4.5 Comparison with Related Work . . . 61

5. On the Use of DRAM with Unrepaired Weak Cells in Computing Systems . . . 63
5.1 Introduction . . . 63
5.2 Proposed Design Solutions . . . 67
5.2.1 Effect of Fragmented Non-critical Memory Region . . . 67
5.2.2 Controller-based Page Re-mapping . . . 69
5.2.3 Re-cycling Error-prone Pages for zRAM . . . 73
5.2.3.1 zRAM Basics . . . 74
5.2.3.2 Proposed Software-based Error Tolerance . . . 74
5.3 Evaluation . . . 77
5.3.1 Controller-based Page Re-mapping . . . 77
5.3.1.1 Latency Overhead . . . 78

5.3.1.2 Hardware Overhead 79
5.3.2 Re-cycling Error-prone Pages for zRAM 80
6. Conclusion and Future Work 85
6.1 Conclusion 85
6.2 Future Work 87
References 88

LIST OF TABLES

4.1 Error distribution of one codeword-cell-group when using (160, 128, 9) BCH code. 52

4.2 Comparison with related prior work. 62

5.1 Mapping table entry structure 70

LIST OF FIGURES

1.1 DRAM storage cell . . . 2

1.2 Organization of hybrid memory cube. . . . 3

1.3 MTJ structure (a) anti-parallel state(high resistance) (b) parallel state(low resistance). . . . 4

2.1 Simulated sensing error vs. read disturbance under different sensing current. 11

3.1 Estimated weak cell rate under (a) different scaling factor α, and (b) different retention time. . . . 14

3.2 Illustration of proposed data-dependent error-tolerance design strategy. . . . 15

3.3 Calculated storage efficiency when storage node capacitance scaling factor α is (a) 0.53 (i.e., weak cell rate is 3.9×10^{-5}) and (b) 0.6 (i.e., weak cell rate is 1.2×10^{-6}). . . . 17

3.4 Illustration of (a) current design practice and (b) proposed data-dependent error-tolerance design strategy. . . . 22

3.5 Comparison of effective storage capacity in terms of user data page number when (a) k_a is 4B, and (b) k_a is 8B. . . . 28

3.6 Effective user data storage capacity under different (a) MTJ resistance deviation-to-average ratio σ, and (b) different maximum read intensity m_h. . . . 29

3.7 Estimated coding energy consumption per bit. . . . 30

3.8 Normalized ratio between effective storage capacity and coding energy consumption. . . . 31

4.1 Illustration of HMC chip structure. . . . 35

4.2 Flow diagram of the developed weak-cell-aware memory ECC design solution for (a) memory write, and (b) memory read. . . . 40

4.3 Illustration of the virtual repair module. . . . 45

4.4 Illustration of parallel architecture for implementing the developed design solution to meet the very high throughput of HMC chip. . . . 45

4.5 Codeword failure rate vs. weak cell rate ρ_w under different ECC codeword length and code rate with $\alpha = 1\%$, $\beta = \rho_w$ and $\rho_s = 10^{-12}$. Decoding failure rate is the ratio of codewords which cannot be successfully decoded over the total number of codewords. . . . 48

4.6 Codeword failure rate vs. weak cell rate ρ_w under different t_{ran} with $\alpha = 1\%$, $\beta = \rho_w$, $\rho_s = 10^{-12}$ and the use of BCH (572, 512, 13) code. 48

4.7 Codeword failure rate vs. weak cell rate ρ_w under different α with $\beta = \rho_w$, $\rho_s = 10^{-12}$ and the use of BCH(572, 512, 13) code. 49

4.8 Codeword failure rate vs. weak cell rate ρ_w under different β with $\alpha = 1\%$, $\rho_s = 10^{-12}$and the use of BCH(572, 512, 13) code. Decoding failure rate is the ratio of codewords which cannot be successfully decoded over the total number of codewords. 49

4.9 Hardware implementation overhead of (a) virtual repair, and (b) address look-up. 53

4.10 Estimated total silicon area cost of all the BCH coding modules (45nm node) in one HMC chip. 54

4.11 Estimated total power cost of all the BCH coding modules (45nm node) in one HMC chip. 55

4.12 Estimated average data read latency overhead in terms of the number of clock cycles. 56

4.13 Latency overhead measured from FPGA-based prototyping. 57

4.14 Latency overhead under different read intensity η. 58

4.15 Simulated CPU2006 benchmark execution time under the detected weak cell rate of 10^{-4} (normalized to the baseline scenario with error-free DRAM). 59

4.16 Normalized full-system IPC performance when using benchmarks of (a) Parsec and (b) Splash2 under different ρ_w with BCH (283, 256, 7) code. 60

4.17 Simulated read latency with (a) random read, and (b) sequential read using the modified HMC simulator. The parameter η represents that the ECC decoding latency is η times longer than the HMC vault average access latency. 61

5.2 Normalized system performance under different memory fragmentation. . . 66

5.1 Free block statistics under different memory fragmentation. 68

5.3 Illustration of the bijective mapping between error-prone pages in critical memory region and error-free pages in non-critical memory region. 69

5.4 One example to illustrate the mapping table research procedure with one collision. 73

5.5 Illustration of data write and read procedure with concatenated coding. . . . 75

5.6 Measured DRAM bit error rate vs. refresh period under different temperature. 76

5.7 Illustration of Ezspage structure with concatenated-CRC/BCH memory error tolerance. 77

5.8 Average table search latency under different error-prone page rate. 79

5.9 Page mapping table capacity under different error-prone page rate. 80

5.10 Measured throughput of (a) zRAM write and (b) zRAM read operations. . . 82

5.11 Read throughput of zRAM-CRC/BCH under different BCH code decoding occurrence probability. 83

ACKNOWLEDGEMENTS

I would like to take this opportunity to acknowledge the people helped me along the way I finish my PhD study.

First of all, I would like to say thank you to my advisor, Professor Tong Zhang, for his precious support and patient guidance to my research and study during the years at Rensselaer Polytechnic Institute. His profound knowledge, perpetual enthusiasm, and insightful ideas had motivated me all the years. Professor Zhang is sagacious and could always provide me the most important support and invaluable suggestions for me to conquer the various problems and difficulties in my research work.I will forever remember this rewarding and enjoyable experience working with him.

I would also like to say thank you to the professors serving on my PhD committee, Prof. Yannick L. Le Coz, Prof. Gary J. Saulnier, and Prof. Christopher D. Carothers. Their are providing me precious suggestions and comments from the thesis proposal to thesis completion. Those suggestions are so important for me to improve my research work .

My sincere thanks also go to all my lab colleagues who make our office a harmony place to work. I would like to thank Dr. Kai Zhao, Dr. Jiangpeng Li, Dr. Ning Zheng, Dr. Xuebing Zhang, Minjie Lyu, Yin Li and Xubin Chen for their encouragement and assistance in the past years.

Finally, and most importantly, I would like to thank my parents for their love and supports in the past years. Their encouragement is the greatest motivator and the most important spiritual support for me to finally finish this achievements.

ABSTRACT

DRAM (dynamic random access memory) technology has been fueling the computing industry for almost five decades and plays an essential role in enabling modern information technology infrastructure. However, as the DRAM technology scaling approaches 20nm and below, it has become increasingly challenging to maintain the historical bit cost reduction. In particular, with the DRAM technology scaling towards sub-20nm, it becomes more and more difficult to achieve sufficient DRAM data retention time. The DRAM cells with relatively shorter retention time are referred as weak cells and may fail to keep stored data in certain refresh period (e.g., 64ms or 128ms in current practice). Thus, tremendous efforts have been devoted to seeking alternative memory technologies. Several emerging memory technologies have been considered as the promising candidates, for example, Spin-Transfer-Torque(STT) RAM and Phase Change Memory (PCM). Although these emerging memory technologies may have advantages in scaling, they inevitably face cost, capacity and reliability challenges. In conventional practice, all the erroneous memory cells are masked by redundancy repair and error control codes (ECC), which are invisible to outside. However, it becomes impractical for memory industry to keep this design philosophy in sub-20 nm region.

This thesis presents a series of orthogonal memory system design techniques that leverage the characteristics of various applications to optimize memory fault tolerance in highly scaled memory technologies. This thesis advocates a system-aided scaling of memory and data-dependent error-tolerance design strategy that allows memory chips to provide erroneous bits. These erroneous bits are directly visible to and tolerated by system-level memory controller instead of memory chips themselves. This design is evaluated in the case of using DRAM and STT-RAM in solid-state drives (SSDs). By dynamically and jointly adjusting ECC configurations, the memory controller is able to adapt to the runtime data access characteristics. This technology contributes significant ECC redundancy saving and data reliability improvement.

3D memory chip stacking is also a promising technology which is an entirely new category of high-performance memory, delivering unprecedented system performance and

bandwidth. Although the emerging 3D DRAM products can significantly improve the computing system performance, the relatively high cost is one of the most critical issues that prevent their wide real-life adoption. Fortunately, system-aided DRAM scaling can very naturally fit the emerging 3D DRAM-controller integrated chips such as the hybrid memory cube (HMC). Under such a system-aided DRAM scaling design framework, the most crucial challenge is how to most effectively compensate the memory errors caused by the erroneous cells at minimal overheads in terms of data access latency and redundancy. Conventional ECC designs for memory focus on random errors while paying no attention to the feature of error patterns introduced by weak cells. Design strategy proposed in this thesis can tolerate the weak cell rate of as high as 10^{-4} and 6×10^{-5} if 100% and 90% of all the weak cells are known in prior. Using Micron's HMC 3D DRAM chips as the test vehicle, the evaluated implementation results show that it only consumes less than $0.4mm^2$ (45nm node) on the logic die. Using CPU and DRAM simulators, simulations are further carried out over a variety of computing benchmarks and the results show that this design solution only incurs less than 2% performance degradation on average.

Besides hardware based strategies, this thesis also presents a software-based solution on the use of DRAM with unrepaired weak cells in computing systems. The solution is based on the simple idea that operating system (OS) reserves all the error-prone pages, which contain at least one unrepaired weak cell, from being used. Under a relatively high error-prone page rate (e.g., 8%), it is almost impossible for OS to allocate a continuous fragmentation-free physical memory space for some critical operations. Moreover, reserving all the error-prone pages from practical usage could cause noticeable memory resource waste. Aiming to address these issues, this thesis presents a controller-based selective page remapping strategy to ensure a continuous critical memory region for OS and develops a software-based memory error tolerance scheme to recycle all the error-prone pages for the zRAM function in Linux. Experiments are carried out using SPEC CPU2006 and further study is performed on the latency, hardware cost and the effectiveness of recycling error-prone pages for zRAM in Linux. The experimental results show that the proposed software-based error tolerance scheme degrades the speed performance of zRAM by only up to 7%.

1. Introduction

1.1 DRAM basics

1.1.1 DRAM memory Architecture

Random-access memory(DRAM) is one type of random-access memory(RAM) which stores each bit of data in a separate capacitor within an integrated circuit. As Fig.1.1 shows, the capacitor has two states, either charged with electrons or discharged which are used to represent two value in digital system, 0 and 1. Since the transistors are not perfect devices, their eventual leakage current cause that the capacitors will slowly discharge, and the information eventually fades. To retain the information stored there, each capacitor needs to be refreshed periodically. Because of this refresh requirement, it is a dynamic memory as opposed to static random access memory (SRAM) and other static types of memory [1].

For a read operation, row decoder selects corresponding word line need to read and open all the transistors connected to the word line. Thus, the information stored in the capacitor will be transport to sense amplifier and select by column decoder. Than the required data will be sent to I/O buffer. At the end of read, sense amplifier will recharge the accessed capacitor so the information in the capacitor will be refreshed after once

---

Portions of this chapter previously appeared as: H. Wang, K. Zhao and T. Zhang, "Efficiently realizing weak cell aware DRAM error tolerance for sub-20nm technology nodes," in *7th IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2015, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang and T. Zhang, "An exploratory study on system-aided dram scaling," in *6th IEEE Int. Memory Workshop (IMW)*, Taipei, Taiwan, 2014, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang, K. Zhao, J. Li and T. Zhang, "Optimizing the use of STT-RAM in SSDs through data-dependent error tolerance," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2743-2747, Nov. 2015.

Portions of this chapter previously appeared as: H. Wang, K. Zhao, M. Lv, X. Zhang, H. Sun and T. Zhang, "Improving 3D DRAM fault tolerance through weak cell aware error correction," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 820-833, Oct. 2016.

Portions of this chapter previously appeared as: H. Wang, Y. Li, X. Zhang, X. Zhao, H. Sun and T. Zhang, "On the Use of DRAM with unrepaired weak cells in computing systems," in *Proc. ACM 2nd Int. Symp. on Memory Syst.*, Washington DC, USA, 2016, pp. 327-337.

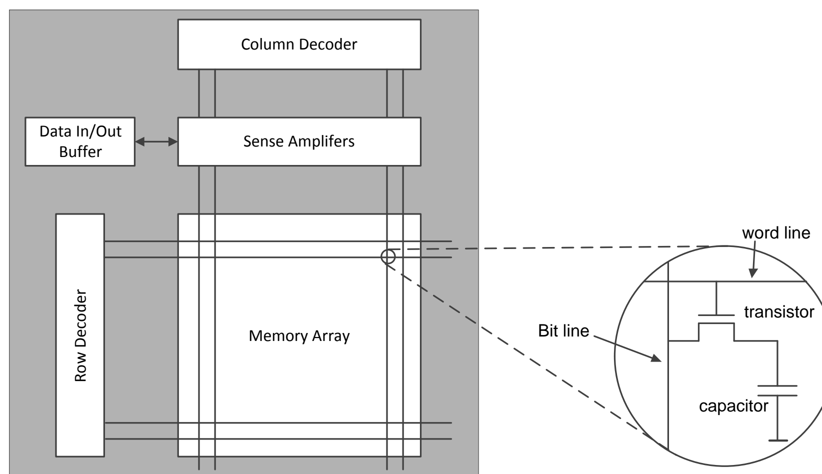


Figure 1.1: DRAM storage cell

read. So essentially, refresh is automatic read arranged by memory controller to keep data integrity in DRAM array. For modern memory design, The refresh interval (time between refreshes for a given cell) has remained constant at 64 ms for several DRAM generations [2].

1.1.2 High throughput 3D memory

Modern memory technology has started to explored 3D architecture design in order to meet increasing requirement in memory density and throughput. New technology has already been commercialized such as HMC and high bandwidth memory(HBM) which are able to provide remarkable throughput performance [3].

An HMC is vertically organized memory device that consists of one logic layer and multiple DRAM layers using through-silicon via (TSV) technology. It has two capacity configurations, 4 GB and 8 GB. The structure of an 8 GB HMC is shown in Fig. 4.1. It has eight DRAM layers and one logic die. Each DRAM layer has 1 GB capacity and is divided into of 32 partitions which contains four DRAM banks so one 8-GB HMC has 256 DRAM banks [3]. All vertically adjacent DRAM die partitions form a *vault* and controlled by vault controller in logic die. Vault controller which realizes refresh function of each vault has a reference queue used to buffer references for its own vault memory. Thus, vault controller can independently reorder memory references for latency optimization rather than executing references exactly as order of arrival. Independent

vault controllers enables 32 concurrent operations in one HMC.

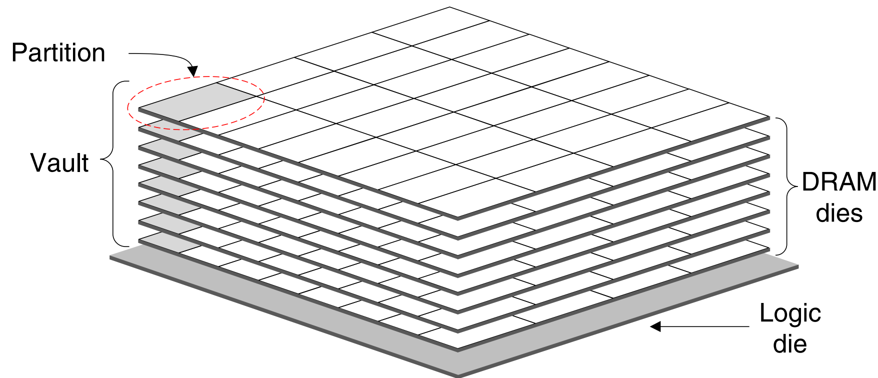


Figure 1.2: Organization of hybrid memory cube.

HMC uses I/O links to communicate with host or other HMC. Each link is comprised of 16 input lanes and 16 output lanes of which speed ranges from 12.5 to 30 Gb/s. Thus one link is capable to provide maximum $32 \times 30 = 120$ GB/s bandwidth in full duplex operation. HMC has 2 link package and 4 link package which provides maximum aggregated bandwidth as $120 \times 4 = 480$ GB/s. Inside HMC, each link is associated with eight local vault denoted as *quadrant*. link communicates with all vaults through a crossbar but access to a vault outside of quadrant has relatively higher latency than to one inside quadrant. HMC supports both read and write granularity from 16 bytes up to 128 bytes block size. HMC can also achieve low energy consumption. Energy per bit in for HMC prototype is measured as 10.48 pj/bit which saves 83% compared with 65 pj/bit for DDR 3 [4].

In general, HMC or other 3D memory technology are evolutionary solution of memory bandwidth wall which improves memory latency, bandwidth, power and density characteristics.

1.2 STT-RAM Memory Basics

Spin-transfer torque(STT)-RAM is a new technology which is considered most promising as competitor of SRAM for cache and DRAM for main memory [5]. Different from DRAM, STT-RAM stores data information in a magnetic tunneling junction(MTJ) instead of a capacitor. As shown in Fig. 1.3, the MTJ is consisted of two ferromagnetic layers and one oxide barrier layer, e.g. MgO. Magnetization direction of one ferromagnetic

layer is fixed and it is regarded as reference layer while magnetization direction of the other ferromagnetic layer can be change by passing a enough current though the MTJ. The resistance of MTJ depends on the relative magnetization directions of the two ferromagnetic layers, i.e., When free layer and reference layer have different magnetization direction, MTJ shows a high resistance state referred as anti-parallel state, leading to 1-bit storage per MTJ. On the other hand, when these two layers have the same magnetization direction, the MTJ is in a a low resistance state denoted as parallel state. Anti-parallel and parallel states are used to represent 1 and 0 in digital system respectively [6]. Let R_L and R_H denote low and high resistance of the MTJ. The tunneling magneto-resistance ratio(TMR) is conventionally defined as $(R_H - R_L)/R_L$. Clearly, a larger TMR makes it easier to distinguish the two resistance states and hence is highly desirable.

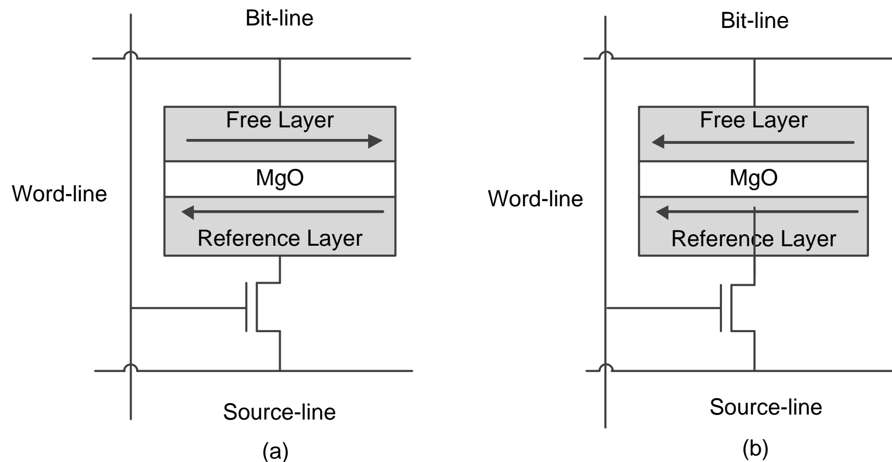


Figure 1.3: MTJ structure (a) anti-parallel state(high resistance) (b) parallel state(low resistance).

Though various design of STT-RAM have been proposed, transistor-one-MTJ(1T1J) structure is the most popular because of it simplicity and potential for scaling [6–8]. Each STT-RAM cell contains one MTJ as the storage element and one nMOS transistor as the access control device. In order to ensure correct write operations, the size (or width) of the nMOS transistor within each STT-RAM cell must be sufficiently large to sustain a write current larger than the switching threshold. As a result, it is very critical to reduce the switching threshold current of MTJ devices as much as possible, which can be accomplished by technology scaling and/or material/device innovations. Recent breakthroughs on the development of MTJ with perpendicular magnetization (also called perpendicular-

MTJ) successfully reveal the practical feasibility of sub-100μA switching threshold, e.g., perpendicular-MTJ with switching threshold of 50μA at 30nm node has been recently demonstrated [9]. Because of the virtually unlimited cycling endurance of MTJ devices, the development of perpendicular-MTJ makes STT-RAM become the most viable candidate for large-capacity on-chip embedded memory (e.g., last-level cache) and DRAM-replacement.

The parallel and anti-parallel magnetization are realized by steering a write current, which must be larger than a switching threshold, directly through MTJs along opposite directions. When enough current pass the MTJ from free layer to reference layer, MTJ switch from 1 to 0 and vice versa. STT-RAM read is realized by steering a sensing current through MTJ to transform the MTJ-based data storage information from the resistance domain into the voltage domain. Since both write and read involve through-MTJ current, the sensing current for memory read has to be sufficiently lower than the switching current threshold in order to avoid read disturbance. Considering read current with state 0 is greater than than rad current with state 1, read current is designed to apply from bit-line to source line in or to minimize read disturbance [10]. Let I_{c_0} denote the MTJ switching current threshold and I_c denote the sensing current (apparently we have $I_c < I_{c_0}$). The probability of read disturbance reduces as we reduce I_c (and hence I_c/I_{c_0}) [11, 12]. Let $f_r(I_c)$ denote the read disturbance probability with the sensing current I_c, it can be estimated as

$$f_r(I_c) = 1 - \exp\{-\frac{\tau_p}{\tau_0}\exp[\frac{E}{k_B T}(1 - \frac{I_c}{I_{c_0}})]\}, \quad (1.1)$$

where τ_p is sensing time applied to MTJ (which is set as 10ns in our simulations), timing constant τ_0 is 1ns, E is the energy barrier, k_B is the Boltzmann constant, and T is the temperature.

1.3 Thesis Outline

The rest of thesis is organized as follows: Chapter. 2 introduced the motivations of methods presented in this thesis on memory system design. Chapter 3 introduces system-aided memory scaling and data-dependent error tolerance design, including two evaluation cases which use DRAM/STT-RAM as cache in SSDs. In Chapter 4, I introduced a

novel error tolerance design for highly scaled memory, weak cell aware decoding, which naturally fit 3D DRAM chip as HMC and shows significant improvement in memory reliability and saving in ECC redundancy. Another brand new software error tolerance design is given in Chapter 5 which adopts fast data compression and software decoding technology to reutilize error prone physical DRAM pages. And finally, conclusions and future work are drawn in Chapter 6.

2. Summary of Motivations

2.1 Scaling Challenge of Memory Industry

2.1.1 System-aided DRAM Scaling

Over the past decade, tremendous amount of research efforts has been devoted to new memory technologies, e.g., STT-RAM (spin-transfer torque RAM), PCRAM (phase-change RAM), and ReRAM (resistive RAM). This has been largely driven by the well-recognized grand challenges faced by DRAM scaling [14, 15]. In current DRAM design practice, the storage node capacitance must be large enough to guarantee a sufficient bit-line sensing voltage margin under the worst-case scenario, i.e., with the leakiest access transistor, retention time of the refresh period (e.g., 64ms or 128ms), and highest operating temperature (e.g., 85 °C). Therefore, as the industry continues to reduce DRAM cell size with the technology scaling down, the aspect ratio (A/R) of the storage node must accordingly scale up in order to maintain almost the same amount of storage node capacitance, i.e., the scaling down of memory cell size must be strictly coupled with the scaling up of storage node A/R. It becomes increasingly challenging to sustain such a *coupled* scaling as the DRAM industry strives to move into the sub-25nm regime.

Since it is so difficult to maintain the conventional coupled DRAM scaling, we

---

Portions of this chapter previously appeared as: H. Wang and T. Zhang, "An exploratory study on system-aided dram scaling," in *6th IEEE Int. Memory Workshop (IMW)*, Taipei, Taiwan, 2014, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang, K. Zhao and T. Zhang, "Efficiently realizing weak cell aware DRAM error tolerance for sub-20nm technology nodes," in *7th IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2015, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang, K. Zhao, J. Li and T. Zhang, "Optimizing the use of STT-RAM in SSDs through data-dependent error tolerance," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2743-2747, Nov. 2015.

Portions of this chapter previously appeared as: H. Wang, K. Zhao, M. Lv, X. Zhang, H. Sun and T. Zhang, "Improving 3D DRAM fault tolerance through weak cell aware error correction," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 820-833, Oct. 2016.

Portions of this chapter previously appeared as: H. Wang, Y. Li, X. Zhang, X. Zhao, H. Sun and T. Zhang, "On the Use of DRAM with unrepaired weak cells in computing systems," in *Proc. ACM 2nd Int. Symp. on Memory Syst.*, Washington DC, USA, 2016, pp. 327-337.

argue system-aided scaling in which the demand on the scaling up of storage node A/R, or even keep the same A/R can be relaxed with the continuous scaling down of DRAM cell size [16]. At the first glance, such a decoupled DRAM scaling does not appear to be a viable option because it apparently cannot maintain the error-free DRAM data storage (except radiation-induced soft errors) as demanded by current DRAM products. Nevertheless, it is evident that almost all the new memory technologies are fundamentally subject to various reliability issues (such as write failures, read disturb, sensing errors, and cycling endurance), and hence most likely cannot achieve the same error-free data storage as in current DRAM products, especially for sub-25nm technology nodes. Therefore, as an alternative to shifting from DRAM to other new memory technologies, evolving from coupled DRAM scaling to decoupled DRAM scaling may not be as unfeasible as it may appear.

We argue that the decoupled DRAM scaling is actually an attractive option for three main reasons: (i) It is well known that, in current DRAM chips, the storage node capacitance is large enough to ensure hundreds of ms and even a few seconds of data retention for the vast majority of DRAM cells, and the typical refresh period of 64ms or 128ms is set by relatively few DRAM cells with very leaky access transistors [17–19]. Therefore, although decoupled DRAM scaling inevitably introduces weak memory cells that cannot ensure the data storage integrity, the number of weak cells may not be significant even if we keep the same refresh period of 64ms or 128ms. (ii) Although conventional redundancy repair schemes may not be able to handle all the weak cells in decoupled DRAM scaling, we can expose the bit errors caused by weak cells to the system-level memory controller, and hence maintain the existing standard memory interfaces (e.g., DDRx). As a result, the industry can continue to benefit from the tremendous investment over the decades on DRAM-specific computer architecture and software optimizations (e.g., various techniques for exploiting page-mode DRAM access). Essentially, such a system-aided DRAM scaling resembles the scaling of NAND flash memory that increasingly relies on system-level error tolerance. (iii) Even though system-level error tolerance for mitigating weak cells may introduce DRAM access latency overhead, it may be easily off-set by the noticeable advantages of DRAM over other new memory technologies in terms of memory cell write/sensing latency. In addition, such a system-aided DRAM

scaling can very naturally fit to the emerging 3D DRAM-controller integrated chips such as the HMC from Micron.

As DRAM cell scaled down, critical problem is to tolerate increasing weak cells in DRAM dies for memory reliability. HMC already uses built-in self test (BIST) technique to remove faulty cells in DRAM dies which is a traditional approach and based on extra memory redundancy and hardwired remapping. For 3D stacked DRAM, though additional resource can be further shared in neighbour DRAM dies, rerouting memory request to reservoir cells becomes increasingly challenging as weak cell number augments [20, 21]. Direct-mapped cache is also used in DRAM memory design to replace faulty words [22]. Nevertheless, it is still costly to replace all the faulty words without combination of other efficient error tolerance solution.

ECC is another important technique to address memory fault. Several kinds of two-tired ECC protection strategies have been proposed to achieve balance between data reliability and storage efficiency [23–25]. Generally, they use simple first tier ECC to detect local errors while second tier extra ECC redundancy are stored in DRAM as data. When first tier ECC detects error or fail to correct error, second tier ECC needs to be accessed for further error correction. Though several methods are proposed to reduce the second tie error correction redundancy in all previous work, memory overhead at storage efficiency is still remarkable. Moreover, previous ECC design for memory treat weak cell errors and soft errors identically while pay no attention to characteristics of error introduced by weak cells.

Different from work above, ArchShield is another error tolerance solution which utilizes weak cell location information to assist DRAM scaling. It can tolerate weak cell rate as high as 10^{-4} [26]. However, ArchShield needs to keep fault map of faulty words in on-chip cache and accesses the fault map every memory read request. Then it duplicates these faulty words in extra memory space referred as replication area. For a 8 GB DIMM, ArchShield costs 64 MB fault map and 256 MB replication area. Moreover, further modification in operating system is also necessary to implement ArchShield.

Above all, previous developed error tolerance solutions are not suitable to assist DRAM scaling considering their efficiency. Thus, the use of system-level memory error tolerance for mitigating memory errors is subject to two major issues: (i) *Storage*

redundancy overhead: DRAM dies has to store certain redundancy to facilitate the error tolerance, which reduces the effective DRAM storage density and hence increases the effective bit cost. (ii) *Latency and energy consumption overhead*: Memory error-tolerance operations will increase the DRAM access latency and consume energy, which can degrade the overall HMC speed performance and energy efficiency. In essence, the practical feasibility of system-aided DRAM scaling in HMC depends on how well these two issues can be addressed.

2.1.2 System-aided STT-RAM Scaling

Recent breakthroughs on the development of MTJ (magnetic tunnel junction) with perpendicular magnetization [9,28] reveal the practical feasibility of STT-RAM with sub-100μA switching current and sub-10ns switching time. Although it largely reduces the memory cell size, significant reduction of switching current can meanwhile degrade STT-RAM read reliability [12]. This is mainly because of the smaller operational window of memory sensing current I_c and the fundamental confliction between read disturbance $f_r(I_c)$ and memory sensing error. To be more specific, the magnitude of sensing current affects read disturbance and sensing error in an opposite manner, i.e., a larger sensing current can improve the sensing accuracy but meanwhile increase the probability of read disturbance, and vice versa. Ideally, memory sensing current should fall into a *safety zone* that ensures sufficiently low (or negligible) probabilities of both read disturbance and sensing error. However, under the continuous reduction of switching current (e.g., even only 50μA at 32nm node [9]), the sensing current safety zone increasingly diminishes, leading to non-negligible and even alarmingly high probabilities of read disturbance and/or sensing error [13].

This clearly suggests a fundamental trade-off between read disturbance and read error, which strongly depends on the sensing current I_c. Ideally, I_c should fall into a safety zone that can ensure sufficiently low (or negligible) probabilities for both read disturbance and sensing error. Unfortunately, as STT-RAM write current approaches 100μA and below with the technology scaling, the sensing current safety zone quickly diminishes [12]. This can be clearly observed from Fig. 2.1, which quantitatively shows the dependency of read disturbance vs. sensing error trade-offs on sensing current at 45nm and 22nm

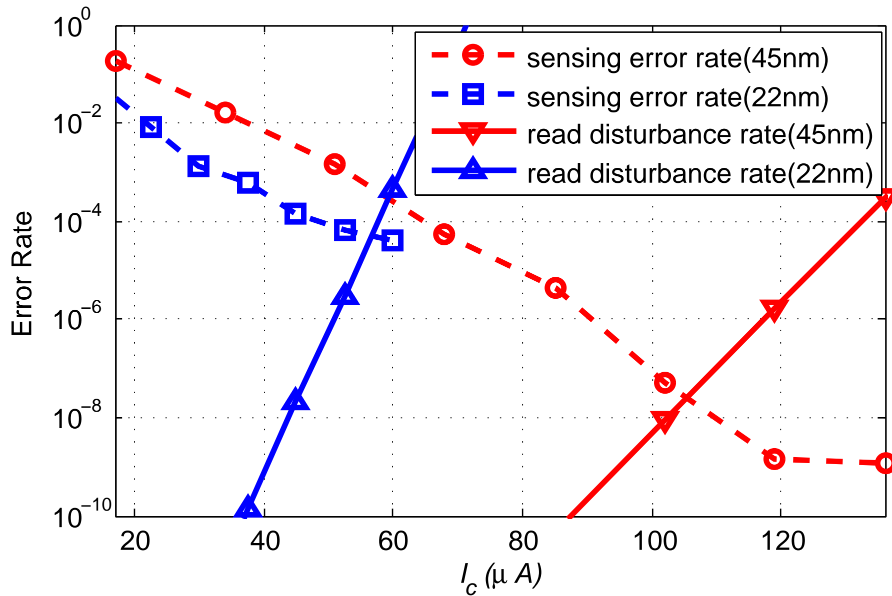


Figure 2.1: Simulated sensing error vs. read disturbance under different sensing current.

nodes. In our simulations, we set the MTJ TMR as 100% and the MTJ resistance standard deviation σ as 6% of the average resistance, and set I_{c_0} for 45nm and 22nm nodes as 170μA and 75μA, respectively [11]. The sensing current varies from 10% I_{c_0} to 80% I_{c_0}. We designed the memory sensing circuit using the structure presented in [29], and obtained the memory sensing error rate under different sensing current through Monte Carlo SPICE simulations.

Much effort have been donated to enhance reliability of STT-RAM to cell architecture [30, 31]. To eliminate read error, more sophisticated sensing and reference circuit design are developed [32–34]. Moreover, various designs have been proposed to make write current configurable or aware of STT-RAM asymmetry of write operation [35–37]. STT-RAM are explored as future non-volatile replacement of SRAM cache or even competitor of main memory with DRAM and STT-RAM shows competitive performance and remarkable advantage in energy efficiency [5, 38–40].

Just as other memory technology, memory fault tolerance techniques, in particular ECC because of the random and transient nature of read disturbance and sensing error, become increasingly indispensable. Many ECC strategies have been proposed to improve STT-RAM reliability and denisity [41–43]. [44] even use data-dependent ECC to utilize

the asymmetry of STT-RAM write operation. However, compared with DRAM, STT-RAM has more complicated sensing and reference circuit. Variation between STT-RAM cells make it challenging to maintain data integrity in deep sub-micron.

2.2 Increasing demand of in-memory database

In-memory database(IMDB) is a database management system that implemented in main memory for computer storage in contrast with traditional database system relies in disk storage. Main memory is able to respond query from CPU with much faster speed than disk system, even NAND flash based disk system. Moreover, since it is byte-addressable, it provide the opportunity to further optimize the database system design. In recent decades, many in-memory database system has been developed such as Memcached and RAMcloud [45, 46]. Even though density of DRAM keeps increasing, memory industry has already been confronted with the challenge for sub-25nm scaling of DRAM which contrasts with the growing demand of DRAM. The cost of memory constrains development of the high performance IMDB.

Thus, it becomes practical to further explore the boundary of modern memory utilization in the IMDB. Relaxing memory scaling for DRAM or even emerging memory technology such as STT-RAM will contribute to reduce memory cost of IMDB while it requires special fault tolerance technique to ensure reliability of the IMDB.

3. System-Aided Scaling of Memory and Data-dependent Error-tolerance

3.1 Exploratory Study on System-aided DRAM Scaling

3.1.1 Weak Cell Rate Estimation

This section discusses the estimation of weak cell rate in the presence of reduced storage node capacitance, and presents experimental results. Let C_{BL} and C_{SN} denote the bit-line capacitance and storage node capacitance, respective, V_{DD} denote the power supply voltage, i_L denote the overall leakage current of memory cell, and t_{ret} denote the data retention time, we can estimate the bit-line sensing voltage swing as

$$\Delta V_{BL} = \frac{\frac{V_{DD}}{2} \cdot C_{SN} - i_L \cdot t_{ret}}{C_{BL} + C_{SN}}. \tag{3.1}$$

Since the leakage current i_L exhibits a large degree of variability due to the process variation [17, 47], ΔV_{BL} can noticeably vary among all the DRAM cells. To ensure the integrity of memory sensing in the presence of various noises in high-density DRAM arrays, conventional DRAM design practice demands that the worst-case ΔV_{BL} should not be less than a minimum bit-line sensing voltage swing V_{min}. Memory cells, which cannot ensure $\Delta V_{BL} \geq V_{min}$ due to their leaky access transistors, are called weak cells. In conventional coupled DRAM scaling, the storage node capacitance remains sufficiently large so that weak cells are very few and hence can be easily handled with redundancy repair. As a result, DRAM chips can always ensure error-free operation (except radiation-induced soft errors).

Portions of this chapter previously appeared as: H. Wang and T. Zhang, "An exploratory study on system-aided dram scaling," in *6th IEEE Int. Memory Workshop (IMW)*, Taipei, Taiwan, 2014, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang, K. Zhao and T. Zhang, "Efficiently realizing weak cell aware DRAM error tolerance for sub-20nm technology nodes," in *7th IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2015, pp. 1-4.

Portions of this chapter previously appeared as: H. Wang, K. Zhao, J. Li and T. Zhang, "Optimizing the use of STT-RAM in SSDs through data-dependent error tolerance," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2743-2747, Nov. 2015.

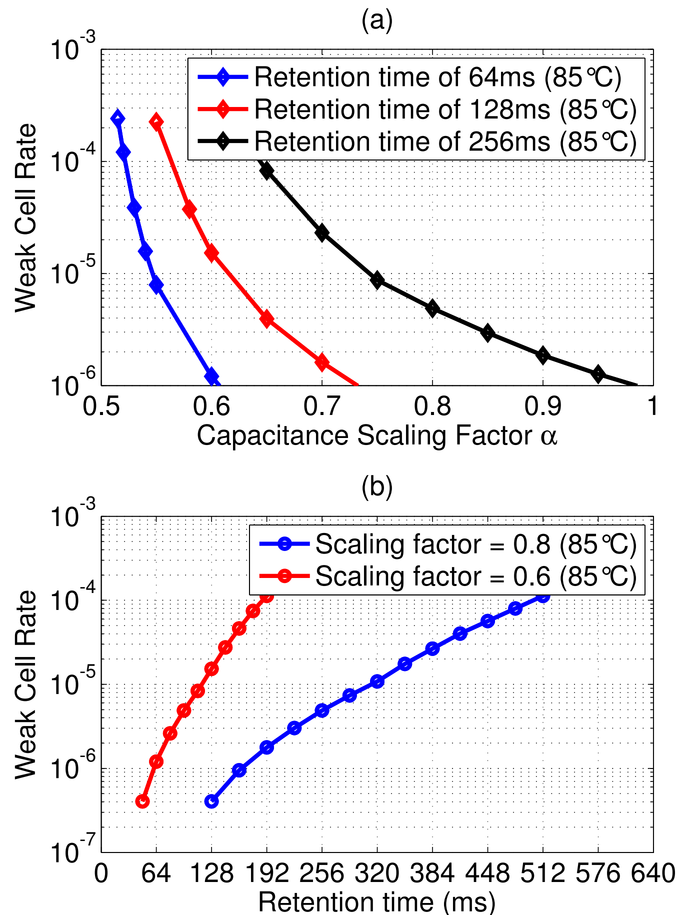


Figure 3.1: Estimated weak cell rate under (a) different scaling factor α, and (b) different retention time.

To emulate the scenario of relaxing the scaling up of storage node A/R, we assume that the storage node capacitance C_{SN} scales down with a factor α, while V_{DD}, C_{BL}, V_{min}, and the distribution of i_L all remain the same. As we reduce the scaling factor α, more and more DRAM cells will become weak cells. We define *weak cell rate* as the probability that one DRAM cell is a weak cell. To quantify the dependency of weak cell rate on the scaling factor α, we can use the following procedure: (i) With commercial DRAM chips, we measure the weak cell rates under different retention time t_{ret} that is (much) longer than the typical value; (ii) Based upon the measured weak cell rates, we estimate the distribution of the leakage current i_L according to (3.1) with realistic values of the parameters V_{DD}, C_{BL}, C_{SN}, and V_{min}; (iii) We accordingly estimate the distribution of ΔV_{BL} under different scaling factors α with the normal retention time, based upon which

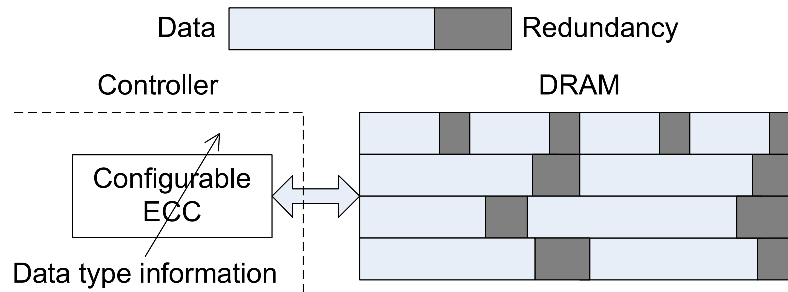


Figure 3.2: Illustration of proposed data-dependent error-tolerance design strategy.

we can estimate the dependency of weak cell rate on the scaling factor α.

This work uses the measurement results of 3xnm DRAM error characteristics reported in [19] under various retention time that is as long as 1024ms under 85°C, and set V_{DD} as 1.5V, C_{BL} as 100fF, C_{SN} as 32fF, and V_{min} as 0.1V. Following the procedure described above, we estimate the weak cell rate vs. scaling factor α under retention time of 64ms, 128ms, and 256ms and the results are shown in Fig. 3.1(a). Fig. 3.1(b) shows the estimated dependency of weak cell rate on data retention time under two different scaling factors. The noticeable weak cell rate demands the use of powerful fault-tolerance to ensure the overall data storage integrity. Conventionally, weak memory cells are handled by redundancy repair with redundant word/bit-lines and sub-arrays. This however fails to work as weak cell rate becomes significant (e.g., beyond 10^{-6}). Hence, a combination of redundancy repair and error correction coding (ECC) is necessary.

3.1.2 Reducing Redundancy Overhead through Data-dependent Error-tolerance

3.1.2.1 Basic Concept

With the full knowledge of the data being stored in DRAM, the host can address the redundancy and access latency issues in a data-dependent manner by fully exploiting the runtime data characteristics. Focusing on reducing redundancy overhead, this work proposes to adjust ECC codeword length in adaptation to the data access granularity. It is well-known that a longer ECC codeword length leads to a higher code rate (i.e., less coding redundancy), which nevertheless comes with a coarse data access granularity. Hence, if the host memory controller is fully aware of the access unit length of current data, it can accordingly adjust the ECC codeword length in order to reduce the coding redundancy without affecting the data accessability, as illustrated in Fig. 3.2.

To evaluate the potential of this data-dependent error-tolerance design concept, this work considers the case of using DRAM in SSDs, where DRAM stores a few different types of data, including address mapping table and buffered/cached user data. The size of each entry in DRAM buffer/cache equals to the size of each basic I/O, which typically ranges between 512B and 4kB in most computing systems. Therefore, to protect each buffer/cache entry, the ECC codeword length can be as large as the size of the basic I/O. On the other hand, each entry in address mapping table is much smaller. For example, using flash memory with 8kB physical page size, a 512GB SSD contains about 64M (i.e., 2^{26}) physical pages. Hence, we only need less than 4B to represent each entry in the address mapping table. Clearly, we can use different ECC codeword length (with different coding redundancy) for protecting these two different types of data in order to minimize the overall ECC coding redundancy.

3.1.2.2 Evaluation Results

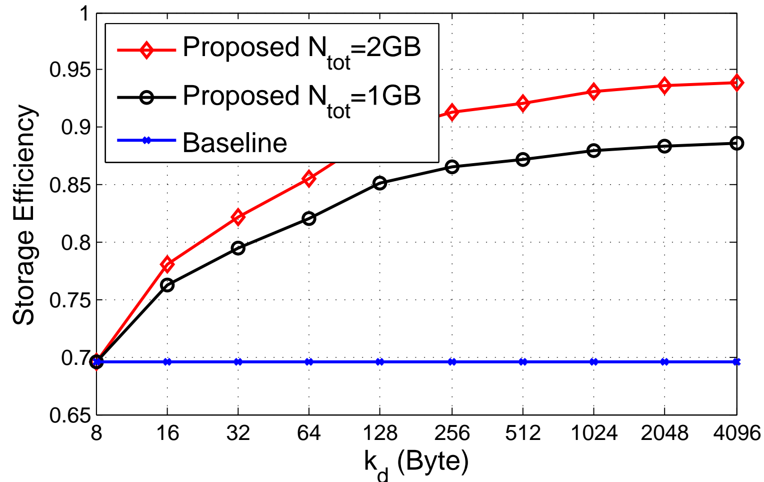
We carried out simulations to evaluate the effectiveness of using the proposed design technique to reduce the redundancy overhead in system-aided DRAM scaling. We use binary BCH codes as ECC in this study, and one BCH code is denoted as (n, k, t), i.e., each BCH codeword contains n bits, protects k user bits, and can correct up to t errors. Given a (n, k, t) code, we define *weak word rate* p_{weak_w} as the probability that a codeword is stored in n cells in which more than t cells are weak cells (i.e., the BCH code cannot ensure the storage integrity). The weak word rate p_{weak_w} should be sufficiently low in order to avoid noticeable impact on DRAM yield. Given the DRAM weak cell rate p_{weak_c}, target weak word rate p_{weak_w}, and k, and assuming that weak cells randomly distribute among all the DRAM cells, we can calculate n and t using

$$n = k + \lceil \log_2 n \rceil \cdot t \tag{3.2}$$

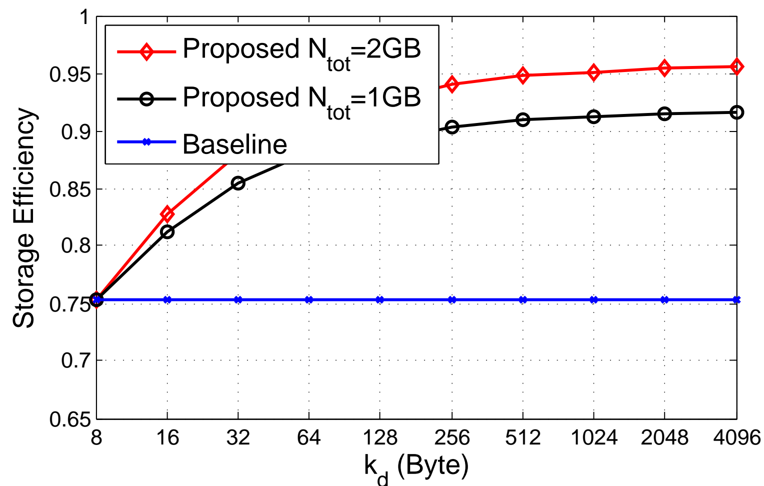
$$p_{weak_w} \geq \sum_{s=t+1}^{n} \binom{n}{s} \cdot p_{weak_c}^{s} \cdot (1 - p_{weak_c})^{n-s} \tag{3.3}$$

As discussed above, data-dependent error-tolerance treats the mapping table and cached/buffered user data in DRAM differently. Let k_a and k_d, where $k_a << k_d$, de-

note the number of bits being protected by each codeword for mapping table and all the cached/buffered data, respectively. We can determine the BCH codes for protecting address mapping table and cached/buffered data, denoted as $(n_a,\ k_a,\ t_a)$ and $(n_d,\ k_d,\ t_d)$, based upon (3.9) and (3.3).



(a) Scaling factor α=0.53



(b) Scaling factor α=0.6

Figure 3.3: Calculated storage efficiency when storage node capacitance scaling factor α is (a) 0.53 (i.e., weak cell rate is 3.9×10^{-5}) and (b) 0.6 (i.e., weak cell rate is 1.2×10^{-6}).

We use the conventional design practice as the baseline (i.e., all the data are protected by the same ECC). Since each mapping table entry is much shorter than the user data being stored in DRAM, we use the BCH code $(n_a,\ k_a,\ t_a)$ as the ECC for the base-

line scenario. In this work, we set the target weak word rate p_{weak_w} as 10^{-13} to ensure negligible impact on DRAM yield, e.g., assuming there are total 1G words in DRAM, the probability that none of the words is a weak word is larger than 99.99%. Assume the DRAM has a total capacity of N_{tot} bits and must store N_a bits of address mapping table data. Hence the DRAM uses $\lceil N_a/k_a \rceil \cdot n_a$ bits to store the address mapping table, including $\lceil N_a/k_a \rceil \cdot (n_a - k_a)$ bits of coding redundancy. This leaves $N_d = N_{tot} - \lceil N_a/k_a \rceil \cdot n_a$ bits available to store user data. Let L_p denote the size of each user data page (e.g., 512B or 4kB). For the baseline scenario using current design practice, the number of user pages that can be stored in DRAM is

$$s^{(base)} = \frac{\lfloor N_d/n_a \rfloor}{\lceil L_p/k_a \rceil}, \tag{3.4}$$

and the total amount of redundancy stored in DRAM is

$$r^{(base)} = \left(\lceil N_a/k_a \rceil + \lfloor N_d/n_a \rfloor \right) \cdot (n_a - n_k). \tag{3.5}$$

When using the data-dependent error-tolerance, the number of user pages that can be stored in DRAM increases to

$$s^{(prop)} = \frac{\lfloor N_d/n_d \rfloor}{\lceil L_p/k_d \rceil}, \tag{3.6}$$

and the total amount of redundancy stored in DRAM is

$$r^{(prop)} = \lceil N_a/k_a \rceil \cdot (n_a - k_a) + \lfloor N_d/n_d \rfloor \cdot (n_d - k_d). \tag{3.7}$$

Let us consider a 512GB SSD, and suppose the flash memory has 8kB physical page size, the 512GB SSD contains at least 64M (i.e., 2^{26}) physical pages (i.e., we need less than 4B to represent each entry in the address mapping table). Let each mapping table entry takes 4B, the entire mapping table contains N_a=256MB of data and hence occupies a total $(256\lceil n_a/k_a \rceil)$MB of storage space after taking into account of the BCH coding redundancy. We consider two different DRAM storage capacities of N_{tot}=1GB and N_{tot}=2GB. Hence, the rest storage space of $(1024 - 256\lceil n_a/k_a \rceil)$MB and $(2048 - 256\lceil n_a/k_a \rceil)$MB are available to store cached/buffered user data, respectively. As shown in Fig. 3.1, with the refresh time of 64ms, the weak cell rate is around 3.9×10^{-5} and

1.2×10^{-6} when the scaling factor α is around 0.53 and 0.6, respectively. Let r denote the total amount of coding redundancy stored in DRAM and recall that N_{tot} denote the total DRAM storage capacity, we define the DRAM storage efficiency as $\gamma = (N_{tot} - r)/N_{tot}$.

By setting k_a as 8B, we calculate the DRAM storage efficiency γ under various k_d for the two different scaling factors, as shown in Fig. 3.3. Baseline scenario uses the same BCH code to protect all the data, hence its storage efficiency is independent from k_d. As shown in Fig. 3.3, when using the proposed data-dependent error-tolerance design strategy, the achieved storage efficiency noticeably increases as we increase k_d. Let $m(\alpha)$ denote the number of DRAM cells that can be manufactured within a unit area under the storage node capacitance scaling factor α, we define the storage density scaling factor $\eta = m(\alpha)/m(1)$. Given a scaling factor α, as long as the corresponding storage efficiency γ is greater than $1/\eta$, the proposed system-aided DRAM scaling can achieve a net reduction of effective bit cost. As shown in Fig. 3.3, given the relatively small scaling factor of 0.53 and 0.6, the achievable storage efficiency γ of even the baseline scenario may already ensure a net gain, and the proposed design strategy can largely improve the storage efficiency and hence achieve a much more noticeable net reduction of DRAM bit cost.

This section presents an exploratory study on system-aided DRAM scaling that aims to relax the scaling up of storage node A/R in the presence of continuous scaling down of DRAM cell size. Such a relaxed scaling directly results in storage node capacitance reduction and hence more weak memory cells. I propose to explicitly expose bit errors caused by the weak cells to system-level memory controllers, leading to a system-aided DRAM scaling. As the first step to explore this unconventional DRAM scaling strategy, I carried out analysis and calculations to estimate weak cell rate in the presence of reduced storage node capacitance based upon measurement results with 3xnm DRAM chips. I further presented a data-dependent error-tolerance design strategy to reduce the redundancy overhead of system-aided weak cell mitigation, and carried out quantitative evaluation in the case of using DRAM in SSDs. System-aided DRAM scaling certainly faces many open research issues, and it is our hope that this exploratory study will lead to much more thorough and cohesive investigations from the system and circuit/device research community on this possible option for future DRAM scaling.

3.2 Optimizing the Use of STT-RAM in SSDs through Data-Dependent Error-Tolerance

3.2.1 Basics of STT-RAM

As the basic storage element in STT-RAM, each MTJ has two ferromagnetic layers separated by one oxide barrier layer. The resistance of MTJ depends on the relative magnetization directions of the two ferromagnetic layers, i.e., when the magnetization is parallel (or anti-parallel), MTJ is in low (or high) resistance state, leading to 1-bit storage per MTJ. The parallel and anti-parallel magnetization are realized by steering a write current, which must be larger than a switching threshold, directly through MTJs along opposite directions. Let R_h and R_l denote the high and low MTJ resistance, respectively, $(R_h - R_l)/R_l$ is conventionally referred to as tunneling magneto-resistance ratio (TMR). Clearly, a larger TMR makes it easier to distinguish the two resistance states and hence is highly desirable.

Each STT-RAM cell contains one MTJ as the storage element and one nMOS transistor. as the access control device. In order to ensure correct write operations, the size (or width) of the nMOS transistor within each STT-RAM cell must be sufficiently large to sustain a write current larger than the switching threshold. As a result, it is very critical to reduce the switching threshold current of MTJ devices as much as possible. which can be accomplished by technology scaling and/or material/device innovations. Recent breakthroughs on the development of MTJ with perpendicular magnetization (also called perpendicular-MTJ) successfully reveal the practical feasibility of sub-100μA switching threshold, e.g., perpendicular-MTJ with switching threshold of 50μA at 30nm node has been recently demonstrated [9]. Because of the virtually unlimited cycling endurance of MTJ devices, the development of perpendicular-MTJ makes STT-RAM become the most viable candidate for large-capacity on-chip embedded memory (e.g., last-level cache) and DRAM-replacement.

3.2.2 Replacing DRAM in SSDs

When being used to replace DRAM in SSDs, STT-RAM stores a few different types of data, including address mapping table and buffered/cached user data. Upon a data access request from the host, which typically comes with the logical address associated

with the data, SSDs must internally translate the logical address into a physical address pointing to the flash memory physical page location. Therefore, SSDs must maintain an address mapping table to store the logical-to-physical address mapping information. In current practice, SSD controllers maintain the up-to-date mapping table in DRAM, and periodically copy the table to flash memory. Meanwhile, in order to support system rebuild after power failure, each flash memory physical page stores the logical address associated with the data being stored in it. In case of address mapping table lost due to power failure, SSD controllers have to scan the all the flash memory physical pages in order to rebuild the address mapping table. Hence, the rebuild time is proportional to the storage capacity, and for large-capacity (e.g., 512GB) SSDs the rebuild time can be be over tens of minutes. Apparently, the non-volatile nature of STT-RAM essentially eliminates the rebuild process after power failure, which is particularly desirable for high-end and mission-critical applications.

Besides storing address mapping table, STT-RAM can meanwhile serve as non-volatile buffer/cache to improve the speed performance of SSDs. In current practice, although SSDs always first buffer or cache incoming write data in DRAM, most mission-critical applications prevent SSDs from claiming write completion until data have been written into flash memory. This is to prevent data loss in case of power failure, because of the volatile nature of DRAM. With non-volatile STT-RAM based buffer/cache, SSDs can significantly reduce the write response time for mission-critical applications.

3.2.3 Proposed Design Strategy

3.2.3.1 Rationale and Key Ideas

Due to the random and transient nature of read disturbance and sensing error, we have to use ECC to ensure STT-RAM read reliability. As demonstrated above, as we continue to scale down the technology, the probabilities of read disturbance and sensing error may largely increase, which directly demands stronger error correction strength. As a result, ECC encoding/decoding engine should reside in the host (e.g., memory controller) instead of STT-RAM chips in order to consolidate the ECC implementation cost and keep existing memory chip interface (e.g., DDRx) intact. Moreover, with the full knowledge of the data being stored in memory, the host can optimize the memory error-tolerance design

in a data-dependent manner, i.e., optimize the memory error-tolerance by fully exploiting the runtime data characteristics. To evaluate the potential of this data-dependent error-tolerance design concept, this work considers the case of using STT-RAM in SSDs. As pointed out above, when being used in SSDs, STT-RAM stores both address mapping table and buffered/cached user data, which have distinctively different characteristics. Accordingly, we propose a data-dependent error-tolerance design strategy, as illustrated in Fig. 3.4. It contains three specific data-dependent error-tolerance design techniques, for which the rationale and key ideas are described as follows.

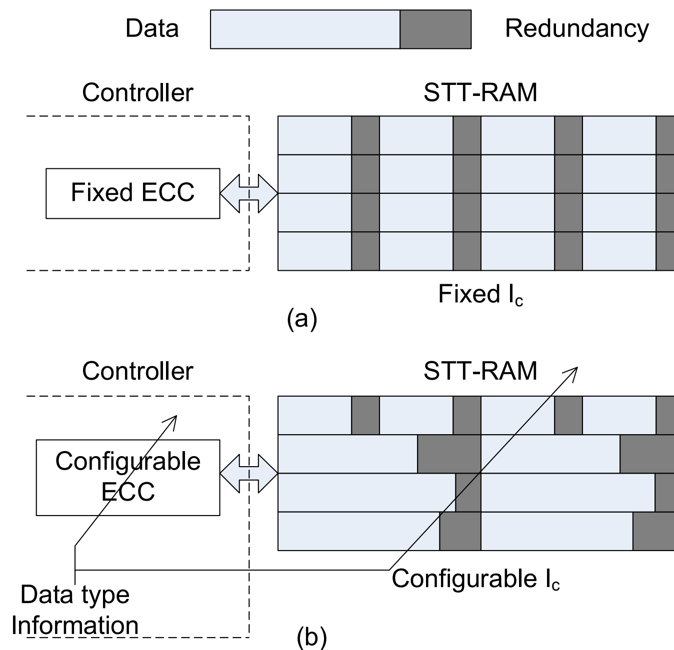


Figure 3.4: Illustration of (a) current design practice and (b) proposed data-dependent error-tolerance design strategy.

- *Data access unit adaptive ECC*: It is well-known that a longer ECC codeword length leads to a higher code rate (i.e., less coding redundancy), which nevertheless comes with a coarse-grained data access granularity. Hence, if the host memory controller is fully aware of the access unit length of current data, it can accordingly adjust the ECC codeword length in order to reduce the coding redundancy without affecting the data accessability. The size of each entry in STT-RAM buffer and cache equals to the size of each basic I/O, which typically ranges between 512B and 4kB in most computing systems. Therefore, to protect each buffer/cache entry, the ECC

codeword length can be as large as the size of each basic I/O. On the other hand, each entry in address mapping table is much smaller. For example, using flash memory with 8kB physical page size, a 512GB SSD contains at least 64M (i.e., 2^{26}) physical pages. Hence, we only need less than 4B to represent each entry in the address mapping table. Clearly, we can use different ECC codeword length (with different coding redundancy) for protecting these two different types of data in order to minimize the overall ECC coding redundancy.

- *Data read intensity sensing current configuration*: As discussed above, read disturbance results in data loss and its effect accumulates over consecutive read operations. Therefore, the errors due to read disturbance strongly depend on memory read intensity, in particular the number of read operations between two adjacent write operations on each memory cell. In comparison, sensing errors are caused by process/environmental variations and are completely independent from read intensity. Compared with other types of data, buffered write user data in STT-RAM may only be read once or few times before being evicted from STT-RAM and written to flash memory. The other types of data, including mapping table and cached data, may be read many times. Given the same sensing error probability, different read intensity will demand different sensing current in order to minimize the overall bit error probability and hence reduce the required coding redundancy.
- *Cleanness adaptive error protection*: The required ECC coding redundancy also depends on the target ECC decoding failure probability. For example, to protect 64B user data with binary BCH code under the raw bit error rate of 10^{-3}, if we want to achieve an extremely strong error tolerance by setting the target decoding failure probability as 10^{-15}, 17B of coding redundancy are required; on the other hand, if we relax the error tolerance by setting the decoding failure probability as 10^{-5}, only 7B of coding redundancy are required. Because of its backup copy in flash memory, cached clean data in STT-RAM can tolerate relatively less stringent memory storage reliability. In case of decoding failure of cached clean data, the controller simply treat it as a cache miss and fetch the data from flash memory. As long as the decoding failure probability is reasonably low (e.g., 10^{-5}), its impact on the cache miss rate and hence system performance will be negligible. Hence, in order to

further reduce the ECC coding redundancy, the SSD controller can appropriately reduce the error correction strength for read cache data in STT-RAM.

3.2.3.2 Practical Realization

To practically implement the above presented three data-dependent error-tolerance design techniques, we need to appropriately and jointly determine the configurations of ECC and STT-RAM sensing current I_c for different types of data. Notice that, in order to keep the memory chip interface intact, the STT-RAM sensing latency always remains the same. In this subsection, we derive the mathematical formulation based on which the ECC and STT-RAM sensing current can be jointly determined.

Let $f_s(I_c)$ and $f_r(I_c)$ denote the probabilities of memory sensing error and read disturbance, respectively, both of which are functions of sensing current I_c. If a memory cell has been read m times before being re-written, the probability of read disturbance is $1-\left(1-f_r(I_c)\right)^m$, and its aggregated error probability can be calculated as

$$\begin{aligned} f_t(I_c,m) \;\; = \;\; & f_s(I_c)\cdot\left(1-f_r(I_c)\right)^m + \\ & \left(1-f_s(I_c)\right)\cdot\left(1-\left(1-f_r(I_c)\right)^m\right), \end{aligned} \tag{3.8}$$

which is a function of both sensing current I_c and read intensity m. In this work, we only consider the scenario with two different values of m among all the data, i.e., m_h (e.g., 10^6) for data with high read intensity including address mapping table and cache, and m_l (e.g., 10) for buffered write data with low read intensity. Hence, according to (3.8), we can search for the sensing current to minimize the aggregated bit error probability under these two different read intensity. Let $p_e^{(m_h)}$ and $p_e^{(m_l)}$ denote the minimal bit error probability corresponding to read intensity of m_h and m_l, respectively, for which the sensing current is $I_c^{(m_h)}$ and $I_c^{(m_l)}$.

We assume the use of binary BCH codes for all the data, and each BCH code is denoted as $(n,\ k,\ t)$, i.e., each BCH codeword contains n bits, protects k user bits, and can correct up to t errors. Given the bit error probability p_e, target BCH code decoding failure probability P_{dec}, and k, we can calculate n and t based upon

$$\begin{cases} n &= k + \lceil \log_2 n \rceil \cdot t \\ P_{dec} &\geq \sum_{s=t+1}^{n} \binom{n}{s} \cdot p_e^s \cdot (1-p_e)^{n-s} \end{cases} \quad (3.9)$$

As discussed above, the proposed data-dependent error-tolerance treat the following types of data in STT-RAM differently: address mapping table, buffered write data, clean read cache data, and other cached and/or buffered data. Let k_a and k_d, where $k_a << k_d$, denote the number of bits being protected by each codeword for mapping table and all the cached/buffered data, respectively. In addition, let $P_{dec}^{(h)}$ and $P_{dec}^{(l)}$, where $P_{dec}^{(h)} >> P_{dec}^{(l)}$, denote the target BCH decoding failure probability for protecting clean read cache data and other data, respectively. We can determine the BCH code configuration for different types of data as follows:

1. To obtain a BCH code (n_a, k_a, t_a) for protecting address mapping table, solve (3.9) to determine n_a and t_a by setting $p_e = p_e^{(m_h)}$, $P_{dec} = P_{dec}^{(l)}$ and $k = k_a$;
2. To obtain a BCH code (n_{wd}, k_d, t_{wd}) for protecting buffered write data, solve (3.9) to determine n_{wd} and t_{wd} by setting $p_e = p_e^{(m_l)}$, $P_{dec} = P_{dec}^{(l)}$ and $k = k_d$;
3. To obtain a BCH code (n_{cd}, k_d, t_{cd}) for protecting clean read cache data, solve (3.9) to determine n_{cd} and t_{cd} by setting $p_e = p_e^{(m_h)}$, $P_{dec} = P_{dec}^{(h)}$ and $k = k_d$;
4. To obtain a BCH code (n_{od}, k_d, t_{od}) for protecting all the other cached and/or buffered data, solve (3.9) to determine n_{od} and t_{od} by setting $p_e = p_e^{(m_h)}$, $P_{dec} = P_{dec}^{(l)}$ and $k = k_d$;

3.2.4 Evaluations

We carried out extensive simulations and analysis to demonstrate the potential effectiveness of the proposed data-dependent error-tolerance design solutions.

3.2.4.1 STT-RAM Modeling

In this study, we set the MTJ TMR as 100%, and set the average resistance of MTJ in anti-parallel and parallel as 2000Ω and 1000Ω (45nm node) and 3000Ω and 1500Ω (22nm node). Regarding the MTJ resistance variation, we assumed the resistance follows Gaussian distribution and considered three different deviation-to-average ratios, including 4%, 6%, and 8%. We use (1.1) presented in Section 1.2 to evaluate the impact of sensing

current on STT-RAM read disturbance, where we set the parameter $\frac{E}{k_B T}$ as 52 and 50 for 45nm and 20nm nodes [48]. Because of the well-recognized advantage of current sensing over voltage sensing [49], we designed a current sensing circuit using the structure presented in [50], where the sense amplifier offset voltage is set as 20mV. Based upon this sensing circuit and above parameters, we carried out extensive Monte Carlo SPICE simulations to evaluate the impact of sensing current on STT-RAM sensing error rate.

3.2.4.2 Effective Storage Capacity

We first studied the improvement on the effective storage capacity when using the proposed data-dependent error-tolerance design strategy. We use the conventional design practice as the baseline (i.e., all the data are protected by the same ECC and the same sensing current is used throughout the entire STT-RAM). In particular, for the baseline scenario, we use a single BCH code that can ensure the highest read intensity (i.e., m_h) and lowest decoding failure rate (i.e., $P_{dec}^{(l)}$) for all the data being stored in STT-RAM. Since each mapping table entry is much shorter than the user data being stored in STT-RAM, the codeword length of this single BCH code is limited by the mapping table data. As discussed in Section 3.2.3.2, let k_a denote the number of bits being protected by each codeword for mapping table, we can obtain a $(n_a,\ k_a,\ t_a)$ BCH code, by setting $p_e = p_e^{(m_h)}$ and $P_{dec} = P_{dec}^{(l)}$, for the baseline scenario. To obtain all the four BCH codes when using the data-dependent error-tolerance, i.e., $(n_a,\ k_a,\ t_a)$ code for mapping table, $(n_{wd},\ k_d,\ t_{wd})$ code for buffered write data, $(n_{cd},\ k_d,\ t_{cd})$ code for clean read cache data, and $(n_{od},\ k_d,\ t_{od})$ code for all the other cached and/or buffered data, we set $P_{dec}^{(l)}$ as 10^{-15}, $P_{dec}^{(h)}$ as 10^{-5}, m_h as 10^6, m_l as 10, and the MTJ resistance deviation-to-average ratio σ as 6%.

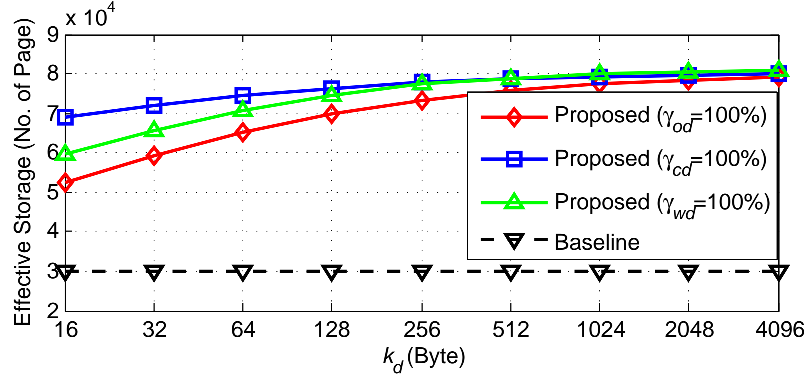
Assume the STT-RAM has a total storage capacity of N_{tot} bits and must store N_a bits of address mapping table data. Hence the STT-RAM uses $\lceil N_a/k_a \rceil \cdot n_a$ bits to store the address mapping table. This leaves $N_d = N_{tot} - \lceil N_a/k_a \rceil \cdot n_a$ bits available to store user data. Let L_p denote the size of each user data page (e.g., 512B or 4kB). For the baseline scenario using current design practice, the number of user pages that can be stored in STT-RAM is

$$s^{(base)} = \frac{\lfloor N_d/n_a \rfloor}{\lceil L_p/k_a \rceil}. \tag{3.10}$$

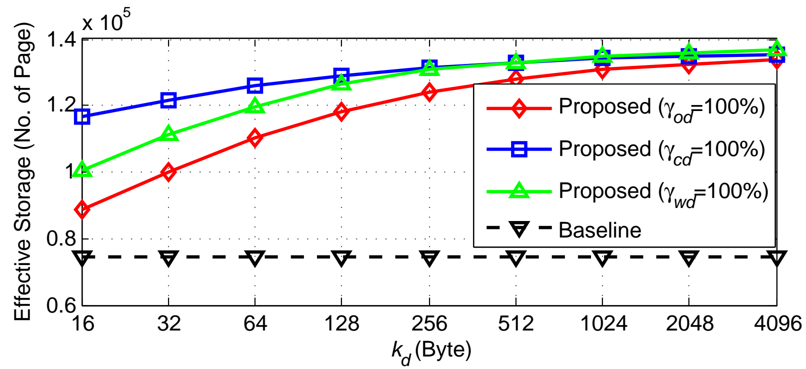
When using the proposed data-dependent error-tolerance, let γ_{wd}, γ_{cd}, and γ_{od} denote the percentage of the available storage capacity being allocated to buffered write data, clean read cache data, and other cached and/or buffered data (we have $\gamma_{wd} + \gamma_{cd} + \gamma_{od} = 100\%$), we can calculate the number of user pages for these three different types of data, denoted as $s^{(wd)}$, $s^{(cd)}$, and $s^{(od)}$, as

$$\frac{\lfloor N_d \cdot \gamma_{wd}/n_{wd} \rfloor}{\lceil L_p/k_{wd} \rceil}, \ \frac{\lfloor N_d \cdot \gamma_{wd}/n_{cd} \rfloor}{\lceil L_p/k_{cd} \rceil}, \ \frac{\lfloor N_d \cdot \gamma_{wd}/n_{od} \rfloor}{\lceil L_p/k_{od} \rceil}. \tag{3.11}$$

Hence, we can calculate the overall effective storage capacity as $s^{(prop)} = s^{(wd)} + s^{(cd)} + s^{(od)}$ and estimate the gain over the baseline scenario as $\left(s^{(prop)} - s^{(base)}\right)/s^{(base)}$. To quantitatively evaluate and compare the effective storage capacity, let us consider a 512G-B SSD with a N_{tot}=1GB STT-RAM. Suppose the flash memory has 8kB physical page size, the 512GB SSD contains at least 64M (i.e., 2^{26}) physical pages (i.e., we need less than 4B to represent each entry in the address mapping table). Let each mapping table entry takes 4B, the entire mapping table contains N_a=256MB of data and hence occupies a total $(256\lceil n_a/k_a \rceil)$MB of storage space after taking into account of the BCH coding redundancy. The rest storage space of $(1024 - 256\lceil n_a/k_a \rceil)$MB is available to store cached/buffered user data. Recall that k_d denotes the number of bits being protected by each codeword for all the cached/buffered data. The choice of k_d involves a trade-off between effective storage capacity and BCH code decoding energy consumption: As we increase the value of k_d (i.e., use a longer BCH codeword), we can reduce the coding redundancy and hence increase the effective storage capacity, while meanwhile the BCH code decoding will consume higher energy. Fig. 3.5 shows the results and comparison of the effective storage capacity in terms of the user data page number (the size of each user data page L_p is set as 4kB) under two different k_a values. In Fig. 3.5(a) and Fig. 3.5(b), we set that each $(n_a,\ k_a,\ t_a)$ code protects one and two mapping table entry (i.e., k_a is 4B and 8B), respectively. To clearly demonstrate the effect of different data types, as shown in Fig. 3.5, we considered three extreme scenarios when all the available storage space is used for storing the three different type of user data, i.e., $\{\gamma_{wd}, \gamma_{cd}, \gamma_{cd}\}$ is $\{100\%$, $0\%, 0\%\}$, $\{0\%, 100\%, 0\%\}$, $\{0\%, 0\%, 100\%\}$, respectively. The results clearly show that the proposed design strategy can largely improve the effective storage capacity.



(a)



(b)

Figure 3.5: Comparison of effective storage capacity in terms of user data page number when (a) k_a is 4B, and (b) k_a is 8B.

In the above evaluations, we set the MTJ resistance deviation-to-average ratio σ as 6% and the maximum read intensity parameter m_h as 10^6. Since these two parameters may largely vary in practice, particularly the MTJ resistance deviation-to-average ratio σ, we carried out further simulations to evaluate the sensitivity regarding these two parameters. Fig. 3.6(a) shows the effect of σ on the effective storage capacity, where we set $\gamma_{wd} = \gamma_{cd} = \gamma_{od} = 33.3\%$. In addition, Fig. 3.6(b) shows the impact of different values of m_h. The results show that the proposed design strategy can consistently outperform the current practice over a wide range of σ and m_h.

3.2.4.3 ECC Decoding Power Consumption

We further carried out studies to evaluate the ECC decoding energy consumption overhead of the proposed design strategy. We designed all the BCH code decoders using

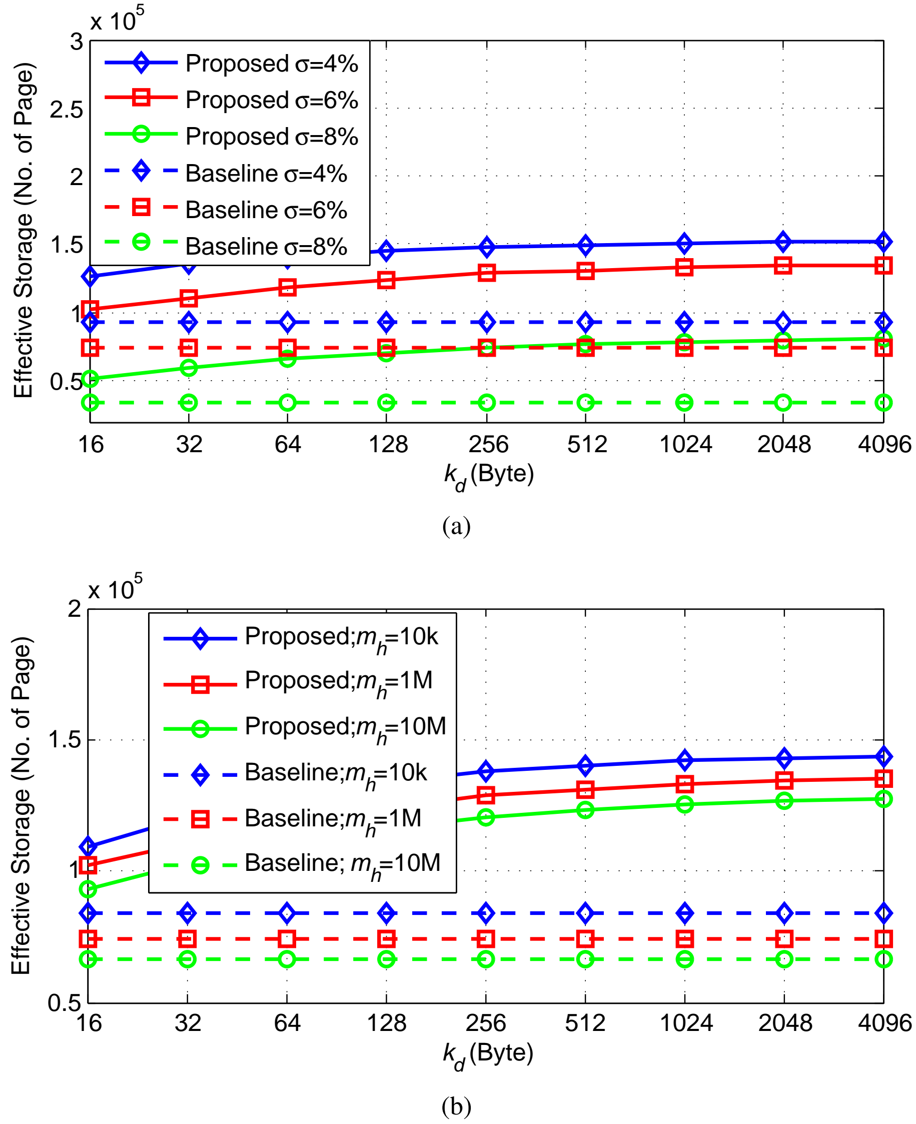


Figure 3.6: Effective user data storage capacity under different (a) MTJ resistance deviation-to-average ratio σ, and (b) different maximum read intensity m_h.

the simplified inverse-free Berlekamp-Massey (SiBM) decoding algorithm [51]. BCH code encoders and decoders are implemented using Synopsys tool set with TSMC 65nm cell library, where the power estimation is carried out using Prime Time PX. Fig. 3.7 shows the estimated BCH coding energy consumption per user bit for different types of pages when using the proposed design strategy, which are normalized to the baseline scenarios with k_a as 4B. It also shows the results for the baseline scenario with k_a as 8B. We note that the energy consumption of accessing the address mapping table is the same

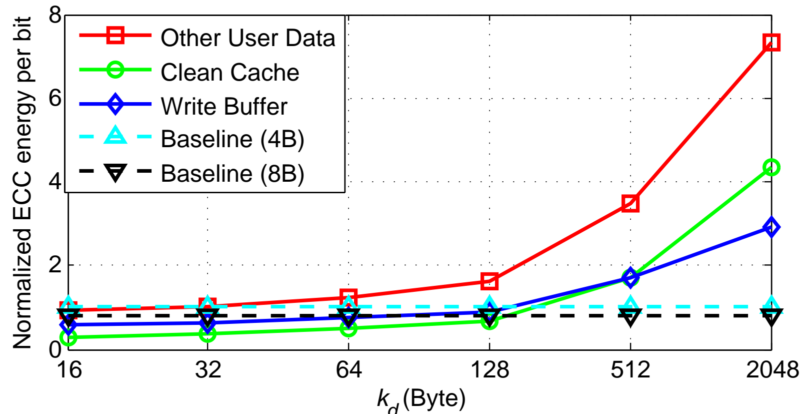


Figure 3.7: Estimated coding energy consumption per bit.

for both baseline and proposed design strategy, because they use the same BCH codes to protect the address mapping table. BCH code decoding energy consumption strongly depends on the value of t (i.e., maximum number of correctable errors per codeword) in a super-linear manner, and a longer codeword tends to have a larger value of t. When using the proposed design strategy, since buffered write data and cached clean data require less error correction strength than the other user data, their energy consumptions are consistently lower than that of other user data, as shown in Fig. 3.7. As we use a large k_d (i.e., over 128B) in the proposed design strategy, all the three type of data are subject to higher coding energy consumption compared with the baseline scenarios, as shown in Fig. 3.7. On the other hand, for smaller value of k_d (i.e., less than 128B), buffered write data and cached clean data may have less energy consumption than the baseline scenarios. The other user data, which are most heavily protected, consistently consume more energy than the baseline scenarios.

The results presented above clearly show a trade-off between effective data storage capacity gain and ECC coding energy consumption overhead. To further illustrate this point, Fig. 3.8 shows the normalized ratio between effective data storage capacity and coding energy consumption per bit, where a larger ratio is more desirable. In the baseline scenario, we set k_a as 8B. We also consider the case when $\gamma_{wd} = \gamma_{cd} = \gamma_{od} = 33.3\%$.

In this section, a data-dependent error-tolerance design strategy is presented which can reduce the error-tolerance redundancy overhead and hence increase effective storage capacity of STT-RAM without sacrificing its reliability. The underlying rationale is to

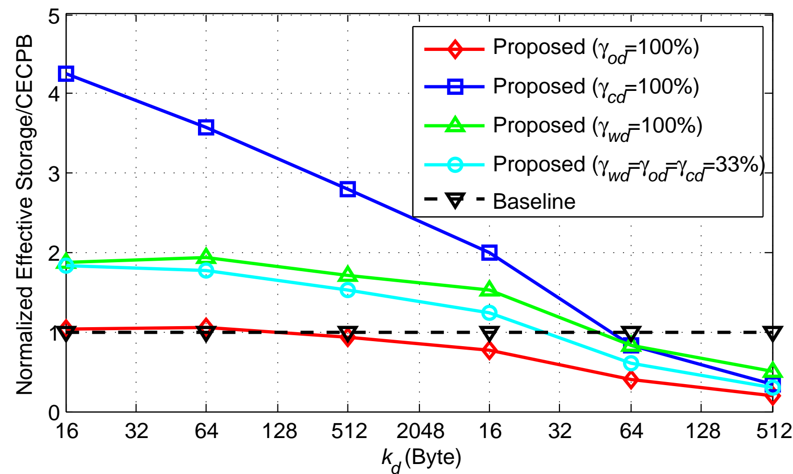


Figure 3.8: Normalized ratio between effective storage capacity and coding energy consumption.

cohesively exploit the run-time data characteristics and the read disturbance vs. sensing error trade-off in STT-RAM. I developed presents three specific data-dependent error-tolerance design techniques, and demonstrate their effectiveness in the context of using STT-RAM to replace DRAM in SSDs. I carried out detailed study on both effective storage capacity gain and energy consumption overhead, and the results show that these design solutions can increase the effective STT-RAM storage capacity by 26%, compared with conventional design practice.

4. Improving 3D DRAM Fault Tolerance through Weak Cell Aware Error Correction

4.1 Challenge in 3D memory

Stacking multiple DRAM dies and a logic die within the same package using through-silicon vias (TSVs), 3D DRAM has attracted significant attentions as a promising option to address the looming memory wall [52]. Compared with traditional dual in-line memory module (DIMM), 3D memory remarkably improves memory bandwidth and density at higher energy efficiency [53]. Two current 3D DRAM products are hybrid memory cube (HMC) by Micron [4, 54] and high bandwidth memory (HBM) by Hynix [55], both of which can achieve over 200GB/s data transfer bandwidth. Although 3D DRAM chips can greatly improve the overall computing system performance, their relatively high cost is one of the most critical issues preventing their wide real-life adoption. Continuous technology scaling is (almost) the only viable option to address this cost challenge. Unfortunately, sub-20nm DRAM is clearly subject to a significant increase of weak memory cells that cannot guarantee the data storage integrity under the worse-case operating condition, i.e., the longest data retention time (e.g., 64ms or 128ms) of the refresh period and the highest operating temperature (e.g., 85 °C) [18, 19].

This work aims to improve the weak cell tolerance in the context of 3D DRAM. Under very high weak cell rates (e.g., 10^{-5} and even above), a large percentage of columns/rows in each memory sub-array may contain at least one weak cell, which will make conventional redundancy-repair subject to prohibitive redundancy overhead and even inapplicable. In the context of 3D DRAM, the stacked logic die could integrate memory access re-mapping to achieve more flexible redundancy-repair [21, 22, 56–58]. The memory access re-mapping demands the implementation of SRAM-based address mapping tables. A high weak cell rate results in a large mapping table, which causes significant overhead in terms of both silicon cost and memory access latency. Therefore, redundancy-

Portions of this chapter previously appeared as: H. Wang, K. Zhao, M. Lv, X. Zhang, H. Sun and T. Zhang, "Improving 3D DRAM fault tolerance through weak cell aware error correction," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 820-833, Oct. 2016.

repair may be a feasible option only for relatively low weak cell rates (e.g., 10^{-7} and below) [26].

This work focuses on applying error correction code (ECC) to improve the weak cell tolerance [27]. In particular, we aim to fully exploit one well-known fact: An ECC with the minimum Hamming distance of d can correct up to $\lfloor \frac{d-1}{2} \rfloor$ *errors* or up to $d-1$ *erasures*, where an erasure is defined as an error with the known location. Hence, due to the detectability of weak cells, if the ECC module is aware of the location of all the weak cells, each ECC codeword can tolerate twice as many as errors assuming all the errors are caused by weak cells. 3D DRAM for the first time makes it practically feasible to implement weak-cell-location-aware ECC coding because: (a) ECC can be completely handled by the logic die inside 3D DRAM, which will be transparent to the external components such as host CPU; (b) With stacked logic die, 3D DRAM (e.g., HMC) can employ serial-link packet-based data transfer, which can naturally embrace the variable latency of more sophisticated ECC inside 3D DRAM package; (c) Being responsible for the design and implementation of both DRAM dies and stacked logic dies, DRAM manufacturers are able to identify the weak cells through comprehensive in-house testing and accordingly integrate such information into the ECC module on the stacked logic die.

Although the basic idea of weak-cell-location-aware ECC erasure decoding is very straightforward, its practical implementation is non-trivial because of the following issues: (i) *Storage of weak cell location information*: Although we can use low-cost one-time programmable non-volatile memory (OTP NVM) [59] and even emerging NVM (e.g., STT-RAM [7] and phase-change memory [60]) to store the weak cell information in the stacked logic die, high weak cell rates will make it prohibitively expensive to store the location of *all* the weak cells. (ii) *Latency overhead*: In the most straightforward manner, when reading one ECC codeword from DRAM, the ECC module should look-up the location of all the weak cells within that codeword before carrying out decoding. Under high weak cell rates, it could take prohibitively long latency to carry out such location information look-up. (iii) *Inaccuracy of weak cell identification*: Under highly scaled DRAM technology nodes, it could be very difficult to precisely identify all the weak cells, and new weak cells may still develop in the field even after thousands rounds of testing by the DRAM manufacturers [61]. (iv) *Soft-error tolerance*: In addition to

errors caused by weak cells, the ECC must be able to handle radiation-induced soft errors as well [62].

Aiming to address the above issues and hence effectively exploit ECC erasure decoding for improving weak cell tolerance, this paper presents a design solution with the following properties: (1) It only needs to explicitly store the location of less than 0.1% of all the weak cells, i.e., reducing the weak cell location information storage overhead by over 1000x. (2) It employs a progressive ECC decoding procedure that incurs very small average data read latency overhead, e.g., as shown later the average read latency overhead is less than 5% even under the weak cell rate of 10^{-4}. (3) This design framework can gracefully embrace the inaccuracy of weak cell detection, and cohesively handle errors caused by both weak cells and radiation. This design solution is based upon the following key observations: First, it is not necessary to carry out erasure decoding for all the codewords. Given an ECC code with the minimum Hamming distance of d and define $t = \lfloor \frac{d-1}{2} \rfloor$, we can simply apply t-error-correction ECC decoding without demanding any weak cell location information if the codeword contains no more than t weak cells. This clearly can obviate the storage of weak cell information for the majority of codewords. Second, we note that not all the weak cells always cause bit errors. Recall that a DRAM cell is marked as a weak cell if it fails to ensure storage integrity under the worst-case scenario, in particular the highest operating temperature. Hence, under normal operational environment, it is reasonable to expect that most weak cells may not cause storage errors for most of the time. As elaborated later in Section 4.3, the developed design solution aims to take advantage of the above two facts in order to largely reduce the storage and latency overhead. We present thorough mathematical formulation of this developed design solution, while fully taking into account of the inaccuracy of weak cell detection and the necessity of handling errors induced by both weak cells and radiation.

We further carried out extensive analysis and simulations to evaluate its effectiveness. We first present thorough mathematical formulation and analysis to study the effects of various parameters including ECC code length and code rate, weak cell rate, inaccuracy of weak cell detection, and desired soft error correction strength. We further studied the implementation cost and latency overhead in the context of HMC 3D DRAM. Using a cycle-accurate x86 simulator [63] and CPU 2006 suite [64], we finally carried out system-

level simulations and the results show that our developed solution only incur less than 2% performance degradation on average even in the presence of weak cell rate of 1×10^{-4}.

4.2 Background

4.2.1 3D DRAM Structure

Using Micron's HMC as an example, we briefly describe emerging 3D DRAM chip structure. An HMC 3D DRAM chips consists of one logic die and multiple DRAM dies connected using TSV technology. At the time of 2016, HMC has two capacity configurations, 4GB and 8GB. The structure of an 8GB HMC is shown in Fig. 4.1. It has eight DRAM dies and one logic die. Capacity of each DRAM die is 1GB and each DRAM die is divided into 32 partitions. One partition contains four DRAM banks so an 8GB HMC has total 512 DRAM banks [3]. All vertically adjacent DRAM partitions together form a *vault* so that there are 32 vaults in one HMC chip. Every vault is controlled by one individual vault controller in the logic die. Each vault controller has a reference queue for buffering references to its associated vault. This feature allows vault controller to independently reorder memory references for latency optimization rather than executing references exactly as order of arrival. Moreover, all the vault controllers can work independently from each other, leading to a high operating parallelism inside each HMC chip.

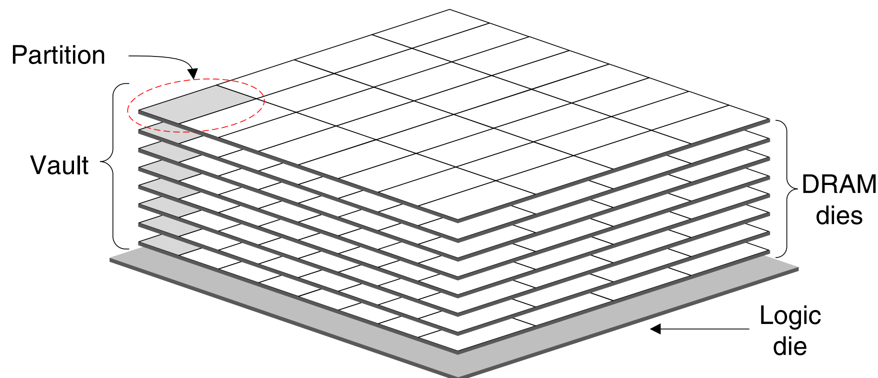


Figure 4.1: Illustration of HMC chip structure.

In sharp contrast to current DDR interface, HMC chips employ packet-based command/data interfaces over 16-lane full-duplex differential serial links, where the speed of each lane ranges from 12.5Gb/s to 30Gb/s. Current HMC product has either 2 or 4 sets

of 16-lane serial links, leading to the maximum aggregated bandwidth of 480GB/s. In addition to higher bandwidth, serial links also has better energy efficiency, e.g., energy per bit in current HMC products is 10.48pj/bit, representing 83% reduction compared with 65pj/bit in DDR-3 [4]. The packet-based command/data interface can naturally embrace variable data access latency inside HMC chip, and the size of each packet ranges from 16 to 128 bytes.

4.2.2 Related Work

Conventionally, weak cells are handled inside DRAM chips by row/column-based redundancy repair schemes which use redundant DRAM storage to replace these weak cells. Instead of self-repair for 2D DRAM chip, 3D DRAM can benefit from sharing the memory array or redundancies across multiple DRAM dies at the architecture level [21]. In addition to conventional row/column-based redundancy repair, some recent work [22, 26, 56, 57] suggest to use more flexible pointer/cache-based schemes to replace faulty memory cells using redundant memory cells. Intuitively, the stacked logic die inside 3D DRAM can be leveraged to improve redundancy repair efficiency [58]. However, most prior work demand significant change of memory circuit design and even involve operating systems in order to reduce the repair-induced redundancy overhead. In addition, they tend to assume the perfect knowledge of all the weak cells, which could be very difficult (if not impossible) in practice.

Error control code (ECC) is another widely used scheme for improving memory fault tolerance. Single-error-correction double-error-detection (SEC-DED) codes [65] are the *de facto* standard DRAM ECC. In current practice, each SEC-DED codeword protects every 64-bit user data at 8-bit redundancy (i.e., 12.5% ECC overhead). Nevertheless, SEC-DED is primarily used to handle radiation-induced soft errors. Advanced ECC-based schemes (e.g., see [23–25, 66]) have been proposed to achieve stronger memory error tolerance than conventional SEC-DED. Nevertheless, none of prior work cohesively exploits the weak cell detectability and ECC erasure decoding to minimize the coding redundancy.

4.3 Proposed Design Solution

4.3.1 Key Observations

We first describe the key observations underlying this design solution. It is well known that, given an ECC with the minimum Hamming distance of d, we can achieve different combinations of error/erausre-correction strength, i.e., suppose we use the ECC to correct up to t errors and meanwhile up to e erasures, we can flexibly adjust the values of t and e subject to

$$d-1 \geq 2t+e. \tag{4.1}$$

For our interested memory fault tolerance, suppose we aim to correct up to t_{ran} errors and e_{max} erasures, we should employ an ECC with the minimum Hamming distance of $d=2t_{ran}+e_{max}+1$. We note that the t_{ran}-error-correction strength is used to handle errors induced by radiation and undetected weak cells. According to Eq. (4.1), if we eliminate erasure correction (i.e., $e=0$), the ECC can correct up to $t_{max}=\lfloor\frac{d-1}{2}\rfloor$ errors. Let n denote the ECC codeword length, and we call each group of n memory cells being protected by one ECC codeword as a *codeword-cell-group*. For one codeword-cell-group that contains no more than $t_{max}-t_{ran}$ detected weak cells, it will never experience more than t_{max} bit errors (including up to $t_{max}-t_{ran}$ errors caused by detected weak cells and up to t_{ran} errors caused by radiation and undetected weak cells), for which we can simply configure the ECC to operate in the t_{max}-error-correction mode without erasure correction. Clearly, it is not necessary to store the weak cell location information for this codeword-cell-group. This can be leveraged to obviate the explicit storage of weak cell location information for most codewords. For example, suppose we set $t_{ran}=1$ and $e_{max}=2$ (hence we have $d=5$ and $t_{max}=2$) and the codeword length as 32 bytes. Under the weak cell rate of 10^{-5} and 10^{-4}, we have that 99.9999% and 99.99% of all the codeword-cell-groups contain no more than $t_{max}-t_{ran}=1$ weak cells and hence do not demand the explicit storage of weak cell location information.

For codeword-cell-groups that contain more than $t_{max}-t_{ran}$ detected weak cells, we have to incorporate erasure decoding to guarantee the storage integrity, for which we should explicitly store the weak cell location information. Although this applies to only a tiny portion of entire memory space (as demonstrated above), the absolute number of such codeword-cell-groups may not be small in large-capacity memory chips. As a result,

it may still take a long latency to look-up the location of weak cells for realizing erasure decoding. To address such a latency issue, this work presents a solution based upon the following key points:

- *Occurrence of bit errors*: Not all the weak cells always cause bit errors. Recall that a DRAM cell is marked as a weak cell if it fails to ensure the storage integrity under the worst-case scenario, in particular the highest operating temperature. Hence, under normal operational environment, it is reasonable to expect that most weak cells may not cause storage errors most of the time. Meanwhile, recall that the parameter t_{ran} is allocated for handling random errors induced by radiation and undetected weak cells, and it is reasonable to expect that the occurrence probability of such random errors is very low. Therefore, for a codeword-cell-group containing e detected weak cells, the number of bit errors experienced by this codeword-cell-group tends to be *much less* than $t_{ran} + e$ most of the time.

- *Opportunistic use of error correction/detection*: It is well known that, if we use an ECC with the minimum Hamming distance of d to correct up to t errors and meanwhile detect up to c errors (where $c > t$), we can flexibly adjust the values of t and c subject to $d - 1 \geq t + c$. If the number of weak cells in one codeword-cell-group is no more than $e_{max} - i$ weak cells (where $i \geq 0$), then this codeword-cell-group will never experience more than $e_{max} + t_{ran} - i$ errors. Hence, we can safely configure the ECC to operate in the $(t_{ran} + i)$-error-correction and $(e_{max} + t_{ran} - i)$-error-detection mode. It will succeed if the number of runtime bit errors is no more than $t_{ran} + i$, which clearly obviates the explicit execution of erasure decoding.

Leveraging the above key observations, we accordingly develop a design solution that can effectively reduce the storage and latency overhead, which will be formulated in the next sub-section.

4.3.2 Formulation of the Design Solution

Our design solution protects the entire memory space using the same ECC with the minimum Hamming distance of $d = 2t_{ran} + e_{max} + 1$, where t_{ran} and e_{max} are two critical design parameters. As discussed in the previous sub-section, the parameter t_{ran}

specifies the random error correction strength reserved for errors caused by radiation and undetected weak cells, and the parameter e_{max} specifies the tolerable maximum number of detected weak cells. Recall that $t_{max} = \lfloor \frac{d-1}{2} \rfloor$, and define $r = t_{max} - t_{ran}$. Let n_{weak} denote the number of detected weak memory cells in each codeword-cell-group, we partition all the codeword-cell-groups into $2+r$ sets Q_v, $Q_e^{(1)}$, $\cdots$, $Q_e^{(r)}$, and Q_c according to the value of n_{weak}:

- The set Q_v consists of all the codeword-cell-groups with $n_{weak} > e_{max}$. Since the ECC is constructed to realize t_{ran}-error-correction and e_{max}-erasure-correction, it cannot guarantee the data storage integrity of the data being stored in Q_v.

- For the r sets $Q_e^{(1)}$, $\cdots$, $Q_e^{(r)}$, each set $Q_e^{(i)}$ consists of all the codeword-cell-groups with $n_{weak} = e_{max} + 1 - i$, where $1 \leq i \leq r$. We should explicitly store the weak cell location information of these codeword-cell-groups since erasure decoding may be needed.

- The set Q_c consists of all the other codeword-cell-groups, i.e., those with $n_{weak} \leq t_{max} - t_{ran}$. We may consider that Q_c can be further partitioned into $r+1$ subset $Q_c^{(0)}$, $Q_c^{(1)}$, $\cdots$, $Q_c^{(r)}$, where each $Q_c^{(i)}$ consists of all the codeword-cell-groups with $n_{weak} = i$ for $0 \leq i \leq r$. As discussed above, we can safely configure the ECC to operate in the t_{max}-error-correction mode to handle all the codeword-cell-groups in the set Q_c, i.e., the erasure decoding can be completely avoided.

Leveraging the key observations discussed in the previous sub-section, we develop a design solution with the data write and read flow diagrams as shown in Fig. 4.2, which can be described as follows:

- *Data write*: To write data at the address A (i.e., the address of the codeword-cell-group to be written), we first check whether A is in the set $\in Q_v$. If $A \in Q_v$, we store the data into a small memory, referred to as *virtual repair memory*, embedded in the memory controller on the logic die instead of writing to the DRAM dies. If $A \notin Q_v$, we carry out the ECC encoding and write the codeword to the codeword-cell-group with the address of A. We note that the size of the set Q_v tends to be very small. For example, suppose we set $t_{ran} = 1$ and $e_{max} = 6$ and the codeword length as 32

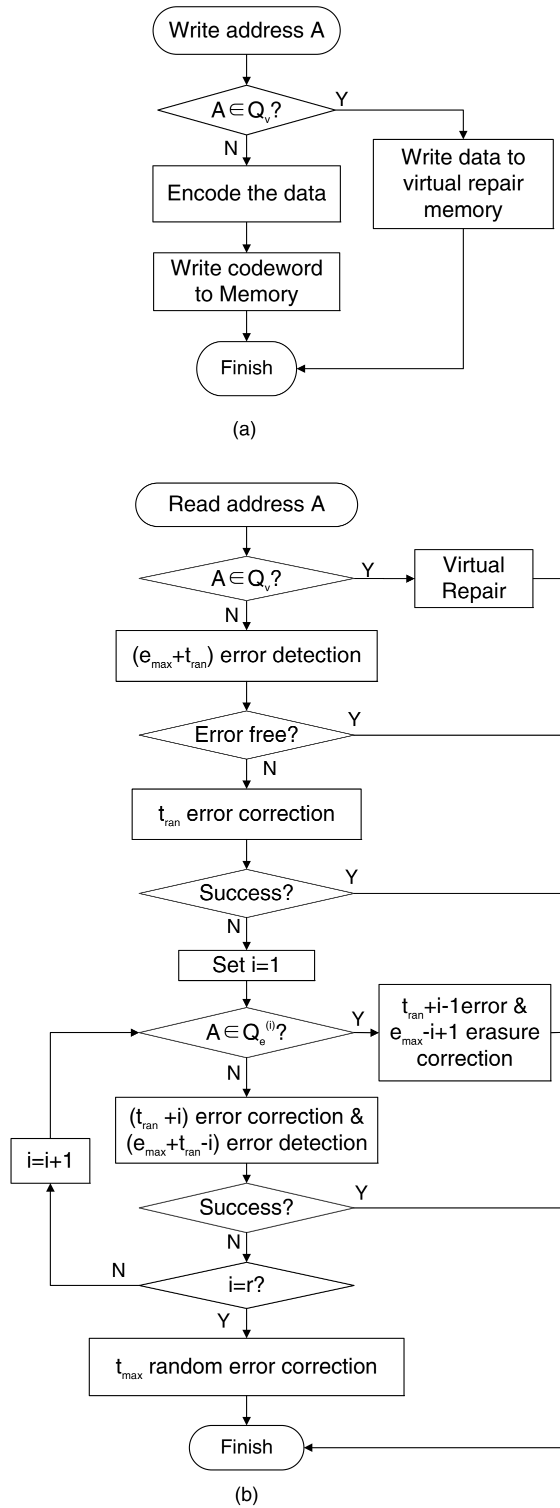


Figure 4.2: Flow diagram of the developed weak-cell-aware memory ECC design solution for (a) memory write, and (b) memory read.

bytes. Under the weak cell rate of 10^{-5} and 10^{-4}, the probability that the set Q_v contains 4 entries is 7.6×10^{-39} and 7×10^{-18} in 32GB DRAM.

- *Data read*: To read data from the address A, we first check whether A is in the set Q_v. If $A \in Q_v$, we directly read the data from the virtual repair memory embedded in the memory controller on the logic die. If $A \notin Q_v$, we read the entire codeword from the DRAM dies and carry out a recursive procedure as shown in Fig. 4.2(b) to correct all the errors at minimal latency overhead. We always start with $(e_{max} + t_{ran})$-error-detection, and t_{ran}-error-correction if errors are detected. If this step fails (i.e., the codeword suffers from more than t_{ran} errors), we check whether A is in the set $Q_e^{(1)}$. If $A \in Q_e^{(1)}$ (i.e., the codeword-cell-group contains e_{max} detected weak cells), we fetch the weak cell location information, and carry out the t_{ran}-error-correction and e_{max}-erasure-correction ECC decoding, which will guarantee to correct all the bit errors. If $A \notin Q_e^{(1)}$ (i.e., the codeword-cell-group contains no more than $e_{max} - 1$ detected weak cells and hence will never experience more than $e_{max} + t_{ran} - 1$ bit errors), we first carry out $(t_{ran} + 1)$-error-correction and $(e_{max} + t_{ran} - 1)$-error-detection ECC decoding without invoking erasure decoding. Since runtime number of errors tends to be very small as discussed above, such a direct error-correction/detection operation most likely will succeed. If the error detection fails (i.e., indeed more than $t_{ran} + 1$ errors occurred), we proceed to check against with the set $Q_e^{(2)}$ and repeat the same operations as shown in Fig. 4.2(b). If the procedure still continues after checking against the set $Q_e^{(r)}$, then we know that $A \in Q_c$ and t_{max} bit errors occurred. Therefore, we simply carry out the t_{max}-error-correction as the last step, as shown in Fig. 4.2(b).

In the context of data write, since we only need to check whether the address is in the smallest set Q_v, it is reasonable to expect that the latency overhead tends to be very small. In the context of data read, the recursive procedure presented above can effectively reduce the average latency overhead caused by weak cell information look-up. In particular, Let $p(Q)$ denote the probability that we have to explicitly check whether an address is in the set Q during data read, and $f(Q)$ denote the latency induced by checking whether an address is in the set Q. We can express the average latency overhead caused

by weak cell information look-up as

$$f(Q_v)+\sum_{i=1}^{r}\left(p(Q_e^{(i)})\cdot f(Q_e^{(i)})\right). \tag{4.2}$$

Since runtime number of bit errors tend to small, we have that $p(Q_v)=1>>p(Q_e^{(1)})>>\cdots>>p(Q_e^{(r)})$. Meanwhile, since $|Q_v|<<|Q_e^{(1)}|<<\cdots<<|Q_e^{(r)}|<<|Q_c|$, we have that $f(Q_v)<<f(Q_e^{(1)})<<\cdots<<f(Q_e^{(r)})$. Therefore, we can conclude that the average latency overhead is significantly less than directly checking all the weak cell information across the entire memory space.

4.3.3 Decoding Failure Probability Estimation

This sub-section presents further mathematical formulations that form a framework for estimating the overall failure probability when using our design solution. In order to accommodate the worst-case scenario, we assume that all the true weak cells always invoke errors throughout the following analysis. First, we note that it may not always be possible to accurately detect all the weak cells. To accommodate the inaccuracy of weak cell detection, we introduce two parameters: (1) the miss-detection probability that one true weak cell is not detected, denoted as α; (2) the false-detection probability that one normal cell is detected as a weak cell, denote as β. Let ρ_w represent the true weak cell rate and ρ_d denote the probability that one cell is detected as a weak cell, we have that

$$\rho_d=\rho_w\cdot(1-\alpha)+(1-\rho_w)\cdot\beta. \tag{4.3}$$

Let ρ_s denote the soft error rate. Given one cell that is not detected as a weak cell, we can express the probability that this cell experiences a bit error as

$$\rho_f=1-(1-\rho_s)(1-\alpha\cdot\rho_w). \tag{4.4}$$

As discussed above, we partition the entire memory space into $2+r$ sets Q_v, $Q_e^{(1)}$, $\cdots$, $Q_e^{(r)}$, and Q_c according to the number of detected weak cells in each codeword-cell-group. Due to the use of virtual memory repair, data being written to the set Q_v will not experience failures. Define the set $Q_e=\{Q_e^{(1)},\cdots,Q_e^{(r)}\}$, and let F_e and F_c denote the

probability that our design solution fails to correct one codeword-cell-group in the set Q_e and Q_c, respectively. In the following, we discuss the estimation of F_e and F_c.

Recall that each codeword-cell-group in the set $Q_e^{(i)}$ contains $e_{max}+1-i$ detected weak cells. Let n and k denote the codeword length and data bits respectively, we can express the probability that one codeword-cell-group belongs to the set $Q_e^{(i)}$ as

$$P_e^{(i)} = \binom{n}{j}(\rho_d)^j(1-\rho_d)^{n-j}, \text{ where } j = e_{max}+1-i. \tag{4.5}$$

For one codeword-cell-group within the set $Q_e^{(i)}$, according to the decoding procedure described above, data decoding failure occurs when more than $t_{ran}+i-1$ errors are caused by radiation and/or undetected weak cells. Let $F_e^{(i)}$ denote the probability of such decoding failures and define $j = e_{max}+1-i$. According to Eq. (4.4), we have

$$F_e^{(i)} = \sum_{h=t_{ran}+i}^{n-j} \binom{n-j}{h}(\rho_f)^h(1-\rho_f)^{n-j-h}. \tag{4.6}$$

Therefore, we can calculate F_e as

$$F_e = \sum_{i=1}^{r} P_e^{(i)} \cdot F_e^{(i)}. \tag{4.7}$$

Next, let's consider the calculation of F_c. To facilitate the discussion, we further partition the set Q_c into $r+1$ sets $Q_c^{(0)}, Q_c^{(1)}, \cdots, Q_c^{(r)}$, where each $Q_c^{(i)}$ consists of all the codeword-cell-groups with i detected weak cells. We can express the probability that one codeword-cell-group belongs to the set $Q_c^{(i)}$ as

$$P_c^{(i)} = \binom{n}{i}(\rho_d)^i(1-\rho_d)^{n-i}. \tag{4.8}$$

Suppose one codeword-cell-group in $Q_c^{(i)}$ contains $j \le i$ true weak cells, let $\Phi_c^{(i)}(j)$ denote the decoding failure probability for this codeword-cell-group, we have

$$\Phi_c^{(i)}(j) = \sum_{h=t_{max}+1-j}^{n-j} \binom{n-j}{h}(\rho_f)^h(1-\rho_f)^{n-j-h}. \tag{4.9}$$

Meanwhile, given one codeword-cell-group in $Q_c^{(i)}$, let γ denote the probability that one detected weak cell in this codeword-cell-group is actually a normal cell, we have

$$\gamma = \frac{(1-\rho_w)\cdot\beta}{\rho_w\cdot(1-\alpha)+(1-\rho_w)\cdot\beta}. \tag{4.10}$$

Accordingly, let $F_c^{(i)}$ denote the probability that one codeword-cell-group in $Q_c^{(i)}$ experiences decoding failure, we have

$$F_c^{(i)} = \sum_{j=0}^{i}\binom{i}{j}\gamma^{i-j}(1-\gamma)^j\Phi_c^{(i)}(j). \tag{4.11}$$

Therefore, we can express F_c as

$$F_c = \sum_{i=1}^{r} P_c^{(i)}\cdot F_c^{(i)}. \tag{4.12}$$

Based upon F_e and F_c, we can obtain the overall decoding failure probability as $F_f = F_e + F_c$.

4.3.4 Practical Implementation

Using Micron's HMC as the test vehicle, we discuss the practical implementation of the developed design solution in this sub-section and will present evaluation results in Section 4.4. We first briefly describe the basics of HMC. Stacking multiple DRAM dies and one logic die with TSVs, HMC chip has a structure as illustrated in Fig. 4.1. Each DRAM die is divided into a certain number of partitions (32 partitions in current product). All the vertically adjacent partitions form a *vault*, which is controlled by one vault controller on the logic die. All the vault controllers operate independently from each other, leading to a very high operational parallelism of HMC. In addition, the logic die contains a single central controller that is responsible for a variety of critical tasks, e.g., HMC packed-based chip I/Os, management of all the vault controllers, and realization of HMC-specific commands such as intra-HMC data copy.

As pointed out earlier, the logic die inside the 3D DRAM chip is fully responsible for implementing the developed design solution, where the major functions include: (1) *ECC coding*: We need to implement all the ECC encoding, error detection, and er-

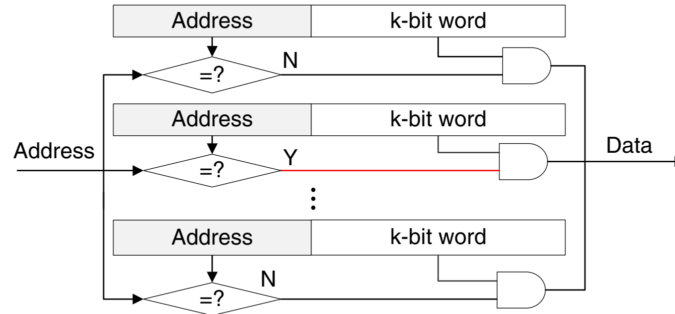


Figure 4.3: Illustration of the virtual repair module.

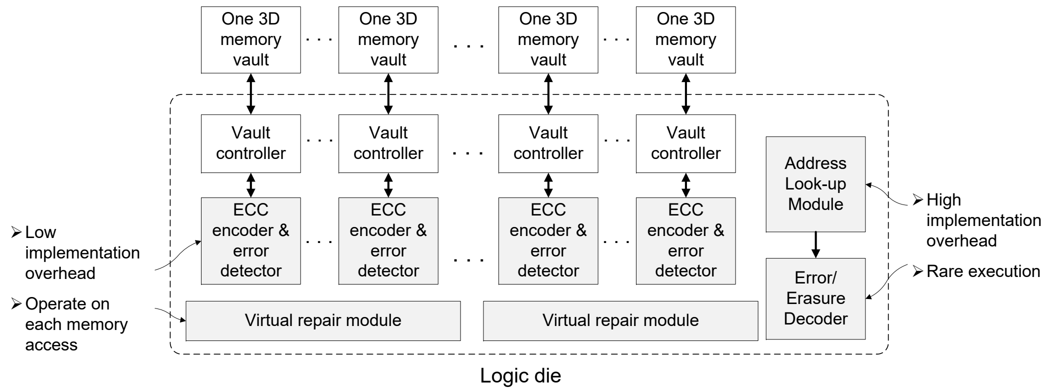


Figure 4.4: Illustration of parallel architecture for implementing the developed design solution to meet the very high throughput of HMC chip.

ror/erasure correction operations, as shown in the flow diagram in Fig. 4.2. (2) *Address look-up*: We need to check whether one codeword-cell-group address A is in the set $Q_e^{(1)}$, $\cdots$, or $Q_e^{(r)}$, and if $A \in Q_e^{(i)}$ we must further fetch the locations of the corresponding $e_{max} - i + 1$ detected weak cells. To facilitate the look-up operation, we can sort the addresses within each set in order to enable simple binary search. (3) *Virtual repair*: Aiming to virtually repair all the codeword-cell-groups belonging to the set Q_v, it functions as a fully associative cache, as illustrated in Fig. 4.3. Assume each codeword-cell-group stores k-bit user data. Each entry in the virtual repair module contains the address of one codeword-cell-group in set Q_v and the associated k-bit user data. Upon each memory access request, we always check its address against the fully associative cache through parallel comparison. In case of cache hit, the memory access request is directly served by the virtual repair module.

In order to meet the beyond-20GB/s throughput of HMC, we must implement this

design solution with sufficient operational parallelism. However, if we directly duplicate all the major functional engines (i.e., ECC coding, address look-up, and virtual repair), it could suffer from prohibitive silicon cost overhead, especially due to the cost of implementing error/erasure decoding and address look-up engines. Fortunately, as pointed out above, runtime memory error rate under normal operational environment tends to be much less than the weak cell rate. Hence, it is reasonable to expect that there is a relatively low probability to invoke error/erasure decoding and address look-up. Therefore, we propose a parallel architecture for implementing our design solution as illustrated in Fig. 4.4. Recall that each HMC chip partitions all the DRAM space into a large number of vaults (e.g., 32 in current practice). We group a certain number of vaults to form an ECC domain that contains a single ECC error/erasure decoding engine and a single address look-up engine. Each vault has its own ECC encoding and error detection engine. We note that the error detection engine here only carries out $(e_{max}+t_{ran})$-error-detection, and once it detects errors, the ECC error/erasure decoding engine will be invoked. Because virtual repair engine could operate at much higher speed than ECC encoding and error detection, a certain number of vaults could share one virtual repair engine as illustrated in Fig. 4.4.

Finally, as demonstrated in [61], new weak cells could even develop over the DRAM chip lifetime. If the in-field new weak cell development has a non-negligible probability, we must accordingly enhance the proposed design solution. For each codeword-cell-group, we know its maximum number of run-time errors (denoted as $s_{\max}$) based upon which set this codeword-cell-group belongs to. For example, for one codeword-cell-group belonging to $Q_e^{(i)}$ $(1 \leq i \leq r)$ or Q_c, it should at most experience $e_{\max}+1-i+t_{ran}$ or $t_{\max}$ errors as discussed above. As pointed out above, weak cells do not always induce errors, especially under normal operating temperature. Moreover, the occurrence of additional t_{ran} random errors tends to have very low probability. Hence, it is reasonable to expect that, for one codeword-cell-group, occurrence of $s_{\max}$ errors is indeed a very low-probability event. Nevertheless, if one or more new weak cells develop in one codeword-cell-group, the probability of occurrence of $s_{\max}$ errors could significantly increase. Therefore, we could enhance the above design solution to improve the tolerance to those in-field new weak cells as follows: When decoding the data being stored in this codeword-cell-group, if the ECC decoder indeed corrects $s_{\max}-v$ errors (where v is a

small constant, e.g., 0 or 1), we immediately promote this codeword-cell-group to the next-level set (e.g., from $Q_e^{(i)}$ to $Q_e^{(i-1)}$ or from Q_c to $Q_e^{(r)}$). Using such a conservative design enhancement, we can improve the runtime tolerance to in-field developed new weak cells.

4.4 Evaluations

To demonstrate the potential effectiveness of the developed design solution, this section presents a variety of evaluation results, including (1) fault tolerance strength analysis/simulations and the sensitivity to different parameters; (2) estimation of the hardware implementation overhead; (3) memory read latency analysis and cycle-accurate system performance simulation results under various computing benchmarks. We note that, throughout the study, we use binary BCH codes as the ECC.

4.4.1 Fault Tolerance Strength Analysis

In addition to the ECC configuration (i.e., codeword length, code rate, and t_{ran}), the fault tolerance strength further depends on the weak cell detection inaccuracy (i.e., the values of the miss-detection probability α and false-detection probability β). In the following, we present quantitative results that show the effect of ECC codeword length, code rate, t_{ran}, and α and β. Meanwhile, we present the comparison with the *baseline* scenario that applies ECC error correction without using weak-cell-location-aware erasure decoding. To avoid unfair comparison against baseline scenarios, we set that each baseline scenario also has a virtual repair module that is 10x larger than the one being used in our proposed design solution. In our study, we fix the probability of radiation-induced soft errors as 10^{-12}, which is essentially a very conservative assumption according to the statistics presented in [62].

- *ECC codeword length*: Under a fixed code rate, a longer ECC codeword length leads to a stronger error correction capability with a larger t_{max}. Fig. 4.5 shows the decoding failure rate vs. the weak cell rate ρ_w under different codeword length and code rate, where we conservatively set the miss-detection probability α as 1%. In addition, since the value of false-detection probability β heavily depends on the true weak cell rate ρ_w, we set $\beta = \rho_w$. Let (n, k, d) denote the ECC codeword length,

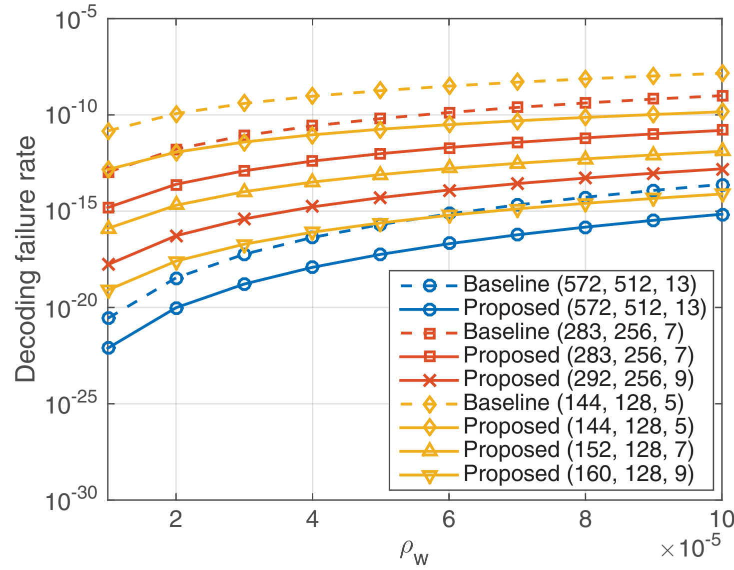


Figure 4.5: Codeword failure rate vs. weak cell rate ρ_w under different ECC codeword length and code rate with $\alpha = 1\%$, $\beta = \rho_w$ and $\rho_s = 10^{-12}$. Decoding failure rate is the ratio of codewords which cannot be successfully decoded over the total number of codewords.

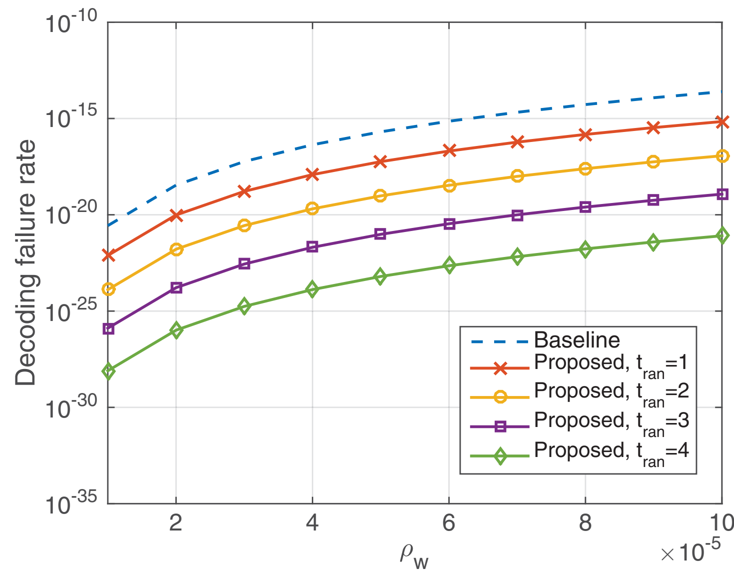


Figure 4.6: Codeword failure rate vs. weak cell rate ρ_w under different t_{ran} with $\alpha = 1\%$, $\beta = \rho_w$, $\rho_s = 10^{-12}$ and the use of BCH (572, 512, 13) code.

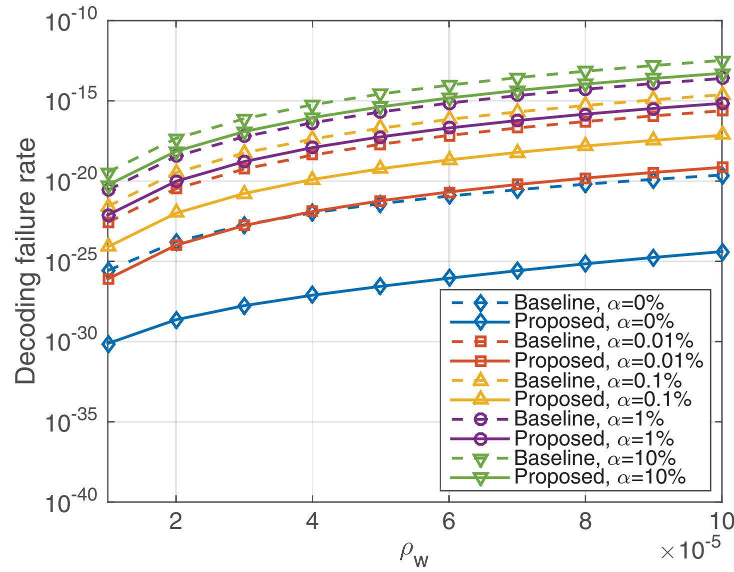


Figure 4.7: Codeword failure rate vs. weak cell rate ρ_w under different α with $\beta = \rho_w$, $\rho_s = 10^{-12}$ and the use of BCH(572, 512, 13) code.

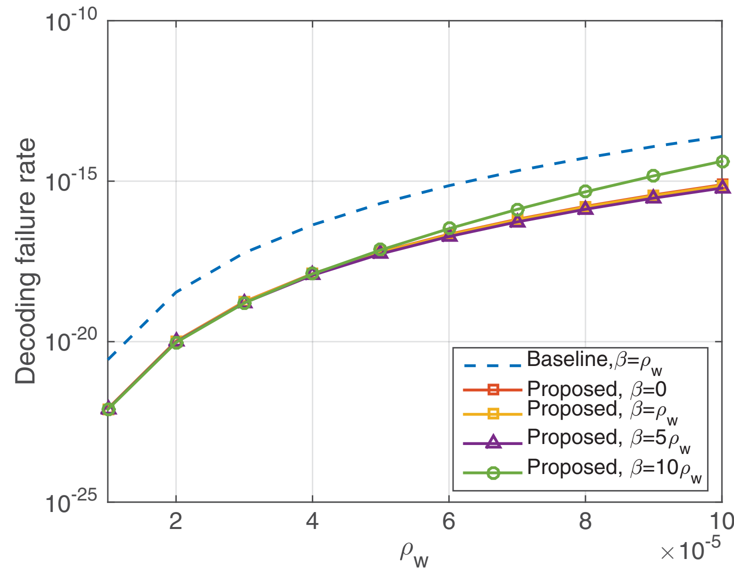


Figure 4.8: Codeword failure rate vs. weak cell rate ρ_w under different β with $\alpha = 1\%$, $\rho_s = 10^{-12}$and the use of BCH(572, 512, 13) code. Decoding failure rate is the ratio of codewords which cannot be successfully decoded over the total number of codewords.

user data length, and minimum Hamming distance. The results clearly show that, compared with the baseline scenario, the developed design solution can reduce the decoding failure probability by at least 3 orders of magnitude, even under the miss-detection probability of 1%. We note that the ECC configurations of (572, 512, 13), (283, 256, 7) and (144, 128, 5) all have the same code rate of 8/9 as commercial ECC DIMM. The benefit of using a longer codeword length is also very obvious, e.g., compared with the codeword of 256 bits, the codeword length of 512 bits could reduce the failure probability by more than 5 orders of magnitude.

- *ECC code rate*: We can improve the fault tolerance by reducing the code rate at the cost of larger redundancy storage overhead. Fig. 4.5 further shows the results when using three different code rates under the same k of 128. For example, compared with the rate-8/9 (144, 128, 5) code, the rate-8/10 (160, 128, 9) code can reduce the failure probability by 5 orders of magnitude.

- *Random error correction strength t_{ran}*: In addition to the ECC codeword length and code rate, the parameter t_{ran} also plays a critical role. Once we determine the value of t_{ran}, the other parameter e_{max} equals $d-1-2t_{ran}$. Fig. 4.6 shows the impact of t_{ran} on the fault tolerance strength where we use (572, 512, 13) BCH code with $t_{max}=6$ and set $\alpha=1\%$. The results clearly show that a large value of t_{ran} directly results in a worse fault tolerance strength.

- *Miss-detection probability α*: The parameter t_{ran} in ECC construction aims to handle random errors caused by radiation and undetected weak cells. Fig. 4.7 shows the impact of α on the decoding failure probability, where we use (572, 512, 13) BCH code with $t_{max}=6$ and set $t_{ran}=1$ and $e_{max}=10$. The results clearly show that the fault tolerance strength significantly degrades as the miss-detection probability α increases. When α increases to 10%, the strength degrades to almost the same as the baseline scenario. Hence, it is very critical for the DRAM manufacturers to minimize the miss-detection probability α.

- *False-detection probability β*: Since β heavily depends on ρ_w, we assume a linear dependence between them and study the scenarios of β being ρ_w, $5\rho_w$, and $10\rho_w$, respectively. Fig. 4.8 shows the impact of β on the fault tolerance strength,

where we use (572, 512, 13) BCH code with $t_{max} = 6$ and set $\alpha = 1\%$, $t_{ran} = 1$ and $e_{max} = 10$. The results show that, in sharp contrast to α, the fault tolerance strength is very weakly affected by the false-detection probability β. Therefore, DRAM manufacturers could be more aggressive on the weak cell detection by trading the accuracy of β for reducing α in order to enhance the achievable fault tolerance strength.

4.4.2 Hardware Overhead Estimation

In this sub-section, we use 8GB 4-link (each link contains 16 full-duplex lanes) HMC as a test vehicle to demonstrate hardware implementation overhead of our developed design solution. The parallel hardware architecture is discussed in Section 4.3.4 and presented in Fig. 4.4. First, we note that the hardware implementation overhead depends on three parameters, including detected weak cell rate ρ_d, ECC code rate and codeword length.

- *Detected weak cell rate ρ_d*: As shown in Eq. (4.3), ρ_d depends on the true weak cell rate ρ_w, miss-detection probability α, and false-detection probability β. Clearly, a larger value of ρ_d results in an increase of the hardware implementation overhead, in particular the hardware overhead of the address look-up and virtual repair. For example, assume we use the (160, 128, 9) BCH code with $t_{max} = 4$ and set $t_{ran} = 1$ and $e_{max} = 6$, we can calculate $P_c^{(i)}$ $(0 \leq i \leq 3)$ or $P_e^{(i)}(4 \leq i \leq 7)$ for different i as listed in the Table 4.1. As ρ_d increases from 10^{-5} to 10^{-4}, $P_c^{(i)}$ (or $P_e^{(i)}$) $(i \neq 0)$ increases by at least one order of magnitude, leading to much more virtual repair entries and address look-up entries.

- *ECC code rate*: A lower code rate directly corresponds to a larger t_{max}. According to Section 4.3.2, the size of sets Q_e and Q_v shrink dramatically as t_{max} increases, leading to less implementation overhead of the address look-up and virtual repair. However, this comes with the penalty of higher DRAM storage overhead.

- *ECC codeword length*: Given the same code rate, a longer codeword length leads to larger t_{max} and e_{max}, leading to less implementation overhead of the address

Table 4.1: Error distribution of one codeword-cell-group when using (160, 128, 9) BCH code.

ρ_d	10^{-5}	10^{-4}
$P_c^{(0)}$	99.84×10^{-2}	98.41×10^{-2}
$P_c^{(1)}$	1.6×10^{-3}	1.57×10^{-2}
$P_c^{(2)}$	1.27×10^{-6}	1.25×10^{-4}
$P_c^{(3)}$	6.69×10^{-10}	6.59×10^{-7}
$P_e^{(4)}$	2.62×10^{-13}	2.59×10^{-9}
$P_e^{(5)}$	8.19×10^{-17}	8.08×10^{-12}
$P_e^{(6)}$	2.12×10^{-20}	2.09×10^{-14}
$1 - \sum_{i=0}^{3} P_c^{(i)} - \sum_{i=1}^{3} P_e^{(i)}$	4.65×10^{-24}	4.59×10^{-17}

look-up and virtual repair. Nevertheless, a longer codeword length increases the implementation complexity of the error/erasure correction.

Throughout the rest of this sub-section, we will present the quantitative estimation of the implementation cost. We set the storage capacity of HMC chip as 8GB, and consider the use of five different BCH codes, including rate-0.80 (160, 128, 9), rate-0.83 (152, 128, 7), rate-0.89 (144, 128, 5), rate-0.88 (292, 256, 9), rate-0.90 (283, 256, 7), and rate-0.89 (574, 512, 13). We set $t_{ran} = 1$ for all the cases.

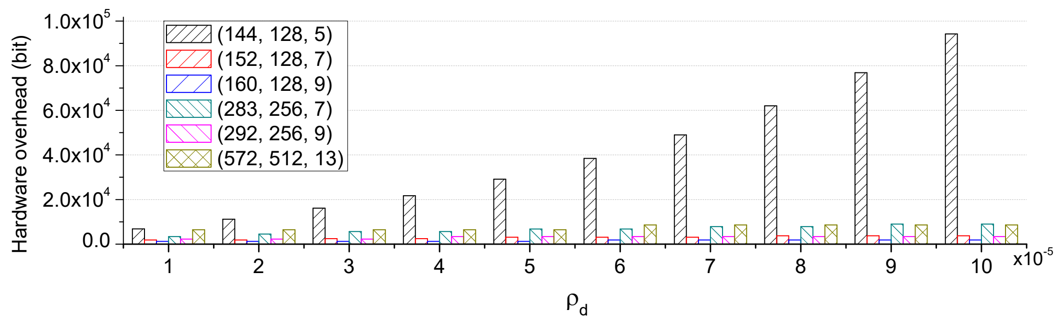
4.4.2.1 Virtual Repair Module

Let m denote the number of vault groups, where all the vaults in each vault group share one virtual repair module. Let $N_v^{(i)}$ $(1 \leq i \leq m)$ represent the number of entries in each virtual repair module and $Q_v^{(i)}$ denote the set consisting all the codeword-cell-groups that contain more than e_{max} detected weak cells in the i-th vault group. Let $L_v^{(i)}$ denote the probability that $|Q_v^{(i)}| > N_v^{(i)}$ (i.e., the i-th virtual repair module does not has enough entries), we have

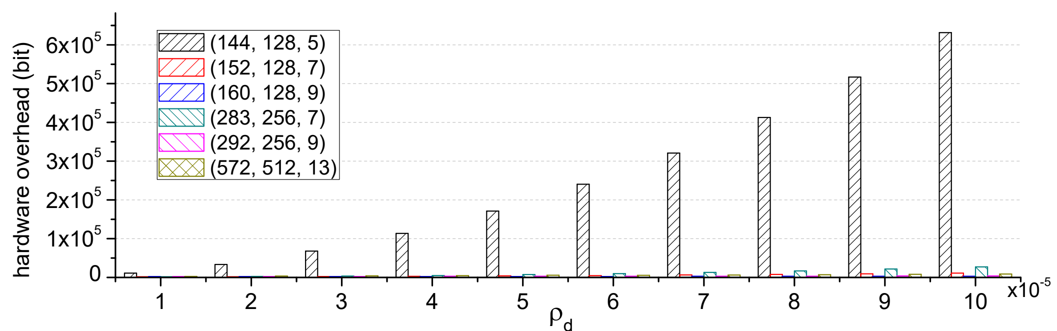
$$L_v^{(i)} = \sum_{j=N_v^{(i)}+1}^{C/m} \binom{C/m}{j} \left(P_v^{(i)}\right)^j \left(1 - P_v^{(i)}\right)^{(C/m-j)}, \tag{4.13}$$

where C is the total number of codeword-cell-groups within each vault group and $P_v^{(i)}$ is the probability that one codeword-cell-groups belongs to $Q_v^{(i)}$.

We must use a large enough value of $N_v^{(i)}$ (i.e., the size of the virtual repair module) to ensure sufficiently low $L_v^{(i)}$. In this study, we set the target $L_v^{(i)}$ as 10^{-20} and set $m = 4$. Recall that the total storage capacity of HMC chip is 8GB. Therefore, for each BCH code, we can search for the minimum value of $N_v^{(i)}$ subject to the constraint of $L_v^{(i)} < 10^{-20}$. Given the 8GB storage space of the HMC chip, Fig. 4.9(a) shows the total number of bits in all the four virtual repair modules under different detected weak cell rate ρ_d. Except the case of (144, 128, 5) BCH code, all the other five cases have similar storage cost of around 1×10^4 bits. The significantly higher cost in the case of (144, 128, 5) BCH code is due to its very weak error correction strength compared with the others. Even in the case of (144, 128, 5) BCH code, the total storage overhead is only 1×10^5 bits.



(a)



(b)

Figure 4.9: Hardware implementation overhead of (a) virtual repair, and (b) address look-up.

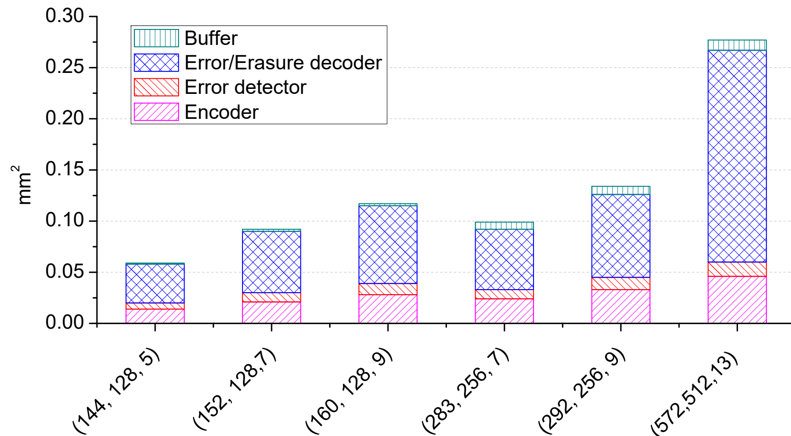


Figure 4.10: Estimated total silicon area cost of all the BCH coding modules (45nm node) in one HMC chip.

4.4.2.2 Address Look-up Module

The address look-up module must record the address of all the codeword-cell-groups belonging to $Q_e^{(i)}$ $(1 \le i \le r)$ and the location of all the detected weak cells in each codeword-cell-group. Suppose the address look-up module can record up to $M_e^{(i)}$ codeword-cell-groups for the set $Q_e^{(i)}$, the overflow probability (i.e., the probability of $|Q_e^{(i)}| > M_e^{(i)}$) can be calculated as

$$L_e^{(i)} = \sum_{j=M_e^{(i)}+1}^{C} \binom{C}{j} \left(P_e^{(i)}\right)^j \left(1-P_e^{(i)}\right)^{(C-j)}. \tag{4.14}$$

By setting $L_e^{(i)} < 10^{-20}$, we search minimum value of each $M_e^{(i)}$ under different configurations (i.e., the ECC code parameters and detected weak cell rate ρ_d), based upon which we can calculate the total address look-up storage overhead for an 8GB HMC chip. Fig. 4.9(b) shows the results. Similar to the results shown in Fig. 4.9(a) for virtual repair, except the (144, 128, 5) BCH code, the cost of address look-up is almost the same for all the codes about 5×10^4 bits. Even in the case of (144, 128, 5) BCH code, the total storage overhead is only about 6×10^5 bits.

4.4.2.3 BCH Coding Engines

To estimate the overall silicon cost of all the BCH coding modules, we carried out ASIC design with Verilog HDL using Synopsys synthesis tool. BCH encoder uses polynomial division to calculate the coding redundancy, for which we use a highly

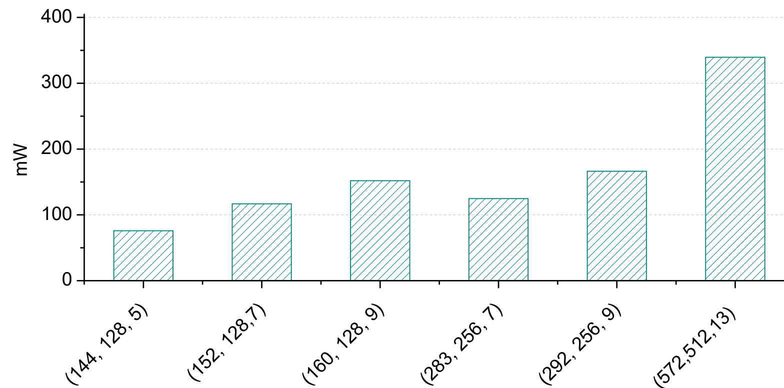


Figure 4.11: Estimated total power cost of all the BCH coding modules (45nm node) in one HMC chip.

parallel structure presented in [67]. Error detectors only need to detect error instead of correcting errors, so they only compute syndrome of each codeword and simply check whether syndromes are all zeros. The BCH decoding employs the Berlekamp-Massey algorithm and uses the parallel architecture presented in [68, 69].

Since a single error/erasure decoder is shared among multiple vault groups as shown in Fig. 4.4, the error/erasure decoder must contains a input buffer. Suppose the buffer can hold up to K codewords from all the vault groups, and let P_b denote the blocking probability that the buffer is full when a new codeword arrives. We simplified this problem as a queue problem. More specifically, we use the $M/M/1/K$ model to estimate the buffer size that can ensure a sufficiently low P_b. Maximum data bandwidth of HMC is 320GB/s [3] and we assume that link lane configuration is symmetric for both input and output. Thus, the maximum output data bandwidth is 160GB/s. Suppose $\rho_d = 10^{-4}$ and we use (160, 128, 9) BCH code, we can estimate that the average throughput of data demanding error correction is about $1.59 \times 10^{-2} \times 160 = 2.5$GB/s. Based upon our ASIC design results, the error/erasure decoder can achieve a throughput of 6.2GB/s. Thus in the $M/M/1/K$ queue model, the arrival intensity $\lambda = 2.5$ and service time $1/\mu = 1/6.2$. Therefore, according to [70], we have

$$P_b = \frac{(1-\delta)\delta^K}{1-\delta^{(K+1)}}, \tag{4.15}$$

where $\delta = \lambda/\mu$. Accordingly, we have that the blocking probability P_b is 5.6×10^{-4} and

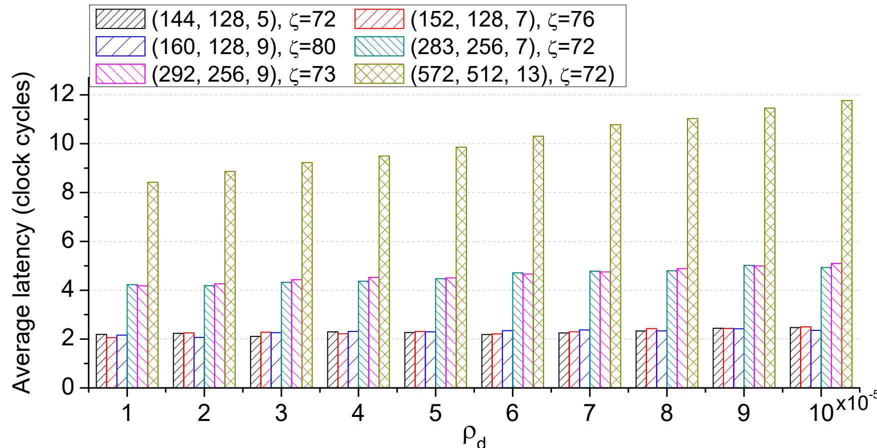


Figure 4.12: Estimated average data read latency overhead in terms of the number of clock cycles.

1.7×10^{-8} when the buffer can hold up to 6 and 10 codewords, respectively.

We further estimate the total silicon area occupied by all the coding modules under the following configuration: One HMC chip contains 32 vaults (hence 32 encoders and 32 error detectors), and contains a single error/erasure decoder. Based upon our ASIC design results, Fig. 4.10 shows the total silicon area at 45nm node when using different BCH codes. The results clearly show that the error/erasure decoder largely dominates the overall silicon area, which further justifies the design choice of sharing a single error/erasure decoder among all the vaults in each HMC chip. Fig. 4.11 shows the estimated power consumption when using different BCH codes. Even the most energy-hungry case of using (572, 512, 13) BCH code only consume less than 0.5W of power consumption.

4.4.3 Read Latency Overhead

Based upon the data read flow diagram shown in Fig. 4.2(b), we carried out simulations to estimate the read latency overhead when using the proposed design solution. Since the data read latency overhead strongly depends on the number of bit errors that actually occurred within one codeword-cell-group, we consider the worse-case scenario, i.e., all the weak cells always cause bit errors. Let n denote the codeword length and ζ denote the implementation parallelism factor (i.e., ζ bits are processed every clock cycle). In the context of BCH coding, error detection is realized by calculating all the syndromes, which takes about n/ζ clock cycles. In contrast, error/erasure correction in-

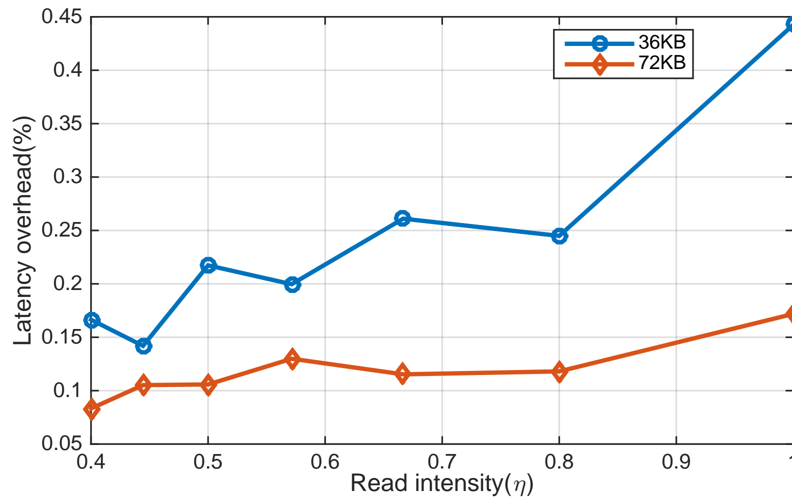


Figure 4.13: Latency overhead measured from FPGA-based prototyping.

volves four pipeline stages, including syndrome computation, key-equation solver, Chien search, and verification (i.e., error detection), where each pipeline state takes about n/ζ clock cycles [51, 71]. Hence, each round of BCH error/erasure decoding takes total $4n/\zeta$ clock cycles. Meanwhile, recall that the address look-up module can record up to $M_e^{(i)}$ codeword-cell-groups for the set $Q_e^{(i)}$, hence it takes $\lceil \log_2 M_e^{(i)} \rceil$ clock cycles to check the set $Q_e^{(i)}$ assuming the use of simple binary search.

I carried out simulations to measure the average data read latency under different BCH code configurations, implementation parallelism factor ζ, and different detected weak cell rate ρ_d. Fig. 4.12 shows the measurement results. The average latencies of proposed design are less than 10 clock cycles for (160, 128, 9), (152, 128, 7), (144, 128, 5), (292, 256, 9) and (283, 256, 7) BCH codes. Even with (572, 512, 13) BCH code, the average latency is only 12 clock cycles when ρ_d is 1×10^{-4}.

Fig. 4.14 shows the impact of the data read intensity η on read latency overhead, where the design parameters α and β are fixed as 1 and 0.1, respectively. The results clearly show a strong dependency of read latency overhead on the read intensity, i.e., as the read intensity slightly reduces from 100% to 90%, the read latency overhead significantly reduces, especially under relatively low runtime memory read BER.

I also implemented a DDR3 DRAM controller with the developed design solution on a PCIe FPGA development board hosting an Altera Arria V GX device and 9126MB

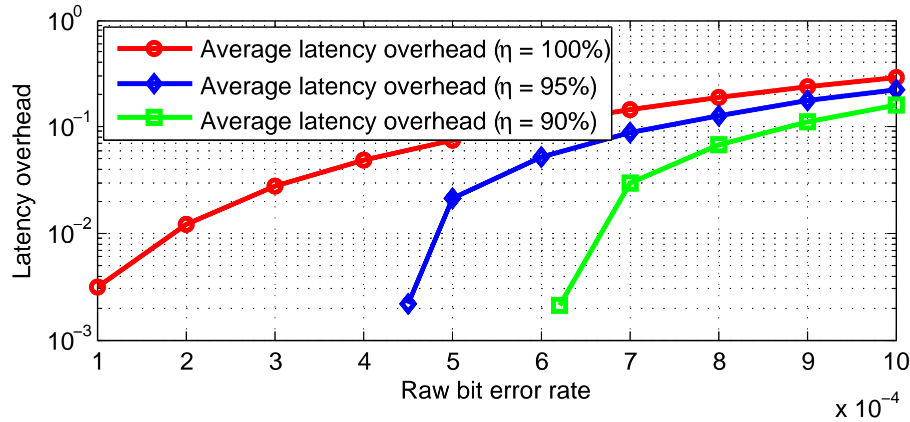


Figure 4.14: Latency overhead under different read intensity η.

DRAM with 72-bit data bus width. The design uses (283, 256, 7) BCH code with $t_{ran} = 1$. To implement our developed design solution, the entire coding module consumes 9k ALUTs, 3k registers and 40kB block memory. Since the DRAM chips are error free, I randomly inject bit errors with the BER of 10^{-4} when writing codewords into DRAM. The decoding module can correct all the errors. Using the Powerplay power analyzer in Quartus 13.0, total power consumption is estimated as 212 mW. We tested successive read request of 36kB (32kB user data plus 4kB coding redundancy) and 72kB (64kB user data plus 8kB coding redundancy) under different read intensity.

Fig. 4.13 shows the measured read latency overhead. The results further demonstrate that our proposed design solution can effectively minimize the read latency overhead.

4.4.4 System performance Overhead

To evaluate the impact of memory read latency overhead on the overall computing system speed performance, we further carried out full-system simulations using a cycle-accurate full-system simulator Marssx86 [63] and DRAM simulator DRAMsim2 [72]. The configuration of Marssx86 has a 4-core, out-of-order CPU operating at 2.4GHz with 128kB L1 and 8MB shared L2 caches. Fig. 4.15 shows the simulated execution time of a variety of SPEC CPU2006 benchmarks under the detected weak cell rate of 10^{-4}. For each benchmark, the execution time is normalized against the baseline scenario where the DRAM is completely error-free and hence ECC is not used at all. As shown in Fig. 4.15,

we used two different BCH codes: (283, 256, 7) and (572, 512, 13) codes. The results clearly suggest that the proposed design solution does not cause significant computing system performance degradation even under the weak cell rate of 10^{-4}. More specifically, when using the (283, 256, 7) BCH code, the average performance penalty is less than 2% over all the benchmarks. In comparison, the use of (572, 512, 13) BCH code tends to cause larger performance penalty, especially on benchmarks with relatively high memory traffic, such as *lbm*, *mcf*, *cactusADM* and *milc* [73]. Nevertheless, its average performance degradation is still less than 4% even in the case of using (572, 512, 13) BCH code.

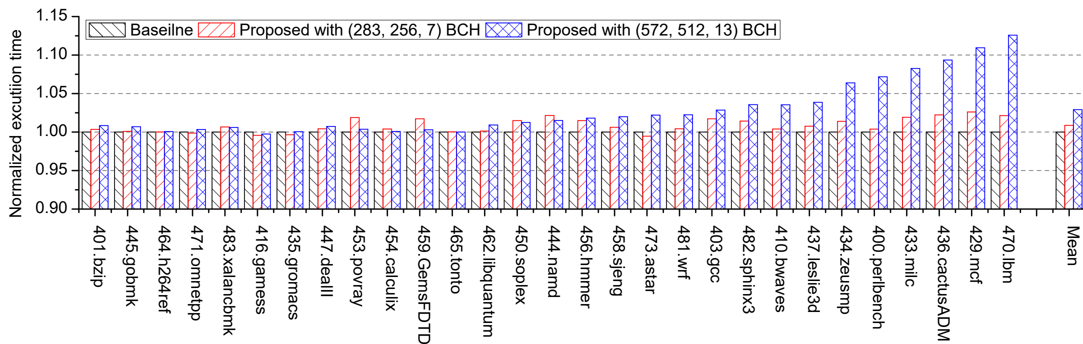
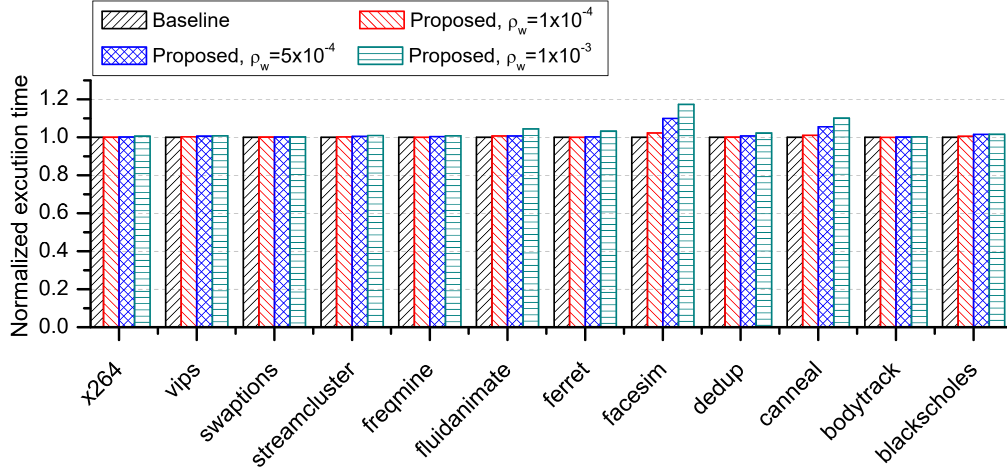


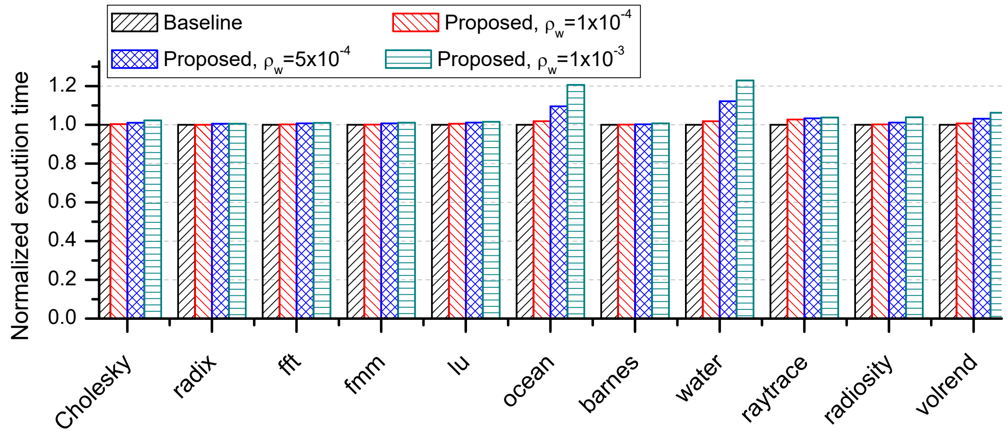
Figure 4.15: Simulated CPU2006 benchmark execution time under the detected weak cell rate of 10^{-4} (normalized to the baseline scenario with error-free DRAM).

I also used the popular workload benchmarks PARSEC 2.1 and Splash2 in our simulations, and Fig. 4.16 shows the average instruction per cycle (IPC) performance under different runtime memory read BER, which are normalized with the baseline scenario for which DRAM does not contain any weak cells. Under the runtime memory read BER of 10^{-4}, our design solutions only induce (much) less than 5% IPC loss over all the workloads. Even when we increase the BER to 10^{-3}, only four workloads suffer from IPC loss of much higher than 5%.

To further analyze the latency overhead of our error tolerance solution, we simulate proposed design with the HMC simulator *HMC-sim* [74]. We use a 4-link HMC with capacity of 4GB and (144, 128, 5) BCH code, and integrated the decoding function into the simulator. We generate memory read access requests and send them through HMC interface. The read latency overhead is calculated by comparing the clock cycles consumed in each task with different parameters, where the decoder buffer length is 64.



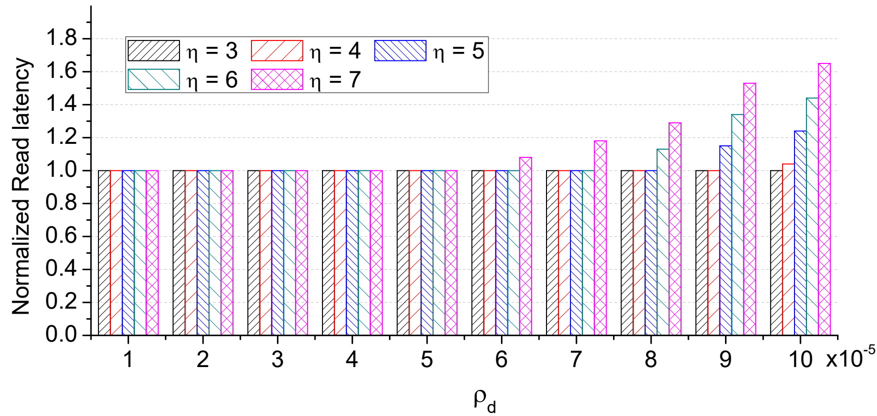
(a)



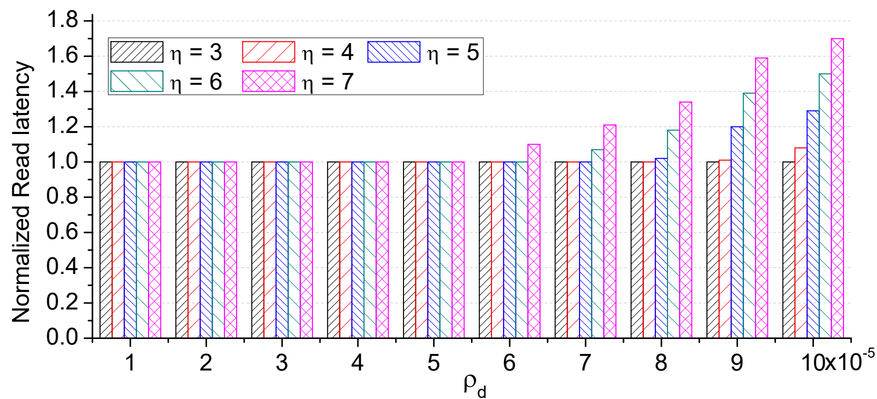
(b)

Figure 4.16: Normalized full-system IPC performance when using benchmarks of (a) Parsec and (b) Splash2 under different ρ_w with BCH (283, 256, 7) code.

Fig. 4.17(a) and Fig. 4.17(b) show the latency overhead with different average decoding latency. We simulated both random read access and sequential read access using the STREAM benchmark. Since *HMC-sim* does not provide memory device timing parameters, we set the ECC decoding latency is η times longer than the HMC vault average access latency and examined different values of η from 3 to 7. The simulation result shows as long as $\rho_d < 6 \times 10^{-5}$, read latency overhead induced by the proposed design is negligible. Considering the fact that DRAM die access latency is dozens of cycles in HMC [75], we expect that the value of η an could be less than 4 in practice.



(a)



(b)

Figure 4.17: Simulated read latency with (a) random read, and (b) sequential read using the modified HMC simulator. The parameter η represents that the ECC decoding latency is η times longer than the HMC vault average access latency.

4.4.5 Comparison with Related Work

Table 4.2 further summarizes the difference between existing related work and our proposed design solution. VECC [24], LOT-ECC [23], and RAIM+ECC Parity [25, 76] do not focus on tolerating weak cells that may randomly distribute across the entire memory die. ECP [56] demands the perfect knowledge on the location of all the weak cells, and CiDRA [57] essentially combines cache repair and ECP with SEC-DED. Both CiDRA [57] and Archshield [26] claim the capability of tolerating weak cell rates as high as $\rho = 10^{-4}$. Archshield requires the support (and hence modification) of OS. None of prior work explicitly employs ECC erasure decoding to leverage the detectability of weak memory cells.

Table 4.2: Comparison with related prior work.

	Granularity	Storage overhead	Error tolerance strength	Focus on weak cell tolerance	Tolerance to weak cell miss-detection	Trans-parent to OS
SEC-DED [65]	8B	12.5%	1b error/64B	N	N/A	Y
VECC [24]	64B	18.75%	4B error/64B	N	N/A	N
LOT-ECC9 [23]	64B	26.5%	1B error/8B	N	N/A	N
RAIM+ECC Parity [25, 76]	64B	19.1%	6B error/64B	N	N/A	N
ECP_6 [56]	64B	11.9%	6b error/64B	Y	N	Y
CiDRA [57]	8B	17.5%	$\rho = 10^{-4}$	Y	N	Y
ArchShield [26]	64B	17.2%	$\rho = 10^{-4}$	Y	Y	N
Proposed (16B, $t_{ran} = 1$)	16B	12.5%	3b error/16B	Y	Y	Y
Proposed (32B, $t_{ran} = 1$)	32B	12.5%	7b error/32B	Y	Y	Y
Proposed (64B, $t_{ran} = 1$)	64B	12.5%	11b error/64B	Y	Y	Y

This chapter presents a design solution that can largely improve the 3D DRAM fault tolerance by cohesively leveraging the memory ECC and detectability of weak cells. The stacked logic die inside 3D DRAM chips makes it a practically viable option for the DRAM manufacturers to employ ECC to tolerate unrepaired weak memory cells. In spite of its very simple basic concept, its practical realization is non-trivial and subject to memory data access latency and silicon cost overhead. In this work, we develop a design solution that can efficiently implement this simple concept at minimal latency and silicon cost penalty. This design solution can naturally embrace the inevitable weak cell detection inaccuracy and radiation-induced soft errors. The thorough mathematical formulation and analysis of this design solution are presented. Our analysis results show that, under the same redundancy overhead of 1:8 as today's ECC DIMM, this design solution can tolerate the weak cell rate of as high as 10^{-4} and 6×10^{-5} if 100% and 90% of all the weak cells are known in prior. We further present a parallel hardware architecture for implementing this design solution in the context of Micron's HMC 3D DRAM chips, and our ASIC design results show that the overall silicon cost is less than 0.4mm^2 at 45nm node. Our cycle-accurate simulations over a variety of computing benchmarks show that this design solution only incurs less than 2% performance degradation on average.

5. On the Use of DRAM with Unrepaired Weak Cells in Computing Systems

5.1 Introduction

Due to the inevitable process variability, DRAM chips typically contain some weak (or tail) cells that cannot satisfy the target data retention time (e.g., 64ms or 128ms) under the highest operating temperature (e.g., 85 °C). In current practice, before shipping DRAM chips to their customers, DRAM manufacturers must ensure that all the weak cells in each chip have been decommissioned through redundancy-repair [77]. Therefore, customers can safely expect error-free DRAM operations (except soft errors caused by radiation). As the industry is pushing the DRAM scaling into the sub-20nm regime, it is evident that the weak cell rate will significantly increase [78, 79], which could quickly make the convenient redundancy-repair fail to work at reasonable cost overhead. Intuitively, one possible choice is to relax the requirement on weak cell repairing (i.e., DRAM manufacturers do not have to repair all the weak cells in each DRAM chip) [80], and rely on computing systems to handle the data storage unreliability caused by the unrepaired weak cells.

This work concerns how computing systems could effectively embrace DRAM chips with unrepaired weak cells [81]. One apparent option is to apply error correction code (ECC) in the hardware stack (e.g., memory controller in CPU) to correct any errors caused by unrepaired weak cells during the runtime. Current mainstream computing systems support single-error correction and double-error detection (SEC-DED) by adding 8-bit redundancy to each 64-bit user data. Nevertheless, this is primarily for handling soft errors, and using existing SEC-DED for weak cell tolerance will leave memory vulnerable to soft errors. In addition, due to the coding redundancy overhead (i.e., 8-bit redundancy per 64-bit user data), most cost-sensitive computing systems (e.g., mobile phones) do not employ memory ECC in the hardware stack at all.

Portions of this chapter previously appeared as: H. Wang, Y. Li, X. Zhang, X. Zhao, H. Sun and T. Zhang, "On the Use of DRAM with unrepaired weak cells in computing systems," in *Proc. ACM 2nd Int. Symp. on Memory Syst.*, Washington DC, USA, 2016, pp. 327-337.

In mainstream computing systems, memory is managed in the unit of pages with the typical page size of 4kB. If one physical page contains at least one unrepaired weak memory cell, we call this page as an *error-prone* page. This work is interested in the scenarios with modest rate of unrepaired weak cells, where the percentage of error-prone pages is not significant (e.g., less than 10%) within the entire physical memory space. Intuitively, one may expect to handle those error-prone pages directly through page reservation: The OS can simply reserve all the error-prone pages and prevent them from being allocated to any applications. In fact, the GRand Unified Bootloader (GRUB) in Linux supports to reserve a few bad pages (which are identified through on-line memory testing) and prevent them from being used. Nevertheless, as the number of error-prone pages increases, such a simple treatment will become increasingly inadequate for two main reasons: (1) *Physical memory space fragmentation*: The error-prone page reservation directly results in fragmentation in the physical memory space. Although virtual memory management can naturally embrace physical memory fragmentation, OS tends to demand that certain physical memory region must be continuously available without any fragmentation, which is called *critical* memory region in this work. For example, during Linux boots up, its bootloader must copy the kernel image into a continuous physical memory space. In addition, the DMA buffer in Linux must occupy a continuous physical memory space without any fragmentation. In Linux, the critical memory region typically corresponds to the lowest 128MB in the physical memory space. For our interested error-prone page rate (e.g., 5%), it is almost impossible to have even a few MB of continuous physical memory space without encountering one error-prone page. Moreover, although it is intuitive to expect that fragmented memory could degrade computing system performance, we are not aware of any prior work in the open literature that empirically measured the computing system performance when OS reserves up to a few percentages of physical pages throughout the entire memory space. (2) *Memory resource waste*: Each error-prone page may contain few unrepaired weak cells. If we simply prevent all the error-prone pages from being used, a noticeable amount of memory capacity (e.g., 120MB out of 4GB under error-prone page rate of 3%) will be wasted.

This paper aims to address the above two issues. To mitigate physical memory space fragmentation, the simplest solution is to employ controller-based page re-mapping,

i.e., the memory controller internally maintains a physical page re-mapping table so that the error-prone pages are re-mapped and coalesced into one small continuous region. Since the re-mapping table implementation complexity is proportional to the number of error-prone pages being re-mapped, it may not be economically feasible to re-map all the error-prone pages. For example, given a computing system with 4GB memory (hence total 1M 4kB pages), an error-prone page rate of 3% corresponds to 30,000 error-prone pages. Implementing a mapping table with 30,000 entries could cause prohibitive silicon and latency overhead. The natural option is to only re-map error-prone pages within the critical memory region that must be free-of-fragmentation as demanded by OS. This is referred to as controller-based *selective* page re-mapping. Under this design framework, we carried out study from two aspects: (1) We developed a hash-based page re-mapping table design strategy and the associated table initialization and search algorithms, and quantitatively estimate its latency and hardware resource overhead under a variety of error-prone page rates. (2) Since we only re-map the error-prone pages within the critical memory region, the remaining non-critical memory region may be highly fragmented due to the reserved error-prone pages. Using SPEC CPU2006 suite [64], we carried out experiments to measure the computing system performance under different degrees of non-critical memory region fragmentation. The results suggest that the speed performance degradation is not significant even when reserving one page for every 16 pages.

Regarding the second issue above (i.e., the memory resource waste due to page reservation), we investigate the feasibility of applying software-based memory error tolerance to re-cycle error-prone pages for the zRAM function in Linux kernel. In essence, zRAM applies in-memory data compression to avoid paging to disk. We are particularly interested in zRAM mainly for three reasons: (1) zRAM is not sensitive the long latency of software-based memory error tolerance. As long as the latency of software-based memory error tolerance is lower than disk access latency, it is beneficial to re-cycle error-prone pages for zRAM through software-based memory error tolerance. (2) Since zRAM carries out data compression and decompression in the unit of 4kB page, its memory data access granularity is very coarse (e.g., at least a few hundreds of bytes), which can be leveraged to reduce the redundancy of memory error tolerance. (3) zRAM has been widely used in practice [82], e.g., ChromeOS and Android systems. In addition, recent work has

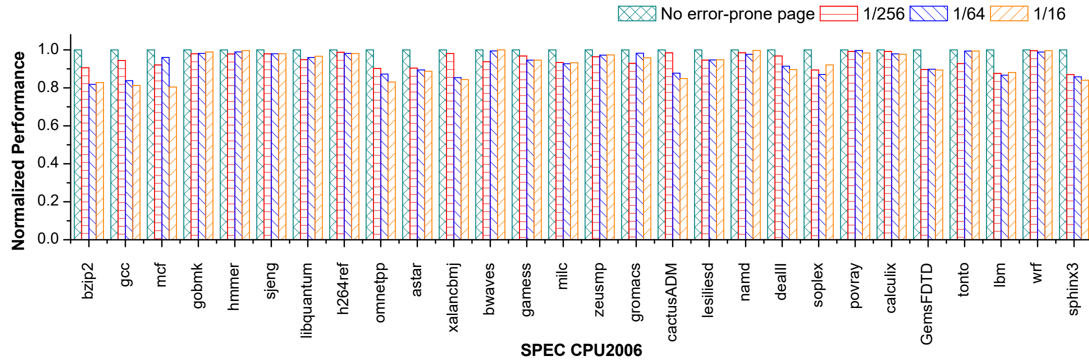


Figure 5.2: Normalized system performance under different memory fragmentation.

well demonstrated its effectiveness in virtual machines on servers [83–85], virtualized consumer electronics [86] and GPU buffer management [87]. In this work, we developed a specific software-based memory error tolerance design solution to re-cycle error-prone pages for zRAM with minimal speed performance impact. We carried out experiments by integrating this design solution into the latest zRAM, and the results show that we can readily re-cycle error-prone pages for zRAM at only 0.4% redundancy overhead and 7% speed performance penalty. In summary, this work for the first time investigates the use of DRAM with error-prone pages in computing systems, and its main contributions include:

- We present a simple re-mapping/re-cycling design framework to facilitate the practical use of DRAM chips with error-prone memory pages. This could contribute to enabling continuous DRAM bit cost reduction in the presence of significant technology scaling challenges.

- We develop a controller-based selective error-prone page re-mapping design strategy to eliminate the fragmentation in critical memory region, and present experimental results to show the effect of non-critical memory region fragmentation on computing system performance.

- We develop a specific software-based memory error tolerance design solution to re-cycle error-prone pages for zRAM in Linux kernel. By integrating this design solution into the latest zRAM module, we carried out experiments and the results well demonstrate its effectiveness.

5.2 Proposed Design Solutions

This work is interested in how to practically use DRAM chips with error-prone pages in computing systems. Under modest error-prone page rates (e.g., 5%), simply reserving all the error-prone pages from being used is subject to two issues, including the difficulty of ensuring fragmentation-free critical memory region and waste of memory resource. To address these two issues, this paper presents a design framework based upon the following two simple concepts: (1) Memory controller internally implements physical page re-mapping to move all the error-prone pages out from the critical memory region; (2) All the error-prone pages are re-cycled for zRAM in Linux kernel through software-based memory error tolerance. Since we only re-map error-prone pages in the critical memory region to minimize the re-mapping implementation complexity, the non-critical memory region could be highly fragmented, leading to obvious concerns on the computing system performance. Therefore, this section first presents our experimental results on the computing system performance in the presence of highly fragmented non-critical memory region, and then presents the specific design solutions to realize controller-based page re-mapping and error-prone page re-cycling for zRAM.

5.2.1 Effect of Fragmented Non-critical Memory Region

As pointed out above, modern OS always demands a continuous fragmentation-free physical memory space for critical operations such as system booting and DMA buffering. In Linux, such critical memory region typically spans over the lowest 128MB physical memory space. Therefore, the majority of physical memory space tends to be non-critical, for which the fragmentation caused by error-prone page reservation can be readily embraced by memory management. For example, Linux uses the *binary buddy allocator* to manage memory pages. The buddy allocator manages all the free memory space pool in the unit of blocks, each block contains 2^n consecutive pages where $0 \leq n \leq 10$. Hence, given the page size of 4kB, the capacity of one block ranges from 4kB to 4MB [88]. The buddy allocator always tries to serve a memory allocation request with as few number of free blocks as possible. When serving a small memory allocation request, the buddy allocator may need to decompose a large free block into a set of smaller blocks. When the buddy allocator puts de-allocated memory pages back to the free memory space

pool, it always tries to coalesce these pages with their neighboring pages in order to form large blocks [89].

To evaluate the impact of reserved error-prone pages in non-critical memory region, we modified the *start_kernel*() function in Linux kernel 3.14 so that we could manually reserve some memory pages from being used during the runtime. In particular, we studied three scenarios with different error-prone page rates: (1) we reserve one page every 256 pages (i.e., error-prone page rate of 0.39%), (2) we reserve one page every 64 pages (i.e., error-prone page rate of 1.56%), and (3) we reserve one page every 16 pages (i.e., error-prone page rate of 6.25%). Fig. 5.1 shows the free block statistics when Linux just boots up, under the three different scenarios (denoted as 1/256, 1/64, and 1/16, respectively). For the purpose of comparison, it also shows the free block statistics when there are no error-prone pages.

Figure 5.1: Free block statistics under different memory fragmentation.

We run the SPEC CPU2006 suite under the different scenarios with different non-critical memory region fragmentation, and Fig. 5.2 shows the measured performance results. For each benchmark, the performance is normalized to the baseline scenario free of error-prone pages. The results show that the performance degradation caused by error-prone page reservation is not significant (mostly less than 10%) even when we reserve one page every 16 pages. To the best of our knowledge, such results have not been presented in open literature. The relatively modest impact of non-critical memory re-

gion fragmentation further justifies the feasibility of only re-mapping error-prone pages in critical memory region, which will be discussed in the next subsection.

5.2.2 Controller-based Page Re-mapping

In order to ensure a fragmentation-free critical memory region, we propose to apply controller-based page re-mapping to move all the error-prone pages out from the critical memory region. Let E_c and E_o denote the sets that contain all the error-prone pages within the critical memory region and non-critical memory region, respectively, and let G_o denote the set that contains all the error-free pages in the non-critical memory region. Hence, the error-prone page re-mapping between E_c and G_o is a bijective function, i.e., each page in E_c has its unique replacement page in G_o. As shown in Fig. 5.3, when a request of physical page $P \in E_c$ is issued, the memory controller internally re-maps this request to its unique replacement page $R \in G_o$. Thus, the critical memory region is continuous (i.e., fragmentation-free) from the OS's perspective. Clearly, such physical memory page mapping is transparent (i.e., invisible) to the OS.

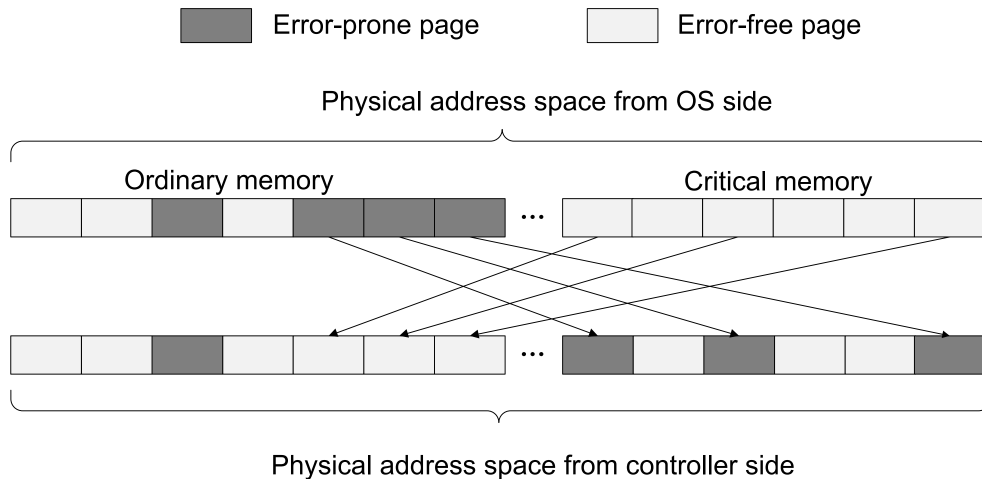


Figure 5.3: Illustration of the bijective mapping between error-prone pages in critical memory region and error-free pages in non-critical memory region.

To realize such OS-transparent page re-mapping, the memory controller internally implements a page mapping table that stores the mapping relation between E_c and G_o. Suppose the entire memory space contains total 2^A physical memory pages. Hence, the set $E_c = \{P_i,\ 1 \le i \le |E_c|\}$, where each P_i is an A-bit address of one error-prone physical

page. Suppose the mapping table consists of 2^B entries. Note we also need to record the replacement pages address in $R_c = \{R_i,\ 1 \leq i \leq |E_c|\}$ in the mapping table, hence the memory controller needs at least $2|E_c|$ entries in the mapping table, i.e., $2^B \geq 2|E_c|$. Each mapping table entry corresponds to one physical page and consists of four parts: $Valid$, Tag, $Index_m$ and $Index_c$, where $Valid$ indicates whether current entry is occupied, Tag confirms the physical page address, $Index_m$ presents mapping relationship, and $Index_c$ is used to handle address collision. Let $Table[l][j]$, where $0 \leq l \leq 2^B - 1$ and $0 \leq j \leq (A+B)$, denote the j-th bit in the l-th entry in the mapping table. Each entry needs total $A+B+1$ bits, and table entry structure is shown in Table. 5.1.

Table 5.1: Mapping table entry structure

$Table[l].Valid$	$Table[l][A+B]$
$Table[l].Tag$	$Table[l][A+B-1:2B-1]$
$Table[l].Index_m$	$Table[l][2B-1:B]$
$Table[l].Index_c$	$Table[l][B-1:0]$

The mapping table initialization procedure is illustrated in Algorithm 1. There are three major steps for the mapping table initialization. First, for an error-prone page whose address $P_i \in E_c$, we use its lower B bits (i.e., $P_i[B-1:0]$) as its hash address in the mapping table. If the table entry is empty (i.e., $Valid = 0$), we set $Valid$ as 1 and assign the value of the Tag as the higher $A-B$ bits in the address of P_i (i.e., $P_i[A-1:B]$). This entry is called a base entry. If the $Valid$ bit is already 1, it means the base entry is already used by another page, i.e., there are at least two error-prone pages sharing the same lower B bits in their addresses. This is referred to as a collision. As shown in Algorithm 1, we scan the entire set E_c and put all the error-prone pages, which encounter collision, into a set C.

Next, as described in Algorithm 1, we find a free table entry (i.e., an entry with $Valid$ as 0) for each page P_i which encounters collision. $Index_c$ represents the table address of the entry for another page with the same lower B bits page address, which is similar to the next pointer in a list structure. We can just simply walk through the table, starting from $Table[P_i[B-1:0]]$, until we find the first empty entry which we call it collision entry. Then we set its $Valid$ as 1 and set the Tag of the collision entry as $P_i[A-1:B]$. Since the first step is already finished, using collision entry does not introduce any

Algorithm 1: Mapping table initialization algorithm

```
Input: E_c = {P_i}(1 ≤ i ≤ |E_c|), C = ∅, prev = 0, Y_i = 0(1 ≤ i ≤ |E_c|),
  Table[i][j] = 0, (0 ≤ l ≤ 2^B − 1, 0 ≤ j ≤ (A+B)), m = 0
Output: Table[i][j] = 0, (0 ≤ l ≤ 2^B − 1, 0 ≤ j ≤ (A+B))

  Step 1:
  for i = 1 : |E_c| do
    if Table[P_i[B−1 : 0]].Valid == 0 then
      Table[P_i[B−1 : 0]].Valid = 1;
      Table[P_i[B−1 : 0]].Tag = P_i[A−1 : B];
      Y_i = P_i[B−1 : 0]
    else
      C = C ∪ P_i
    end if
  end for

  Step 2:
  for P_i ∈ C do
    prev = P_i[B−1 : 0], m = 1;
    while Table[P_i[B−1 : 0] + m].Valid ≠ 0 do
      m++
      if Table[prev].Index_c == P_i[B−1 : 0] + m then
        prev = P_i[B−1 : 0] + m
      end if
    end while
    Table[P_i[B−1 : 0] + m].Valid = 1;
    Table[prev].index_c = P_i[B−1 : 0] + m;
    Y_i = P_i[B−1 : 0] + m
  end for

  Step 3:
  for Y_i do
    l = 1, m = 2^(A−B) − 1
    while Table[l].Valid ≠ 0 || (m << B + l) ∈ E_o do
      l++
      if l = 2^B then
        m−−
      end if
    end while
    Table[l].Valid = 1
    Table[l].Tag = m
    Table[l].Index_m = Y_i
    Table[l].Index_c = 0
    Table[Y_i].Index_m = l
  end for
```

further collision. What differs from the first step is that we need to store the newly found collision entry index in its base entry, $Table[P_i[B-1:0]$, as $Table[P_i[B-1:0]].Index_c$. If there are more than one collision, the index of the second collision entry is stored in first collision entry's $Index_c$.

As described in Algorithm 1, the third step aims to search a physical page in G_o as the replacement page for each $P_i \in E_c$. This search step skips all the error-prone pages in E_o, and can start from anywhere in non-critical memory region with an arbitrary high $(A-B)$-bit address, denoted as m. As described in Algorithm 1, m is set to $2^{A-B}-1$ and the search starts from the highest physical memory address. Replacement pages only use empty entries in the mapping table, hence there is no collision for replacement pages. Once a proper replacement page R_i is found, $Index_m$ of P_i and R_i point to the table address of each other.

During the runtime, upon receiving each memory access request, the memory controller must check whether the corresponding page has been re-mapped, and find out the re-mapped address if the page indeed has been re-mapped. In particular, the memory controller must search through the page mapping table based upon its data structure as described above. The search latency could vary dependent upon whether the page is re-mapped and whether collision occurs. Let L denote the number of mapping table entries. Algorithm 2 shows the procedure when search the page re-mapping table.

Algorithm 2: Remapping search algorithm

Input: $Table[L]$, $P[A-1:0]$,$y=0$
Output: $R[A-1:0]$
  $y = P[B-1:0]$
  while $Table[y].Tag \neq P[A-1:B] \& Table[y].Index_c \neq 0$ **do**
    $y = Table[y].Index_c$
  end while

  if $Table[y].Tag == P[A-1:B]$ **then**
    $R[B-1:1] = Table[y].Index_m$
    $R[A-1:B] = Table[Table[y].Index_m].Tag$
  else
    $R[A-1:0] = P[A-1:0]$
  end if

For the purpose of illustration, Fig. 5.4 shows one example. When a memory re-

quest is issued to page P, we first use its lower N bits $P[B-1:0]$ to find the corresponding table entry. Suppose $P[B-1:0] = 001$, then we check $Valid$ bit of the $1st$ entry. If $Valid = 1$, Tag in this entry is read out to compare with $P[A-1:B]$. If the Tag, denoted as $P'[A-1:B]$, is not equal to $P[A-1:B]$, $Index_c$ is read out. As described in Algorithm 1, collision handling starts from $l = 1$. Thus, if $Index_c = 0$, it means there is no collision. Otherwise, we read collision entry pointed by $Index_c$ which is 004 as shown in Fig. 5.4. Since the Tag of table entry 004 matches $P[A-1:B]$, $Index_m = 008$ is the index of the replacement page R_i for P_i. Since there is no collision for replacement page, we can directly get the page address of R_i by concatenating $Index_m = R_i[B-1] = 008$ and $Tag = R_i[A-1:B]$. The mapping procedure of R_i is similar but the $Index_c$ is always 0 for replacement page entry since R_i can be chosen arbitrarily.

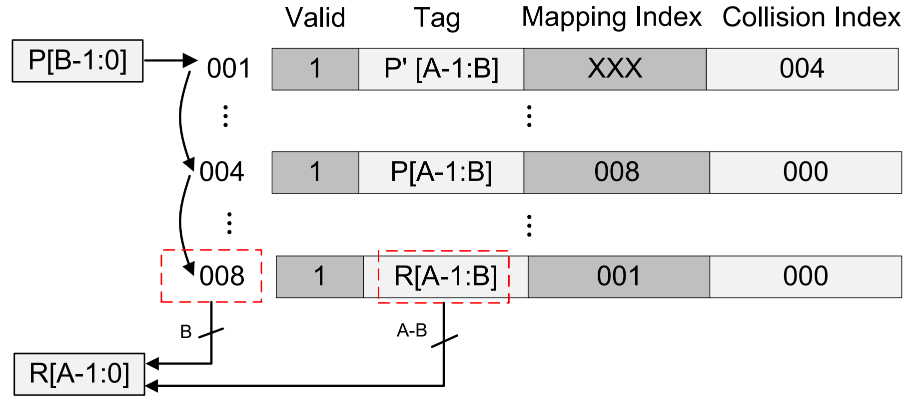


Figure 5.4: One example to illustrate the mapping table research procedure with one collision.

5.2.3 Re-cycling Error-prone Pages for zRAM

In order to re-cycle error-prone pages, we have to apply software-based memory error tolerance to ensure their data storage integrity. Clearly, those re-cycled error-prone pages can only be used by applications that can natively tolerate the relatively long latency of software-based memory error tolerance. The zRAM in Linux kernel is one such application with wide real-world use. In the following, we first discuss the basics of zRAM, then present a specific design solution to effectively re-cycle error-prone pages for zRAM.

5.2.3.1 zRAM Basics

zRAM is a Linux kernel feature to provide swapping on DRAM. It was merged into Linux kernel mainline in version 3.14 and also available in Chrome OS and Android 4.4. When zRAM is enabled, pages which need to be swapped out of memory are compressed and written to zRAM instead of being swapped to disk. When a page is written to zRAM, Linux attempts to compress it first, where the default compression algorithm is Lempel-Ziv-Oberhumer (LZO) compression algorithm [90]. The compressed page is referred as *zpage*. If the size of compressed page exceeds a pre-defined threshold (e.g., 3kB in Linux 3.14), the compression is regarded as unsuccessful and this page is stored in its original uncompressed form. Otherwise, zRAM uses *zsmalloc* allocator to allocate memory for the compressed page. The *zsmalloc* allocator compounds multiple noncontiguous pages together as one *zspage*. When Linux needs to swap in a page from zRAM, it first checks whether this page is compressed as a *zpage*. If it is a *zpage*, Linux decompresses it and copies it to memory. Otherwise, Linux directly copies this page without any decompression.

To re-cycle error-prone pages for zRAM, we introduce a new structure called *Ezspage* in zRAM, which contains several error-prone pages. Whenever zRAM uses an *Ezspage* to store a compressed page, it first protects the compressed page with software-based error tolerance, and stores both the compressed page and its associated error tolerance redundancy into the *Ezspage*. When reading the page from *Ezspage*, we first carry out software-based error detection and correction to obtain the original compressed page data, and then decompress the data to recover the original page data.

5.2.3.2 Proposed Software-based Error Tolerance

Clearly, one key design issue is the design of software-based memory error tolerance for ensuring the data storage integrity of the *Ezspages*. In this work, we propose to use a concatenated coding scheme, i.e., given one compressed page to be stored in zRAM, we first protect the compressed page data using an error detection code (EDC) as an inner code to add certain error detectability, and then apply an outer ECC to add sufficient error correctability. Fig. 5.5 illustrates the corresponding data write and read procedure.

Compared with the straightforward ECC-only data protection, such concatenated

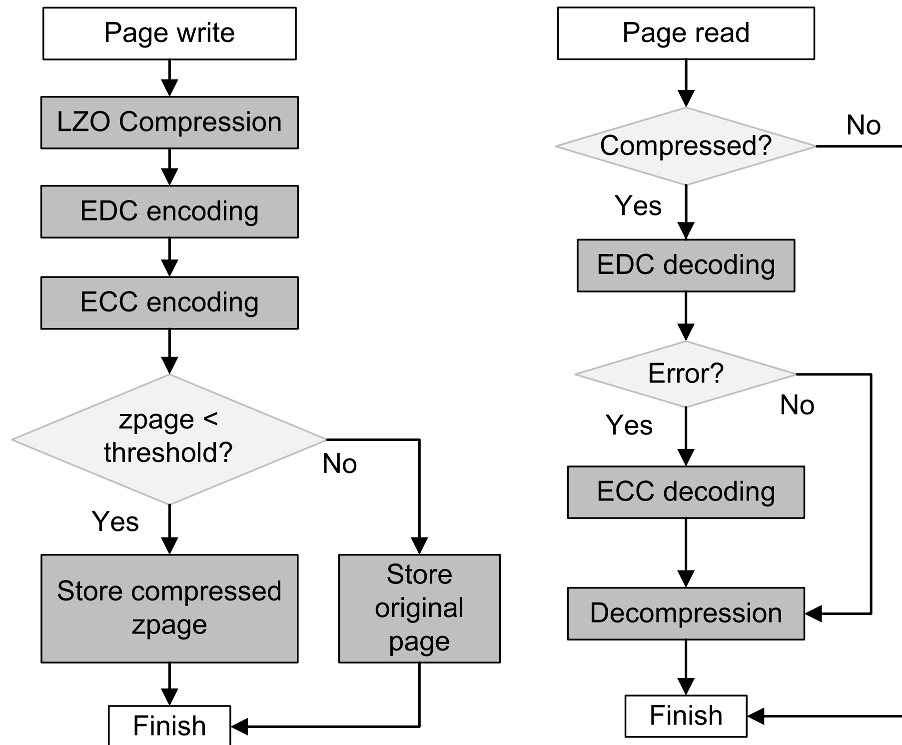


Figure 5.5: Illustration of data write and read procedure with concatenated coding.

EDC/ECC data protection can significantly reduce the software-based error tolerance latency overhead at small amount of extra redundancy. This is for two main reasons: (1) EDC decoding tends to be much faster than ECC decoding, which will be further discussed later; (2) Error-prone pages do not necessarily always experience memory bit errors in the runtime. As discussed above, weak memory cells are those memory cells that cannot meet the data retention time under the highest operating temperature (e.g., 80°C). Due to the strong dependency of memory cell data retention time on the temperature, weak cells most likely will not cause bit errors during the runtime with normal operating temperature. For the purpose of further quantitative demonstration, we measured bit error statistics of a 1Gb DRAM chip under different operating temperature and refresh period, and the results are shown in Fig. 5.6. Under each combination of operating temperature and refresh period, we write random data to DRAM chip and repeatedly read data over a long period to record all the DRAM cells that have ever generated errors. Accordingly we calculate the DRAM bit error rate by dividing the total number of DRAM cells with the number of DRAM cells that have ever generated errors. The measurement results

clearly show that most weak cells do not result in bit errors during runtime under normal operating conditions. Therefore, it is reasonable to expect that error-prone pages most of time do not experience memory bit errors during the normal operation.

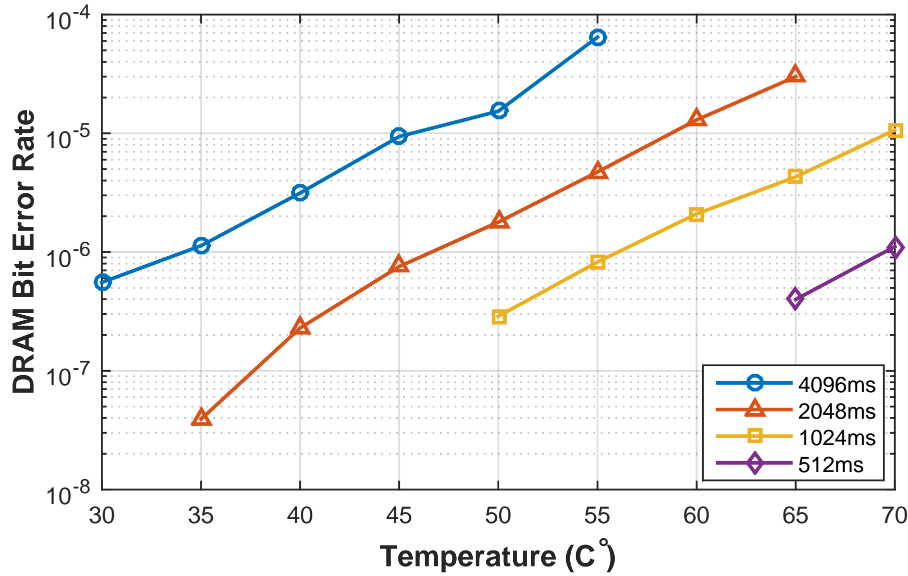


Figure 5.6: Measured DRAM bit error rate vs. refresh period under different temperature.

In this work, we choose the well-known cyclic redundancy check (CRC) code to implement the inner EDC for two main reasons. First, CRC code has relatively large minimum Hamming distance hence good error detection capability. For instance, the iSCSI CRC polynomial (0x11EDC6F41) can provide minimum Hamming distance of 4 with only 32-bit CRC redundancy for k-bit data ($5276 \leq k \leq 2^{31} - 1$) [91, 92]. Second, CRC encoding/decoding computation is relatively simple and only involves binary polynomial remainder calculation. Moreover, modern Intel processors can accelerate CRC computation for the iSCSI CRC (0x11EDC6F41) using the *CRC32C* instruction in SSE4 (Streaming SIMD Extensions 4) [93].

We choose binary BCH code [94] to implement the outer ECC. Although prior work have developed a variety of multi-error-correction ECCs [23, 73, 77, 95] for memory error correction, they primarily focused on ECC engine implementation in the hardware stack. Unfortunately, the encoding and decoding algorithms of those hardware-oriented ECC typically do not well match with the byte-oriented processing of CPUs. In comparison,

the encoding and decoding of classical BCH codes can be formulated as byte-oriented, which is a much better fit to CPUs. As a result, we choose binary BCH codes to realize the outer ECC in the software-based memory error tolerance. A binary BCH code is constructed over a GF(2^m), where the BCH codeword length is not larger than $2^m - 1$. BCH code encoding and decoding algorithms have been well studied and interested readers are referred to [96]. Both its encoding and decoding heavily involve GF(2^m) arithmetics. In order to accelerate CPU-based BCH code encoding and decoding, the key is to carry out the GF(2^m) arithmetics in the byte-oriented form. To achieve this objective, we must employ the table look-up based strategy. In particular, to process p bytes of data at one time in GF(2^m) arithmetics, we pre-compute basic GF(2^m) arithmetics (in particular multiplication) and accordingly build a look-up table (LUT). During the runtime, we can directly use the pre-computed LUT to accelerate the GF(2^m) arithmetics. When using the CRC code as the inner EDC and BCH code as the outer ECC, each compressed *zpage* are stored into error-prone pages together with its CRC redundancy and BCH redundancy. Hence the *Ezspage* structure can be illustrated in Fig. 5.7 where the *Ezspage* is formed by four error-prone pages and the *zpage* size is 2kB, half of the physical memory page.

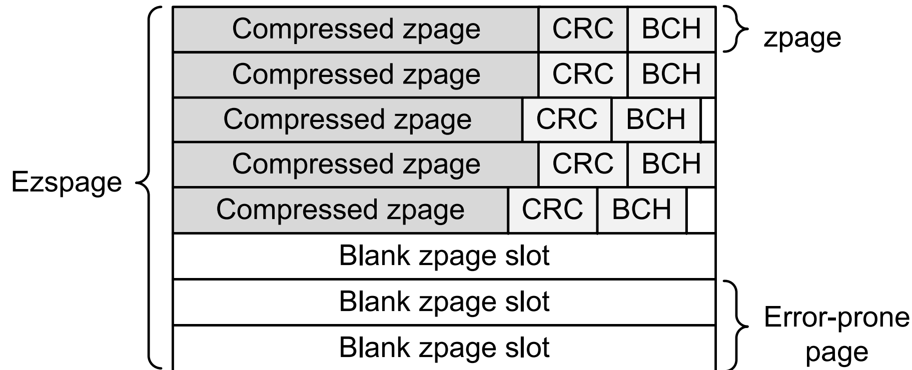


Figure 5.7: Illustration of Ezspage structure with concatenated-CRC/BCH memory error tolerance.

5.3 Evaluation

5.3.1 Controller-based Page Re-mapping

We first studied the implementation overhead of the proposed controller-based page re-mapping in terms of both latency and hardware resources.

5.3.1.1 Latency Overhead

When a memory request on page address P is sent to the memory controller, we first check whether this page has been re-mapped. As discussed above, the lower $P[B-1:0]$ bits of page address are used as the entry address in re-mapping table, and we should read Tag, $Index_m$ and $Index_c$ out. There are the following three possible scenarios:

1. $Tag \neq P[A-1:B] \& Index_c = 0$: Since Tag does not match, P does not belong to R_c. Meanwhile, as described in Algorithm 1, when $Index_c = 0$, there is no collision on this page. Thus P does not belong to E_c either. Hence the memory controller can safely conclude that the page p is not re-mapped. In this case, the latency overhead is the latency of accessing one table entry.

2. $Tag \neq P[A-1:B] \& Index_c \neq 0$: In this case, there may be collision, and meanwhile P does not belong to R_c. The memory controller needs to read the Tag from another entry pointed by $Index_c$ in current entry to further compare with $P[A-1:B]$ to determine whether current page P is indeed re-mapped. In this case, the latency overhead is the latency of accessing two or more table entries.

3. $Tag = P[A-1:B]$: The page P is in E_c or R_c, i.e., the page P is re-mapped. According to the data structure as described in Section 5.2.2, the memory controller can determine the re-mapped page location. In this case, the latency overhead is the latency of accessing two table entries.

Among the three cases above, the latency overhead of the case 2 depends on the number of collisions encountered during the table search. Let Pr_e denote the error-prone page rate, and assume all the error-prone pages randomly distribute across the entire physical memory space. Since total 2^{A-B} pages share each unique lower B bit address, we can calculate the probability that collision occurs i times during the mapping table search as

$$Pr_c(i) = \binom{2^{A-B}}{i} Pr_e^i (1-Pr_e)^{2^{A-B}-i}. \quad (5.1)$$

To minimize its silicon cost, the page re-mapping table should be implemented using SRAM. Hence, we assume that, during one clock cycle, the memory controller can only read one table entry. In the case of page address collision, the memory controller has to

spend multiple clock cycles for each table search. It takes one clock cycle to read one entry so one more collision leads to one more clock cycle latency. Fig. 5.8 shows the estimated average table search latency under different error-prone page rate. As the error-prone page rate increases, the page address collision probability accordingly increases, leading to a longer average table search latency as shown in Fig. 5.8. We note that the latency is not significant, which is essentially ignorable compared with DRAM access latency. For example, even under the error-prone rate of 10%, the average table search latency is only 2.35 clock cycles.

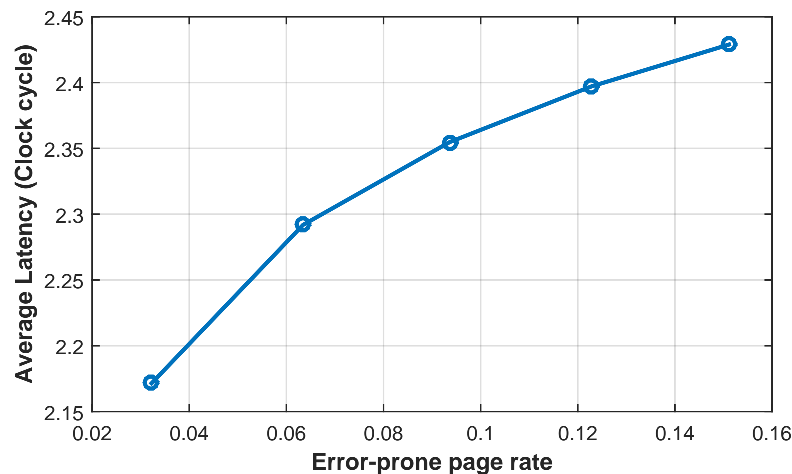


Figure 5.8: Average table search latency under different error-prone page rate.

5.3.1.2 Hardware Overhead

As discussed above in Section 5.2.2, the memory controller only aims to re-map all the error-prone pages in the critical memory region. The number of the mapping table entries must be at least twice of the number of error-prone pages in the critical memory region. Let L denote the number of mapping table entries, M_c denote the capacity of critical memory region, and S_p denote the page size (i.e., typically 4kB in current practice). Recall that Pr_e denotes the error-prone page rate, and define Pr_o as the mapping table overflow probability, i.e., the the probability that the mapping table size L is not large enough to re-map all the error-prone pages in the critical memory region. Given the value

of Pr_e and L, we can estimate the mapping table overflow probability as

$$Pr_o = \sum_{i=L+1}^{M_c/S_p} \binom{M_c/S_p}{i} Pr_e^i (1 - Pr_e)^{M_c/S_p - i}. \tag{5.2}$$

Hence, by setting a sufficiently low mapping table overflow probability threshold, we can calculate the minimum table size L given the current error-prone page rate Pr_e. Fig. 5.9 shows the required mapping table size under different error-prone page rate when the critical memory region capacity M_c is 64MB, 128MB, and 256MB, respectively. We set the target mapping table overflow probability as 10^{-50}. The results clearly show that, by only re-mapping error-prone pages in the critical memory region, the hardware complexity of the mapping table is not significant, e.g., the mapping table capacity only needs to be 70kB even when the critical memory region is 256MB and error-prone page rate is as high as 10%.

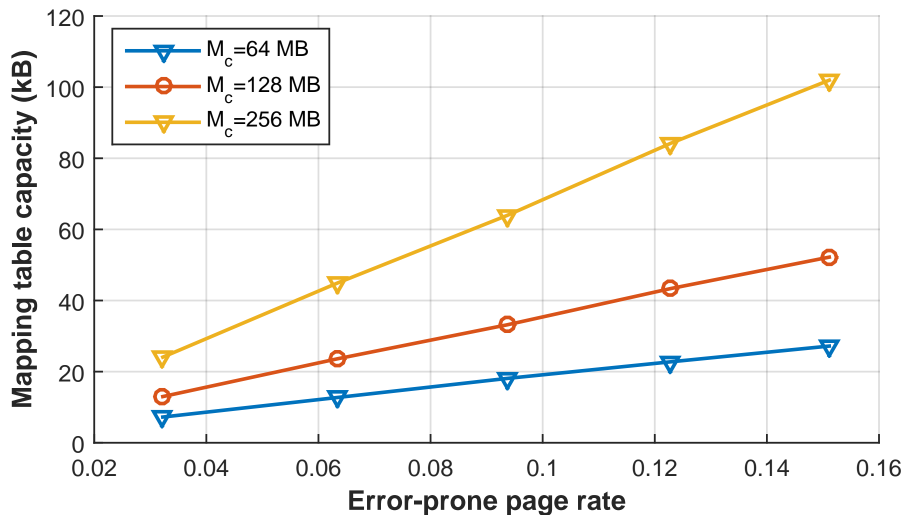


Figure 5.9: Page mapping table capacity under different error-prone page rate.

5.3.2 Re-cycling Error-prone Pages for zRAM

As discussed above in Section 5.2.3, we can re-cycle error-prone pages for zRAM in order to eliminate the memory resource waste. The key is to apply a concatenated-CRC/BCH software-based memory error tolerance design solution to ensure the data storage integrity of error-prone pages. The use of software-based error tolerance clearly

induces latency overhead, which will degrade the operational throughput of zRAM. In this work, we carried out detailed experiments to evaluate the impact, and the results demonstrate that the proposed design solution incurs very small zRAM throughput degradation.

First, we note that CRC encoding and decoding involve the essentially the same operations of calculating the CRC bits. Hence, CRC encoding and decoding have the same latency. In this work, we implement CRC encoding/decoding using the CRC32C instruction in modern x86 processors. For example, a single core of an Intel i5 processor 650 can compute the CRC at the throughput of 0.145 cycles/byte with a single thread [93]. Assuming the average size of a compressed page in zRAM is 2kB, we measured that the CRC coding latency is only about 0.54μs on a 2.4GHz Intel i7 processor, which is much less than the latency of data compression/decompression and DRAM data access.

The BCH code encoding is realized through multiplication with a generator polynomial on a Galois Field. The encoding throughput depends on the size of the generator polynomial size, which further depends on the target error correction strength. BCH code decoding involves three steps, including syndrome calculation, key equation solver, and Chien search. We implemented a software-based library to implement BCH code encoding and decoding, where we use LUT-based design strategy to accelerate the Galois Field multiplications for both encoding and decoding. Assuming the average size of a compressed page in zRAM is 2kB, we measured that the BCH code encoding latency is 3.48μs, 3.70μs, and 3.78μs under the error correction strength of 2, 3, and 4, respectively (i.e., each BCH codeword can correct up to 2, 3, and 4 errors, respective). We measured that the BCH code decoding latency is 3.84μs, 15.40μs, and 24.00μs when correcting 2, 3, 4 errors.

In order to investigate the impact of software-based error tolerance on the speed performance of zRAM, we modified the zRAM in Linux 3.14 by integrating the CRC and BCH coding library. Since the runtime data compressibility plays an important role on the zRAM operational throughput, we carried out experiments with a variety of data pages with different compressibility. We quantify data compressibility using compression ratio, which is defined as the ratio between the size of compressed page and the original uncompressed page, i.e., the compression ratio is in (0,1] and a lower compression ratio means better data compressibility. In addition to the proposed concatenated-CRC/BCH

design approach, we also studied the straightforward design option of BCH-only memory error tolerance for the purpose of comparison. Moreover, since the latency of BCH code encoding/decoding depends on the error correction strength, we further evaluate the zRAM throughput performance under different error correction strength of BCH code.

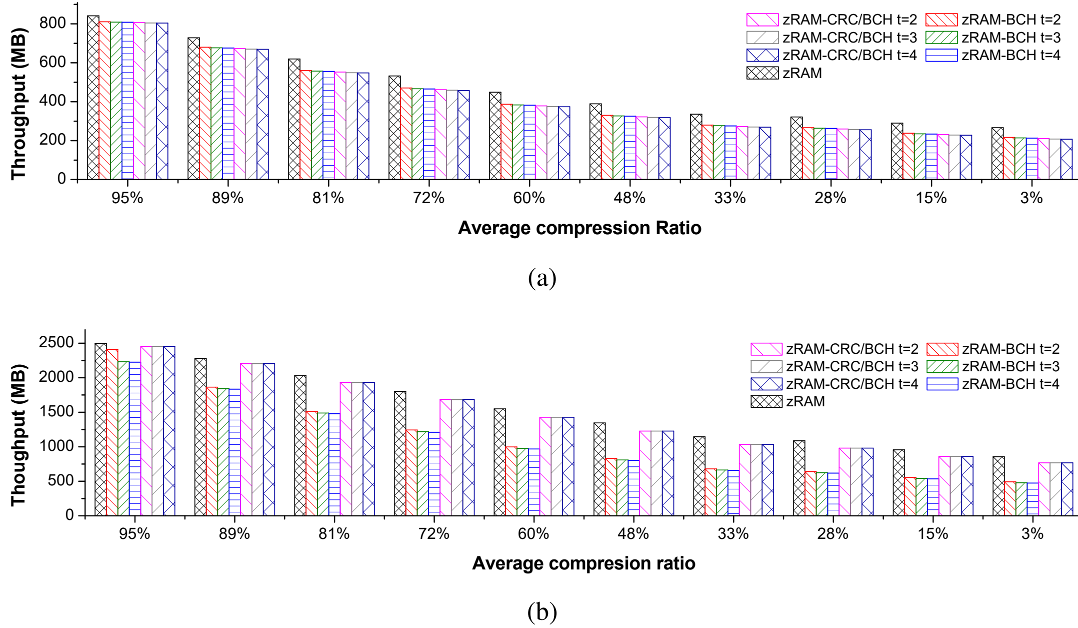


Figure 5.10: Measured throughput of (a) zRAM write and (b) zRAM read operations.

In our experiments, we use a large variety of files with different compressibility, and categorize all the data pages into 10 groups with different average compression ratios including 95%, 89%, 81%, 72%, 60%, 48%, 33%, 28%, 15% and 3%. Fig. 5.10 shows the measured zRAM write and read throughput under three cases: (1) the baseline case *zRAM*, which is the original zRAM without re-cycling any error-prone pages, (2) the case *zRAM-BCH*, which applies BCH code alone to realize software-based memory error tolerance for error-prone pages, and (3) the case *zRAM-CRC/BCH*, which applies the proposed concatenated-CRC/BCH scheme to realize software-based memory error tolerance for error-prone pages. For the last two cases, we further consider different error correction strength of the BCH code (i.e., the error correction strength t is 2, 3, and 4).

Fig. 5.10 shows the results, which clearly show the strong dependency of zRAM operational throughput on data compressibility (i.e., a higher data compressibility leads to

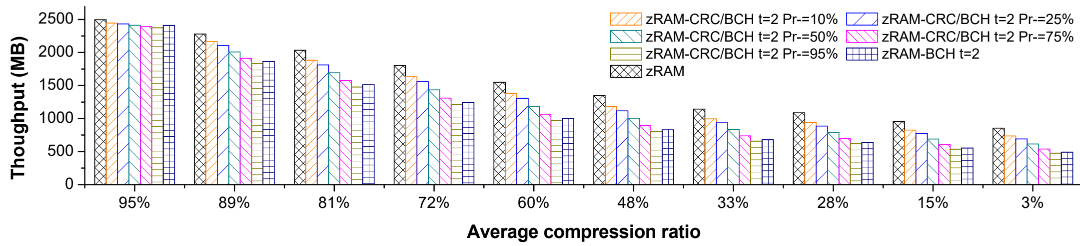


Figure 5.11: Read throughput of zRAM-CRC/BCH under different BCH code decoding occurrence probability.

a lower zRAM operational throughput). This is because that the compression engine LZO has to carry out more data manipulations for pages with higher compressibility, leading to a lower throughput. For zRAM write, as shown in Fig. 5.10(a), various software-based memory error tolerance options (i.e., zRAM-BCH and zRAM-CRC/BCH with different BCH error correction strength) tend to incur similar and relatively small throughput degradation compared with the original zRAM. Meanwhile, we can observe that the zRAM write throughput degradation increases as the compression ratio reduces (i.e., as the data become more compressible). For example, under the average compression ratio of 89%, the zRAM write throughput degradation is only about 6%. When the average compression ratio reduces to 60% and 15%, the zRAM write throughput degradation increases to 13% and 18%, respectively.

The results in Fig. 5.10(b) clearly show noticeably different impact on zRAM read throughput when using zRAM-BCH or zRAM-CRC/BCH. As discussed above, the unrepaired weak cells in error-prone pages do not induce bit errors most of time, especially under normal operating temperature. Hence, the latency overhead in the case of zRAM-CRC/BCH is mainly dominated by the CRC decoding, which fortunately has a very high throughput. Nevertheless, for the case of zRAM-BCH, we have to always invoke the BCH decoding, which has a much lower throughput than CRC. Therefore, as shown in Fig. 5.10(b), zRAM-BCH has much lower read throughput than the other two design options. In addition, due to the very high CRC decoding throughput, zRAM-CRC/BCH only incurs very small read throughput degradation. Similar the zRAM write, the zRAM read throughput degradation also increases as the data compression ratio reduces. Under the average compression ratio of 89%, the zRAM-BCH and zRAM-CRC/BCH incur 18%

and 3% read throughput degradation. When the average compression ratio reduces to 60% and 15%, zRAM-BCH incurs 35% and 42% read throughput degradation, while zRAM-CRC/BCH only incurs 7% and 10% read throughput degradation.

Finally, for our proposed concatenated-CRC/BCH design strategy, we further evaluate the impact on zRAM read throughput when the BCH decoding is indeed invoked with a non-negligible probability. Fig. 5.11 shows the measured zRAM read throughput when BCH code decoding is invoked with different probability (denoted as Pr_z), where the BCH code error correction strength is $t = 2$. As shown in Fig. 5.11, we consider five different values of Pr_z, including 10%, 25%, 50%, 75%, and 95%. For the purpose of comparison, it also shows the read throughput of the original zRAM and zRAM-BCH. The results show that the read throughput degradation of zRAM-CRC/BCH increases as the probability Pr_z increases. Results show that, if the Pr_z is very high (e.g., 95%), the read throughput of zRAM-CRC/BCH could be even worse than that of zRAM-BCH. Fortunately, it is reasonable to expect that the probability Pr_z should be small, even if it is not negligible, in practice. As shown in Fig. 5.11, even under the Pr_z of 25%, zRAM-CRC/BCH only degrades the zRAM read throughput by 16% compared with the original zRAM, when the average compression ratio is 60%.

6. Conclusion and Future Work

6.1 Conclusion

This thesis focuses on architecting memory system upon highly scaled error-prone memory technologies. In particular, this thesis presents a set of design techniques for memory controller and memory system to optimize error tolerance adopting specific memory architecture and runtime data characteristics.

The first contribution of this thesis is the development of design solutions that relax the scaling up of storage node A/R in the presence of continuous scaling down of DRAM cell size. Such a relaxed scaling directly results in storage node capacitance reduction and hence more weak memory cells. The thesis proposes to explicitly expose bit errors caused by the weak cells to system-level memory controllers, leading to a system-aided DRAM scaling. As the first step to explore this unconventional DRAM scaling strategy, analysis and calculations are carried out to estimate weak cell rate in the presence of reduced storage node capacitance based upon measurement results with 3xnm DRAM chips. This thesis further presented a data-dependent error-tolerance design strategy to reduce the redundancy overhead of system-aided weak cell mitigation, and carried out quantitative evaluation in the case of using DRAM in SSDs. For STT-RAM, the underlying rationale is to cohesively exploit the run-time data characteristics and the read disturbance vs. sensing error trade-off in STT-RAM. The thesis presents three specific data-dependent error-tolerance design techniques, and demonstrate their effectiveness in the context of using STT-RAM to replace DRAM in SSDs. Detailed study is performed on both effective storage capacity gain and energy consumption overhead, and the results show that these design solutions can increase the effective STT-RAM storage capacity by 26%, compared with conventional design practice.

The second contribution is the development of design solutions that can largely improve the 3D DRAM fault tolerance by cohesively leveraging the memory ECC and detectability of weak cells. The stacked logic die inside 3D DRAM chips makes it a practically viable option for the DRAM manufacturers to employ ECC to tolerate unrepaired weak memory cells. In spite of its very simple basic concept, its practical realization is

non-trivial and subject to memory data access latency and silicon cost overhead. In this work, we develop a design solution that can efficiently implement this simple concept at minimal latency and silicon cost penalty. This design solution can naturally embrace the inevitable weak cell detection inaccuracy and radiation-induced soft errors. The thorough mathematical formulation and analysis of this design solution are presented. Our analysis results show that, under the same redundancy overhead of 1:8 as today's ECC DIMM, this design solution can tolerate the weak cell rate of as high as 10^{-4} and 6×10^{-5} if 100% and 90% of all the weak cells are known in prior. We further present a parallel hardware architecture for implementing this design solution in the context of Micron's HMC 3D DRAM chips, and our ASIC design results show that the overall silicon cost is less than 0.4mm^2 at 45nm node. Our cycle-accurate simulations over a variety of computing benchmarks show that this design solution only incurs less than 2% performance degradation on average.

The third contribution of this thesis is the study on using DRAM chips with unrepaired weak cells in computing systems. In particular, this thesis develops a controller-based selective error-prone page re-mapping design strategy to eliminate the physical memory space fragmentation in the critical memory region, and develops a software-based concatenated-CRC/BCH memory error tolerance design solution to re-cycle all the error-prone pages for zRAM function in Linux. We evaluated the latency and hardware cost of implementing the controller-based page re-mapping under different error-prone page rates. By integrating the developed software-based memory error tolerance into zRAM in Linux 3.14, we carried out experiments to evaluate the impact on zRAM operational throughput. The results show our proposed software-based error tolerance scheme only incurs about 7% zRAM speed performance degradation. Moreover, since the remaining non-critical memory region is still subject to physical memory space fragmentation caused by error-prone page reservation, we carried out experiments using SPEC CPU2006 under a variety of error-pone page rates. The results show that even highly fragmented non-critical memory region (e.g., 1 page is reserved every 16 pages) may not cause significant computing system performance degradation.

6.2 Future Work

Both data dependent fault tolerance, weak cell aware ECC and software error tolerance solution all reduce storage redundancy and provide high-reliable data integrity. Future work should focus on how to further utilize data characteristic in high performance, mobile computing system and IMDB to optimize memory design for error-prone memory. So for the future work, some possible research directions should include:

- Weak cell aware decoding needs extra ECC redundancy to recover data from error-prone memory which costs about 11% memory storage. However, in mobile device there is no ECC memory interface which complicates implementation of WECAS. So next work will focus on how to utilize weak-cell information in mobile device system to reduce power consumption and hardware cost.
- Relational and key-value databases have different data structure and query characteristic especially when they are implemented as IMDB which develops rapidly. It becomes more challenging to meet both the latency and reliability in fine-grain accessible memory system. Thus future work also focus on cross-layer co-design techniques of database system in error-prone memory.
- The progressive storage system design also includes data compression technique which reduces storage overhead remarkably. Different compression approach has distinct requirement of data protection. This is another perspective we can look into for more specific error tolerance design.

System-aided DRAM scaling certainly faces many open research issues, and it is our hope the exploratory study in this thesis will lead to much more thorough and cohesive investigations from the OS, system and circuit/device research community on this possible option for future DRAM scaling.

References

[1] B. Jacob, S. Ng and D. Wang, "DRAM device organization: Basic circuits and architecture," in *Memory Systems: Cache, DRAM, Disk*, Burlington, MA, USA: Morgan Kaufmann, 2010, ch. 8, pp. 333-335.

[2] J. Liu, B. Jaiyen, R. Veras and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. 39th Annu. Int. Symp. Comput. Architecture (ISCA)*, Portland, OR, USA, 2012, pp. 1-12.

[3] H. M. C. Consortium, "Hybrid memory cube specification 2.1," Beaverton, OR, USA, 2014. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads, Accessed on May 15, 2017.

[4] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *IEEE Symp. on VLSI Technol. (VLSIT)*, Honolulu, HI, USA, 2012, pp. 87-88.

[5] E. Kultursay, M. Kandemir, A. Sivasubramaniam and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *IEEE Int. Symp. Performance Anal. Syst. Software (ISPASS)*, Austin, TX, USA, 2013, pp. 256-267.

[6] Y. Chen, X. Wang, H. Li, H. Xi, Y. Yan and W. Zhu, "Design margin exploration of spin-transfer torque RAM (STT-RAM) in scaled technologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 12, pp. 1724-1734, Dec. 2010.

[7] M. Hosomi *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *IEEE Int. Electron Devices Meeting (IEDM) Tech. Dig.*, Washington, DC, USA, Dec. 2005, pp. 459-462.

[8] T. Kawahara *et al.*, "2 Mb SPRAM (spin-transfer torque RAM) with bit-by-bit bi-directional current write and parallelizing-direction current read," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 109-120, Jan. 2008.

[9] E. Kitagawa *et al.*, "Impact of ultra low power and fast write operation of advanced perpendicular MTJ on power reduction for high-performance mobile CPU," in *IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2012, pp. 29.4.1-29.4.4.

[10] J. Kim, K. Ryu, S. H. Kang and S. Jung, "A novel sensing circuit for deep submicron spin transfer torque MRAM (STT-MRAM)," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 181-186, Jan. 2012.

[11] E. Chen *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Trans. Magn.*, vol. 46, no. 6, pp. 1873-1878, May 2010.

[12] Y. Zhang, W. Wen and Y. Chen, "The prospect of STT-RAM scaling from readability perspective," *IEEE Trans. Magn.*, vol. 48, no. 11, pp. 3035-3038, Nov. 2012.

[13] H. Wang, K. Zhao, J. Li and T. Zhang, "Optimizing the use of STT-RAM in SSDs through data-dependent error tolerance," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2743-2747, Nov. 2015.

[14] K. Prall, "Scaling non-volatile memory below 30nm," in *IEEE 22nd Non-Volatile Semiconductor Memory Workshop*, Monterey, CA, USA, 2007, pp. 5-10.

[15] K. Kim, "Future silicon technology," in *Proc. Eur. Solid-State Device Research Conf. (ESSDERC)*, Bordeaux, France, 2012, pp. 1-6.

[16] H. Wang, K. Zhao and T. Zhang, "Efficiently realizing weak cell aware DRAM error tolerance for sub-20nm technology nodes," in *7th IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2015, pp. 1-4.

[17] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled DRAMs," *IEEE Electron Device Lett.*, vol. 30, no. 8, pp. 846-848, Jul. 2009.

[18] 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 *Proc. 40th Annu. Int. Symp. Comp. Architecture (ISCA)*, Tel-Aviv, Israel, 2013, pp. 60-71.

[19] S. Baek, S. Cho and R. Melhem, "Refresh now and then," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3114-3126, Dec. 2014.

[20] X. Wang, D. Vasudevan and H. Lee, "Global built-in self-repair for 3D memories with redundancy sharing and parallel testing," in *IEEE Int. 3D Syst. Integr. Conf. (3DIC)*, Osaka, Japan, 2012, pp. 1-8.

[21] L. Jiang, R. Ye and Q. Xu, "Yield enhancement for 3D-stacked memory by redundancy sharing across dies," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, USA, 2010, pp. 230-234.

[22] N. Axelos, K. Pekmestzi and D. Gizopoulos, "Efficient memory repair using cache-based redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 12, pp. 2278-2288, Nov. 2012.

[23] A. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis and N. Jouppi, "LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems," in *Proc. 39th Annu. Int. Symp. Comp. Architecture (ISCA)*, Portland, OR, USA, 2012, pp. 285-296.

[24] D. Yoon and M. Erez, "Virtualized ECC: Flexible reliability in main memory," *IEEE Micro*, vol. 31, no. 1, pp. 11-19, Jan. 2011.

[25] X. Jian and R. Kumar, "ECC parity: a technique for efficient memory error resilience for multi-channel memory systems," in *Proc. Int. Conf. High Performance Comput., Netw., Storage, Anal.*, New Orleans, LA, USA, 2014, pp. 1035-1046.

[26] P. Nair, D. Kim, and M. Qureshi, "Archshield: architectural framework for assisting DRAM scaling by tolerating high error rates," in *Proc. 40th Annu. Int. Symp. Comp. Architecture (ISCA)*, Tel-Aviv, Israel, 2013, pp. 72-83.

[27] H. Wang, K. Zhao, M. Lv, X. Zhang, H. Sun and T. Zhang, "Improving 3D DRAM fault tolerance through weak cell aware error correction," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 820-833, Oct. 2016.

[28] A. Driskill-Smith *et al.*, "Latest advances and roadmap for in-plane and perpendicular STT-RAM," in *IEEE Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2011, pp. 1-3.

[29] J. Kim, K. Ryu, J. Kim, S. Kang and S. Jung, "STT-MRAM sensing circuit with self-body biasing in deep submicron technologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 7, pp. 1630-1634, Jul. 2014.

[30] Y. Zhang, I. Bayram, Y. Wang, H. Li and Y. Chen, "ADAMS: asymmetric differential STT-RAM cell structure for reliable and high-performance applications," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, USA, 2013, pp. 9-16.

[31] W. Kang, L. Zhang, J. Klein, Y. Zhang, D. Ravelosona and W. Zhao, "Reconfigurable codesign of STT-MRAM under process variations in deeply scaled technology," *IEEE Trans. Electron Devices*, vol. 62, no. 6, pp. 1769-1777, Mar. 2015.

[32] K. Huang, N. Ning and Y. Lian, "Optimization scheme to minimize reference resistance distribution of spin-transfer-torque MRAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1179-1182, Jul. 2014.

[33] W. Kang *et al.*, "Variation-tolerant high-reliability sensing scheme for deep submicrometer STT-MRAM," *IEEE Trans. Magn.*, vol. 50, no. 11, pp. 1-4, Dec. 2014.

[34] J. Kim, T. Na, J. Kim, S. Kang and S. Jung, "A split-path sensing circuit for spin torque transfer MRAM," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 61, no. 3, pp. 193-197, Jan. 2014.

[35] Y. Zhang, X. Wang, Y. Li, A. Jones and Y. Chen, "Asymmetry of MTJ switching and its implication to STT-RAM designs," in *Proc. Design, Autom. Test Europe (DATE) Conf. and Exhibition*, Dresden, Germany, 2012, pp. 1313-1318.

[36] C. Chen, S. Wang and C. Wu, "Write current self-configuration scheme for MRAM yield improvement," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 7, pp. 1260-1270, Jul. 2013.

[37] K. Kwon, S. Choday, Y. Kim, and K. Roy, "Aware (asymmetric write architecture with redundant blocks): A high write speed STT-MRAM cache architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 712-720, May 2014.

[38] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi and M. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *IEEE 17th Int. Symp. High Performance Comput. Architecture (HPCA)*, San Antonio, TX, USA, 2011, pp. 50-61.

[39] W. Xu, H. Sun, X. Wang, Y. Chen and T. Zhang, "Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM)," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 3, pp. 483-493, Dec. 2011.

[40] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. IEEE 44th Annu. Int. Symp. Microarchitecture (MICRO)*, Porto Alegre, Brazil, 2011, pp. 329-338.

[41] W. Xu, Y. Chen, X. Wang and T. Zhang, "Improving STT MRAM storage density through smaller-than-worst-case transistor sizing," in *Proc. 46th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2009, pp. 87-90.

[42] J. Ahn, S. Yoo, and K. Choi, "Selectively protecting error-correcting code for area-efficient and reliable STT-RAM caches," in *IEEE 18th Asia and South Pacific Design Autom. Conf. (ASP-DAC)*, Yokohama, Japan, 2013, pp. 285-290.

[43] B. Del Bel, J. Kim, C. Kim and S. Sapatnekar, "Improving STT-MRAM density through multibit error correction," in *Proc. Design, Autom. Test Europe (DATE) Conf. and Exhibition*, Dresden, Germany, 2014, pp. 1-6.

[44] W. Wen *et al.*, "CD-ECC: content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, USA, 2013, pp. 1-8.

[45] B. Fitzpatrick, "Distributed caching with memcached," *Linux Journal*, vol. 2004, no. 124, p. 5, Aug. 2004.

[46] J. Ousterhout *et al.*, "The case for RAMClouds: scalable high-performance storage entirely in DRAM," *ACM SIGOPS Operating Syst. Review*, vol. 43, no. 4, pp. 92-105, Jan. 2010.

[47] T. Hamamoto, S. Sugiura and S. Sawada, "On the retention time distribution of dynamic random access memory (DRAM)," *IEEE Trans. Electron Devices*, vol. 45, no. 6, pp. 1300-1309, Jun. 1998.

[48] X. Wang, Y. Chen, H. Li, D. Dimitrov and H. Liu, "Spin torque random access memory down to 22nm technology," *IEEE Trans. Magn.*, vol. 44, no. 11, pp. 2479-2482, Dec. 2008.

[49] T. Maffitt *et al.*, "Design considerations for MRAM," *IBM J. Research and Development*, vol. 50, no. 1, pp. 25-39, Jan. 2006.

[50] J. Kim, K. Ryu, J. Kim, S. Kang and S. Jung, "STT-MRAM sensing circuit with self-body biasing in deep submicron technologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 7, pp. 1630-1634, Jul. 2014.

[51] W. Liu, J. Rho and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," in *IEEE Workshop Signal Process. Syst. Design Implementation*, Banff, Canada, 2006, pp. 303-308.

[52] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Comput. Architecture News*, vol. 23, no. 1, pp. 20-24, Mar. 1995.

[53] G. Loh, "3D-stacked memory architectures for multi-core processors," in *Proc. 35th Int. Symp. Comput. Architecture (ISCA)*, vol. 36, no. 3, 2008, pp. 453-464.

[54] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *IEEE Hot Chips 23rd Symp. (HCS)*, Stanford, CA, USA, 2011, pp. 1-24.

[55] D. U. Lee *et al.*, "25.2 A 1.2 V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *IEEE Int. Solid-State Circuits Conf. Dig. of Tech. Papers (ISSCC)*, San Francisco, CA, USA, 2014, pp. 432-433.

[56] S. Schechter, G. Loh, K. Strauss and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. 37th Annu. Int. Symp. Comput. Architecture (ISCA)*, Saint-Malo, France, 2010, pp. 141-152.

[57] Y. Son, S. Lee, O. Seongil, S. Kwon, N. Kim and J. Ahn "CiDRA: A cache-inspired DRAM resilience architecture," in *IEEE 21st Int. Symp. High Performance Comput. Architecture (HPCA)*, Burlingame, CA, USA, 2015, pp. 502-513.

[58] M. Lv, H. Sun, Q. Ren, B. Yu, J. Xin and N. Zheng, "Logic-DRAM co-design to exploit the efficient repair technique for stacked DRAM," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 5, pp. 1362-1371, Apr. 2015.

[59] M. Lankhorst, B. Ketelaars and R. Wolters, "Low-cost and nanoscale non-volatile memory concept for future silicon chips," *Nature Materials*, vol. 4, no. 4, pp. 347-352, Mar. 2005.

[60] B. Lee, E. Ipek, O. Mutlu and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. 36th Annu. Int. Symp. Comput. Architecture (ISCA)*, Austin, TX, USA, 2009, pp. 2-13.

[61] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson and O. Mutlu, "The efficacy of error mitigation techniques for DRAM retention failures: a comparative experimental study," in *ACM Int. Conf. Measurement and Modeling of Comput. Syst.*, Austin, TX, USA, 2014, pp. 519-532.

[62] A. Hwang, I. Stefanovici and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design," in *Proc. 17th Int. Conf. on Architectural Support for Programming Languages and Operating Syst. (ASPLOS)*, London, United Kingdom, 2012, pp. 111-122.

[63] A. Patel, F. Afram, S. Chen and K. Ghose, "MARSS: A full system simulator for x86 CPUs," in *Proc. 48th Design Automation Conf. (DAC)*, San Diego, California, USA, 2011, pp. 1050-1055.

[64] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Architecture News*, vol. 34, no. 4, pp. 1-17, Sep. 2006.

[65] C.L. Chen and M. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM J. Research and Development*, vol. 28, no. 2, pp. 124-134, Mar. 1984.

[66] D. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 397-408, Dec. 2010.

[67] X. Liu and Q. Hu, "10Gb/s orthogonally concatenated BCH encoder for fiber communications," in *2nd IEEE Int. Symp. on Instrumentation and Measurement, Sensor Netw. and Autom. (IMSNA)*, Toronto, Canada, 2013, pp. 1018-1021.

[68] E. R. Berlekamp, "Binary BCH codes for correcting multiple errors," in *Algebraic Coding Theory*, revised ed. Singapore, Singapore: World Scientific, 2015, ch. 7, pp. 195-199.

[69] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory*, vol. 15, no. 1, pp. 122-127, Jan. 1969.

[70] D. Gross, J. Shortie, J. Thompson and C. Harris, "Simple markovian queueing models," in *Fundamentals of Queueing Theory*. Hoboken, NJ, USA: Wiley, 2013, ch. 2, pp. 73-81.

[71] T. Zhang and K. K. Parhi, "On the high-speed VLSI implementation of errors-and-erasures correcting Reed-Solomon decoders," in *Proc. 12th ACM Great Lakes Symp. on VLSI*, Banff, Canada, 2002, pp. 89-93.

[72] P. Rosenfeld, E. Cooper-Balis and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Architecture Lett.*, vol. 10, no. 1, pp. 16-19, Jan. 2011.

[73] J. Kim, M. Sullivan and M. Erez, "Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory," in *Proc. IEEE 21st Int. Symp. on High Performance Comput. Architecture (HPCA)*, New Orleans, LA, USA, 2015, pp. 101-112.

[74] J. D. Leidel and Y. Chen, "HMC-SIM: A simulation framework for hybrid memory cube devices," *IEEE Int. Parallel Distributed Process. Symp. Workshops*, Washington, DC, USA, 2014, pp. 1465-1474.

[75] P. Rosenfeld, "Performance exploration of the hybrid memory cube," Ph.D. dissertation, Elect. Eng., Univ. of Maryland, College Park, 2014.

[76] P. Meaney *et al.*, "IBM zEnterprise redundant array of independent memory subsystem," *IBM J. Research and Development*, vol. 56, no. 1.2, pp. 4.1-4.11, Jan. 2012.

[77] K. Chakraborty and P. Mazumder, "Diagnosis, repair, and reconfiguration," in *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Upper Saddle River, NJ, USA: Prentice Hall, 2002, ch. 2, pp. 58-73.

[78] S. H. Lee, "Scaling trends and challenges of advanced memory technology," in *IEEE Int. Symp. VLSI Design, Autom. and Test (VLSI-DAT)*, Hsinchu, Taiwan, 2014, pp. 1.

[79] S. K. Park, "Technology scaling challenge and future prospects of DRAM and NAND flash memory," in *IEEE 7th Int. Memory Workshop (IMW)*, Monterey, CA, USA, 2015, pp. 1-4.

[80] H. Wang and T. Zhang, "An exploratory study on system-aided dram scaling," in *IEEE 6th Int. Memory Workshop (IMW)*, Taipei, Taiwan, 2014, pp. 1-4.

[81] H. Wang, Y. Li, X. Zhang, X. Zhao, H. Sun and T. Zhang, "On the use of DRAM with unrepaired weak cells in computing systems," in *Proc. ACM 2nd Int. Symp. on Memory Syst.*, Washington, DC, USA, 2016, pp. 327-337.

[82] E. Shiu and S. Prakash, "System challenges and hardware requirements for future consumer devices: From wearable to chromebooks and devices in-between," in *IEEE Symp. on VLSI Citcuits*, Kyoto, Japan, 2015, pp. 1-5.

[83] J. Chiang, T. Chiueh and H. Li, "Memory pressure balancing on virtualized servers," in *IEEE 21st Int. Conf. Embedded and Real-Time Comput. Syst. and Applications (RTCSA)*, Hong Kong, China, 2015, pp. 70-79.

[84] J. Chiang, T. Chiueh and H. Li, "Memory reclamation and compression using accurate working set size estimation," in *IEEE 8th Int. Conf. on Cloud Comput. (CLOUD)*, New York, NY, USA, 2015, pp. 187-194.

[85] J. Chiang, "Optimization techniques for memory virtualization-based resource management," Ph.D. dissertation, Comput. Sci., Stony Brook University, Stony Brook, NY, USA, 2012.

[86] C. Lee *et al.*, "Compressed and shared swap to extend available memory in virtualized consumer electronics," *IEEE Trans. Consumer Electron.*, vol. 60, no. 4, pp. 628-635, Nov. 2014.

[87] C. Lee, C. Hong, S. Yoo and C. Yoo, "Managing GPU buffers for caching more apps in mobile systems," in *Proc. 12th Int. Conf. on Embedded Software*, Amsterdam, Netherlands, 2015, pp. 207-216.

[88] H. Park, J. Choi, D. Lee and S. Noh, "iBuddy: Inverse buddy for enhancing memory allocation/deallocation performanceon multi-core systems," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 720-732, Jan. 2015.

[89] R. Love, "Memory management," in *Linux Kernel Development*, 2nd ed. Newbury, United Kingdom: Novell Press, 2005, ch. 11, pp. 181-209.

[90] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337-343, May 1977.

[91] G. Castagnoli, S. Brauer and M. Herrmann, "Optimization of cyclic redundancy-check codes with 24 and 32 parity bits," *IEEE Trans. Commun.*, vol. 41, no. 6, pp. 883-892, Jun. 1993.

[92] P. Koopman, "32-bit cyclic redundancy codes for internet applications," in *Proc. Int. Conf. Dependable Syst. and Netw.*, Washington, DC, USA, 2002, pp. 459-468.

[93] V. Gopal, J. Guilford and E. Ozturk, "Fast CRC computation for iSCSI polynomial using CRC32 instruction," Santa Clara, CA, USA, Apr. 2011. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers, Accessed on May 15, 2017.

[94] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Inf. and Control*, vol. 3, no. 1, pp. 68-79, Mar. 1960.

[95] H. Chen, A. Arunkumar, C. Wu, T. Mudge and C. Chakrabarti, "E-ECC: Low power erasure and error correction schemes for increasing reliability of commodity dram systems," in *Proc. 1st Int. Symp. Memory Syst.*, Washington, DC, USA, 2015, pp. 60-70.

[96] S. Lin and D. J. Costello, "Binary BCH codes," in *Error Control Coding: Fundamentals and Applications*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004, ch. 6, pp. 194-231.