Designing Low Cost Error Correction Schemes for Improving Memory Reliability

by

Hsing-Min Chen

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2017 by the
Graduate Supervisory Committee:

Chaitali Chakrabarti, Chair
Trevor Mudge
Carole-Jean Wu
Umit Ogras

ARIZONA STATE UNIVERSITY

May 2017

ABSTRACT

Memory systems are becoming increasingly error-prone, and thus guaranteeing their reliability is a major challenge. In this dissertation, new techniques to improve the reliability of both 2D and 3D dynamic random access memory (DRAM) systems are presented. The proposed schemes have higher reliability than current systems but with lower power, better performance and lower hardware cost.

First, a low overhead solution that improves the reliability of commodity DRAM systems with no change in the existing memory architecture is presented. Specifically, five erasure and error correction (E-ECC) schemes are proposed that provide at least Chipkill-Correct protection for x4 (Schemes 1, 2 and 3), x8 (Scheme 4) and x16 (Scheme 5) DRAM systems. All schemes have superior error correction performance due to the use of strong symbol-based codes. In addition, the use of erasure codes extends the lifetime of the 2D DRAM systems.

Next, two error correction schemes are presented for 3D DRAM memory systems. The first scheme is a rate-adaptive, two-tiered error correction scheme (RATT-ECC) that provides strong reliability (10^{10}x reduction in raw FIT rate) for an HBM-like 3D DRAM system that services CPU applications. The rate-adaptive feature of RATT-ECC enables permanent bank failures to be handled through sparing. It can also be used to significantly reduce the refresh power consumption without decreasing the reliability and timing performance.

The second scheme is a two-tiered error correction scheme (Config-ECC) that supports different sized accesses in GPU applications with strong reliability. It addresses the mismatch between data access size and fixed sized ECC scheme by designing a product code based flexible scheme. Config-ECC is built around a core unit designed for 32B access with a simple extension to support 64B and 128B accesses. Compared to fixed 32B and 64B ECC schemes, Config-ECC reduces the failure in time (FIT) rate by 200x and 20x, respectively. It also reduces the memory energy by 17% (in the dynamic mode) and 21% (in the static mode) compared to a state-of-the-art fixed 64B ECC scheme.

DEDICATION

To my parents

ACKNOWLEDGMENTS

I wish to express my gratitude to my advisor and mentor, Dr. Chaitali Chakrabarti, who has patiently taught me to do research, speak and write. I cannot remember how many weekends we have worked together to discuss my research. I am deeply indebted for her guidance, encouragement and support throughout my graduate study.

I wish to thank Dr. Trevor Mudge, Dr. Carole-Jean Wu and Dr. Umit Ogras for serving on my thesis committee, and for their insightful comments in improving my research. Dr. Mudge brought me into the DARPA project. I do appreciate his feedback during our weekly meetings. I would also like to thank Dr. Wu for her critical comments. Her tough words and critical thinking always pushed me to improve my work.

I would like to thank all my collaborating researchers, Supreet Jeloka, Nilmini Abeyratne and Byoungchan Oh, at the University of Michigan as well as Akhil Arunkumar and Shin-Ying Lee at Arizona State University. I am also grateful to DARPA, the National Science Foundation and the School of Electrical, Computer and Energy Engineering at ASU for funding my graduate study.

I wish to thank my colleagues in the Low Power Systems Lab, Chengen Yang, Lifeng Miao, Ming Yang, Siyuan Wei, Madhu Vasudevan, Manqing Mao, Asha Sasikumar, Jian Zhou, Vamsi Chagari Reddy, Jaideep Gour, Jiang Xiang and Shunyao Wu. I have learnt a lot from them. It is because of them that the Low Power Systems Lab was a great place to work in. I also want to thank my girl friend, Rui Liu, for her endless support and care.

I wish to thank all my friends and roommates in Arizona. Without them I would not have such colorful memories to cherish. Without them I may not have survived the PhD program!

Finally and most importantly, I would like to thank my parents. I am grateful to them for fostering my interests in science and engineering. Their unconditional love and sustained support has been and will always be the force with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Motivation

In large scale computing systems, guaranteeing the reliability of its memory system is a major challenge. More than 40% of hardware related failures are attributed to errors in memory systems [1]. These errors cause computing systems to malfunction and also lead to serious security vulnerabilities [2]. To make it worse, due to the scaling down and the high density of memory cells, the number of errors is projected to increase in the future.

In this dissertation, we propose schemes to improve the reliability of both 2D dynamic random access memory (DRAM) and 3D DRAM systems. 2D DRAM system is used as main memory in most computer systems. Unfortunately, these systems are vulnerable to different kinds of faults (e.g. hard, intermittent or random) [3,4]. These faults manifest as single bit errors, multiple errors along a row and/or along a column of a chip and even whole chip failure. Error correction coding (ECC) [5] schemes are typically used to deal with DRAM errors. Designing an ECC code with low storage overhead and good error correction performance is quite a challenge.

2D DRAM has the advantages of high density, high capacity and low cost but its performance is limited by memory latency and bandwidth. 3D die-stacked DRAM is a promising solution to improve the system performance [6] and reduce the power consumption. It can provide higher bandwidth, lower power consumption, lower latency and has smaller footprint compared to 2D DRAM. There are several prototypes for 3D DRAM, including high bandwidth memory (HBM) [7], hybrid memory cube

(HMC) [8] and Octopus from Tezzaron [9]. However, 3D DRAM suffers from serious reliability issues due to higher density and thermal issues. Furthermore, since a single data line is not striped across multiple chips like 2D DRAM, it increases the difficulty of designing a strong ECC scheme. Also, through-silicon-via (TSV) failures occur only in the 3D DRAM structure and add to the complexity. In the rest of this chapter, we briefly describe the proposed low cost, low overhead and strong ECC schemes for 2D and 3D DRAM memory systems that improve their reliability without degrading their performance.

1.2 2D DRAM Reliability

Most computer systems use dynamic random access memory (DRAM) as main memory. As the size of DRAM in computer systems increases, DRAM reliability becomes a serious concern. The traditional single bit error correction and double bit error detection (SEC-DED) code is no longer strong enough to guarantee the memory reliability.

High performance servers are typically expected to have Chipkill-Correct level protection, that is the ability to correct errors due to failures of a DRAM chip with 12.5% storage overhead [10, 11]. Chipkill-Correct was first implemented by striping data across multiple chips so that a single error correction and double error detection (SEC-DED) code could be used to correct errors due to a chip failure [10]. The bit-level Chipkill-Correct code had high power consumption and low system performance and so symbol-based Chipkill-Correct was proposed for x4 DRAM systems in [12, 13]. This scheme activated 36 chips across two ranks for every memory access and thus also had high power consumption and lower timing performance.

To reduce the power consumption, many systems moved to x8 or x16 DRAMs, which activate fewer chips per memory access. For instance, V-ECC [14] activates

18 x8 chips across two ranks while LOT-ECC [15] and Multi-ECC [16] only activate nine chips per x8 rank. Although LOT-ECC and Multi-ECC can reduce the memory power by an average of more than 25% compared to Chipkill-Correct, they cannot fully correct a chip failure at run time.

In order to activate fewer chips and maintain high reliability, many systems employ two-tier schemes [14–18], where the first tier is used for error detection and the second tier is used for error correction. For example, V-ECC [14] uses two check symbols in the first tier to perform error detection and the third check symbol in the second tier to perform error correction. The second tier is usually cached to reduce the read latency for error correction and the write frequency for ECC updates.

Existing schemes such as those in [15, 16] also rely on replacement of faulty chips to extend the lifetime of the DRAM memory systems. Some commercial systems use memory sparing or bit-steering [16, 19, 20] to re-route the faulty bits or re-route data from a faulty chip to a healthy chip. This not only reduces the usable physical memory size but also increases the overhead required to perform re-routing or mapping.

In this work we propose a very different approach to providing at least Chipkill-Correct error protection for commodity DRAM systems. Our approach is based on the use of stronger symbol based codes which are chosen to handle the constraints of different memory systems. Use of a stronger code adds very little overhead to existing systems. Our synthesis results in 28nm node show that the decoding latencies of these codes are very small and do not affect the DRAM timing performance. Moreover, unlike the existing multi-tiered schemes, our schemes do not require extra memory read/write operations to access data or to do error correction.

Furthermore, instead of employing chip sparing to increase the lifetime of memory systems, we make use of erasure correction, where an erasure is defined as an error whose location is known. We utilize the machine check architecture (MCA) [20, 21] to record the error information of each chip. Once the number of errors in a certain chip increases beyond a threshold, this chip is marked as faulty and errors due to this chip are treated as erasures.

We present five erasure and error correction (E-ECC) schemes that provide at least Chipkill-Correct reliability for x4, x8 and x16 DRAM systems. All the proposed E-ECC schemes can correct errors due to a chip failure on the fly and can correct one additional random error when the chip is marked as faulty. In order to demonstrate that our proposed schemes have low cost and low overhead, we synthesize all decoding units using industrial 28nm library to obtain the corresponding latency, area and power results. In addition, we analyze the tradeoffs between reliability and system performance (timing, power and energy) of these proposed schemes and compare them to the existing competing schemes.

Overall, we make the following key contributions in the design of reliable 2D DRAM systems:

- For x4 DRAM systems, we present three schemes that all have 12.5% storage overhead but differ in the number of ranks being activated (one or two). The specific ECC codes used in these three schemes are rotational (144,128) code over GF(2^4) and RS (36,32) code over GF(2^8). The schemes that activate two ranks per memory access have lower timing, power and energy performance but higher reliability compared to the one that activates only one rank.

- For x8 DRAM systems, we propose a scheme which is also based on the RS (36,32) code over GF(2^8). It accesses two ranks per memory access and has the lowest power consumption and highest energy efficiency among all five schemes.

- For x16 DRAM systems, we propose a scheme which is based on the RS (20,16) code over GF(2^8). This scheme activates two ranks per memory access, has storage overhead of 25% but has the highest timing performance among all the schemes.

This work appeared in ACM MEMSYS Conference [22] and IEEE Transactions on Computers [23].

1.3 3D DRAM Reliability

The 3D die-stacked DRAM is an excellent candidate for high performance computer systems. It has lower access latency, lower power consumption and higher bandwidth [6, 24]. In this work, we consider an HBM-like structure where multiple dies are stacked on top of the logic die; stacking is achieved by the use of through silicon vias (TSV).

Unfortunately, 3D DRAM is likely to be less reliable than 2D DRAM because of additional errors due to its higher integration density. The increase of errors in 2D DRAM due to higher density has been well documented in [3, 4, 25, 26]. The 2D DRAM errors that were due to transient and permanent single bit, column, row and bank failures are expected to be present in 3D DRAM as well. In addition, 3D DRAM will incur errors caused by TSV failures. Thus, designing a reliable memory system using an error-prone 3D die-stacked DRAM is quite a challenge.

Added to this challenge is the fact that in a 3D DRAM (such as HBM), data block from a lower level cache is stored in a single bank and not striped across multiple banks as in 2D DRAM. While this results in lower power and higher performance as shown in [27, 28], it makes the design of a reliable system even more challenging. For instance, if a bank fails, an entire cache line is corrupted. Traditional ECC schemes for 2D DRAMs such as Chipkill-Correct [11], virtualized-ECC [14], LOT-ECC [15] and Multi-ECC [16] were all based on data being striped across different banks. Hence, these schemes cannot be extended to handle 3D DRAM.

In recent years, several schemes have been proposed for improving the reliability of 3D DRAM. These include Subarraykill [29], Resilient-DRAM-Caches [30], Efficient-RAS [28], Citadel [27] and Parity-Helix [31]. Subarraykill [29] proposes an ECC scheme for the specialized Tezzaron Octopus structure, which allows for the data to be striped across several subarrays. The ECC design in [30] focuses on the protection of the last level cache; the single bit error correction code and small size of cyclic redundancy code (CRC) [5] are not strong enough for 3D DRAM memory. Citadel [27] and Efficient-RAS [28] are two contemporary systems that have been designed for HBM-like memory. Both use two tiered schemes, where the first tier is used to only detect [27], or detect as well as correct [28], and the second tier is used to correct errors detected but not corrected by the first tier. Efficient-RAS [28] uses symbol based rotational code to correct errors due to small granularity faults while Citadel [27] uses CRC-32 to detect errors and relies on tier-2 code to correct all errors. Since small granularity faults account for 70% of all faults [26], this can affect the timing performance. Parity-Helix [31] focuses on the tier-2 code design where the ECC parity bits are twisted so that the system can recover from errors due to a single dimensional failure in a multi-dimensional memory system.

1.3.1 ECC For 3D DRAM In CPU Systems

In this work, we propose rate-adaptive two-tiered error correction scheme (RATT-ECC) to deal with errors due to transient and permanent 3D failures in an HBM-like 3D DRAM system that services CPUs. Here, the memory is organized into 8 data dies and one additional die is used to store the error correction (ECC) bits. CPU systems typically have data access size of 64B and so the proposed ECC scheme is optimized for 64B access.

In RATT-ECC, the tier-1 code is based on a Reed-Solomon (RS) code [5] to provide strong error detection and correction capability. The strong detection capability reduces error leaks and the strong error correction capability decreases the frequency of tier-2 activation. RATT-ECC can correct errors due to all small granularity faults such as single bit, column or TSV failures by using tier-1 code. It needs tier-2 code only for correcting errors due to large granularity faults such as row or bank failures. RATT-ECC uses RS (70,64) as the tier-1 code. This code can be used to correct one symbol error and detect five symbol errors. Since the undetected error probability of tier-1 code is as low as $2.3 \cdot 10^{-10}$, almost all errors can be corrected by the tier-2 code. The tier-1 code can also be used as an erasure code and errors due to permanent TSV failure can be easily corrected.

Furthermore, RATT-ECC has a mechanism to free up banks in the ECC die to handle faulty data banks. It alters the rate of tier-1 and tier-2 codes so that these codes need less parity storage allowing for 1 or 2 spare banks. Specifically, by puncturing RS (70,64) code to RS (69,64) code, it frees up one ECC bank since RS (69,64) requires five ECC banks to store its parity symbols. A second ECC bank can be freed up by XORing the tier-2 parity symbols that were originally stored in two ECC banks. Thus RATT-ECC can provide for a highly reliable 3D DRAM system that is plagued with all types of faults.

Overall, we make the following key contributions in the design of a reliable 3D DRAM system that services CPUs:

- Design of a rate-adaptive two-tiered ECC scheme with a total storage overhead of 12.5% to correct errors due to small and large granularity faults. The strong correction capability of tier-1 code reduces the number of activations of tier-2 code and the strong detection capability of tier-1 code avoids substantial error leaks. Overall, the raw failure-in-time (FIT) rate is reduced by more than 10^{10}x.

- The rate adaptive feature of tier-1 code and tier-2 code allows for up to two spare banks in the ECC die. Thus the memory system can reliably function even after two data banks are marked faulty with very little overhead.

- The erasure correction capability of the tier-1 code is utilized to handle permanent TSV failures with no performance overhead.

- The synthesis results in 28nm technology show that the proposed tier-1 code has very small overhead in area, power and latency. Simulations on SPEC2006 benchmark shows that the timing performance drops by less than 2% compared to the baseline scheme with no ECC.

This work appeared in ACM Transactions on Architecture and Code Optimization [32].

1.3.2 ECC For 3D DRAM In GPU Systems

Graphics processing units (GPUs) play an increasingly significant role in high performance computers. 3D DRAMs, e.g., high bandwidth memory (HBM) [33], are projected to be widely adopted in GPU memory systems in the near future. In fact, 3D DRAMs have already been applied in some commercial GPU products, such as Nvidia Tesla P100 [34] and AMD Radeon R9 FURY X GPU cards [35]. In this work, we assume that 3D DRAM is organized into 8 dies, where each die holds both data bits and ECC bits.

Designing an ECC scheme for GPU memory systems is more challenging than for CPUs. The data fetch granularity from GPUs can vary and so designing an ECC scheme for fixed sized data access is sub-optimal. Current GPU products support different cache line sizes. For example, AMD GPUs, such as the RADEON series, adopt 64B as the cache line size [36], whereas Nvidia uses 128B L2 cache line size in many of their GPU products. In some GPU products, the L2 cache line size can be statically configured to operate in 2 different modes (32B or 128B) before a GPU kernel is launched [37]. This is proposed with compiler level analysis to fully exploit the varying degree of spatial locality in GPGPU applications. Furthermore, many recent studies [38, 39] have shown that GPGPU applications exhibit mixed locality data access patterns over application execution. Therefore, a lot of designs have been proposed to dynamically predict and accommodate varying cache line sizes for the L1 cache [39] as well as for the L2 cache [38].

Traditionally, ECC algorithms used in DRAMs were optimized for fixed sized data accesses. Examples include CRC-16 for 32B accesses in [33], symbol-based code for 64B accesses in [28, 32]. However, applying an ECC scheme that is optimized for a single data access size leads to suboptimal memory reliability or higher energy consumption for the other data access sizes. For instance, an ECC algorithm optimized for 64B access leads to additional energy consumption if the GPU data fetch size is 32B. Alternatively, an ECC algorithm optimized for 32B data results in lower reliability compared to one that results in been designed for 64B data. Clearly, to better support the diversity of cache configurations in GPUs and the performance feature of the variable data fetch granularity runtime support for future GPUs, there is a strong need to develop an adaptive and flexible ECC scheme.

In this work, we propose Config-ECC, an ECC scheme that not only supports a diverse set of cache hierarchy configurations in different GPU cards in the static mode but is also designed to handle variable data fetch granularities at runtime in the dynamic mode with strong reliability and/or low energy consumption. Config-ECC is based on a two-tiered ECC code: The tier-1 code is a product code [5] that has strong error detection and correction capabilities. It can correct errors due to small granularity faults (single bit, column, and TSV failures) and detect errors with large granularity fault (row or bank failures) with very low silent data corruption rate. The tier-2 code, which is based on an XOR structure and stored in the data banks, is launched for correcting errors due to large granularity faults.

Config-ECC is built around a core 32B ECC scheme that can be easily extended to support 64B and 128B accesses. The product code structure of Config-ECC helps address the mismatch between data access size and fixed size ECC scheme. Basically, the inner code is optimized to have very strong detection capability for 32B access and the combination of inner code and outer code provides for very strong reliability for 64B access. Config-ECC improves the silent data corruption rate (SDC) by 200 times compared to the 32B ECC scheme in the HBM standard and by 20 times compared to the latest 64B ECC schemes [27, 32]. For applications that prefer small cache line size (32B), Config-ECC has significantly lower energy consumption compared to 64B and 128B ECC schemes and also provide stronger reliability than that of a fixed 32B ECC scheme.

Our main contributions are as follows:

- We designed a two-tiered ECC scheme to provide strong error protection for mixed 32B, 64B and 128B accesses in 3D HBM memory systems used in high performance GPUs. These three ECC schemes share the same core structure and can be configured statically and dynamically to support different sized accesses.

- Compared to a fixed 32B ECC scheme, Config-ECC increases the reliability by 200 times for both static and dynamic modes.

- Compared to a fixed 64B ECC scheme, Config-ECC increases the reliability by 20 times when the access size is 64B or larger. Also, Config-ECC can choose to only read 32B of data to reduce the energy by 21% in the static mode and 17% in the dynamic mode.

This work has been submitted to MICRO 2017 [40].

1.4 Dissertation Organization

The rest of the dissertation is organized as follows.

In Chapter 2, basics of 2D and 3D DRAM architecture, error characteristics, existing ECC schemes for 2D and 3D DRAM and the machine check architecture (MCA) are introduced.

Chapter 3 focuses on ECC schemes for 2D DRAM systems. The data access pattern and decoding flowchart for all E-ECC schemes are presented. The tradeoff between system performance and reliability for each scheme is also analyzed.

Chapter 4 focuses on an ECC scheme, RATT-ECC, for an HBM system that services CPU requests. The operation of RATT-ECC is described in detail and the timing performance, reliability and implementation overhead are also analyzed.

Chapter 5 focuses on an ECC scheme, Config-ECC, for an HBM system that supports GPU data access sizes of 32B, 64B and 128B. Config-ECC design is described in detail along with the reliability and energy analysis.

Chapter 6 concludes the thesis.

Chapter 2

BACKGROUND

2.1 DRAM Memory Systems

2.1.1 2D Commodity DRAM Architecture

A DRAM memory system is organized into channels, ranks, chips and banks [41]. The DRAM memory controller (MC) acts as an interface between the last level cache and the DRAM. It can access data from one or more channels. Each channel consists of dual in-line memory modules (DIMMs), each of which consists of one or more ranks. A rank is the minimum unit that is activated in a read or write access. Each rank is composed of multiple chips (also called devices) and the number of chips to be activated depend on the size of the data bus width.

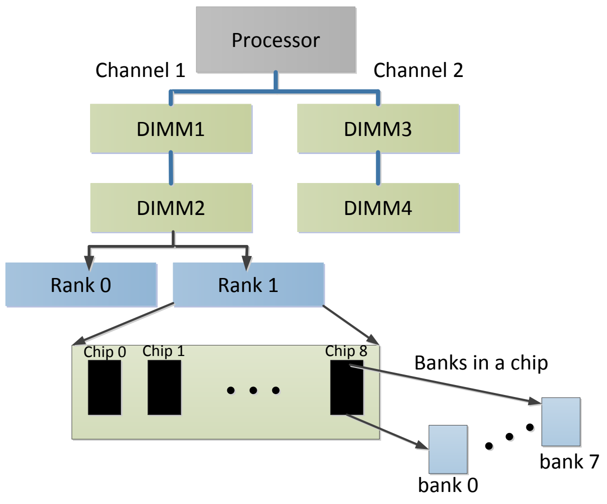


Figure 2.1: Logical View Of A DRAM Based Memory System

For DDR3, the I/O width (N) is typically 4, 8 or 16 bits. Since the 64 bit data path is fixed, a rank consists of 64/N chips. A DRAM system built with sixteen x4 DRAM chips is referred to as a 16x4 system. The common DRAM system configurations are 16 x4, 8 x8 and 4 x16 (number of chips per rank x data I/O width). In an x4 system, there are 8 extra ECC bits for every 64 bits data resulting in two extra ECC chips per rank. In an x8 system, one extra chip is used for ECC chip per rank. The DRAM architecture for a x8 system is shown in Figure 2.1.

In DDR3 systems, data is accessed in the burst mode; typically, burst length is eight or four (chopped burst) [42]. A burst length of eight means that eight beats of data are transferred per memory access [41]. Some DRAM systems operate in the lock-step mode [14, 41]. In such a mode, two physical channels operate as a single logical channel. A single 64B cache line is fetched using two memory channels; one half of the cache line is accessed in the first channel while the second half is obtained from the second channel.

2.1.2 3D DRAM Memory Architecture

There are several prototypes of 3D DRAM, for example, high bandwidth memory (HBM) [7, 33], hybrid memory cube (HMC) [43] and Wide-I/O [44]. In this thesis, we consider a 3D DRAM architecture to be similar to HBM [7, 33]. For each die-stacked HBM, typically, there are eight dies to store data, where each die has eight or sixteen banks. A single stack is anticipated to support up to eight channels. Since each stack supports multiple channels, it provides larger bandwidth than 2D DRAM. Each channel accesses an independent set of DRAM banks. A data line is stored in the same bank in HBM in contrast to a data line being striped across multiple banks in conventional 2D DRAMs. Through-silicon-via (TSV) connects different data dies [45]. Each channel has data width of 128b as the most common setting in [7]; hence, there

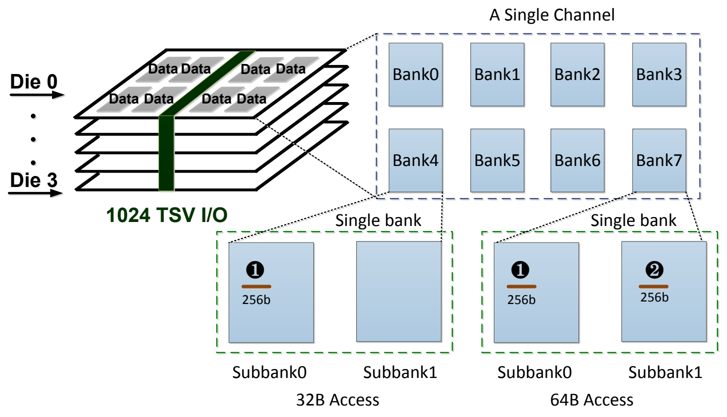


Figure 2.2: 32B vs. 64B Accesses in HBM2 (Pseudo Channel Mode)

are 128 TSV lines per channel. Since there are eight channels, there are a total of 1024 TSV lines per stack.

HBM has two different architectures: HBM1 and HBM2. The access unit is only 32B in HBM1 [7] while the access unit can be 32B or 64B in HBM2 [33]. Each bank in HBM2 consists of two sub-banks and these two sub-banks can be controlled by two sub-channels. The HBM2 architecture is shown in Figure 2.2. The I/O for each channel is 128b with additional 16b for ECC in [33,46], which is different from HBM1. Two modes are supported in HBM2: the legacy mode and the pseudo channel mode. In the legacy mode, in each request, 256b of data is read out from a single bank. Therefore, a 32B DRAM access consists of 2 bursts (1 cycle) with 128b accessed per burst through a single channel. Here, a burst means the rising edge or falling edge of a clock cycle.

To reduce activation power, the pseudo channel mode is proposed in the HBM2 standard [1]. In the pseudo channel mode, a single channel is divided into two sub-channels each of size 64 bits. So a 32B access consists of 4 bursts (2 cycles) with 64b

[1]HBM1 only supports the legacy mode while HBM2 supports both legacy and pseudo channel mode.

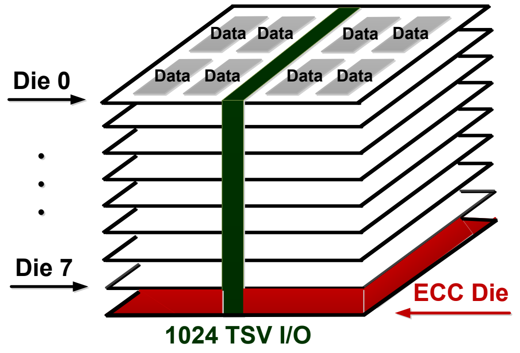


Figure 2.3: 3D HBM-Like Structure With An ECC Die

per access. HBM2 has separate pins for row and column commands and this dual command feature allows HBM2 to simultaneously issue row and column commands to two different banks. Thus, in a 64B access, two sub-channels are activated at the same time and the 64B access consists of 4 bursts (2 cycles).

In Chapter 4, we assume an HBM structure that consists of 8 data dies and an additional ECC die to store the ECC bits as in [27]. Figure 2.3 shows the internal stack organization of such a memory. The parity check symbols for tier-1 and tier-2 code are stored in the ECC die. In each memory request, 512b of data is accessed from a certain bank of a single data die and the corresponding ECC bits are read from a certain bank in the ECC die. The data is read over four beats with 128 bits being read in each beat as shown in Figure 2.4. Eight data bits form a symbol and so a single TSV failure leads to only one symbol error as shown in Figure 2.4 as well.

In Chapter 5, we assume that the memory system that services GPUs uses the latest HBM2 standard [33] and develop ECC schemes for the pseudo channel mode. The ECC bits are stored in the data banks instead of being housed in another die as in [27]. In each access, the ECC bits are read along with the data bits. For 32B and

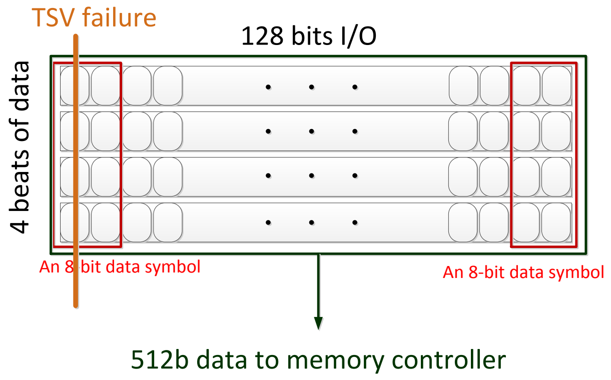


Figure 2.4: 512b or 64 Symbols Are Read From A Bank In A Data Die

64B accesses, the burst length is 4 and considered as one request. For 128 access, we assume that the system issues two 64B requests.

2.2 DRAM Error Characteristics

2.2.1 2D Commodity DRAM Errors

DRAM errors can be broadly classified into soft errors and hard errors. Soft errors are caused by transient faults which occur randomly and cause incorrect data to be read from a memory location; they disappear when the location is overwritten. Hard errors are caused by permanent faults or intermittent faults. A permanent fault causes a memory location to consistently return an incorrect value, such as a stuck-at-0 fault. An intermittent fault causes a memory location to sometimes return incorrect values. Note that a single fault can result in multiple error instances [3, 4].

DRAM faults consist of single-bit, single-word, single column, single row, single-bank, multiple-bank and multiple-rank failures [3]. We use the definition from [3] and present the terminology here:

1. **Single-bit:** All errors map to a single bit.

Table 2.1: DRAM Fault Modes in Hopper

Fault Mode	Total Faults	Transient	Permanent
Single-bit	78.9%	42.1%	36.8%
Single-column	5.9%	0.0%	5.9%
Single-row	9.2%	1.8%	7.4%
Single-bank	4.3%	0.4%	3.9%
Multiple-bank	0.6%	0.0%	0.6%
Multiple-rank	1.0%	0.2%	0.8%
Summary	99.9%	44.5%	55.4%

2. **Single-word:** All errors map to a single word.

3. **Single-column:** All errors map to a single column.

4. **Single-row:** All errors map to a single row.

5. **Single-bank:** All errors map to a single bank.

6. **Multiple-bank:** Errors map to multiple banks.

7. **Multiple-rank:** Errors map to multiple DRAMs in the same lane.

Table 2.1 shows a breakdown of DRAM faults from [26].

DRAM errors have been analyzed in detail in [2–4, 25, 47]. The study in [2, 3, 25] shows that a large fraction of errors are hard errors and these manifest as repeating errors occurring at the same address, row, column or chip. In addition, permanent faults tend to be clustered [25]; these errors have strong correlations in space and time. The repeated errors contaminate the nearby rows and columns and increase dramatically in the presence of prior errors. The study in [2, 25] also shows that the number of errors in any memory system increase over time. A more recent analysis of DRAM faults performed over a period of 15 months shows that while the failure rate due to transient faults increases mildly, the failure rate due to permanent faults is higher in the beginning and becomes almost the same as transient faults around months six to eight [4]. It is projected that the failure rate would again increase towards the end of the device's lifetime.

In general, if a chip has persistent errors, then that chip can be marked as faulty and all data from that chip can be treated as erasures. Note that erasures are defined as errors whose locations are known [5]. Thus if erasures can be corrected, the faulty chip can continue to be used instead of being retired. In this work, we utilize the error recording mechanism of machine-check architecture (MCA) [20, 21] to mark a chip as faulty.

2.2.2 3D Die-Stacked DRAM Errors

For 3D DRAM errors, a better classification is based on the granularity of faults. Small granularity faults are those due to single bit or single column and account for 70% of all faults, and large granularity faults are due to row and bank failures [3,4,26]. 3D DRAM also has errors due to TSV failures [27,28], which fall under small granularity faults. In this work, we analyze the capability of the system to handle errors due to small granularity and large granularity faults (transient and permanent). To handle permanent faults such as TSV faults (small granularity) and bank faults (large granularity), we also utilize the error recoding mechanism of MCA [21].

2.3 Existing ECC Mechanisms

2.3.1 Existing Schemes in 2D DRAM

Chipkill-Correct is the most common error protection scheme for DRAM memory systems [2,10,11,48]. It can correct errors due to failure of one chip and also detect errors due to two chip failures. The original Chipkill-Correct solution from IBM [10] used single bit error correction and double bit error detection code (SEC-DED). Current systems such as Sun UltraSPARC-T1/T2 [49] and AMD Opteron [50] systems use symbol based Chipkill-Correct codes. An example of such a code is the rotational (144,128) code [12], which is a (36,32) code over GF(2^4). In an x4 memory system, this code results in activation of 36 devices across 2 ranks and thus consumes a lot of power. Next we describe several methods that try to achieve a balance between reduction in power consumption and Chipkill-Correct reliability.

Virtualized ECC (V-ECC) [14] provides Chipkill-Correct capability for x4 and x8 DRAM systems. It is based on a 3 check symbol code where 2 check symbols are used for detection (tier-1) and a third check symbol is used for correction (tier-2). In an x8 system, V-ECC activates 18 chips in 2 ranks. It caches the tier-2 symbols to reduce the read latency and write frequency. However, it still incurs extra read/write operations to perform error correction or to update the ECC bits. The storage overhead of V-ECC is 18.75%.

Localized and tiered ECC (LOT-ECC) [15] activates only nine chips per rank in x8 DRAM systems to reduce power consumption. It uses multiple layers of XOR operations to deal with memory errors. Data along with local and global ECC parity bits are stored in the same DRAM row to improve access efficiency. If errors are detected, global parity bits are read by a second access. LOT-ECC is not a Chipkill-Correct solution since it can only correct a stuck-at-0 or stuck-at-1 chip failure. The storage overhead of LOT-ECC is 26.5%, which is higher than the existing schemes.

Multi-line error correction (Multi-ECC) [16] also activates only nine chips per rank in x8 DRAM systems. Multi-ECC uses a different approach where errors are first detected along rows and then column checksums are used to locate the errors. The row parity bits are then used to correct these errors. This method requires a large number of data reads when an error is detected. In addition, since Multi-ECC uses one's complement for column checksums, it cannot fully detect errors due to column failures. The storage overhead of this method is only 12.9%, which is a small increase compared to Chipkill-Correct.

Adaptive Reliability Chipkill Correct (ARCC) [51] also provides two tiers of error protection. It reduces the power consumption by activating only one rank when there are no errors. When errors are detected, ARCC adaptively adjusts the ECC strength by combining adjacent codewords to perform error correction. Once two ranks are combined, the cache line size is increased from 64B to 128B. ARCC does not increase the ECC storage overhead; the only overhead is that the last level cache needs to be modified to accommodate both 64B and 128B cache lines.

Bamboo-ECC [52] is a recently proposed single-tier error protection scheme that provides good system reliability with low storage overhead. It uses an 8-bit symbol based RS code for x4 DRAM systems to handle error events ranging from correcting single bit errors with 3.1% storage overhead to correcting errors due to two chip failures with 25% storage overhead. Furthermore, by grouping per-pin data to form ECC symbols, it is able to correct double pin failures and thus provides higher error protection compared to Chipkill-Correct.

2.3.2 Existing Schemes In 3D DRAM

We give a brief description of prior approaches on improving the reliability of 3D Die-stacked DRAM [27–30]. Of these schemes, Efficient-RAS [28] and Citadel [27] also focus on an HBM-like structure and are closest to our proposed scheme.

Subarraykill [29] corrects errors due to a subarray failure in Tezzaron Octopus structure. A cache line is striped across a subset of subarrays; the number of subsets depends on the page size. Due to this striped pattern of data storage, the ECC schemes based on symbol-based code can be used.

Resilient-DRAM-Caches [30] uses ECC codes to protect tags and cache lines in 3D DRAM caches. Single bit error correction (SEC) code is used to correct single bit errors while cyclic redundancy check codes (CRCs) are used to detect multiple bit errors. To recover multiple bit errors due to row or bank failures in a contaminated cache line, a duplicate dirty cache line that is stored in another DRAM cache bank is utilized. This approach has high storage overhead and is not strong enough to handle errors in 3D DRAM.

Efficient-RAS [28] proposes a two tier error correction scheme with higher than 12.5% storage overhead. The first tier uses a symbol-based ECC code to correct errors due to small granularity faults, while the second tier is a XOR-based correction code (XCC) to correct the detectable but uncorrectable errors due to large granularity faults. It proposes a strategy to deal with permanent TSV (or row) failures by using spare TSVs (or rows). However, it does not have any strategy to handle permanent bank failures.

Citadel [27] also uses a two tier ECC protection to handle small and large granularity faults. For the first tier, Citadel uses CRC-32 to provide strong error detection capability. After errors are detected, Citadel relies on multiple levels of parity (3DP) to recover from errors due to single bit, single column, single TSV faults as well as row and bank failures. In this scheme, the cost of correcting errors is very high. Citadel allocates 64 bits for each cache line; the metadata is located in ECC logic die resulting in a 12.5% additional storage. Additional bits are used to address TSV failures and for data recovery by 3DP. In addition, it provides for two spare banks to handle bank failures.

Parity-Helix [31] focuses on design of the tier-2 code and allows the system to recover from errors due to a single channel failure or a single die failure in a die-stacked DRAM. The ECC core structure is based on RAID-5. The ECC parity blocks in vertical and horizontal directions are twisted to form a helix structure which can avoid a single dimensional fault in a multi-dimensional memory system. The storage overhead is 26.8% for the overall ECC scheme with 14.3% for the tier-2 code.

2.4 Error Logging Architecture

The memory controller not only manages the data access from and to DRAM but also performs error detection and correction. Many server systems utilize machine-check architecture (MCA) [3, 21, 53, 54] to record detected errors in each machine. MCA defines the facilities by which processor and system hardware errors are logged and reported to the system software (operation system) [21]. These detected errors include corrected and uncorrected error events but the MCA does not record the errors due to miss error detection. Once errors are detected, the CPU sends a hardware exception signal called a machine check exception (MCE) which must be handled by the processor.

When an MCE event occurs, the memory controller logs corrected error events in registers provided by the MCA [3,21]. Each machine's operating system is configured to poll the MCA registers once in a certain period (e.g., few seconds) and record any exception event. These console logs contain a variety of information about the physical address associated with the error, the time that the error was corrected, the type of memory access (e.g., read or write), the type of error (corrected or uncorrected) and the ECC syndrome associated with the error.

For 2D DRAM memory systems, if the number of errors is larger than a threshold value (the threshold value is system-dependent), the chip is marked as faulty by the memory controller. The erasure correction scheme is launched and the corresponding ECC mechanisms are discussed in Chapter 3.

For 3D DRAM memory systems, if the number of errors is larger than a threshold value (the threshold value is system-dependent), the specific TSV line or bank can be marked as faulty by the memory controller. If a TSV is marked as faulty, we utilize the erasure correction capability of the tier-1 code and if the data banks are marked as faulty, we reduce the parity storage of the tier-1 and tier-2 codes to free up banks in the ECC die as discussed in Chapter 4.

Chapter 3

2D DRAM ECC SCHEMES

In this chapter we describe our work on designing reliable ECC schemes that provide at least Chipkill-Correct capability for different commodity DRAM systems, namely, x4, x8 and x16 systems. The goal is to balance the tradeoffs between performance, reliability, storage overhead and power consumption by designing ECC schemes tailored to the characteristics of the different DRAM systems.

Our solution is to use erasure and error correction code (E-ECC) to deal with random errors as well as errors due to chip failures. We make use of erasure codes to correct the errors due to a chip that has already been marked as faulty by MC. This helps in increasing the lifetime of the device. We believe that this is a more cost-effective way of increasing the lifetime compared to systems that use chip sparing or bit steering. Furthermore, our proposed schemes do not incur extra read or write operations to perform ECC correction or to update ECC bits.

We describe the proposed E-ECC schemes for x4 DRAM systems in Section 3.1, for x8 DRAM systems in Section 3.2 and for x16 DRAM systems in Section 3.3. For each scheme, we describe the data access pattern and the decoding flowchart. The actual decoding algorithms and the corresponding synthesis results are included in the Appendix A and Section 3.4, respectively. We also analyze the tradeoff between the reliability and system performance (timing, power and energy efficiency) for our proposed schemes and the existing schemes in Section 3.5.

3.1 Proposed E-ECC Schemes For x4 DRAM Systems

For x4 DRAM systems, we present three schemes that all have 12.5% storage overhead but differ in the number of ranks being activated. E-ECC Schemes 1 and 2 operate in lockstep mode while E-ECC Scheme 3 only activates one rank per access. The specific ECC codes used in these three schemes are rotational (144,128) code over GF(2^4) and RS (36,32) code over GF(2^8). The schemes that activate two ranks per memory access have lower timing, power and energy performance but higher reliability compared to the one that activates only one rank.

3.1.1 E-ECC Scheme 1 - x4 DRAM System

Chipkill-Correct uses 4-check symbol codes to correct errors due to a single chip failure and detect errors due to two chip failures. We use rotational (144,128) code [12] as the representative Chipkill-Correct code in this work. Here, 36 devices are activated across two ranks in each access. Each device provides four bits of data per beat, that is, 36x4 = 144 bits per beat, to the ECC decoding unit. Each set of 144 bits is decoded to obtain 128 data bits and a total of 4x128 = 512 data bits is sent to the last level cache.

The rotational (144,128) code has a minimum distance of 4 and so this code can support the following cases: (i) single symbol correction and double symbol detection, (ii) single erasure correction, (iii) single erasure and single error correction, (iv) double erasure correction and (v) double erasure and single error detection. Current Chipkill-Correct x4 systems implement only single symbol correction and double symbol detection (case (i)). Here, we propose an enhancement which makes use of the same (144,128) code to handle erasures; we refer to it as **E-ECC Scheme 1**.

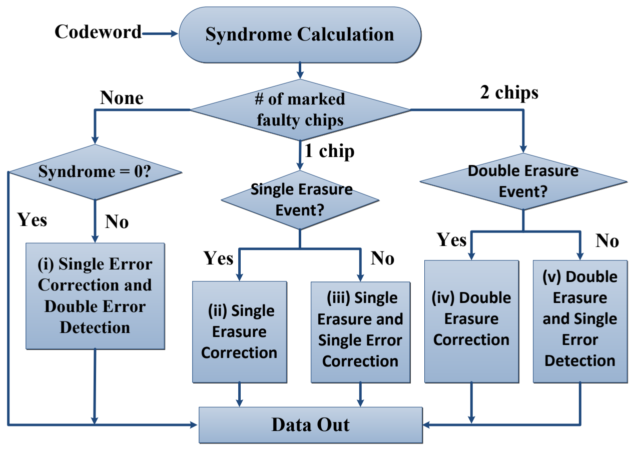

Codeword
Syndrome Calculation
# of marked faulty chips
None
1 chip
2 chips
Syndrome = 0?
Yes
No
(i) Single Error Correction and Double Error Detection
Single Erasure Event?
Yes
No
(ii) Single Erasure Correction
(iii) Single Erasure and Single Error Correction
Double Erasure Event?
Yes
No
(iv) Double Erasure Correction
(v) Double Erasure and Single Error Detection
Data Out

Figure 3.1: Overview Of The Decoding Algorithm For E-ECC Scheme 1

If a chip is marked as faulty, it leads to a single erasure and Scheme 1 performs single erasure correction (case (ii)). When one more random error occurs in another chip, Scheme 1 can still correct it (case (iii)). The decoder first checks if it is a single erasure event. If so, case (ii) is executed; otherwise, case (iii) is activated. If a second chip fails, the MC marks it as faulty. The decoder first checks if it is a double erasure event. If so, double erasure correction (case (iv)) is implemented; otherwise, double erasure and single error detection (case (v)) is activated. The decoding flowchart for Scheme 1 is shown in Figure 3.1 and the details of the decoding algorithm are included in Appendix A.

3.1.2 E-ECC Scheme 2 - x4 DRAM System

To enhance the error correction capability of x4 DRAM systems, we investigate ECC codes operating in higher finite field. We combine data from two beats to obtain 256 data bits and 32 ECC bits. Since there is no RS (72,64) code over GF(2^4), we move to GF(2^8). We propose using RS (36,32) in GF(2^8), which can be derived from RS (255,251). RS (36,32) can provide double error correction, that is, it can correct double chip failures on the fly instead of only detecting them as in Scheme 1. We refer to this method as **E-ECC Scheme 2**.

In Scheme 2, two ranks (with 18 chips per rank) are activated per access. In each read/write, two consecutive 4-bit symbols from the same bank form a single 8-bit symbol, and thus a total of 36 symbols are read out. Figure 3.2 illustrates the data access pattern. The proposed RS (36,32) E-ECC code has a minimum distance of 5 and supports the following cases: (i) single error correction, (ii) double error correction, (iii) single erasure correction, (iv) single erasure and single error correction (v) double erasure correction and (vi) double erasure and single error correction.

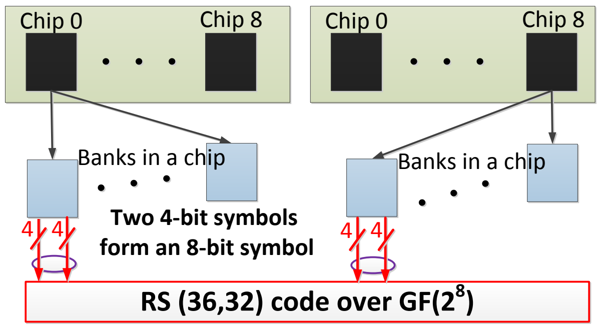

Chip 0
Chip 8
Chip 0
Chip 8
Banks in a chip
Banks in a chip
Two 4-bit symbols form an 8-bit symbol
4
4
4
4
RS (36,32) code over GF(2^8)

Figure 3.2: E-ECC Scheme 2. x4 DRAM Access Pattern Of Two Ranks.

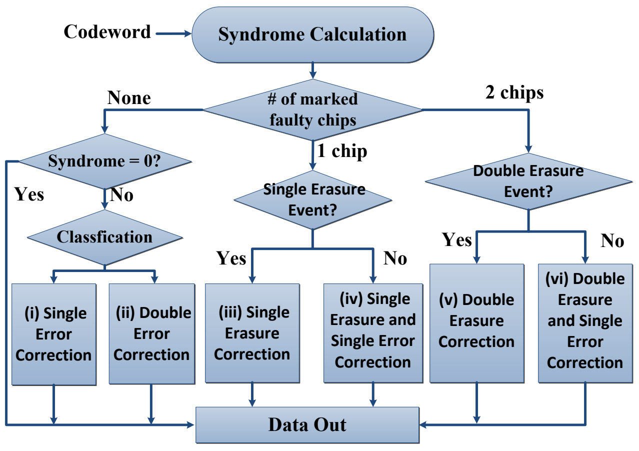

Codeword
Syndrome Calculation
None
# of marked faulty chips
2 chips
1 chip
Syndrome = 0?
Yes
No
Classfication
Single Erasure Event?
Double Erasure Event?
Yes
No
Yes
No
(i) Single Error Correction
(ii) Double Error Correction
(iii) Single Erasure Correction
(iv) Single Erasure and Single Error Correction
(v) Double Erasure Correction
(vi) Double Erasure and Single Error Correction
Data Out

Figure 3.3: E-ECC Scheme 2. Overview Of The Decoding Algorithm.

The default state of Scheme 2 is single error and double error correction. Since RS code has a special algebraic structure, the decoder can use the syndrome to distinguish between case (i) and case (ii) efficiently [55]. When a chip fails, it leads to a single error in the E-ECC codeword and a single error can be corrected easily. When MC marks this chip as faulty, the error becomes an erasure in an ECC codeword and the decoding circuitry for single erasure correction (case (iii)) is activated. Correcting single erasure is simpler compared to correcting single error (see Appendix A). Furthermore, if there is an additional error in another chip, it can be corrected as well (case (iv)). We assume that the errors build up over time and so a second faulty chip can start generating repeated errors. When MC marks the second chip as faulty, the E-ECC decoder activates double erasure correction (case (v)). Once two chips are marked as faulty, it can correct one more random error (case (vi)). The decoding flowchart is shown in Figure 3.3 and details of the decoding algorithm for each case is given in Appendix A.

3.1.3 E-ECC Scheme 3 - x4 DRAM System

Although Scheme 2 improves the reliability compared to Chipkill-Correct and E-ECC Scheme 1, it still activates 36 devices and consumes significant amount of power. This motivates us to find another scheme that activates fewer chips at the cost of some loss in reliability. We investigate codes in GF(2^4) and GF(2^8) with the constraint that 18 chips can be activated in each access. Under this constraint, in each beat, 72 bits (64 data bits + 8 ECC bits) are accessed from 18 chips. Since there is no RS (18,16) code over GF(2^4), we move to GF(2^8).

In GF(2^8), the only available code is RS (9,8), which has a minimum distance of 2 and so cannot even correct one single error. However, if we combine data from two beats (144 bits), there are two candidates: one is the rotational (144,128) code and the other is the RS (18,16) code over GF(2^8). Rotational code cannot correct one chip failure (there are 2 x4 symbol errors due to a chip failure) and hence it is not suitable. The RS (18,16) code has minimum distance of 3 and can perform single error correction. It seems to be used in AMD Chipkill-Correct [52]. Although this code provides Chipkill-Correct capability, we choose a stronger code whose error correction capability is competitive with the other proposed schemes. Specifically, we choose RS (36,32) code over GF(2^8); the corresponding scheme is referred to as **Scheme 3**.

In Scheme 3, four consecutive 4-bit symbols from the same bank contribute towards a codeword. Figure 3.4 describes the corresponding data access pattern. As mentioned earlier, the RS (36,32) code has a minimum distance of 5 and supports the following cases: (i) single error correction, (ii) double error correction, (iii) double erasure correction, (iv) double erasure and single error correction (v) double erasure and double error detection.

The default state of Scheme 3 is also single error and double error correction. When a chip fails, it leads to two errors in the E-ECC codeword and these two errors can be corrected on the fly. When MC marks a chip as faulty, the decoding circuitry for double erasure correction (case (iii)) is activated. The double erasure correction methods for Schemes 2 and 3 are different. In Scheme 2, two chip failures lead to two erasure symbols in a codeword, which may not be adjacent. In Scheme 3, one chip failure leads to double erasures and these two erasure symbols are adjacent in a codeword.

Correcting two erasures is simpler compared to correcting two errors (see Appendix A) and these two erasure addresses are consecutive in Scheme 3. Furthermore, if there is an additional error in another chip, it can be corrected as well (case (iv)). If a second chip fails, there are two erasures (from the chip marked faulty) and two errors (from the other faulty chip) which can be detected but not corrected (case (v)). Note that the system cannot use case (iv) and case (v) at the same time and a choice has to be made. For DRAM systems whose errors increase over time, the MC can activate case (iv) decoder, and once the number of single errors is larger than a threshold, it can activate the circuitry for case (v). The decoding flowchart is shown in Figure 3.5 and details of the decoding algorithm for each case is given in the Appendix A.

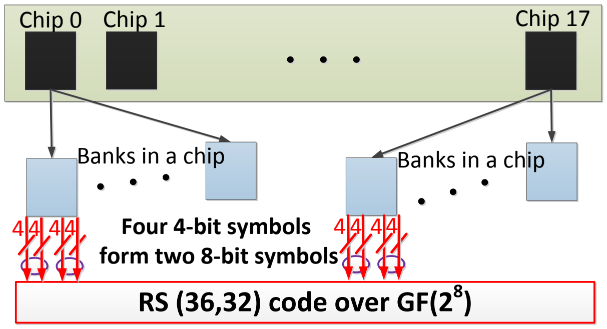


Figure 3.4: E-ECC Scheme 3. x4 DRAM Access Pattern Of One Rank

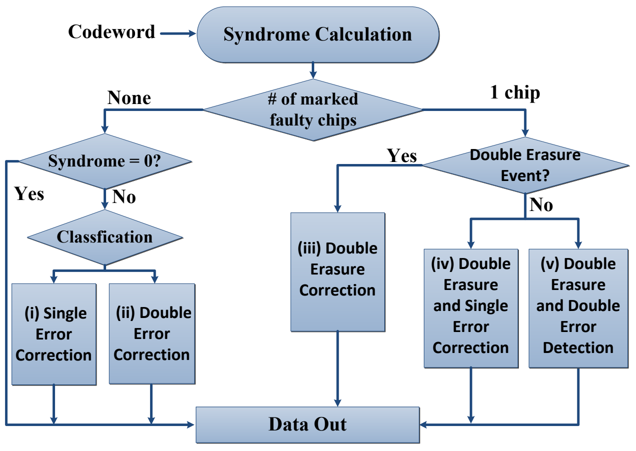

Codeword
Syndrome Calculation
# of marked faulty chips
None
1 chip
Syndrome = 0?
Yes
No
Classfication
(i) Single Error Correction
(ii) Double Error Correction
Double Erasure Event?
Yes
No
(iii) Double Erasure Correction
(iv) Double Erasure and Single Error Correction
(v) Double Erasure and Double Error Detection
Data Out

Figure 3.5: E-ECC Scheme 3. Overview Of The Decoding Algorithm

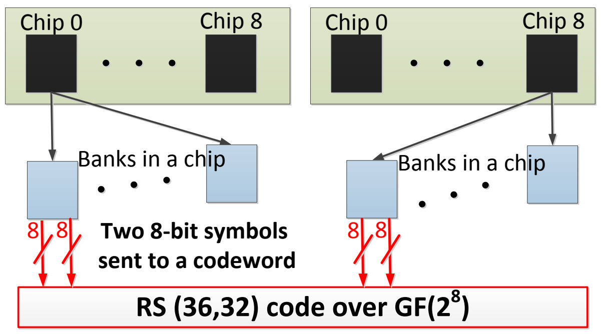

Chip 0
Chip 8
Chip 0
Chip 8
Banks in a chip
Banks in a chip
8
8
8
8
Two 8-bit symbols sent to a codeword
RS (36,32) code over GF(2^8)

Figure 3.6: E-ECC Scheme 4. x8 DRAM Access Pattern

3.2 Proposed E-ECC Schemes For x8 DRAM Systems

3.2.1 E-ECC Scheme 4 - x8 DRAM System

In an x8 DRAM system, nine chips (8 data chips + 1 ECC chip) are accessed every time and since fewer chips are activated compared to x4 DRAM, the power consumption is lower. Hence, a lot of research effort has focused on designing Chipkill-Correct x8 DRAM systems [14–16, 51, 56].

Deriving a Chipkill-Correct solution for x8 DRAM with low overhead is still quite a challenge. If only one rank is activated, in each beat, nine symbols are accessed from nine chips. A possible choice is the RS (9,8) code over GF(2^8). Unfortunately, this code can only detect a single symbol error and thus is not suitable. If two beats of data are combined, RS (18,16) code can be used over GF(2^8) to perform single error correction. However, since one faulty chip leads to two error symbols per codeword, this code can not correct them. To provide Chipkill-Correct protection, either an extra ECC chip (3-check symbol code) has to be used or the extra ECC symbols have to be stored in another rank like [14].

Our solution is to operate the x8 DRAM system in lock-step mode as in Intel [19], HP [57] and Dell [58] systems. In the lock-step mode, two ranks are activated and in each beat, 144 bits (128 data bits + 16 ECC bits) are accessed from 18 chips across two ranks. If we use the rotational (144,128) code, when a chip fails, the faulty chip leads to 2 x4 error symbols in a codeword and since this code cannot correct two random errors, it is not Chipkill-Correct. In GF(2^8), the RS (18,16) code is a possible choice. It can correct errors due to a chip failure. When MC marks a chip as faulty, it can correct single erasure and detect one more random error in another chip. However, we choose a stronger code, namely, RS (36,32), since it has better error correction capability compared to RS (18,16) and the same 12.5% storage overhead. We refer to this scheme as **E-ECC Scheme 4**.

In each beat, 2x9 = 18 symbols are read out from two ranks and a total of 2x18 = 36 symbols are accessed in two beats. Figure 3.6 demonstrates the data access pattern of Scheme 4. As mentioned earlier, the RS (36,32) code supports: (i) single error correction, (ii) double error correction, (iii) double erasure correction, (iv) double erasure and single error correction (v) double erasure and double error detection. Scheme 4 uses the same RS (36,32) code as Scheme 3, though their access patterns are different (Scheme 4 is designed for x8 DRAM systems while Scheme 3 is for x4 DRAM systems). The decoding flow of Scheme 4 is the same as Scheme 3 (shown in Figure 3.5). When none of the chips are marked as faulty, Scheme 4 performs cases (i) and (ii). When a chip is marked as faulty, both symbols from that chip are treated as erasures. The decoder checks whether it is a double erasure event. If so, it launches double erasure correction (case (iii)); otherwise, it activates double erasure and single error correction (case (iv)) or double erasure and double error detection (case (v)). Note that case (iv) and case (v) cannot be chosen at the same time.

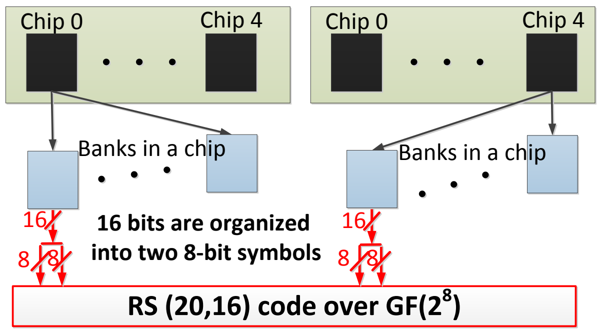

Chip 0
Chip 4
Chip 0
Chip 4
Banks in a chip
Banks in a chip
16
16
8
8
8
8
16 bits are organized into two 8-bit symbols
RS (20,16) code over GF(2^8)

Figure 3.7: E-ECC Scheme 5. x16 DRAM Access Pattern

3.3 Proposed E-ECC Schemes For x16 DRAM Systems

3.3.1 E-ECC Scheme 5 - x16 DRAM System

x16 DRAM activates fewer number of chips per rank compared to x4 and x8 DRAM systems and thus has even lower power. Since each rank has four chips for data, there has to be at least one additional chip per rank to provide Chipkill-Correct reliability. Thus, such a scheme results in a storage overhead of 25%. If only one rank is activated, five 16-bit symbols are read out in each beat. This can be configured into ten 8-bit symbols and the RS (10,8) code over GF(2^8) can be used. While the RS (10,8) code can correct a single symbol error, it cannot correct double errors due to a faulty chip. The RS (5,4) code over GF(2^{16}), on the other hand, can detect a faulty chip but cannot correct the errors due to that faulty chip. If two beats of data are combined, the RS (20,16) code over GF(2^8) or the RS (10,8) code over GF(2^{16}) can be used. Unfortunately, both these codes cannot correct errors due to a faulty chip. Hence, we move to x16 DRAM systems that operate in the lock-step mode.

When two ranks are activated, in each beat, 160 bits (128 data bits + 32 ECC bits) are accessed. If RS (20,16) code over GF(2^8) is used, double errors due to a single faulty chip can be corrected. Although RS (10,8) code over GF(2^{16}) is also a Chipkill-Correct code, RS (20,16) over GF(2^8) provides stronger and more flexible error correction capability with the same 25% storage overhead. It can correct two random symbol errors in two different chips and also can correct two symbol errors in a single chip. Hence, we choose RS (20,16) code over GF(2^8) as our **E-ECC Scheme 5**.

Figure 3.7 illustrates the data access pattern. Here two ranks with five chips per rank are activated each time. The 16 bits per chip are organized into two 8-bit symbols in a codeword. The proposed RS (20,16) code also has a minimum distance of 5 and supports the following cases: (i) single error correction, (ii) double error correction, (iii) double erasure correction, (iv) double erasure and single error correction (v) double erasure and double error detection. Scheme 5 has the same decoding flow as Schemes 3 and 4 (also shown in Figure 3.5), though its access pattern is quite different. As before, when none of the chips are marked as faulty, Scheme 5 performs cases (i) and (ii). When a chip is marked as faulty, one 16-bit symbol failure from that chip is treated as two 8-bit erasures. The decoder checks whether it is a double erasure event. If so, Scheme 5 launches double erasure correction (case (iii)); otherwise, it activates double erasure and single error correction (case (iv)) or double erasure and double error detection (case (v)). Note that case (iv) and case (v) cannot be used at the same time.

3.4 Synthesis Results

We implemented the E-ECC decoders in Verilog hardware description language and synthesized them with a 28nm industrial process. The synthesis was done in collaboration with Supreet Jeloka of University of Michigan. Recall that Scheme 1 is based on the rotational (144,128) code, Schemes 2, 3, 4 are based on the RS (36,32) code over GF(2^8), and Scheme 5 is based on the RS (20,16) code over GF(2^8). The latency, power consumption and area of each block ini the three E-ECC codes are presented in Tables 3.1, 3.2 and 3.3, respectively.

The two main hardware components are finite field multiplication and finite field inversion. We used a fully parallel implementation for finite field multiplication. We implemented finite field inversion using a look-up table of size 16×4 for GF(2^4) and 256×8 for GF(2^8). The syndrome calculation unit is activated in every read operation and so it is important that its latency be minimized. We implemented this unit using 144 GF(2^4), 144 GF(2^8) and 80 GF(2^8) multiplications for rotational (144,128) code, RS(36,32) code and RS (20,16) code, respectively, followed by a tree of XOR gates.

Latency: The syndrome calculation unit of Scheme 1 (based on the (144,128) code) has a latency of 0.41ns. If the syndrome vector is not zero and the error event is single error, correcting the errors takes an additional 0.41ns. This low latency comes at the cost of additional area due to parallelization. If a chip is marked as faulty, single erasure correction takes 0.3ns. In addition to the erasures, if there is one more random error, it takes another 1.1ns. If two chips are marked as faulty, Scheme 1 takes 0.39ns to do double erasure correction. If there is one more random error, Scheme 1 can detect this error but can not correct it, and the corresponding timing delay is also 0.39ns.

The syndrome calculation unit of Scheme 2 (based on RS (36,32) code) takes 0.48ns. If the syndrome vector is not zero, RS (36,32) can classify whether it is a single error event or a double error event. Single error correction takes an additional 0.47ns, making the total latency 0.95ns. Double error correction takes an additional 1.07ns, making the total latency 1.55ns. When a chip is marked as faulty, Scheme 2 checks whether it is a single erasure event. If so, it implements single erasure correction with an additional 0.33ns; otherwise, it implements single erasure and single error correction with an additional 0.33+0.68ns. In Schemes 3 and 4, when a chip is

Table 3.1: Synthesis Results Using A 28nm Library - Latency

	Rotational (144,128) Code over GF(2^4)	Shortened RS (36,32) Code over GF(2^8)	Shortened RS (20,16) Code over GF(2^8)
Syndrome Calculation	0.41ns (Δ)	0.48ns (Λ)	0.42ns(ν)
Corrects 1 error	Δ + 0.41ns	Λ + 0.47ns	ν + 0.47ns
Corrects 1 erasure	Δ + 0.30ns	Λ + 0.33ns	N/A
Corrects 1 erasure & 1 error	Δ + 0.30ns + 1.1 ns	Λ + 0.33ns + 0.68ns	N/A
Corrects 2 errors	N/A	Λ + 1.07ns	ν + 1.07ns
Corrects 2 erasures	Δ + 0.39ns	Λ + 0.78ns	ν + 0.79ns
Corrects 2 erasures & 1 error	N/ A	Λ + 0.78ns + 0.87ns	ν + 0.79ns + 0.86ns

marked as faulty, E-ECC decoder checks for double erasures. If so, it implements double erasure correction with an additional 0.78ns; otherwise, it implements double erasure and single error correction with an additional 0.78+0.87ns.

The syndrome calculation unit in Scheme 5 takes only 0.42ns because Scheme 5 is based on a smaller RS code. When none of the chips are marked faulty, Scheme 5 performs single error correction with an additional 0.47ns and double error correction with an additional 1.07ns. When a chip is marked as faulty, double erasure correction takes an additional 0.79ns. If there is one more random error, it performs double erasure and single error correction with an additional 0.79+0.86ns.

Power: We list the static power and dynamic power consumption for each E-ECC unit in Table 3.2. The dynamic power is based on input switching probability of 50%. The syndrome calculation unit, which is activated in every memory access, also has very small power consumption compared to the off-chip DRAM power consumption (Overall, the power consumption values are very small and can be ignored).

Area: We also estimate the area of the E-ECC decoders. Schemes 1 to 5 have an area of 12,793um^2, 25,215um^2, 20,389um^2, 20,389um^2 and 16,768um^2 respectively. Scheme 2 has the largest number of decoding units and hence it has the largest

Table 3.2: Synthesis Results Using A 28nm Library - Power

(static power/dynamic power)	Rotational (144,128) Code over GF(2^4)	Shortened RS (36,32) Code over GF(2^8)	Shortened RS (20,16) Code over GF(2^8)
Syndrome Calculation	0.0083 mW / 14.5 mW	0.0192 mW / 14.6 mW	0.0141 mW / 11.2 mW
Corrects 1 error	0.0015 mW / 12.5 mW	0.0195 mW / 22.53 mW	0.0158 mW / 13.6 mW
Corrects 1 erasure	0.001 mW / 12.6 mW	0.0144 mW / 25.62 mW	N/A
Corrects 1 erasure & 1 error	0.0043 mW / 9.48 mW	0.0271 mW / 21.44 mW	N/A
Corrects 2 errors	N/A	0.0374 mW / 16.54 mW	0.0326 mW / 11.1 mW
Corrects 2 erasures	0.0131 mW / 15.93 mw	0.0277 mW / 19.1 mW	0.0239 mW / 12.2 mW
Corrects 2 erasures & 1 error	N/A	0.0353 mW / 19.97 mW	0.0306 mW / 13.7 mW

Table 3.3: Synthesis Results Using A 28nm Library - Area

	Rotational (144,128) Code over GF(2^4)	Shortened RS (36,32) Code over GF(2^8)	Shortened RS (20,16) Code over GF(2^8)
Syndrome Calculation	2001 um^2	3970 um^2	2533 um^2
Corrects 1 error	2760 um^2	2024 um^2	1842 um^2
Corrects 1 erasure	980 um^2	1336 um^2	N/A
Corrects 1 erasure & 1 error	5658 um^2	3490 um^2	N/A
Corrects 2 errors	N/A	5043 um^2	4575 um^2
Corrects 2 erasures	1394 um^2	3311 um^2	2935 um^2
Corrects 2 erasures & 1 error	N/A	6041 um^2	4883 um^2

area. Schemes 3 and 4 both have the same area since they activate the same set of decoding units. Scheme 5 has smaller area compared to Schemes 3 and 4 because it uses a smaller sized RS code. Overall the area of the E-ECC decoders is quite small. The largest E-ECC has an area of only $0.025mm^2$, which is fairly small compared to a typical die size ($\approx 100\ mm^2$).

Table 3.4: Simulation System Parameters

Processor	2 GHz single core (quad-core for multiprogrammed workloads), 4-way out-of-order, 128-entry reorder buffer
L1-I Cache L1-D Cache	32KB, 4-way assoc., 1 cycle latency, 64B cache line 32KB, 8-way assoc., 1 cycle latency, 64B cache line
Shared L2 Cache	1MB (4MB for multiprogrammed workloads), 16-way assoc., 10 cycle latency, 64B cache line
Main Memory	DDR3-1600 (800MHz), FCFS, open page policy, 13.75ns precharge time, 13.75ns CAS latency, 13.75ns RAS to CAS latency, 8 banks per chip, 1 kB row buffers

3.5 Evaluation

We evaluate the timing performance of the five different E-ECC schemes by using an open source full-system simulator, gem5 [59]. We model a 4-way out-of-order processor with a two-level cache hierarchy. The L1 instruction and data caches are private to each core while the L2 caches are shared. We also model a detailed DDR3 DRAM with data rate 1600 (MT/s). Table 3.4 describes the configuration of our setup.

We use a detailed cycle-based DRAM simulator, DRAMSim2 [60], to evaluate DRAM power consumption. We generate DRAM access traces from the aforementioned gem5 simulation setup. We then estimate the power consumption of the five different schemes by simulating at least 2 million memory accesses from these traces. We use the Micron 4Gb data sheets [61] for the input parameters of DRAMSim2.

3.5.1 Workloads

We evaluate the timing and power performance impact of the E-ECC schemes using both sequential and multiprogrammed workloads. We use 11 DRAM-sensitive sequential applications from the SPEC2006 benchmark suite (Table 3.5). We use the

Table 3.5: Sequential And Multiprogrammed Workloads.

Benchmarks
bwaves, bzip2, cactusADM, h264ref, lbm, libquantum, mcf, omnetpp, soplex, sphinx3, zeusmp
mix1(bzip2, mcf, omnetpp, libquantum)
mix2(libquantum, bwaves, zeusmp, cactusADM)

Simpoints [62] to identify a single, 250-million instruction representative region for each sequential workload. In addition, we also evaluate the multi-core system performance. We create two 4-core multiprogrammed workload mixes of the SPEC2006 benchmarks to realistically model the multiprogrammed application execution scenarios. The workload mixes are also summarized in Table 3.5. For our multiprogrammed simulations, in order to enable the start of different benchmarks in each workload at the same time, we fast-forward two billion instructions from the program start and simulate in detail until all the benchmarks have simulated for at least 250 million instructions. We collect the statistics after the slowest benchmark has completed simulating 250 million instructions.

3.5.2 Results And Analysis

In this part, we analyze the timing, power and energy performance of the five E-ECC schemes and the three existing schemes (V-ECC, LOT-ECC and Multi-ECC). In order to obtain the timing performance of the competing schemes (V-ECC, LOT-ECC and Multi-ECC), we also synthesized the corresponding ECC units in 28 nm technology. The syndrome calculation unit of V-ECC (using RS (19,16) code over GF(2^8)) is 0.42ns, LOT-ECC (using multi-level one's complement addition) is 0.52ns and Multi-ECC (using RS (9,8) code over GF(2^{16})) is 0.24ns.

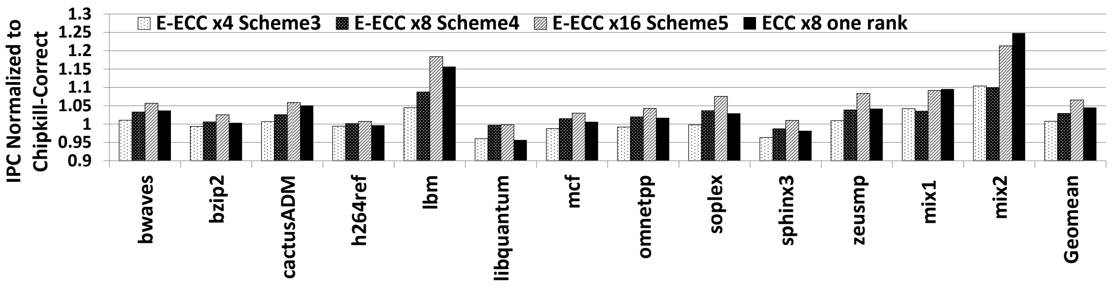


Figure 3.8: IPC Performance Of Sequential And Multiprogrammed Applications. Schemes 1 And 2 Have The Same IPC Performance As Chipkill-Correct (Baseline). LOT-ECC And Multi-ECC Have Performance Comparable To ECC x8 One Rank And V-ECC Has Performance Comparable To E-ECC x8 Scheme 4.

Since the latencies of syndrome calculation of all schemes are within one memory cycle (1.25ns), we add one additional cycle for each memory read access in gem5 simulation. We present the simulation results of all schemes in an error-free system. All performance values are normalized to those of the x4 Chipkill-Correct baseline scheme. Since Schemes 1 and 2 activate 36 x4 chips in two ranks as in the baseline, they have identical timing, power and energy performance and are not shown in the figures.

Timing Performance Comparison

Figure 3.8 shows the IPC performance of E-ECC Schemes 3, 4 and 5 for a subset of our sequential and multiprogrammed workloads. We measure the performance of the multiprogrammed workloads using the weighted speedup, which is given by $\sum_i \frac{(IPC_i^{share})}{(IPC_i^{single})}$ [63, 64]. We use the geometric mean to report the average values.

Overall, the IPC performance increases as the data width increases from x4 to x8 to x16. The performance improvement can be attributed to higher rank-level parallelism. For example, Schemes 1 and 2 operate on one logical rank whereas

Schemes 3 and 4 operate on two logical ranks and thus have better IPC performance. Figure 3.8 shows that Scheme 5, which operates on four logical ranks, has the best IPC performance among all the schemes. The improvement is 7% average, 21% maximum, in multiprogrammed workloads and 18% in sequential workloads.

Of the existing schemes, V-ECC has the same configuration as E-ECC Scheme 4; the only difference is that V-ECC has to update the tier-2 ECC symbols when there is a write request. Since V-ECC uses ECC cache to reduce the number of write operations, we project that V-ECC has timing performance similar to E-ECC Scheme 4.

LOT-ECC and Multi-ECC both activate one x8 rank per memory access. While the tier-2 ECC symbols of LOT-ECC are located in the same row as the accessed data, they are located in another row in Multi-ECC. Both schemes also use ECC cache to reduce the number of extra writes to update the tier-2 ECC symbols. Thus, their best case performance is when the ECC cache has a 100% hit rate. This is shown in Figure 3.8 under ECC x8 one rank.

Overall, we can expect the order of timing performance improvement to be E-ECC Scheme 5 > LOT-ECC and Multi-ECC (ECC x8 one rank) > E-ECC Scheme 4 and V-ECC > E-ECC Scheme 3 > E-ECC Scheme 1 and 2 and Chipkill-Correct.

Power Performance Comparison

We compare the DRAM power consumption of the candidate E-ECC schemes and the existing schemes; the memory configurations are provided in detail in Table 3.6. We generate the power consumption for each scheme by running the simulations in DRAMSim2.

Figure 3.9 shows the power consumption of the sequential and multiprogrammed SPEC workloads normalized to baseline. The power consumption is due to the off-

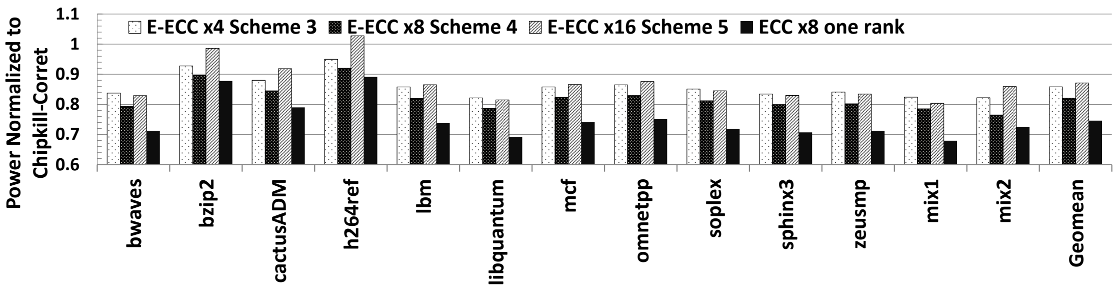


Figure 3.9: Power Consumption Of Sequential And Multiprogrammed Workloads Normalized To Chipkill-Correct. Schemes 1 And 2 Have The Same Power Performance As Chipkill-Correct (Baseline). LOT-ECC And Multi-ECC Have Performance Comparable To ECC x8 One Rank And V-ECC Has Performance Comparable To E-ECC x8 Scheme 4.

chip DRAM; the power consumption due to ECC decoding in logic die is negligible (20mW compared to 3W for DRAM) and is ignored. Scheme 4 obtains the best power reduction among the five schemes; it achieves an average of 18%, a maximum of 23.5% in multiprogrammed workloads and a maximum of 21.3% in sequential workloads power reduction compared to baseline. Scheme 5 does not perform as well due to its higher storage overhead (25% compared to 12.5%) and its 2 kB row buffer compared to 1 kB row buffer used in x4 and x8 DRAM systems. Even though x16 DRAM activates fewer number of DRAM chips, the larger row buffer size adversely affects its power performance.

Of the existing schemes, V-ECC has power performance comparable to our E-ECC Scheme 4. LOT-ECC and Multi-ECC have the lowest power consumption since they only activate one x8 rank per memory access. Overall, we can expect that the power efficiency to be LOT-ECC and Multi-ECC (ECC x8 one rank) > E-ECC x8 Scheme 4 > E-ECC x4 Scheme 3 > E-ECC x16 Scheme 5 > E-ECC Scheme 1 and 2 and Chipkill-Correct.

Table 3.6: Memory Configurations Of The Five E-ECC Schemes

ECC	Channel	I/O pin	Ranks/Channel	Chips/Rank	Burst length	Total Size	Data Size
E-ECC x4 Scheme 1,2	1	x4	1	36	4	18GB	16GB
E-ECC x4 Scheme 3	1	x4	2	18	8	18GB	16GB
E-ECC x8 Scheme 4	1	x8	2	18	4	18GB	16GB
E-ECC x16 Scheme 5	2	x16	2	10	4	20GB	16GB
ECC x8 One rank	2	x8	2	9	8	18GB	16GB

Energy Performance Comparison

We next present our energy efficiency comparison for the different E-ECC schemes. We derive the energy number by multiplying cycle per instruction (CPI) and power for each benchmark with 250 million instructions and normalizing it to the energy number of the baseline. Overall, Scheme 4 outperforms all other proposed schemes in energy efficiency. Figure 3.10 shows that Scheme 3, Scheme 4 and Scheme 5 improve the energy efficiency by a mean of 17.4%, 25.4% and 22%, respectively. Scheme 5 outweighs Schemes 3 or 4 only when the system is heavily used. However, if the system is under utilized, Scheme 5 performs worse compared to other schemes. We conclude that even though Scheme 5 has the best timing performance, it does not have the best power performance and thus not the best energy performance.

Compared to the existing schemes, V-ECC also has energy efficiency comparable to our E-ECC Scheme 4. LOT-ECC and Multi-ECC have the best energy efficiency; they outperform E-ECC Scheme 4 by around 15%. Although LOT-ECC and Multi-ECC have attractive system performance, they are not as reliable as Chipkill-Correct schemes.

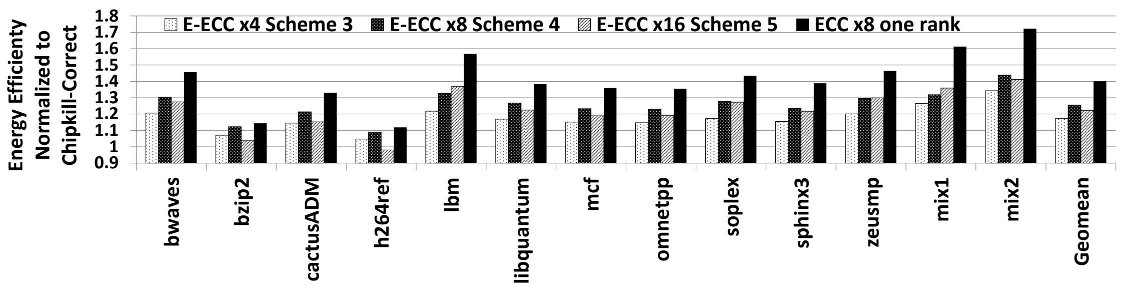


Figure 3.10: Energy Efficiency Of Different E-ECC Schemes. Schemes 1 And 2 Have The Same Energy Efficiency Performance As Chipkill-Correct (Baseline). LOT-ECC And Multi-ECC Have Performance Comparable To ECC x8 One Rank And V-ECC Has Performance Comparable To E-ECC x8 Scheme 4.

3.5.3 Reliability Analysis

In this section, we analyze the reliability of the competing schemes including Chipkill-Correct, V-ECC [14], LOT-ECC [15], Multi-ECC [16] and our E-ECC Schemes. Table 3.7 summarizes the error detection and correction capability of all schemes. For each error event, we provide the rates for detectable and correctable errors (DCE), detectable but uncorrectable errors (DUE) and silent data corruption (SDC). These rates are calculated by performing 10 million Monte Carlo simulations.

First of all, all schemes can correct single bit error with 100% probability. If there are multiple bit errors in a row, LOT-ECC uses 7-bit checksum to perform local error detection and can only detect a whole row being stuck-at-0 or stuck-at-1. However, if there are multiple random bit errors in a row, LOT-ECC can not fully detect it. The DCE rate of this scheme is 87.5% and the SDC rate is 12.5% as given in [15]. The rest of the schemes can correct multiple random bit errors in a single row.

Next we consider multiple bit errors in a single column. Multi-ECC uses one's complement to generate the column check sum and so if there are even number of random errors or combinations of stuck-at-0 and stuck-at-1 failures in a single column, Multi-ECC can detect the errors using row parity bits but cannot use the column checksums to locate them correctly. If every bit in a single column is flipped with 50% probability, then for 10 million runs, Multi-ECC has DCE of 75.01% and DUE of 24.99%. In contrast, the rest of the schemes can deal with any combination of random errors or permanent errors within a single column.

When a chip fails, it results in multiple column failures or row failures. LOT-ECC can detect a chip failure with DCE less than 87.5%, which is the same as its row failure event. Similarly, Multi-ECC has a chip failure probability which is the same as its column failure probability. In contrast, all five E-ECC schemes can correct errors due to a chip failure. When one chip fails and a single bit failure occurs in another chip, Chipkill-Correct and V-ECC can fully detect this event. LOT-ECC can detect this event with less than 87.5% probability while Multi-ECC can detect with $1-2^{-16} \approx 99.9985\%$ probability. Scheme 1 can detect this event with 100% probability and Scheme 2 can correct with 100% probability. However, E-ECC Schemes 3, 4 and 5 cannot handle this error event.

If the MC marks a chip as faulty and there is one more random error in another chip, all 5 E-ECC Schemes can correct it. V-ECC can also be designed to correct one erasure and one error. LOT-ECC, which uses XOR operation to recover from a faulty chip, cannot handle this error event. Multi-ECC can correct the extra error if a spare chip is used to replace the faulty chip. If the MC marks a chip as faulty and there is one more chip failure, Schemes 1 and 2 can fully correct the errors.

Overall, Scheme 2 has the highest error protection capability since it can correct double chip failures on the fly. In addition, when two chips are marked as faulty, it can correct errors due to a third chip failure.

Table 3.7: Error Detection And Correction Comparison

	Chipkill-Correct	V-ECC [14]	LOT-ECC [15]	Multi-ECC [16]	E-ECC Scheme 1	E-ECC Scheme 2	E-ECC Scheme 3	E-ECC Scheme 4	E-ECC Scheme 5
Single bit failure	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Row failures	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:87.5% DUE:0% SDC:12.5%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Column failures	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:75.01% DUE:24.99% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Single chip failure	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:87.5% DUE:0% SDC:12.5%	DCE:75.01% DUE:24.99% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%

3.5.4 Overhead Comparison

All proposed E-ECC Schemes (except for Scheme 5) have 12.5% storage overhead while V-ECC, LOT-ECC and Multi-ECC have storage overheads more than 12.5% (18.5%, 26.5% and 12.9%). Although Scheme 5 has 25% storage overhead, it only utilizes one additional chip per four chips in x16 DRAM and does not incur extra reads/writes to access additional ECC symbols. Furthermore, direct implementations of V-ECC, LOT-ECC and Multi-ECC incur extra reads or writes to access or update tier-2 ECC symbols. In order to reduce the number of additional read/write operations to access the tier-2 ECC symbols, all three schemes need the supports of the ECC cache. OS needs to be modified and extra hardware units (ECC address mapping) and extra ECC cache size are added to support ECC cache as indicated in [14–16].

The E-ECC schemes rely on machine check architecture (MCA) to record the corrected errors in each chip. MCA is already used in several servers [3, 21, 50] and they are not the extra design to support our E-ECC schemes. When errors are caused

Table 3.8: Overhead Comparison

	Infrastructure Overhead
E-ECC	• 12.5% storage overhead (25% for Scheme 5) • ECC logic size from 12,793 to 25,215 um^2 • Utilize MCA to mark chip as faulty
V-ECC	• 18.5% storage overhead • 4,375 um^2 of ECC logic size • ECC cache implementation and extra cache size
LOT-ECC	• 26.5% storage overhead • 4,107 um^2 of ECC logic size (only tier-1) • ECC cache implementation and extra cache size • Rely on MCA to record errors and implement spare rows, page retirement and chip replacement to avoid error correction overhead
Multi-ECC	• 12.9% storage overhead • 762 um^2 of ECC logic size (only tier-1) • ECC cache implementation and extra cache size • Read-before-write operation to update tier-2 • Rely on MCA to record errors and implement spare rows, page retirement and chip replacement to avoid error correction overhead

by permanent faults, V-ECC, LOT-ECC and Multi-ECC also rely on MCA to record the errors and use spare rows, page retirement and chip replacement to avoid the overhead of error correction. However, to use spare rows, page retirement and chip replacement, there is significant hardware and software overhead to reroute, remap or retire the data in DRAM. Overall, E-ECC Schemes use more logic die area but have the lowest storage and infrastructure overhead compared to the existing schemes. All these comparisons are given in Table 3.8.

3.6 Summary

We presented five erasure and error correction (E-ECC) schemes that provide superior error protection for commodity DRAMs. All our E-ECC schemes use strong symbol based codes and provide higher error protection compared to existing systems. Furthermore, when a chip is marked faulty, our schemes make use of erasure correction to increase the lifetime of the memory system with no additional cost. Synthesis results show that the decoding latency of these codes is very small and the additional latency does not affect the timing performance of the memory system. Also, our schemes require no extra memory accesses to perform error correction and more importantly, do not require any change in the memory architecture.

Chapter 4

3D DRAM ECC SCHEMES FOR CPU

In this chapter we describe our work on designing an ECC scheme for an HBM-like 3D DRAM system for CPUs. The current ECC schemes either have low decoding latency to correct errors due to small granularity faults and poor detection capability for errors due to large granularity faults or have strong detection capability for errors due to large granularity faults but very long decoding latency to correct errors due to small granularity faults. Our proposed scheme, rate-adaptive two tiered ECC (RATT-ECC), can correct errors due to small granularity faults with low latency and detect errors due to large granularity faults very reliably. In addition, our scheme allows the system to dynamically free up ECC banks as spare banks, when needed. The low decoding latency of our scheme allows the memory system to increase the refresh periods resulting in lower refresh power consumption without sacrificing the reliability and timing performance.

We describe the high level overview of RATT-ECC in Section 4.1. Section 4.2 presents the detailed design of RATT-ECC. The timing performance, reliability and the implementation overhead are analyzed in Section 4.3.

4.1 Overview Of RATT-ECC

We propose **RATT-ECC**, a robust and low overhead ECC scheme that can cope with all types of faults (transient and permanent, small and large granularity faults) as described in Section 2.2.2. Figure 4.1 shows the high level view of RATT-ECC. RATT-ECC is also a two-tiered scheme like Citadel [27] and Efficient-RAS [28]. For each memory read request, the corresponding tier-1 ECC parity symbols are also read

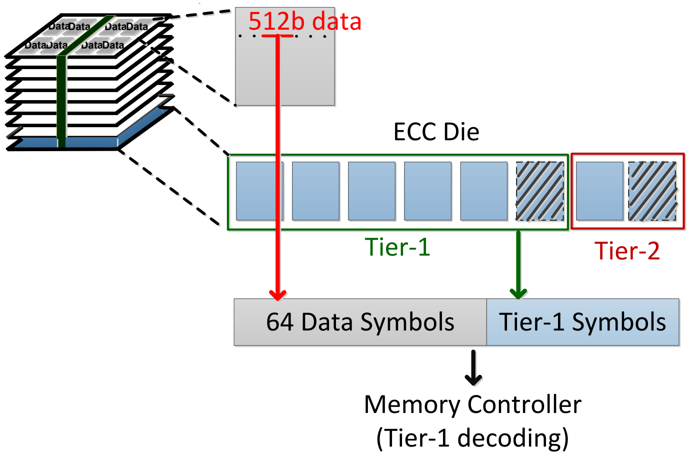


Figure 4.1: RATT-ECC Overview

out and used to perform error correction and detection. The tier-1 code is a strong symbol-based code which can correct all small granularity faults and detect large granularity faults with high probability. The tier-2 code is a XOR based correction code that is used to correct large granularity faults that cannot be corrected by the tier-1 code. Choosing a strong code for tier-1 reduces the number of activations of tier-2 code and also helps reduce error leaks. Furthermore, tier-1 code and tier-2 code can be modified to provide for an extra spare bank when a data bank has been marked as faulty. In the following, we explain how RATT-ECC deals with different types of faults.

4.1.1 Strategies For Handling 3D DRAM Faults

Transient small granularity faults: To correct errors due to transient small granularity faults, we propose to use a stronger tier-1 code compared to Citadel [27] and Efficient-RAS [28]. Our proposed tier-1 code can correct errors due to transient small granularity faults without triggering tier-2 and has the same detection capability as CRC-32 (used in [27]).

Permanent small granularity faults: If the small granularity faults have been marked as permanent by the MCA, we can treat the errors due to these faults as erasures. We use erasure correction to correct errors due to permanent TSV failures. Erasure code is a more efficient way of correcting permanent errors compared to TSV swapping that is used in [27], and will be discussed in Section.

Transient large granularity faults: We rely on the detection of tier-1 code and correction of tier-2 code to recover from errors due to transient large granularity faults. Hence, the detection capability of tier-1 code should be strong to avoid error leaks since tier-2 code only performs correction (not detection).

Permanent large granularity faults: In order to correct errors due to permanent large granularity faults, tier-2 code has to be activated every time. Such a correction process incurs a large timing overhead. We propose relying on spare banks to reduce the decoding latency. For instance, when there is a permanent bank failure, we propose to modify tier-1 or tier-2 code so that one less bank is required to store ECC bits and this bank can now serve as a spare bank.

4.1.2 Design Of Tier-1 And Tier-2 Codes

In order to keep the storage overhead at 12.5%, we use only one extra ECC die to protect eight data dies. Thus, tier-1 and tier-2 parity symbols have to fit into eight banks in the ECC die. Tier-1 code plays a more important role than tier-2 code because tier-1 code needs to have strong error detection and error correction capabilities. If more ECC banks are allocated for tier-2 parity, errors due to large granularity faults would be corrected faster. But this would imply use of a weaker tier-1 code, resulting in weaker correction and detection capabilities, which is undesirable. So we allocate at most 2 ECC banks for tier-2 parity storage.

For tier-1 code, to achieve strong correction and detection capability, we propose using Reed-Solomon (RS) code [5]. If we use RS (72,64) code over GF(2^8), the 12.5% storage overhead budget would go towards supporting this code and there would be no room for tier-2 parity, which is essential to recover from errors due to large granularity faults. Therefore, RS (72,64) is not a suitable candidate. We can choose RS (71,64) or RS (70,64) as tier-1 code since both codes can correct errors due to small granularity faults and detect errors with very high probability (the undetected error rate of both codes is as strong as CRC-32).

If RS (71,64) code is used, seven parity symbols need to be distributed among seven ECC banks (one ECC bank is reserved for tier-2). The minimum number of additional accesses with power-of-two mapping is 3 (three ECC banks store 4 + 2 + 1 parity symbols). This assignment will hurt timing performance and so we only consider RS (70,64) code as our tier-1 code. In this case, tier-1 parity is distributed among six ECC banks and tier-2 parity across two ECC banks.

Finally, we choose RS code because of the embedded structure of these codes which allows for rate adaptation – a feature that is exploited when the memory system has faulty banks. The ECC code rate R is defined as $R = \frac{k}{n}$, where k is the size of information symbols and n is the size of an ECC codeword [5]. For example, the code rate for RS (70,64) code is $\frac{64}{70}$.

As shown in Appendix B, RS (70,64) can be used to derive RS (69,64) and RS (68,64) codes. The encoding of RS (68,64) code is embedded in the RS (69,64) code and the encoding of RS (69,64) code is embedded in the RS (70,64) code. The decoders of these three codes have a similar embedded structure. Thus the codeword of RS (68,64) can be obtained from the codeword of RS (69,64) by removing the last symbol, and the codeword of RS (69,64) can be obtained from the codeword of RS (70,64) by removing the last symbol. Finally, the syndrome vector due to RS (70,64)

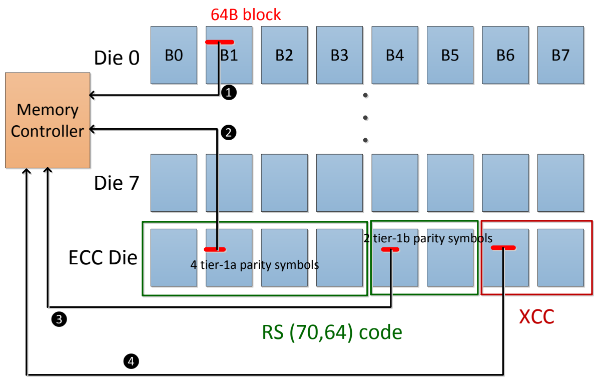


Figure 4.2: Scenario 1: Flowchart For Accessing Data Cache Line, Tier-1 Code And Tier-2 Code

can be used as the syndrome vector of RS (69,64) by removing the last symbol and the syndrome vector due to RS (70,64) can also be used as the syndrome vector of RS (68,64) by removing the first and the last symbols. In the next section we show how this embedded structure can help partition the tier-1 parity symbols into two parts, resulting in fewer memory access without affecting the reliability. Also the embedded structure is used to implement a rate-adaptive tier-1 code that helps a provide a spare bank when needed.

4.2 Details Of RATT-ECC

In this section we describe how RATT-ECC handles the following scenarios: (i) no data banks are marked as faulty (Section 4.2.1), (ii) one data bank is marked as faulty (Section 4.2.2), and (iii) two data banks are marked as faulty (Section 4.2.3).

4.2.1 Scenario 1 - No Banks Are Marked As Faulty

Here, none of the data banks is marked as faulty and so all the eight ECC banks can be used by tier-1 code and tier-2 code. Since we choose RS (70,64) code over

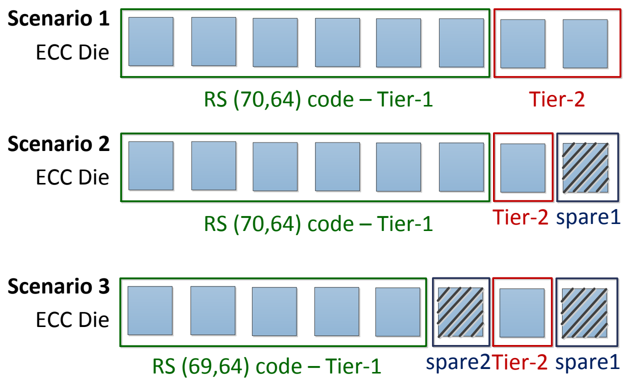


Figure 4.3: 3 Scenarios Of RATT-ECC

GF(2^8) as the tier-1 code, six ECC banks are reserved for storing parity check symbols of tier-1 code; the remaining two ECC banks are used for tier-2 code.

RS (70,64) has a minimum distance of 7 and supports the following error correction and detection cases [5]: (i) detects six errors, (ii) corrects one error and detects five errors, (iii) corrects two errors and detects four errors, and (iv) corrects three errors. Table 4.1 lists the detectable and correctable error (DCE) rate, detectable but uncorrectable error (DUE) rate and silent data corruption (SDC) rate [12, 52] for different types of failures for the four cases. We utilize the formula given in [65] to derive the probability of undetected error or SDC rate [12] of RS (70,64) code.

Case (i) has the highest error detection probability (with SDC rate $3.6 \cdot 10^{-15}$) but cannot correct any errors. Once the errors are detected by tier-1 code, case (i) relies on tier-2 code to recover from the errors. Case (ii) can correct a single error, detect up to five errors and has $2.4 \cdot 10^{-10}$ SDC, while case (iii) can correct double errors, detect four errors and has a higher $7.2 \cdot 10^{-6}$ SDC. Case (iv) can only correct errors without detecting errors, which is a serious problem in case there are row or bank failures. We choose case (ii) over case (iii) due to its stronger error detection probability. Its single error correction capability can deal with errors due to small granularity faults which account for 70% of all faults [26].

Table 4.1: Error Handling Performance Of The Different Decoding Cases of RS (70,64)

Failure Type	Case (i) Detects 6 errors	Case (ii) Corrects 1 error & detects 5 errors	Case (iii) Corrects 2 errors & detects 4 errors	Case(iv) Corrects 3 errors
Single bit	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Single column	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Single TSV	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Double bit/ column/TSV	DCE: 0% DUE: 100% SDC: 0%	DCE: 0.2% DUE: 99.8% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Row or Bank	DCE: 0% DUE: $1 - 3.6 \cdot 10^{-15}$ SDC: $3.6 \cdot 10^{-15}$	DCE: 0% DUE: $1 - 2.4 \cdot 10^{-10}$ SDC: $2.4 \cdot 10^{-10}$	DCE: 0% DUE: $1 - 7.2 \cdot^{-6}$ SDC: $7.2 \cdot 10^{-6}$	DCE: 0% DUE: 0% SDC: 100%

During a memory read request, the 64B block is read from a data bank and its corresponding tier-1 code is read from ECC banks in the ECC die. Since each 64B block is encoded with six parity symbols of tier-1 code, if we put all six symbols in a single row of an ECC bank, it incurs two disadvantages. First, it increases address decoding complexity due to a non power-of-two computation. Also, six symbols cannot fit perfectly if the row size is 2KB or 4KB, resulting in a need for a larger ECC bank size. Second, if that ECC bank fails, it could lead to six symbol errors that cannot even be detected by tier-1 code. Hence, we propose to distribute the six symbols of tier-1 code across multiple banks to support power-of-two address mapping and also higher reliability.

We can distribute the six symbols in one of the following way: (i) 1+1+1+1+1+1, (ii) 2+2+2 or (iii) 4+2, all of which support power-of-two address mapping. In distribution (i), the six symbols are stored in six ECC banks and thus six additional accesses are needed to retrieve tier-1 parity symbols. If a single ECC bank fails, it only leads to one symbol error, which can be easily corrected by tier-1 code. In distribution (ii), the six symbols are stored in three ECC banks and thus each memory request leads to three extra accesses. If an ECC bank fails, it leads to double errors, which can be detected by tier-1 code but require tier-2 code for correction. In distribution (iii), the 6 symbols are stored in two different ECC banks, leading to two additional accesses. Of the 6 symbols, 4 symbols are stored in one ECC bank and another 2 symbols are stored in the second ECC bank. Thus, if an ECC bank fails, it either leads to four symbol errors (first bank) or double symbol errors (second bank). Both error types can be detected by tier-1 code and corrected by tier-2 code.

We choose distribution (iii) since it requires the fewest number of access among the three distributions. The parity symbols of tier-1 code now consist of two parts: tier-1a (four symbols) and tier-1b (two symbols). The tier-2 code is a RAID5-like code based on XOR operations. Since two ECC banks are reserved for storing the parity symbols of tier-2 code, data in a set of 32 data banks are XORed and stored in a single tier-2 bank. For example, data in banks 0 to 3 from dies 0 to 7 are XORed and stored in one tier-2 bank, and data in banks 4 to 7 from dies 0 to 7 are XORed and stored in the second tier-2 bank. When tier-1 code detects errors that it cannot correct, tier-2 code is activated. To recover from errors in a 64B data block, 32 data blocks have to be read from 32 banks.

In each memory read request, tier-1a and tier-1b parity symbols stored in two ECC banks have to be accessed. In order to reduce the number of accesses, we propose to read only tier-1a parity symbols. The 64B data symbols along with the 4 tier-1a parity symbols form a codeword of RS (68,64) over GF(2^8). As described earlier, RS (68,64) is obtained by puncturing RS (70,64) and so the syndrome of this code can be used to detect errors with SDC as low as $2.3 \cdot 10^{-10}$. If errors are detected, then tier-1b parity symbols are retrieved by a second access and RS (70,64) decoder is used to correct the errors.

To reduce the timing performance overhead of these extra accesses, we use a cache to store the ECC bits, referred to as ECC cache. Existing schemes such as [27] or [28] also utilize ECC caches. We cache tier-1a, tier-1b and tier-2 parity symbols in the ECC cache; however, we only check tier-1a for each memory read request. The access flowchart is shown in Figure 4.2.

READ: A 64B block is read from one of the dies (step 1). The MC checks whether tier-1a parity symbols are in ECC cache or not. If it is not cached, MC reads the corresponding ECC symbols of tier-1a from the ECC bank and calculates the syndrome corresponding to RS (68,64) code. If the syndrome vector is zero, no additional accesses for tier-1b are required; otherwise, the ECC symbols of tier-1b have to be accessed. If the ECC symbols of tier-1b are not cached, the MC signals an extra read and this is shown in step 3. The parity symbols of tier-1a and tier-1b and the 64 symbols from data cache line form a codeword of RS (70,64), which can correct single error and detect five errors. If RS (70,64) flags a multiple error event, the MC activates tier-2 code to correct the errors generated by large granularity faults (step 4).

WRITE: When a data cache line has to be written back to memory, the memory controller needs to update the data cache line as well as tier-1a, tier-1b and tier-2 parity symbols. Since tier-1a, tier-1b and tier-2 parity symbols are cached, it can reduce memory traffic contention to the ECC die. The dirty cache lines of tier-1a, tier-1b and tier-2 codes are replaced when they are evicted. Although we have one extra write compared to [27] and [28] for updating tier-1 parity, the extra write operations can be well hidden and our simulation results (see Section 4.3) show that the overhead of this extra write is quite small.

4.2.2 Scenario 2 - 1 Data Bank Is Marked As Faulty

Once the MC marks a bank as faulty, every read access from that bank would require activation of both tier-1 and tier-2 codes. This would result in unnecessary performance overhead. So we propose to use a spare bank to achieve high system reliability without degradation in timing performance.

We have two ways to free up a spare bank from the set of eight banks in the ECC die. We can either modify the rate of tier-1 code or modify the rate of tier-2 code. (a) Modification of tier-1 code: If the tier-1 RS(70,64) code is punctured to RS (69,64) code, five banks are needed to store the tier-1 code (instead of six banks) and so one bank can be used as the spare bank. The tier-2 code is still stored in two banks. Unfortunately, RS (69,64) code has weaker detection capability and thus this modification is not our first choice. (b) Modification of tier-2 code: If only one bank is used to store tier-2 code, then the second bank can be used as the spare; the tier-1 code is still stored in 6 banks. Here data from all 64 banks (instead of 32 banks) have to be XORed and stored in one XCC bank. Since tier-2 code data was previously stored in 2 XCC banks, data in the two XCC banks has to be XORed and stored in one XCC bank. Thus the modification of tier-2 only changes the error correction latency for large granularity faults and so we choose modification (b) over modification (a). Thus in Scenario 2, we still use RS (70,64) code as the tier-1 code. Six ECC banks are still used to store the tier-1 code, one ECC bank is used to store tier-2 code and one ECC bank is utilized as a spare bank.

READ: The sequence of operations is essentially the same as Scenario 1. However, to recover from errors in a block, row or bank failures, 64 blocks, rows or banks have to be accessed instead of 32 in Scenario 1.

WRITE: The write-back operation is the same as Scenario 1.

4.2.3 Scenario 3 - 2 Banks Are Marked As Faulty

If there are two bank failures, we use RS (69,64) code which is obtained by puncturing RS (70,64) code to free up the second spare bank. We choose to perform single error correction and quadruple error detection with $5.9 \cdot 10^{-8}$ SDC as the decoding strategy. Although RS (69,64) code can also be used to support double error detection and triple error detection, the corresponding SDC drops to 1.26×10^{-4}, which is too low for a reliable tier-1 code.

The five parity symbols due to RS (69,64) code cannot be stored in the same ECC bank due to poor storage efficiency. Also, if we store five ECC symbols in the same ECC bank, a single row or bank failure of an ECC bank can lead to five errors and this cannot be detected and corrected by tier-1 code. Hence, we distribute five ECC parity symbols across two ECC banks (four symbols in one ECC bank and the fifth symbol in another ECC bank). This distribution has two advantages: first, tier-1a parity of RS (69,64) is the same as that of RS (70,64). Thus, there is no change in the storage pattern of tier-1a across all three scenarios. Second, the tier-1a continues to have high detection probability. In this scenario, the tier-1b parity symbols need to be mapped from two ECC banks to only one ECC bank; the corresponding overhead is described in Section 4.3.3.

READ: When a 64B block is read from one of the dies, the MC checks whether tier-1a parity symbols are in ECC cache and then sends the 64 + 4 symbols to RS (68,64) unit. If the syndrome vector is not zero, MC reads one tier-1b parity symbol to form the RS (69,64) code. If RS (69,64) decoder flags a multiple error event, MC activates tier-2 code to correct the errors generated by large granularity faults.

WRITE: The write-back operation is the same as Scenarios 1 and 2.

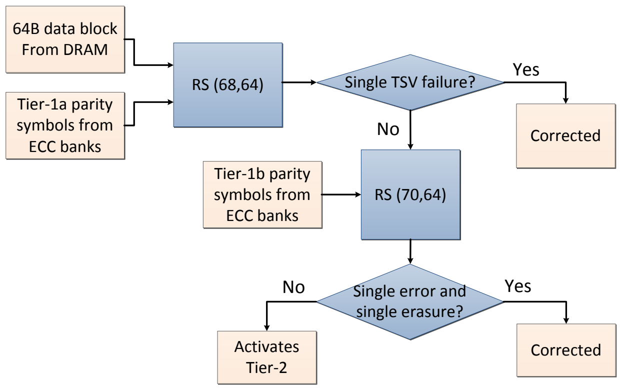


Figure 4.4: Decoding Flowchart Of Erasure And Error Correction Mode Of RS (70,64) Code For Handling Errors Due To A Permanent TSV Failure

4.2.4 Correction Of Errors Due To Permanent TSV Failure

In the organization described in Figure 2.4, a single TSV failure only leads to one symbol error, which can easily be corrected by only tier-1 code. However, both tier-1a and tier-1b parity symbols are needed to perform correction and in the worst case, this leads to two additional memory accesses. Now if the TSV failure is marked as permanent by the MCA, the corresponding symbol can be treated as erasure and the erasure and error correction [5] capability of this code can be used to correct the error. Figure 4.4 gives the decoding flow chart for this case. First, 64 symbols from the data bank and 4 tier-1a parity symbols are read and sent to the RS (68,64) unit (which is embedded in the RS (70,64) unit). The RS (68,64) unit checks whether it is a single erasure event. If it is a single erasure event, the decoder directly corrects it; otherwise, it declares that there are errors (this code can detect up to three extra errors with SDC rate of $5.9 \cdot 10^{-8}$).

Apart from a faulty TSV (that has been marked by the MCA), if there is one more random error, RS (68,64) can detect it. The two tier-1b symbols are then read and sent to the RS (70,64) which now performs single error and single erasure correction,

Table 4.2: Simulation Configuration

Processors	
Core	4 cores, 3.2 GHz out-of-order, 128 ROB
L1 I-cache	4-way, 32KB, 1 cycle; 64B cache line
L1 D-cache	4-way, 32KB, 2 cycle; 64B cache line
L2 cache	8-way, 256KB, 4 cycle; 64B cache line
L3 cache	16-way, shared 4MB, 24 cycles; 64B cache line
3D Stacked DRAM	
Size	8GB, 8 data layers
Bus Frequency	800MHz (DDR3 1.6GHz), 128b per channel
Channels	8 channels per stack
Memory Controller	1 memory controller per channel
Banks	8 banks per channel
Data block size	64B
Scheduling Policy	FR-FCFS
Row Buffer	2KB
tRAS-tCAS-tRP	27-9-9

or single erasure and four error detection (with SDC rate of $5.9 \cdot 10^{-8}$). Hence, the decoder can correct an error caused by a transient fault and an error caused by a permanent TSV fault without activating the tier-2 code. The synthesis results for these units are given in Table 4.8.

4.3 Evaluation

We analyze the timing performance of our proposed scheme with ECC cache implementation in Section 4.3.1, the reliability in Section 4.3.2 and hardware overhead in Section 4.3.3.

4.3.1 Timing Performance Analysis

Simulation Infrastructure & Workloads

To evaluate the performance of our proposed ECC scheme, we utilize macsim [66], a cycle-level x86 simulator. We model a quad-core processor, where each core has a private 32KB(I) cache + a 32KB(D) L1 cache and a unified 256KB L2 cache. The size of the shared L3 cache is 4MB. The memory system uses 3D-stacked DRAM with 8 data dies (1GB per die). The detailed configuration of our system is described in Table 4.2.

We use 17 SPEC CPU2006 benchmarks and simulate a representative set of 250M instructions identified with SimPoint [62]. We categorize the applications into four different groups based on the misses per thousand instructions (MPKI) in the L3 cache. We simulate 4 high memory intensity workloads (MPKI is above 27), 2 high-median intensity workloads (MPKI is between 15 and 27), 4 median intensity workloads (MPKI is between 1 and 15) and 7 low workloads (MPKI is below 1). Table 4.3 summarizes the workload combinations.

ECC Cache Implementation

The proposed ECC cache stores tier-1a, tier-1b, and tier-2 parity symbols. We allocate 50% of the ECC cache to tier-1a, 25% to tier-1b and 25% to tier-2. This distribution is motivated by the fact that in the ECC die, four banks are used to store tier-1a parity symbols, two banks are used to store tier-1b parity symbols and another two banks are used to store tier-2 parity symbols. The ECC cache is housed in the logic die. It is implemented as a directed mapped cache with size varying from 256KB to 2MB.

Table 4.3: Workloads

Mix	Workload	MPKI	Group
WL-1	4 x astar	13.99	M
WL-2	4 x bwaves	30.8	H
WL-3	4 x bzip2	2.28	M
WL-4	4 x dealII	0.77	L
WL-5	4 x gromacs	0.71	L
WL-6	4 x h264ref	1.33	M
WL-7	4 x lbm	45.53	H
WL-8	4 x libquantum	24.97	HM
WL-9	4 x mcf	88.46	H
WL-10	4 x namd	0.1	L
WL-11	4 x omnetpp	23.95	HM
WL-12	4 x perlbench	0.06	L
WL-13	4 x povray	0.02	L
WL-14	4 x sjeng	0.34	L
WL-15	4 x soplex	29.78	H
WL-16	4 x tonto	0.02	L
WL-17	4 x zeusmp	6.42	M

Table 4.4: The Coverage Of Tier-1a, Tier-1b And Tier-2 Parity For Three Scenarios. Each Block Is Of Size 64B.

	Scenario 1	Scenario 2	Scenario 3
tier-1a	Covers 16 blocks	Covers 16 blocks	Covers 16 blocks
tier-1b	Covers 32 blocks	Covers 32 blocks	Covers 64 blocks
tier-2	Covers 32 blocks	Covers 64 blocks	Covers 64 blocks

Table 4.5: Comparison Of The Error Handling Performance Of Tier-1 Codes Used In The Existing Schemes

Error Type	CRC-32 Detect errors	4 x Rotational code Corrects 1 error & detects 2 errors	1 x RS (69,64) Corrects 1 error & detects 4 errors	1 x RS (70,64) Corrects 1 error & detects 5 errors
Single bit failure	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Single column failure	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Single TSV failure	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Row or Bank failure	DCE: 0% DUE: 1 - $2.3 \cdot 10^{-10}$ SDC: $2.3 \cdot 10^{-10}$	DCE: 0 DUE: 97% SDC: 3%	DCE: 0% DUE: $1 - 5.9 \cdot 10^{-8}$ SDC: $5.9 \cdot 10^{-8}$	DCE: 0% DUE: 1 - $2.4 \cdot 10^{-10}$ SDC: $2.4 \cdot 10^{-10}$

Table 4.6: FIT Analysis

		Resultant FIT			
Failure Mode	Raw FIT	Citadel [27]	Efficient-RAS [28]	RATT-ECC Scenario 1 and 2	RATT-ECC Scenario 3
Single bit	320	0*	0	0	0
Single column	70	0*	0	0	0
Single row	80	$1.8 \cdot 10^{-8}$*	2.4*	$1.9 \cdot 10^{-8}$*	$4.7 \cdot 10^{-6}$*
Single bank	100	$2.3 \cdot 10^{-8}$*	3*	$2.4 \cdot 10^{-8}$*	$5.9 \cdot 10^{-6}$*
Single TSV	41	0*	0	0	0
Summary	611	$4.1 \cdot 10^{-8}$	5.4	$4.3 \cdot 10^{-8}$	$1.1 \cdot 10^{-5}$
* means the correction should be performed by tier-2					

In an error-free system, for each memory read request, the MC checks whether the corresponding parity symbols of tier-1a are cached or not. If it is not cached, a read request is launched by MC to retrieve the data from the ECC banks of tier-1a. For each write-back request, MC has to update the data, tier-1a, tier-1b and tier-2 parity symbols. The MC checks whether the parity symbols of tier-1a, tier-1b or tier-2 are cached or not. If these parity symbols are not cached, MC launches the requests to the ECC die.

Timing Performance Of RATT-ECC

We assume that the system is error-free. Also, we only simulate for Scenario 1 since this is projected to have the worst timing performance. The ECC cache entry size is also set to 64B, which is consistent with the rest of the memory system. Figure 4.5 gives the hit rate for different ECC cache sizes. When ECC cache size is larger than 1MB, the hit rate becomes stable at 70%. Figure 4.6 compares the execution time for different ECC cache sizes; the execution time is normalized to the baseline that does not have any ECC scheme. These results show that for ECC cache sizes which are less than 512KB, the execution time is increased by 5%, which is too large. However, if we use ECC cache sizes of 1MB or 2MB, the execution time increases by 2.4% and 1.8% respectively, which is acceptable.

Although we did not perform the simulations for Scenarios 2 and 3, we argue that Scenario 1 captures the worst case timing performance. In Scenario 1, a cache line of tier-1a part of the ECC cache covers 16 blocks of one data bank, a cache line in the tier-1b part covers 32 blocks of one data bank and a cache line in the tier-2 part ECC cache covers 32 data blocks across 32 banks. In Scenario 2, tier-2 part of the ECC cache covers 64 blocks across 64 banks. Thus the hit rate of the ECC cache will be higher in Scenario 2 compared to Scenario 1. Similarly, in Scenario 3, the tier-1b part

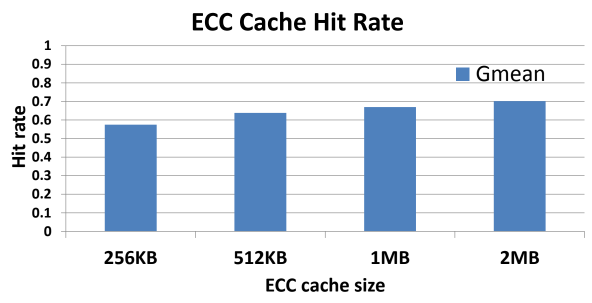


Figure 4.5: Hit Rate Of Different Sized ECC Cache

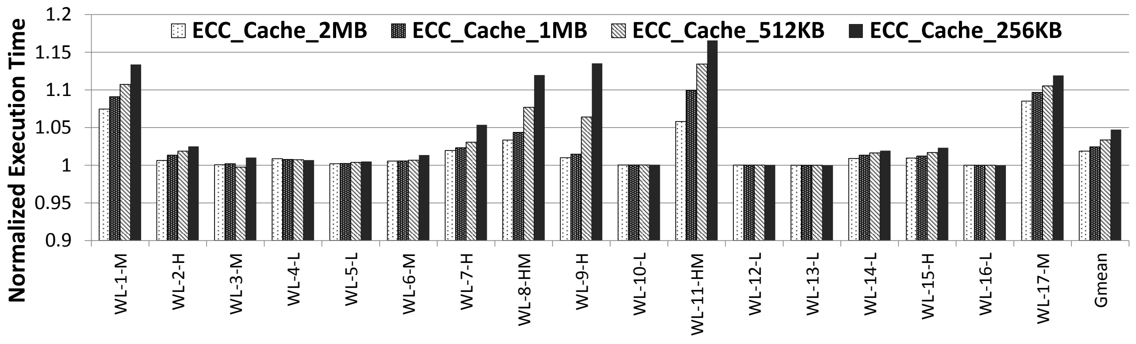


Figure 4.6: Normalized Execution Time For Different ECC Cache Sizes

of the ECC cache covers 64 blocks of a data bank (instead of 32 blocks in Scenario 1) and the hit rate of the ECC cache will be higher. Hence, Scenario 1 will have the worst timing performance followed by Scenario 2 and then Scenario 3.

Refresh Power Evaluation

Current 3D DRAM memories, such as HBM, are 8GB and are projected to become larger in the near future [7]. As the 3D DRAM size increases, the refresh power accounts for a larger part of the DRAM power. In order to reduce the refresh power, we increase the refresh interval and correct the errors caused by the larger interval to maintain the same reliability level as the baseline (64ms). In our simulation, we utilize the error rates due to increase in refresh interval from [67]. Table 4.7 shows the significant increase in the number of single bit errors (retention failures) as the refresh

Table 4.7: Number Of DRAM Cell Retention Errors As A Function Of Refresh Interval In A 60nm Process; The Default Refresh Interval Is 64ms.

	Memory size			
Refresh interval	8GB	16GB	32GB	64GB
128ms	8	15	30	60
256ms	250	500	1000	2000

interval increases for different memory sizes. According to [67], weak cells caused by increasing the refresh interval are randomly distributed throughout the DRAM array. Hence, for each memory read access, there is at most one bit error (among 512 bits), and this error can be corrected by tier-1 code with low decoding latency.

The default refresh rate is set to 64ms (as DDR3), which corresponds to the error free case, and so we normalize all the results to this case. For 8GB DRAM, if we increase the refresh interval to 128ms, there would be 8 additional errors as indicated in Table 4.7. We inject errors in 8 random locations in the memory trace of each benchmark corresponding to the worst case consideration. Similarly, if the refresh interval is 256ms, we inject 250 errors in random locations in the memory trace. We run the simulations 100 times and take the average for each benchmark. The number of additional error corrections for each benchmark is shown in Figure 4.7.

When the memory read access contains data from the leaky cell (single bit error), the memory controller activates the correction operation to correct this error. RATT-ECC relies on tier-1 code to correct this error and the overhead is quite small. The overhead is due to the tier-1 ECC cache miss which has a penalty of 100ns to read data. The syndrome calculation and single error correction takes only 0.52ns and 0.47ns (from Table 4.8), respectively.

Figure 4.8 shows the power reduction of RATT-ECC scheme for refresh intervals of 128ms and 256ms compared to when the refresh interval is 64ms. While the refresh power reduces by 50% when the refresh interval is increased to 128ms and by 75% when the refresh interval is increased to 256ms, the overall power reduction is a lot lower. For instance, the power consumption is reduced by an average of 4.1% (maximum: 7.1% and minimum: 2%) and 7.1% (maximum: 11.2% and minimum: 3.2%) when the refresh interval is 128ms and 256ms, respectively.

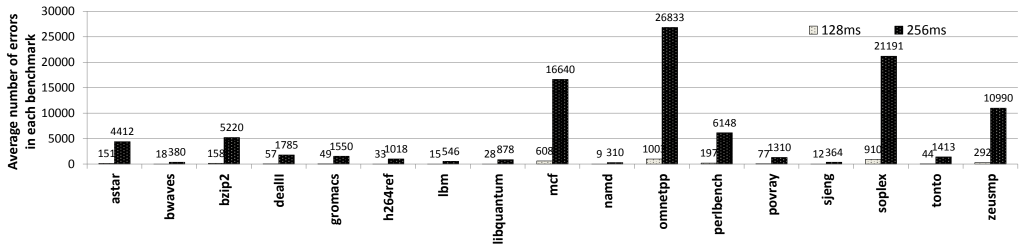


Figure 4.7: Number Of Additional Error Corrections For 8GB DRAM When The Refresh Interval Is Increased.

We also increase the DRAM memory size from 8GB to 64GB and change the corresponding DRAM parameters in the power consumption simulation of DRAMSim2. The result is shown in Figure 4.9. For memory size of 16GB, 32GB and 64GB, the power consumption reduces by an average of 11.9%, 14.6% and 16.3% if the refresh interval is 128ms, and reduces by an average of 15.1%, 17.6% and 21.6%, if the refresh interval is 256ms. Thus for large memory size, the total power consumption reduces significantly and since all additional single bit errors are corrected by tier-1, this is achieved with negligible loss in timing performance.

Finally, we also compare the timing and power performance of RATT-ECC to E-RAS [28], Citadel and [27] for larger refresh intervals. When the refresh interval is increased, RATT-ECC and E-RAS have very little overhead in timing performance, since they can all correct the additional single bit errors using the tier-1 code. How-

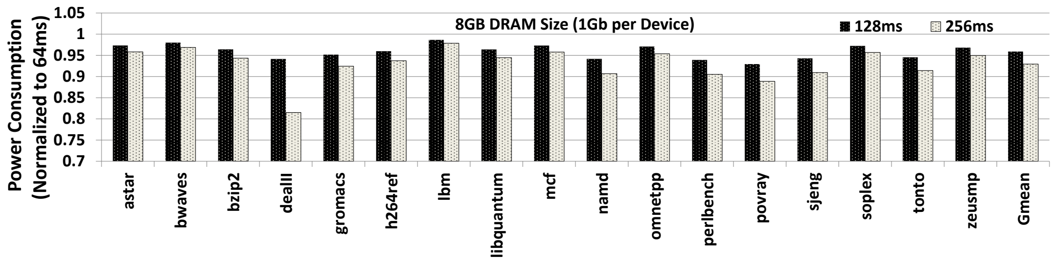


Figure 4.8: Power Reduction Of 8GB Memory For Different Refresh Intervals

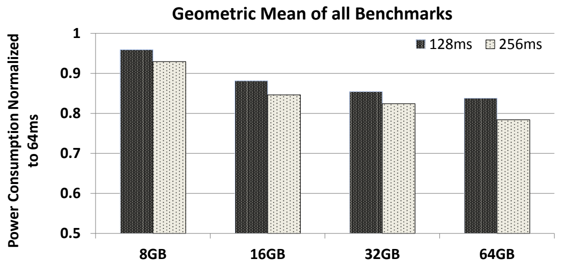


Figure 4.9: Power Reduction Of 8GB To 64GB Memory For Different Refresh Intervals

ever, Citadel takes 0.7s to correct every single bit error, which increases its total execution time and power consumption significantly.

4.3.2 Reliability Comparison

In this section, we compare the reliability of RATT-ECC with respect to the existing methods, namely, Citadel [27] and Efficient-RAS [28]. For tier-1 code, [27] uses CRC-32 to detect errors and there exists a shortened CRC-32 code with data length 512 bits that achieves that the upper bound (2^{-32}) of undetected error probability. CRC-32 code can detect all errors due to the small granularity faults and can detect the errors caused by the large granularity faults with SDC smaller than $2^{-32} \approx 2.3 \cdot 10^{-10}$.

The scheme is [28] uses a symbol-based code (rotational code in [12]) to correct errors due to small granularity faults and detect errors due to large granularity faults which are then corrected by XCC. The symbol-based code uses four bits/symbol and can perform single symbol correction and double symbol detection (SSC-DSD). Since there is no theoretical bound for undetected error probability for this code, we did Monte Carlo simulations to derive the corresponding SDC rate for large granularity faults. The number of trials is more than 10M. We assumed that each bit in a single cache line has 0.5 probability to be flipped. We found that its SDC rate is around 3%, which is too high for a reliable tier-1 code.

CRC-32 has good detection capability but cannot correct any errors while rotational code has good correction capability but poor SDC. RS (70,64) that is used in Scenarios 1 and 2 has strong correction and detection capabilities. Although we only read the first 4 symbols for detection, the corresponding RS (68,64) code still has a very low SDC of $2.3 \cdot 10^{-10}$. Once RS (68,64) code detects errors, other two ECC symbols are read and RS (70,64) decoder is activated. It corrects all errors caused by the small granularity faults and detects errors caused by large granularity faults with $2.4 \cdot 10^{-10}$ SDC. RS (69,64) code, which is used in Scenario 3, can correct one symbol error and detect four symbol errors and has slightly higher SDC ($5.9 \cdot 10^{-8}$). We summarize the DCE, DUE and SDC rates of the CRC-32 code (used by [27]), rotational code (used by [28]), RS (70,64) (RATT-ECC Scenario 1 and 3) and RS (69,64) (RATT-ECC Scenario 3) in Table 4.5.

Transient small granularity faults: All three schemes can correct errors due to transient small granularity faults. Since [27] uses tier-2 code to correct these errors, it has a higher performance overhead.

Permanent small granularity faults: Here, we only discuss errors due to permanent TSV failures. Such failures are handled using spare TSV in [27]. The method in [28] can directly correct the errors due to a single TSV failure, however, it suggests using spare TSV to avoid correction overhead. In our scheme, we use erasure correction capability of the RS code to directly correct these errors by only using tier-1a. The latency of correction is quite small (0.33ns) since correcting an erasure is much easier and faster than correcting an error (in Table 4.8).

Transient large granularity faults: All schemes rely on tier-1 code to detect the errors caused by large granularity faults and launch tier-2 for correction. [28] has significantly higher SDC rate, making it less reliable compared to the others.

Permanent large granularity faults: Large granularity faults tend to have a bimodal distribution [27]. There are either a few row failures (less than four rows) or a large number of row failures (more than 4K rows) in a single bank. To handle the case when there are a few row failures in a single bank, we also allocate some extra spare rows per bank in the logic die, similar to [27] and [28].

When multiple rows develop faults, the bank is marked as a faulty bank by MCA. To handle errors due to faulty banks, [27] allocates two spare banks. [28] considers permanent TSV or row failures but not permanent bank failures. However, permanent bank failures cannot be neglected and spare banks are the most efficient way to avoid the burden of error correction. RATT-ECC adjusts the rates of tier-1 and tier-2 code to free up ECC banks to be used as spare banks.

FIT Analysis: Real-world field data from [4] provides DRAM FIT rate for 1Gb 2D DRAM device. Since field data are not available for 3D DRAM, we used the numbers from [28], which conservatively projects that 3D DRAM has 10x higher error rates than 2D DRAM. We simply borrowed the raw FIT values from [28] and used them to compute the final FIT rates.

Table 4.6 presents the final FIT rate of the competing schemes. All schemes can completely remove the errors due to small granularity faults. RATT-ECC (Scenario 1 and 2) and [27] can reduce the raw FIT rates to around $4 \cdot 10^{-8}$, which improves more than 10^9x compared to the baseline (no ECC) scheme. RATT-ECC Scenario 3 also reduces the total raw FIT rate to $1.1 \cdot 10^{-5}$. However, due to the poor detection capability of rotational code, [28] only decreases the overall raw FIT rate to 5.4.

4.3.3 Hardware Overhead Of RATT-ECC

The overhead of RATT-ECC scheme consists of ECC decoding units, ECC cache and the modification required to support rate-adaptive codes.

***ECC Decoding Units*:** We synthesized the syndrome calculation and the error correction units for rotational code [12] and RS (70,64) code using 28nm industry library. Syndrome and error correction units for RS (69,64) and RS (68,64) have not be designed separately. Instead, the RS (70,64) units are used and small additional logic is used to support the punctured codes (RS (69,64) and RS (68,64)). Table 4.8 gives the area, latency and power information for rotational code and RS (70,64). Since syndrome calculation is on the critical path of memory read, its decoding latency is made as small as possible. The decoding latency of syndrome calculation for a single rotational code is 0.41ns; since 4 rotational codes are used per channel, the area increases to 8000 um^2. The decoding latency of syndrome calculation of RS (70,64) code is slightly higher at 0.52ns and its area is 11,337 um^2.

Considering an average latency of a memory read is around 30ns-40ns [24] and the lower clock frequency of L3 cache (1ns), the latency of syndrome calculation of RS (70,64) code incurs only one extra cycle in L3 cache or DRAM cycle. Since the logic die area of the existing computer system is around 200-500 mm^2, the size of the ECC unit consumes a negligible extra area in the logic die.

***Rate-Adaptive Code Implementation*:** The 64 data symbols and 4 symbols of the tier-1a code are decoded using the RS (68,64) decoder, which is embedded in the RS (70,64) decoder. If an error is detected, 2 additional symbols are read and

Table 4.8: Synthesis Results Of Existing ECC Schemes

	Area	Latency	Power
Rotational code Syndrome Calculation	2,000um^2	0.41ns	0.0083 mW (Static) 14.5 mW (Dynamic)
Rotational code SSC-DSD	2,760um^2	0.41ns	0.0015 mW (Static) 12.5 mW (Dynamic)
RS(70,64) Syndrome Calculation	11,337um^2	0.52ns	0.0645 mW (Static) 36.345 mW (Dynamic)
RS(70,64) Single Error Correction	2,024um^2	0.47ns	0.0195 mW (Static) 22.53 mW (Dynamic)
RS(70,64) Single Erasure Correction	1,336um^2	0.33ns	0.0144 mW (Static) 25.62 mW (Dynamic)
RS(70,64) Single Erasure & Single Error Correction	5,658um^2	1.01ns	0.0271 mW (Static) 21.44 mW (Dynamic)

sent to the RS (70,64) decoder. Our hardware implementation has only one main RS (70,64) decoder and a small control unit to support the other two decoders.

In Scenarios 2 and 3, data in two ECC banks of tier-2 (from Scenario 1) are XORed and stored in one ECC bank in Scenarios 2 and 3. This step can be pipelined with the storage of new data in the spare bank. However, when we move from RS (70,64) to RS (69,64) to free up a bank, only one of the two tier-1b parity symbols from RS (70,64) have to be stored. Reorganizing the data in the new tier-1b ECC bank may take time and so we propose to use tier-1a code for detection and tier-2 code for correction during that time. While activating tier-2 for small granularity faults has a performance overhead, the reliability of our ECC scheme is not compromised.

***ECC cache*:** We consider direct mapped ECC cache with sizes ranging from 256KB to 2MB. The ECC cache size could be reduced if the cache supported set associativity.

4.4 Summary

We presented a two-tiered ECC scheme, RATT-ECC, where the first tier is used to handle errors due to small granularity faults (single bit, column and TSV) and the second-tier is used to handle errors due to large granularity faults (row or bank failures). We chose a strong symbol based RS code in tier-1, namely, RS (70,64) code because of its ability to correct errors due to small granularity faults and reliably detect errors due to large granularity faults. This code can also be punctured and the resulting RS (69,64) and RS (68,64) codes share the same encoder and decoder as the RS (70,64) code. The punctured codes can also be used to free up a ECC bank if a data bank is faulty or help store and decode tier-1 parity symbols in a way that reduces the number of memory accesses. The tier-2 code is an XCC based code which can also be modified to free up a ECC bank. The simulation results with SPEC2006 benchmarks show that the average performance overhead is less than 2%. Due to the strong error correction capability and low latency of decoding, RATT-ECC can be used to reduce the refresh power without sacrificing reliability and performance.

Chapter 5

3D DRAM ECC SCHEMES FOR GPU

In this chapter, we propose **Config-ECC**, a configurable ECC scheme for HBM systems that service GPUs. The ECC design for a GPU system is more challenging than the ECC design for a CPU system. Most CPU systems have fixed cache line size of 64B while the cache line size in today's commercial GPU products have cache line sizes ranging from 32B to 128B. Some GPUs allow the cache line size to be configured before the application starts while others allow change in the configuration at runtime to achieve even better performance. A fixed ECC scheme cannot be used for the varied sized accesses from GPU applications due to the mismatch of the GPU request and the fixed ECC scheme. This results in either lower reliability or higher energy consumption. Hence, we propose a configurable ECC scheme to tackle this mismatch problem.

Section 5.1 gives the motivation for this work. Section 5.2 gives the high level overview of our proposed Config-ECC and Section 5.3 presents the detailed design for 32B access. The 32B access scheme can be extended to support different sized accesses as explained in Section 5.4. The reliability and simulation infrastructure are given in Section 5.5 and the results are shown in Section 5.6.

5.1 Motivation For An ECC Design Supporting GPU's Varying Data Fetch Sizes

The cache line size in today's commercial GPU products is not fixed at 64B as in most CPU systems. For example, the cache line size of NVIDIA Fermi GPUs [68] is 128B, the cache line size of Intel Gen9 GPUs and AMD Graphics [36, 69] is 64B, and the data fetch granularity of NVIDIA Maxwell GPUs [70] is 32B with two different

L2 cache line sizes of 32B and 128B. Different applications favor different cache line sizes and we anticipate that GPU configurations with multiple cache line sizes will be used in future data centers.

Consider the device driver of NVIDIA Tesla GPU which can be modified to launch kernels with two cache configurations before the start of an application. When the L1 cache is turned on, the L1 as well as for the L2 cache lines are 128B. When the L1 cache is turned off, the data fetch size of the L2 cache becomes finer-grained 32B. Figure 5.1 shows the performance characterization results for GPGPU applications running with different cache configurations on a Nvidia Kepler-based GPU. The blue bars represent the performance of our baseline system with the L1 cache enabled, as such, the L2 cache operates at the 128B data fetch granularity whereas the black bars represent the performance speedup when the L1 cache is disabled, resulting a finer-grained 32B data fetch size. Applications in the first category prefer a finer data fetch granularity (32B) while other applications prefer a coarse-grained data fetch size (128B) because of the relative well-exploited spatial locality. Choosing the right configuration results in not only better performance but also lower energy consumption.

Also, within a GPU kernel, optimal data fetch granularities can vary as well. A strong correlation between an issuing instruction, its degree of memory divergence and the optimal data fetch size was demonstrated in [38]. Also [39] showed that a dynamic data fetch granularity design can significantly improve the performance and energy efficiency of GPUs. Our own characterization study (Figure 5.2) illustrates a similar trend for a large set of GPGPU applications and each color of the bars indicates the portion of the 128B cache line before its eviction. We can observe that, within an application, the data access pattern can be mixed, demanding varying size of data. Since the optimal data access granularity changes during the kernel execution at

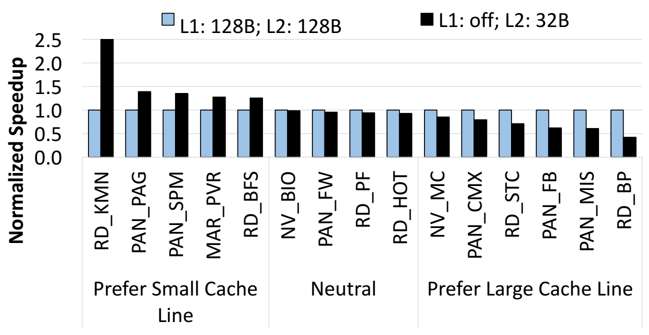


Figure 5.1: The Normalized Execution Time Speedup With Different Two Cache Configurations: 128B Cache Line Size For Both L1 And L2 Caches Versus L1 Cache Disabled And 32B Cache Line Size For The L2 Cache (Provided By Shin-Ying Lee).

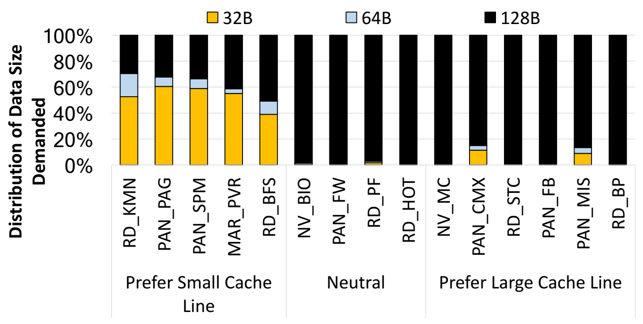


Figure 5.2: The Distribution Of Number Of Bytes Referenced During A Cache Line's Lifetime (Provided By Shin-Ying Lee).

runtime, an optimized GPU design needs to be able to predict and apply the optimal memory configuration to improve resource utilization and performance of the GPGPU application.

Traditional DRAM ECC algorithms are developed and optimized for fixed sized data accesses. Applying an ECC algorithm that is optimized for a certain data fetch size causes configuration mismatch with GPU cache line size, that leads to sub-optimal reliability and larger energy consumption. Here, we use Figure 5.3 to explain the mismatch. An ECC scheme optimized for 32B data access (Figure 5.3(a)) can support 32B, 64B and 128B accesses without additional energy consumption.

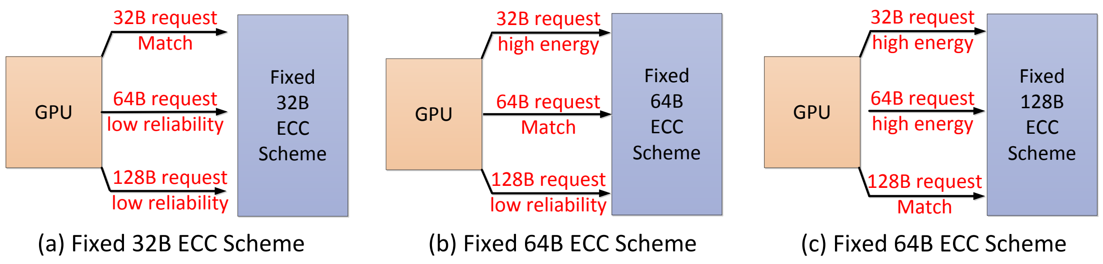


Figure 5.3: Interaction Between Different GPU Access Request Sizes And Fixed Sized ECC Schemes

However, since 64B and 128B requests are processed by a 32B ECC scheme, the reliability is lower than if they were processed by a 64B or 128B fixed ECC scheme. An ECC scheme designed for 64B data access (Figure 5.3(b)) can support 64B and 128B accesses without additional energy overhead but the 128B request has lower reliability compared to if it were processed by an fixed 128B ECC scheme. For 64B ECC scheme, 32B request leads to 64B access from HBM, resulting in an additional 32B read and consuming more energy. Although 64B ECC scheme provides stronger reliability compared to a 32B ECC scheme, it forces the system to read 64B, resulting in higher energy. Similarly, an ECC scheme designed for 128B data access (Figure 5.3(c)) can support 128B requests without additional energy consumption but a 32B or a 64B request forces it to always read 128B data. This overfetch consumes significantly more energy. Thus these are mismatches between GPU data fetch sizes and ECC algorithms, referred to as *configuration mismatches*.

To address the issues due to configuration mismatch, we propose Config-ECC, a mechanism to provide a strong reliability guarantee while minimizing data fetch energy consumption for the GPU memory subsystem for a varying granularity of data accesses. The proposed Config-ECC provides two operation modes:

- **Static Operation Mode** which tackles the mismatch issue by providing ECC protection for small or large data access granularity at the application level.
- **Dynamic Operation Mode** which tackles the mismatch issue by interleaving small and large sized ECC protection for supporting mixed locality behavior during runtime.

5.2 Overview Of Config-ECC

Config-ECC operates in two modes: the dynamic mode configuration is shown in Figure 5.4 and the static mode configuration is shown in Figure 5.5.

In the dynamic mode (see Figure 5.4), for 32B accesses, we only use the inner code to protect 32B of data and use the outer code to correct errors when the inner code detects errors. For 64B accesses, both inner code and outer code are used, resulting in better reliability compared to the 32B accesses. For 128B accesses, two separate 64B decoding units are used resulting in the same protection capability as 64B accesses.

In the static mode (see Figure 5.5), we add interleavers to increase the burst error detection capability. Recall that large granularity faults manifest as burst errors and adding interleavers is a low cost method of improving burst error detection capability. For 64B accesses, an interleaving unit is used to spread the erroneous bits in two 256b of data to two inner code decoding units. Similarly, for 128B accesses, the interleaving unit is used to spread the erroneous bits of four 256b of data to four inner code decoding units.

5.2.1 Design Of Tier-1 Code

In our system, the tier-1 code has the following requirements. First, it should have strong detection capability for different types of accesses. Specifically, it should detect all errors due to small granularity faults and have very low SDC rates for errors

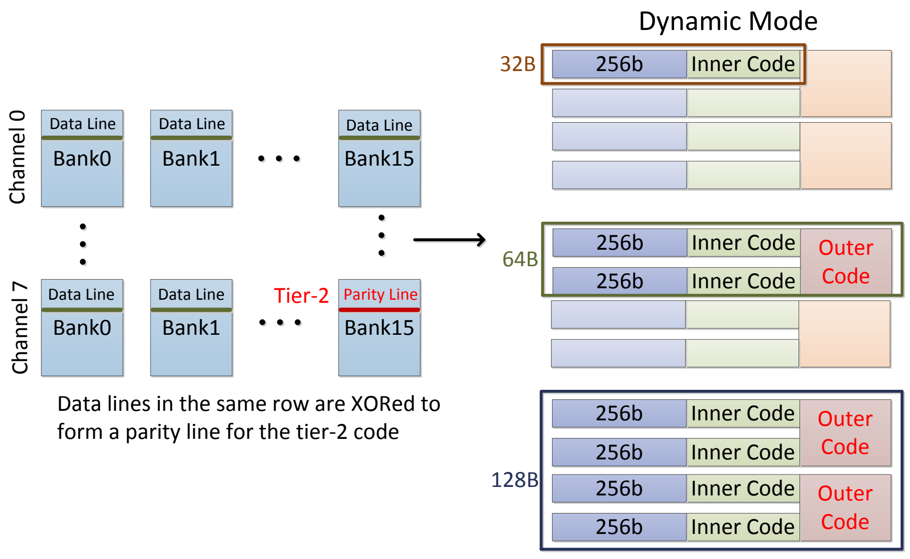
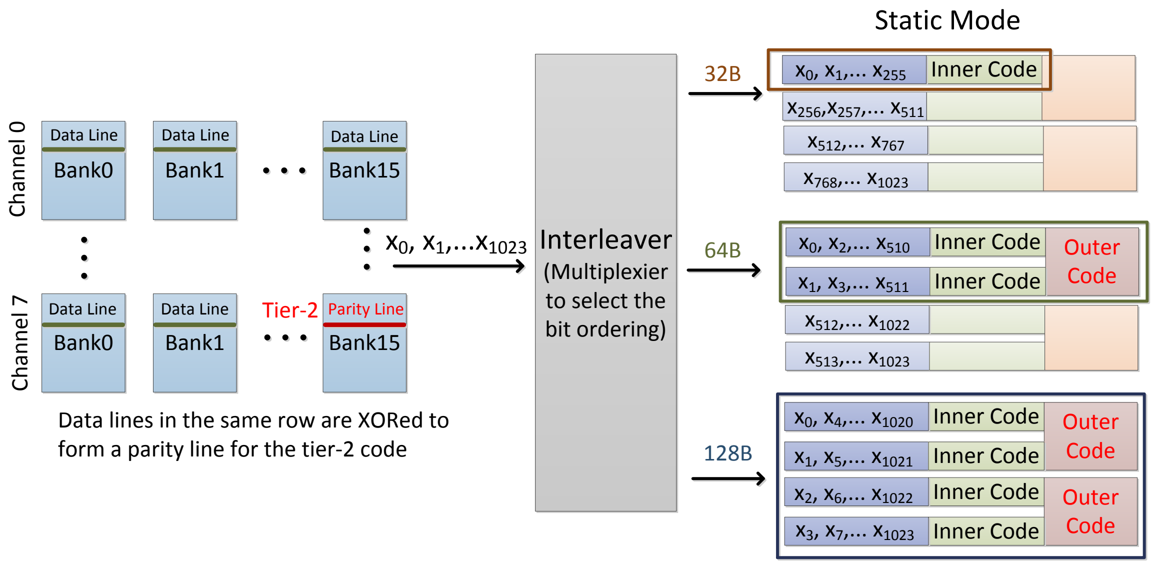


Figure 5.4: Overview Of Config-ECC In The Dynamic Mode.



Static Mode
Channel 0
Data Line
Bank0
Data Line
Bank1
Data Line
Bank15
Channel 7
Data Line
Bank0
Data Line
Bank1
Tier-2
Parity Line
Bank15
$x_0, x_1, \ldots x_{1023}$
Interleaver (Multiplexier to select the bit ordering)
Data lines in the same row are XORed to form a parity line for the tier-2 code
32B
$x_0, x_1, \ldots x_{255}$
Inner Code
$x_{256}, x_{257}, \ldots x_{511}$
$x_{512}, \ldots x_{767}$
$x_{768}, \ldots x_{1023}$
64B
$x_0, x_2, \ldots x_{510}$
$x_1, x_3, \ldots x_{511}$
$x_{512}, \ldots x_{1022}$
$x_{513}, \ldots x_{1023}$
Outer Code
128B
$x_0, x_4, \ldots x_{1020}$
$x_1, x_5, \ldots x_{1021}$
$x_2, x_6, \ldots x_{1022}$
$x_3, x_7, \ldots x_{1023}$

Figure 5.5: Overview Of Config-ECC In The Static Mode.

due to large granularity faults. This feature is very important since the tier-2 code is only activated when the tier-1 code detects errors. Thus, the tier-1 code has to offer a strong detection capability.

Second, the tier-1 code should correct all errors due to small granularity faults. Recent work on DRAM reliability indicates that errors due to small granularity faults account for more than 70% of the total faults [4, 25]. Since the tier-2 code has larger latency and consumes more energy to correct errors, errors due to small granularity faults should be corrected by the tier-1 code. Third, the storage overhead of the tier-1 code should be constrained to 12.5%, as indicated in the HBM standard [33].

Since the ECC storage overhead is 12.5% for the tier-1 code design, each 32B (=256 bits) of data is protected by 32 ECC bits. These 32 ECC bits are stored along with the data in the same bank. Of these 32 bits, N parity bits are reserved for the inner code and the remaining $(32-N)$ bits are reserved for the outer code. The choice of whether more bits should be allotted to inner code or outer code is dictated by the following consideration. If we put more bits in the inner code, it would have strong detection capability but the outer code might not be able to support correction. On the other hand, if we put more bits in the outer code, the inner code might have weak detection capability for errors due to large granularity faults and might lead to miss-error events. So we choose to allot more ECC bits to the inner code to support strong detection capability.

Inner Code: For each set of 32 ECC bits, the N parity bits can be designed to support correction or detection of 32B data. If the inner code is designed to provide for both correction and detection, its overall detection strength would be lower. Note that a linear block code with minimum distance, d_{min}, can be designed to perform (i) correction only (corrects up to $\lfloor \frac{d_{min}}{2} \rfloor$ errors), (ii) detection only (detects up to d_{min} - 1 errors) or (iii) detection and correction (corrects λ errors and detects l errors where $\lambda + l + 1 \leq d_{min}$ and $\lambda < l$) [5]. Here we choose to use the N parity bits for only detection, so that the tier-1 inner code has strong detection capabilities. Designing the inner code for strong detection guarantees that errors due to small and large granularity faults be detected, producing very low SDC rates. If the inner code detects errors, the tier-1 outer code can be used to correct errors due to small granularity faults and the tier-2 code can be used to recover errors due to large granularity faults.

Outer Code: Of the 32 ECC bits, if N bits are allotted for tier-1 inner code, we only have $32 - N$ bits for the outer code. If N is (say) 24, there are not enough bits to design an outer code with the desired detection and correction capabilities. So we combine ECC bits associated with two sets of 32B data. Specifically, we combine $2 \times (32 - N)$ ECC bits to form the parity bits of the outer code.

The proposed outer code corrects all errors due to small granularity faults such as single bit, single column and single TSV faults. Single bit and single column failures lead to 1 bit error in a 32B access. On the other hand, a single TSV failure could lead to 4 bit errors in a 32B access. So, if bit-level correction code is used for the outer code, it would need to have the ability to correct 4 bit errors. However, if we align the TSV bits in a symbol (ex. 8 bits per symbol), a TSV failure would only lead to one symbol error in a codeword and a single symbol error correction code could be used to correct these errors. Therefore, we choose symbol based code as the outer code to handle correction of errors due to single bit, single column and single TSV failures.

5.2.2 Tier-2 Code

Once the errors in the data line are flagged as errors due to large granularity faults (that cannot be corrected by the tier-1 code), the tier-2 code is invoked. The tier-2 code corrects errors due to large granularity faults caused by a single row/bank failure in a die-stacked DRAM. The tier-2 code is based on an XOR-correction code as used in [27,28,31,32]. Specifically, 126 data lines and 1 parity line in the same row location in other data banks are read out and XORed to generate the correct data line. The tier-2 code has an additional storage overhead of 1/127.

Since the tier-1 code can correct errors due to small granularity faults, a simple XOR-correction code is efficient for correcting errors due to large granularity faults such as a single row or single bank failure. To handle errors due to multiple row or bank failures, Config-ECC can be extended to support the 3 dimensional parity scheme in [27] or helix-parity storage scheme in [31].

5.3 Details Of 32B Config-ECC

This section describes the detailed design of our ECC scheme for fixed 32B data fetch size HBMs. We first describe the design of the inner code in Section 5.3.1 and the design of the outer code in Section 5.3.2. Section 5.3.3 summarizes the error detection and correction operations and the decoding flowcharts for 32B accesses.

5.3.1 Design Of the Tier-1 Inner Code

For a strong error detection capability, Reed-Solomon (RS) [65] symbol based code or cyclic redundancy code (CRC) [71] are both good candidates for the tier-1 code. Both RS and CRC codes can provide strong detection capability for burst errors as well as multiple random bit errors (large granularity faults). However, these two codes have different detection capabilities.

Consider two candidate ECC codes that use 24b ECC to protect 256b data, namely, RS (35,32) code in GF(2^8) and CRC-24. RS (35,32) can detect three symbol errors within 35 symbols (8 bits per symbol). However, it cannot guarantee detection

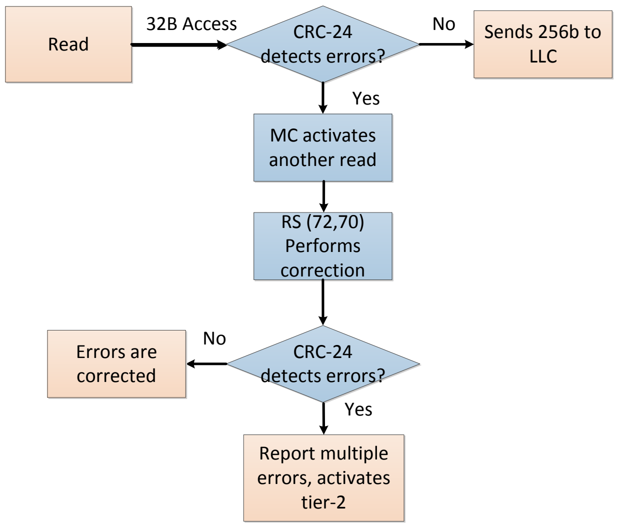


Figure 5.6: Decoding Flowchart for 32B Access

of all 24-bits consecutive bit errors in a 280 bit (35×8) codeword. For example, if 24 consecutive bit errors fall into four consecutive symbols in the RS (35,32) codeword, the RS code cannot fully detect four symbol errors. It can detect up to 17 burst errors.

On the other hand, CRC-24 can detect any 24 consecutive bit errors in the 280 bit codeword. CRC-24 has minimum distance of 6 [72] and so it can detect up to 5 random bit errors. Five random bit errors could lead up to 5 random symbol errors that the RS code cannot detect and thus CRC-24 has stronger detection capability for this error event. When the number of errors is larger than what the CRC-24 or RS (35,32) code can detect, their theoretical silent data corruption rate approaches 2^{-24} as indicated in [73]. Furthermore, CRC provides more flexibility than a RS-based code. For instance, since the data size is 256b,a RS-based code has to be designed over finite field GF(2^8) or larger, resulting in large hardware overhead. In contrast, there is a large set of CRC codes that are available in [74] and so for a fixed data size and storage overhead, a CRC code with the appropriate protection level can be chosen more easily.

Therefore, for the tier-1 inner code, we choose to use the CRC-24 code to perform detection. We select the CRC polynomials for CRC-24 from [72, 74]. We choose the CRC polynomial 0xBD80DE, which has the largest minimum distance of 6 among all CRC-24 codes and also detects all odd number of errors. The largest minimum distance guarantees the strongest detection capability for random bit errors.

5.3.2 Design Of Tier-1 Outer Code

The outer code is designed to perform correction using a symbol-based code, as outlined in Section 5.2. For low storage overhead with strong correction capability, we propose to use a RS-based code to perform correction and detection as the tier-1 outer code.

Since there are 32 ECC bits to protect each group of 32B data and the inner code is CRC-24, then there are only $32 - 24 = 8$ bits to design the outer code, which is not enough even to correct a single bit error. For instance, (265,256) shortened Hamming code from (511,502) Hamming code needs 9 parity bits to perform the single bit error correction. So as mentioned in Section 5.2.1, we combine the parity bits of two 32B accesses to form the parity bits of the outer code. Thus, we now have $2 \times (32 - 24) = 16$ bits for the outer code. The errors due to small granularity faults might occur in the data bits as well as the CRC parity bits, so the outer code protects both data bits and CRC parity bits. We choose the RS (72,70) code over finite field GF(2^8) as the outer code. RS (72,70) can be used to provide single error correction because its minimum distance is 3 [5].

5.3.3 Error Correction & Detection

The flowchart for 32B accesses and the corresponding ECC decoding steps are summarized in Figure 5.6. The decoding process is as follows: (i) if the CRC-24 code reports error-free, the RS decoder is not launched, (ii) if CRC code reports errors (the first pass), an additional read is issued to retrieve the second 256b data and the corresponding 32b ECC and the RS (72,70) decoder performs single error correction. After correction, the two CRC-24 decoders check for errors again (the second pass). If errors are detected, the tier-2 code is launched to correct these errors. Next we analyze the error correction and detection capability of the 32B access scheme.

Errors Due to Small Granularity Faults: CRC-24 successfully detects errors due to a single bit failure and a single column failure. A single TSV failure results in 4-bit error, which can be detected by one CRC-24 since CRC-24 has minimum distance of 6. A small granularity fault only leads to one symbol error in a RS codeword, which can be corrected.

Errors Due to Large Granularity Faults: After the RS (72,70) decoder performs correction, two CRC-24 codes recheck the CRC bits. If any one of the two CRC-24 codes flags errors, the tier-2 code has to be activated to correct errors due to large granularity faults.

Silent Data Corruption: There are two situations when the tier-1 code suffers from silent data corruption. First, when the CRC decoder detects errors, the RS code performs error correction and the two CRC codes miss-detect. If the RS code performs correction, the two CRC codes recheck the decoded data again. If two CRC codes miss-detect the errors, an SDC event occurs.

Second, the CRC miss-detects errors at the first detection; RS(72,70) will not be launched and the RS decoder cannot help in this case. Hence, it is very important that tier-1 inner code to have very strong detection capabilities.

5.4 Extensions Of Config-ECC

In this section, we explain how our 32B ECC scheme can be easily extended to support Dynamic Operation Mode (Section 5.4.1) and the Static Operation Mode (Section 5.4.2) .

5.4.1 Dynamic Mode Protection

64B Access Size

In this case, two 32B of data and two 32b of ECC are read out from two sub-banks and sent to the tier-1 decoder. The tier-1 decoder checks for errors using CRC-24 decoder, corrects single symbol error using RS (72,70) and launches tier-2 if errors are detected.

The decoding steps for 64B access are as follows: (i) if only one CRC-24 code reports errors, the RS (72,70) decoder performs single error correction, followed by CRC-24 detection as in the 32B access scheme, (iii) if two CRC codes report errors, RS decoder is not activated and the tier-2 decoder is launched. (iii) if two CRC-24 codes report error-free, the RS (72,70) decoder performs double error detection.

If two CRC codes report errors at the same time, it means that the errors must not have been caused by the small granularity faults and so tier-2 code has to be triggered. The decoding case (iii) is different from the 32B access scheme where the RS (72,70) decoder was used for single symbol correction. Since RS (72,70) code can perform either single error correction or double error detection, once two CRC codes report error-free, the RS decoder activates double error detection to avoid possible

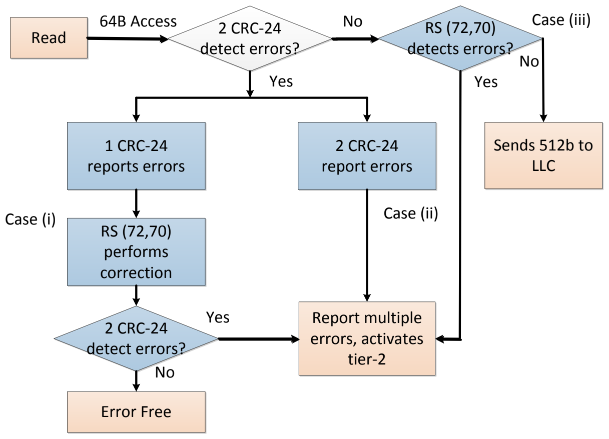


Figure 5.7: Decoding Flowchart For 64B Access

miss-error detection by two CRC codes. The decoding flow chart is presented in Figure 5.7. Our analysis of the error performance of the 64B access scheme is as follows.

No Errors: When there are no errors, 2 CRC-24 codes and RS (72,70) report error free and the memory controller sends the 512b to the lower level cache.

Errors Due to Small Granularity Faults: A single bit failure and a single column failure lead to one bit error which can be fully detected by one CRC-24 code. A single TSV failure results in 4 bit errors in one CRC-24 code and since CRC-24 has minimum distance of 6, the CRC-24 code can fully detect this error event. Thus, when there is a small granularity fault, only one CRC-24 code report errors. The small granularity fault only causes one symbol error in an RS codeword, and can be fully corrected.

Errors Due to Large Granularity Faults: When two CRC-24 codes declare errors, tier-2 code is launched. This is because the number of errors is beyond the correction capability of the RS code.

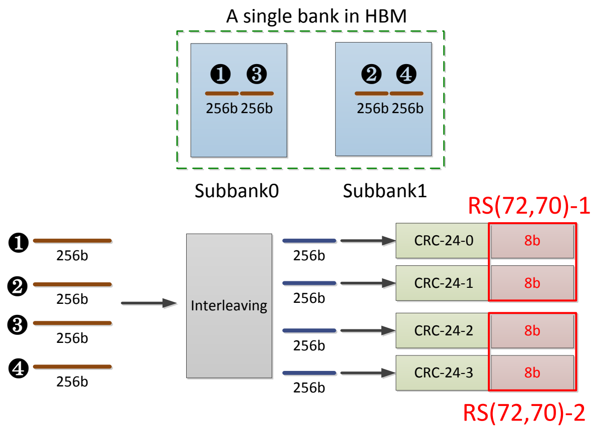


Figure 5.8: Block Diagram Of 128B Access: Static Mode

Silent Data Corruption: Silent data corruption can occur in the following scenarios. First, two CRC-24 codes declare error free, and RS (72,70) also declares error free. Second, one CRC-24 code declares errors, RS (72,70) performs correction and two CRC-24 codes declare error free.

128B Access Size

Here, 128B of data is split into two 64B of data and fed to two Config-ECC 64B units and thus the decoding flow is the same as 64B access (see Figure 5.7). Consequently, the 64B access scheme and the 128B access scheme have identical reliability.

5.4.2 Static Mode Protection

Errors due to large granularity faults manifest as burst errors. So we propose to add interleaving units to improve the overall error detection capability.

64B Access Size

We interleave 32B data from each sub-bank and then encode it with CRC-24 code. Such a scheme increases the burst error detection capability without the use of a larger

CRC code which has higher hardware complexity. The interleaved CRC decoding procedure is based on the method in [75]. The bit sequence of two 32B data lines are redistributed as follows. In each data line, if the bit position modulo 2 is 0, it is sent to the first CRC-24 unit; if the bit position modulo 2 is 1, it is sent to the second CRC-24 unit. After interleaving, the decoding steps are the same as dynamic Config-ECC 64B scheme.

128B Access Size

128B of data are redistributed before sending to four CRC-24 units. Assume that for the first 64B access, 32B are read from sub-banks 0 and 1 as shown in Figure 5.8. If the bit position modulo 4 is 0, 1, 2 and 3, then that bit is send to CRC-24 unit 0, 1, 2 and 3, respectively. Now for the second 64B access, the mapping function is a little different. For data lines 3 and 4, if the bit position modulo 4 is 1, 0, 3 and 2, these bits are sent to CRC-24 units 0, 1, 2, and 3, respectively. This mapping function was designed to avoid 8 bit errors due to a single TSV failure be sent to the same CRC unit.

The decoding steps for static 128B access are as follows: (i) if one or two CRC-24 units report errors, the RS decoder performs single error correction, (ii) if more than two CRC-24 decoder report errors, RS decoder is not activated and tier-2 decoder is launched, (iii) if four CRC units report error free, the RS decoder performs double error detection. The decoding flowchart is given in Figure 5.9.

5.5 Methodology

We introduce the experimental setup used to evaluate the reliability and compare the energy consumption of Config-ECC in this section.

Table 5.1: GPU And HBM System Configuration.

GPU Architecture	
# of SMs,	15
Max. # of Wraps per SM	48
Max. # of Blocks per SM	8
# of Warps per SM	32
# of Schedulers per SM	2
# of Registers per SM	32768
L1 inst. cache	2kB per SM, 128B line size 4 sets, 4-ways
L2 cache	768kB, 128B line size 32 sets, 8-ways, 6 banks
Scheduler	Greedy-then-oldest (GTO)
HBM Configuration	
Capacity	4GB per stack
# of Channels	8
# of banks	16 per channel
I/O	144 per channel
Memory clock	0.5GHz
Row Size (pseudo channel mode)	1kB for 32B access 2kB for 64B access

HBM Memory
Timing (cycles)
	tRC=24, tRCD=7, tRP=7,
	tCL=7, tRAS=17, tRRD=3,
	tCCD=2, tWL=1, tRTP=2

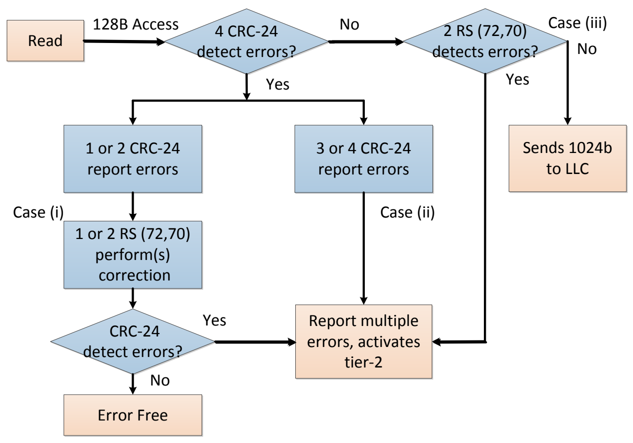

Read
128B Access
4 CRC-24 detect errors?
No
2 RS (72,70) detects errors?
Case (iii)
No
Yes
Yes
1 or 2 CRC-24 report errors
3 or 4 CRC-24 report errors
Sends 1024b to LLC
Case (i)
Case (ii)
1 or 2 RS (72,70) perform(s) correction
CRC-24 detect errors?
Yes
Report multiple errors, activates tier-2
No
Error Free

Figure 5.9: Decoding Flowchart For 128B Access: Static Mode

5.5.1 Setup For Reliability Evaluation

To measure system-level failure probability, we use Monte Carlo based simulations. The number of trials is at least 1 billion. Since the number of simulations is very large, we use the Mersenne Twister algorithm as the pseudo number generator to generate error patterns that ensure randomness [80]. We inject faults into the 3D HBM DRAM system and analyze the results after ECC decoding. Our procedure to calculate the reliability of each ECC scheme is similar to others [27, 28, 31, 32].

For errors due to single bit and single column failure, we assume that there is one single bit error per access (32B, 64B or 128B). We assume that the number of bit errors due to a single TSV failure is 4, 8 and 8 for 32B, 64B and 128B access, respectively. Prior work [25] indicated that a row or bank failure could result in 3 to 31 random bit errors. While [28] simulated this error event, [30, 31] pessimistically assumed that half of the bits in each data line are erroneous. Here, we combine the assumptions of [25, 28, 30, 31] and inject 3 to 128 random bit errors in a data line with flipping probability of 50% to model errors due to large granularity faults.

Table 5.2: Benchmarks For Config-ECC Energy Evaluation.

Abbr	Application	Dataset
`RD_KMN`	K-Means [76]	494020 objects
`PAN_PAG`	Page Rank [77]	1M data entries
`PAN_SPM`	Sparse Vector Multiplication [77]	1M points
`MAR_PVR`	Page View [78]	1M pages
`RD_BFS`	Breadth-First-Search [76]	65536 nodes
`NV_BIO`	Binomial Options [79]	512 Options
`PAN_FW`	Floyd Warshall [77]	256(V), 16K (E)
`RD_PF`	Particle Filter [76]	28x128x10 nodes
`RD_HOT`	Hotspot [76]	512x512 nodes
`NV_MC`	Monte Carol [79]	256 options
`PAN_CMX`	Graph Coloring Max [77]	ecology
`RD_STC`	Streamcluster [76]	32x4096 nodes
`PAN_FB`	Floyd Warshall Block [77]	256(V), 16K (E)
`PAN_MIS`	Maximal Independent Set [77]	ecology
`RD_BP`	Back Propagation [76]	65536 nodes

We examine the ECC decoding results to check whether the errors are detectable and correctable errors (DCE), detectable but uncorrectable errors (DUE) or result in silent data corruption (SDC) [52]. Finally, we also use the raw FIT rate provided from [32] and give the final FIT rates of our three Config-ECC schemes in Table 5.5.

5.5.2 System Simulation Infrastructure

To evaluate the performance of our ECC schemes in terms of energy consumption, we use GPGPU-Sim (version 3.2.2), a cycle-level performance simulator of a general purpose GPU architecture. The key microarchitectual parameters of the baseline configuration are summarized in Table 5.1. The GPGPU-Sim simulator is used to generate DRAM traces for benchmarks. To generate the DRAM traces, we assume that the L2 cache is equipped with a perfect spatial locality predictor. For each DRAM trace, we record the issued time of DRAM requests and number of bytes that have been referenced during a cache line's lifetime at the L2 cache controller. We use the DRAM traces to compute the energy consumption of the different ECC schemes. We select a wide range of benchmarks from Nvidia SDK [79], Mars [78], Pannotia [77], and Rodinia [76, 81] benchmark suites to represent the diversity of memory access patterns in GPUs. The details of benchmarks are listed in Table 5.2.

5.5.3 HBM Energy Modeling

We model HBM energy consumption using DRAMSim2 [60], a cycle-level accurate memory system simulator. For activation/precharge energy, we use the latest value provided by CACTI-3D and plug this value into the simulator. For 1kB page size, we assume that the row energy is 1.5nJ and for 2kB page size, the row energy is scaled to 3nJ. Similarly, we assume 32B read/write energy to be 4nJ based on CACTI-3D. Since there is no publicly available data for HBM background and refresh current, we only show the results when only activation/precharge and read/write energy are considered.

5.6 Results and Analysis

5.6.1 Reliability Comparison

32B Access Schemes

We compare our Config-ECC 32B scheme to the baseline CRC-16 scheme [33] and the single bit correction and double bit detection code (SEC-DED) used by Nvidia Tesla P100 [34]. The SDC rate of Config-ECC 32B scheme is dominated by the detection capability of the CRC-24 code. The reason is that once CRC code miss-detects errors, the RS decoder is not activated. Simulation result shows that CRC-24 has SDC rate of $7E-8$ (10 billion runs), which is very close to its theoretical SDC rate $2^{-24} \cong 5.9E-8$.

Both the baseline CRC-16 scheme and the SEC-DED scheme have higher SDC rate. CRC-16 can detect all errors due to small granularity faults and has an SDC rate of $1.5E-5$ for errors due to large granularity faults. The disadvantage of using CRC-16 as the tier-1 code is that it can only perform detection and all correction has to be done by the tier-2 code.

We assume SEC-DED code in [34] is based on Hamming (72,64), which protects 64 data bits with 8 parity ECC bits. The Hamming (72,64) SEC-DED can correct all errors due to small granularity faults but it has very poor SDC rate for errors due to large granularity faults. The error performance of the three competing schemes is shown in Table 5.3.

Using raw FIT rate number from [32], we calculate the final FIT rates of all schemes. CRC-16 and SEC-DED have final FIT rate of 3.7E-3 and 68.9, respectively. In contrast, our Config-ECC 32B reduces the raw FIT rate to 1.69E-5, which is 200x better than the baseline CRC-16.

Table 5.3: The Error Protection Coverage For 32B Config-ECC And The Existing Schemes

	CRC-16 (Baseline)	SEC-DED (Nvidia)	Config-ECC 32B (Proposed)
Single bit or column failure	DCE:0% DUE:100% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Single TSV failure	DCE:0% DUE:100% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Single row or bank failure	DCE:0% DUE:(1-1.5E-5) SDC:1.5E-5	DCE:0% DUE:72% SDC:28%	DCE:0% DUE:(1-7E-8) SDC:7E-8

64B Access Schemes

We compare Config-ECC 64B scheme to two recent ECC schemes: Citadel [27] and RATT-ECC [32]. For 64B access without interleaving, SDC events can occur due to (i) two CRC-24 decoders declaring error free, and RS (72,70) decoder also declaring error free in spite of errors, (ii) one CRC-24 decoder declaring errors, RS (72,70) performing correction and two CRC-24 decoder declaring error free even when errors are presented. The SDC rate of (ii) is very low because the probability of two CRC-24 codes detecting errors wrongly at the same time is extremely low (around $2^{-24} \times 2^{-24}$). We assume the worst condition when errors are all located in one 32B of data line in case (i). In this case, the SDC rate is calculated theoretically to $2^{-24} \times 2^{-16} \cong 9.0E{-}13$ (close to 10^{-12}). We did not see any error events in 10^{-12} runs and so we conservatively claim that the SDC rate is lower than 10^{-12}. Note that the interleaving 64B access

Table 5.4: The Error Protection Coverage For 64B Config-ECC And The Existing Schemes

	CRC-32 [27]	RS(70,64) [32]	Config-ECC 64B (Proposed)
Single bit or column failure	DCE:0% DUE:100% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Single TSV failure	DCE:0% DUE:100% SDC:0%	DCE:100% DUE:0% SDC:0%	DCE:100% DUE:0% SDC:0%
Single row or bank failure	DCE:0% DUE: 1 - SDC SDC:$2.3E-10$	DCE:0% DUE: 1 - SDC SDC: $2.4E-10$	DCE:0% DUE: $1-SDC$ SDC: $1E-12$

scheme used in static mode has similar SDC rate but stronger burst error detection capability.

Citadel [27] uses CRC-32 as the tier-1 code to detect errors and relies on tier-2 code to perform all correction. RATT-ECC [32] uses a stronger tier-1 code, RS(70,64), and so can correct all errors due to small granularity faults. Both schemes detect errors due to large granularity faults with very low SDC rate. The simulation results show that our Config-ECC 64B scheme has SDC rate lower than $1E-12$, making it at least 40x stronger than [27] and [32]. Our Config-ECC 64B reduces the raw FIT rate to 2.44E-10, which is 20x better than [27] and [32].

128B Access Schemes

Since there is no current ECC scheme designed for 128B access, we only analyze the reliability metrics of our Config-ECC 128B scheme. In the dynamic mode, Config-ECC 128B is built using two Config-ECC 64B units and so the reliability performance

Table 5.5: The Final FIT Rate Of Config-ECC Schemes

Failure Mode	Raw FIT	32B	64B/128B
Single bit	238	0	0
Single column	70	0	0
Single row	84	5.9E-6	8.4E-11
Single bank	162	1.1E-5	1.6E-10
Single TSV	41	0	0
Summary	685	1.69E-5	2.44E-10

of Config-ECC 128B is the same as Config-ECC 64B. In the static mode, Config-ECC 128B uses an interleaver to detect up to 96 consecutive burst errors. Table 5.5 summarizes the final FIT rates of our Config-ECC.

5.6.2 Energy Comparison

Read And Write Request Handling

We compare the energy results of Config-ECC to an existing 32B ECC scheme, namely, CRC-16 [33], an existing 64B ECC scheme, Citadel [27], and an 128B ECC scheme that was designed exclusively for 128B data. Such a scheme would have very high reliability for the same 12.5% parity storage. For better understanding of the energy performance results, we summarize the different types of accesses for a 32B ECC [33], 64B ECC [27], 128B ECC, and Config-ECC. The energy overhead is due to the mismatch between the data access size and data size used for the different ECC schemes. It is a function of the numbers of reads, writes, and row activations.

32B read request: Config-ECC and [33] have no additional reads but Citadel has an additional read of 32B of data since its ECC is based on 64B. Any 128B ECC scheme needs to read additional 96B of data to perform ECC decoding.

32B write request: There is no overhead for [33]. However, both Config-ECC and [27] have to read an additional 32B of data first and after encoding of ECC, write back 64B. The 128B ECC scheme needs to read additional 96B of data and write 128B data back.

64B read request: There is no overhead for Config-ECC, [33] and [27]. The 128B ECC scheme needs to read another 64B of data.

64B write request: There is no overhead for Config-ECC, [33] and [27] as well. The 128B ECC scheme needs to read an additional 64B of data and write 128B data back.

128B request: For both read and write requests, there is no overhead for any of the ECC schemes.

Dynamic Operation Mode

We present the energy results of the four competing schemes in Figure 5.10[1]. For the GPGPU applications that show a mixed data fetch size preference, RD_KMN, PAN_PAG, PAN_SPM, MAR_PVR, and RD_BFS (the first five applications in Figure 5.2), Config-ECC achieves a good balance between strong reliability guarantee and low energy consumption. Compared to the lowest energy 32B ECC design, Config-ECC increases the active energy consumption by 1.5% while providing 200

[1]We observe that, of the 15 benchmarks listed in Table 5.2, PAN_FW, RD_HOT, NV_MC, PAN_FB and RD_BP have mostly 128B accesses (see Figure 5.2). Therefore, there is no difference in the energy performance of the four ECC schemes. Thus, we do not show their energy results in Figure 5.10.

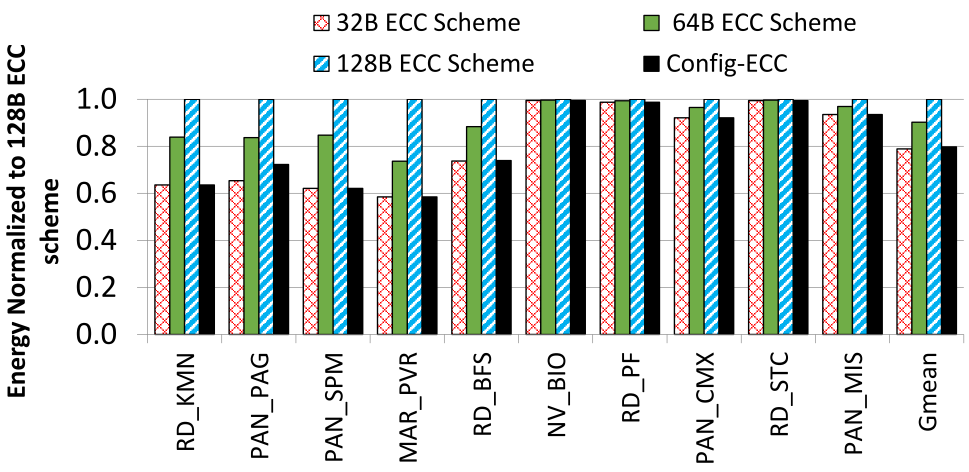


Figure 5.10: Energy Comparison Of The Competing Schemes When Operating In The Dynamic Operation Mode

times stronger reliability guarantee. The last five benchmarks, NV_BIO, RD_PF, PAN_CMX, RD_STC and PAN_MIS have a small fraction of accesses that are 32B and 64B, so the energy results for all schemes are very similar.

When compared to the 64B and 128B ECC designs, Config-ECC offers 20 times stronger reliability guarantee while requiring significantly lower active energy consumption by 17% and 34% less than [27] and the 128 ECC scheme. Both [27] and 128B ECC schemes have higher energy due to the mismatch between access size and ECC data size, resulting in larger number of reads, writes, and row activations. An analysis of the ratio of reads and writes shows that these benchmarks have high frequency of 32B reads but very low frequency of 32B writes. This is why Config-ECC has energy consumption similar to [33] even though it has higher energy consumption for 32B writes compared to [33].

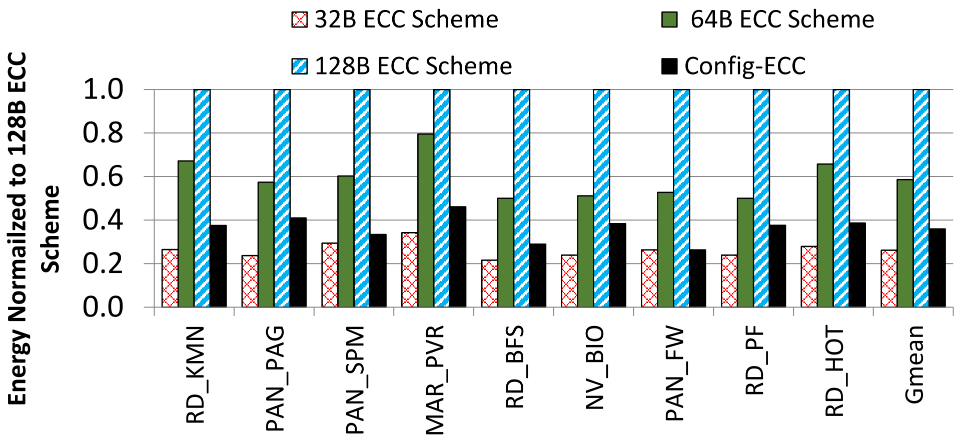


Figure 5.11: Energy Consumption Of The Competing Schemes When Operating In The Static Operation Mode

Static Operation Mode

For the static operation mode, we present the energy consumption of the competing schemes for the L1 cache-off configuration, since the energy results for the L1 cache-on configuration are expected to be the same for the competing schemes. This corresponds to the first nine GPGPU applications in Figure 5.1. In this configuration, the requests are either 32B read or write requests. Figure 5.11 shows the energy components of Config-ECC and other ECC schemes. Config-ECC consumes an additional 10% energy compared to [33] since the number of 32B writes for these benchmarks is high, ranging from 0% to 49%. This energy overhead comes with the benefit of a stronger, more reliable memory system. As shown in Section 5.6.1, Config-ECC provides 200 times lower SDC rate than the 32B ECC scheme. When compared to [27] and 128B ECC scheme, Config-ECC has 21% and 63% lower energy, respectively.

5.7 Summary

We present Config-ECC, a two-tiered error correction scheme that provides high reliability with a flexible structure to support different sized accesses in HBM memory systems. The tier-1 code is a product code that provides strong error detection and correction capabilities to correct all errors due to small granularity faults and detect errors with large granularity faults with very low silent data corruption (SDC) rate. The tier-2 code is launched for correcting errors due to large granularity faults. Config-ECC increases the reliability compared to fixed 32B and 64B ECC schemes significantly. When the access size is 32B, the product code design allows Config-ECC to read only 32B resulting in significant energy reduction compared to a fixed 64B ECC scheme.

Chapter 6

CONCLUSIONS

In this dissertation, we presented several low cost ECC schemes to improve the reliability of 2D and 3D DRAM systems.

6.1 2D DRAM Reliability

2D DRAMs are used in most computing systems. We considered x4, x8, x16 commodity DRAM systems and designed five erasure and error correction (E-ECC) schemes that provide superior error protection for these systems. All our E-ECC schemes use strong symbol based codes and provide higher error protection compared to existing systems. Furthermore, when a chip is marked faulty, our schemes make use of erasure correction to increase the lifetime of the memory system with no additional cost. Synthesis results show that the decoding latency of these codes is very small and the additional latency does not affect the timing performance of the memory system. Also, our schemes require no extra memory accesses to perform error correction and more importantly, do not require any change in the memory architecture.

All the proposed schemes can correct errors due to a chip failure, and when the chip is marked faulty, they can correct one more random error. Of all these schemes, Scheme 2 that is designed for x4 systems and uses RS(36,32) code, has the highest reliability. Simulations on SPEC 2006 benchmarks show that Schemes 3, 4 and 5 have better timing, power and energy performance compared to x4 Chipkill-Correct solutions. Among these schemes, Scheme 4 that is designed for x8 systems and uses the RS(36,32) code, has the lowest power consumption and highest energy performance while Scheme 5 that is designed for x16 systems and uses the RS(20,16) code, has the best timing performance. Overall, our proposed schemes provide a low cost solution to increasing the lifetime of commodity DRAM memory systems with lower power and performance overhead.

6.2 3D DRAM Reliability

3D DRAM systems have lower access latency, lower power consumption and higher bandwidth. However, designing a reliable 3D DRAM system is more challenging because not only is 3D DRAM more error-prone, data block from a lower level is stored in a single bank and so if this bank fails, the data is lost. We designed low cost ECC schemes for HBM-like memory that service CPUs with fixed data access size (64B) as well as those that service GPUs with varied data access size (32B, 64B and 128B).

6.2.1 CPU Systems

For CPU systems, where the data access is 64B, we presented a two-tiered ECC scheme, RATT-ECC, where the first tier is used to handle errors due to small granularity faults (single bit, column and TSV) and the second-tier is used to handle errors due to large granularity faults (row or bank failures). We chose a strong symbol based RS code in tier-1, namely, RS (70,64) code because of its ability to correct errors due to small granularity faults and reliably detect errors due to large granularity faults (with SDC $2.4 \cdot 10^{-10}$). This code can also be punctured and the resulting RS (69,64) and RS (68,64) codes share the same encoder and decoder as the RS (70,64) code. The punctured codes have been used to free up a ECC bank if a data bank is faulty or help store and decode tier-1 parity symbols in a way that reduces the number of memory accesses. The tier-2 code is an XCC based code which can also be modified to free up a ECC bank.

Overall RATT-ECC has excellent error handling performance with very little hardware overhead. Its timing performance is also very good. Simulation results with SPEC2006 benchmarks show that if the ECC cache is of size 2MB, the average performance overhead is less than 2%. Due to the strong error correction capability and low decoding latency of RATT-ECC, this scheme can be used to lower the refresh frequency and the additional errors can easily be corrected. Refresh power accounts for 33% of the total DRAM power dissipated [82] and so this scheme can be used to to obtain lower power consumption without degrading performance of 3D DRAM.

6.2.2 GPU Systems

Future GPU systems are expected to implement a memory subsystem supporting fine and coarse-grained data accesses to match the difference in the spatial locality of GPGPU applications. Given the differences in the cache line sizes, an ECC scheme that is optimized for a fixed data line size is sub-optimal.

We present Config-ECC, a two-tiered error correction scheme that provides high reliability with a flexible structure to support different sized accesses in HBM memory systems. The tier-1 code is a product code that provides strong error detection and correction capabilities to correct all errors due to small granularity faults and detect errors with large granularity faults with very low silent data corruption (SDC) rate. The tier-2 code is launched for correcting errors due to large granularity faults. Config-ECC increases the reliability by 200 times compared to a fixed 32B ECC scheme and by 20 times compared to a fixed 64B ECC scheme. Also, Config-ECC can choose to read only 32B resulting in 17% energy reduction when operating in dynamic mode and 21% reduction when operating in static mode compared to a fixed 64B ECC scheme.

6.3 Future Work

Apart from HBM, there are other 3D DRAM prototypes such as HMC [8] and Wide-I/O [44]. It is projected that HMC will replace 2D DRAM to serve as the CPU memory and Wide-I/O will replace low power DDR as the mobile DRAM. These two classes of memory access a data line in the same way as HBM. Thus, our proposed ECC schemes for HBM can be used for both HMC and Wide-I/O. HMC has a new requirement for error protection when transferring data between different die-stacked HMCs. Such an error protection can be added on top of our proposed scheme. Wide-I/O has larger number of I/O pins (256 bits vs. 64 bits). Our tier-1 code can be designed to correct errors due to TSV failures more efficiently while reducing the ECC storage overhead.

REFERENCES

[1] S. Li, K. Chen, M.-Y. Hsieh, N. Muralimanohar, C. Kersey, J. Brockman, A. Rodrigues, and N. Jouppi, "System Implications of Memory Reliability in Exascale Computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, (SC)*, Nov. 2011, pp. 1–12.

[2] B. Schroeder, E. Pinheiro, and W. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009, pp. 193–204.

[3] V. Sridharan and D. Liberty, "A Field Study of DRAM Errors," in *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2012, pp. 1–11.

[4] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," in *International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2013, pp. 1–12.

[5] S. Lin and D. J. Costello, *Error Control Coding*, Pearson, 2nd edition, 2004.

[6] G.H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *35th International Symposium on Computer Architecture, ISCA '08.*, Jun. 2008, pp. 453–464.

[7] JEDEC Standard, "High Bandwidth Memory (HBM) DRAM, JESD235," 2013.

[8] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube: New Dram Architecture Increases Density and Performance," in *Symposium on VLSI Technology (VLSIT)*, Jun. 2012, pp. 87–88.

[9] Tezzaron Semiconductor, "Die-Stacked DRAM," `http://www.tezzaron.com/`.

[10] T. J. Dell, "A white paper on the benefits of Chipkill-Corret ECC for PC server main memory," in *IBM Microelectronics*, 1997.

[11] D. Locklear, "Chipkill Correct Memory Architecture," *Dell Enterprise System Group*, Aug. 2000.

[12] T. R. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall Inc., 1989.

[13] C. L. Chen, "Symbol error correcting codes for memory applications," in *Proceedings of Annual Symposium on Fault Tolerant Computing*, 1996, pp. 200–207.

[14] D. Yoon and M. Erez, "Virtualized and Flexible ECC for Main Memory," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. Mar. 2010, pp. 397–408, ACM.

[15] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems," in *International Symposium on Computer Architecture (ISCA)*, Jun. 2012, pp. 285–296.

[16] X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar, "Low-power, Low-storage-overhead Chipkill Correct via Multi-line Error Correction," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC13*, 2013, pp. 1–12.

[17] X. Jian, J. Sartori, H. Duwe, and R. Kumar, "High Performance, Energy Efficient Chipkill Correct Memory with Multidimensional Parity," in *Transaction on Computer Architecture Letters*, Jul. 2013, vol. 12, pp. 39–42.

[18] X. Jian and R. Kumar, "ECC Parity: A Technique for Efficient Memory Error Resilience for Multi-Channel Memory Systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC14*, Nov. 2014, pp. 1035–1046.

[19] "Intel Xeon Processor E7 Family: Reliability, Availability and Serviceability: Advanced data integrity and resiliency support for mission-critical deployment," 2011.

[20] "Reliability, Availibility, and Serviceability. Features of the IBM eX5 Portfolio," 2012.

[21] "AMD64 architecture programmer's manual revision 3.17," 2011.

[22] H.-M. Chen, A. Arunkumar, C. J. Wu, T. Mudge, and C. Chakrabarti, "E-ECC: Low Power Erasure and Error Correction Schemes for Increasing Reliability of Commodity DRAM Systems," in *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS)*, 2015, pp. 60–70.

[23] H.-M. Chen, S. Jeloka, A. Arunkumar, D. Blaauw, C. J. Wu, T. Mudge, and C. Chakrabarti, "Using Low Cost Erasure and Error Correction Schemes to Improve Reliability of Commodity DRAM Systems," in *IEEE Transactions on Computers*, Dec. 2016, vol. 65, pp. 3766–3779.

[24] J. Meng, D. Rossell, and A. K. Coskun, "3D Systems with On-Chip Dram for Enabling Low-Power High-Performance Computing," in *IEEE High Performance Embedded Computing, HPEC*, 2011, pp. 1–4.

[25] 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," *SIGARCH Computer Architecture News*, pp. 111–122, Mar. 2012.

[26] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory Errors in Modern Systems: The Good, The Bad, and The Ugly," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15*, 2015, pp. 297–310.

[27] P. Nair, D. Roberts, and M. Qureshi, "Citadel: Efficiently Protecting Stacked Memory from Large Granularity Failures," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2014, pp. 51–62.

[28] H. Jeon, G. Loh, and M. Annavaram, "Efficient RAS Support for Die-Stacked DRAM," in *IEEE International Test Conference (ITC)*, Oct. 2014, pp. 1–10.

[29] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H-M Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, "Exploring Dram Organizations for Energy-Efficient and Resilient Exascale Memories," in *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2013, pp. 1–12.

[30] J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor, "Resilient Die-stacked Dram Caches," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, ISCA '13, pp. 416–427.

[31] X. Jian, V. Sridharan, and R. Kumar, "Parity Helix: Efficient Protection for Single-Dimensional Faults in Multi-Dimensional Memory Systems," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Mar. 2016, pp. 555–567.

[32] H.-C. Chen, C.-J. Wu, T. Mudge, and C. Chakrabarti, "RATT-ECC: Rate Adaptive Two-Tiered Error Correction Codes for Reliable 3D Die-Stacked Memory," in *ACM Transaction on Architecture and Code Optimization (TACO)*, Sept. 2016, vol. 13, pp. 24:1–24:24.

[33] "High Bandwidth Memory (HBM) DRAM," *JEDEC Standard, JESD235A*, 2015.

[34] "Nvidia Tesla P100 - Whitepaper," *WP-08019-001 v01.1*, 2016.

[35] J. Macri, "Amd's Next Generation GPU and High Bandwidth Memory Architecture: FURY," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug. 2015, pp. 1–26.

[36] AMD, "White paper - AMD Graphics Cores Next (gcn) Architecture," Jun. 2012.

[37] W. Jia, K. A. Shaw, and M. Martonosi, "Characterizing and Improving the Use of Demand-Fetched Caches in GPUs," in *Proceedings of the 26th ACM International Conference on Supercomputing*, 2012, ICS '12, pp. 15–24.

[38] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, MICRO, pp. 86–98.

[39] A. Arunkumar, S.-Y. Lee, and C.-J. Wu, "ID-Cache: Instruction and Memory Divergence Based Cache Management for GPUs," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2016, pp. 158–167.

[40] H.-M. Chen, S.-Y. Lee, C.-J. Wu, T. Mudge, and C. Chakrabarti, "Configurable-ECC: Architecting a Flexible ECC Scheme to Support Different Sized Accesses in High Bandwidth Memory Systems (submitted)," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, May 2017.

[41] B. Jacob, S. Ng, and D. Wang, *Memory Systems Cache, DRAM, Disk*, Morgan Kaufmann; first edition, 2007.

[42] "DDR3 SDRAM Standard, JESD79-3F," 2012.

[43] "Hybrid Memory Cube Specification 2.0," *Hybrid Memory Cube HMC-30G-VSR PHY*, 2014.

[44] "Wide I/O 2," *JEDEC Standard, JESD 229-2*, Aug. 2014.

[45] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee, H.-S. Joo, W.-S. Kim, H.-K. Kim, E.-M. Lee, S.-R. Kim, K.-H. Ma, D.-H. Jang, N.-S. Kim, M.-S. Choi, S.-J. Oh, J.-B. Lee, T.-K. Jung, J.-H. Yoo, and C. Kim, "8Gb 3D DDR3 DRAM Using Through-Silicon-Via Technology," in *2009 IEEE International Solid-State Circuits Conference*, Feb. 2009, pp. 130–131.

[46] K. Sohn, W. J. Yun, R. Oh, C. S. Oh, S. Y. Seo, M. S. Park, D. H. Shin, W. C. Jung, S. H. Shin, J. M. Ryu, H. S. Yu, J. H. Jung, K. W. Nam, S. K. Choi, J. W. Lee, U. Kang, Y. S. Sohn, J. H. Choi, C. W. Kim, S. J. Jang, and G. Y. Jin, "18.2 A 1.2V 20nm 307GB/s HBM DRAM with At-Speed Wafer-Level I/O Test Scheme and Adaptive Refresh Considering Temperature Distribution," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Jan. 2016, pp. 316–317.

[47] N. DeBardeleben, S. Blanchard, V. Sridharan, S. Gurumurthi, J. Stearley, K. Ferreira, and J. Shalf, "Extra Bits on SRAM and DRAM Errors - More Data From the Field," in *Silicon Errors in Logic - System Effects (SELSE-10), Stanford University*, Apr. 2014.

[48] T. J. Dell, "System RAS Implications of DRAM Soft Errors," in *IBM Journal of Research and Development*, 2008, vol. 52, pp. 307–314.

[49] "Sun Microsystems, Inc., "T2 core microarchitecture specification", http://www.oracle.com/technetwork/systems/opensparc/t2-06-opensparct2-core-microarch-1537749.html," 2007.

[50] "Advanced Micro Devices (AMD), inc., "Kernel developer's guide for AMD NPT family 0Fh processors", http://developer.amd.com/wordpress/media/2012/10/325591.pdf," 2007.

[51] X. Jian and R. Kumar, "Adaptive Reliability Chipkill Correct (ARCC)," in *IEEE International Symposium on High Performance Computer Architecture, HPCA*, Feb. 2013, pp. 270–281.

[52] J. Kim, M. Sullivan, and M. Erez, "Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory," in *IEEE International Symposium on High Performance Computer Architecture, HPCA*, Feb. 2015, pp. 101–112.

[53] V. Sridharan, N. DeBardeleben, S. Blanchard, B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *SIGARCH Computer Architecture News*, Mar. 2015, pp. 297–310.

[54] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2015, pp. 415–426.

[55] R.-H. Deng and D. J. Costello, "Decoding of DBEC-TBED Reed-Solomon Codes," *IEEE Transactions on Computers*, vol. C-36, no. 11, pp. 1359–1363, 1987.

[56] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balsubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *2010 Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2010, pp. 175–186.

[57] "HP: How memory RAS technologies can enhance the uptime of HP Proliant servers," `https://www.hpe.com/h20195/V2/GetPDF.aspx/4AA4-3490ENW.pdf`, 2013.

[58] "Memory for Dell Poweredge 12^{th} Generation Servers," `https://www.dell.com/downloads/global/products/pedge/poweredge_12th_generation_server_memory.pdf`, 2012.

[59] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, and D. Wood, "The Gem5 Simulator," in *SIGARCH Computer Architecture News*, Aug. 2011, vol. 39, pp. 1–7.

[60] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," in *IEEE Computer Architecture Letters*, Jan. 2011, vol. 10, pp. 16–19.

[61] Micron, "DDR3 SDRAM System-Power Calculator," 2011.

[62] E. Perelman, G. Hamerly, M. Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for Accurate and Efficient Simulation," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, 2003, pp. 318–319.

[63] A. Snavely and M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multi-threaded Processor," in *Proceedings of the international conference on Architectural support for programming languages and operating systems (ASPLOS)*, Dec. 2000, pp. 234–244.

[64] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, May 2008, vol. 28, pp. 42–53.

[65] T. Kasami and S. Lin, "On the Probability of Undetected Error for the Maximum Distance Separable Codes," *IEEE Transactions on Communications*, vol. 32, no. 9, pp. 998–1006, Sep. 1984.

[66] HPArch, "Macsim Simulator," `https://code.google.com/p/macsim/downloads/list`.

[67] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled Drams," in *IEEE Electron Device Letters*, Aug. 2009, vol. 30, pp. 846–848.

[68] NVIDIA, "NVIDIA's next generation CUDA compute architecture: Fermi," September 2009.

[69] Intel, "The compute architecture of Intel processor graphics Gen9," `https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf`, Aug. 2015.

[70] NVIDIA, "NVIDIA GeForce GTX 750 Ti: Featuring first-generation Maxwell GPU technology, designed for extreme performance per watt," Feb. 2014.

[71] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, Jan. 1961.

[72] Philp Koopman, "Best CRC Polynomials," `https://users.ece.cmu.edu/~koopman/crc/`.

[73] T. Kasami and S. Lin, "On the Probability of Undetected Error for the Maximum Distance Separable Codes," in *IEEE Transactions on Communications*, Sep. 1984, vol. 32, pp. 998–1006.

[74] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *International Conference on Dependable Systems and Networks*, Jun. 2004, pp. 145–154.

[75] J. J. Kong and K. K. Parhi, "Interleaved cyclic redundancy check (CRC) code," in *The Thirty-Seventh Asilomar Conference on Signals, Systems Computers*, Nov. 2003, vol. 2, pp. 2137–2141.

[76] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[77] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understading Irregular GPGPU Graph Applications," in *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2013, pp. 185–195.

[78] B. He, W. Fang, K. Govindaraju, Q. Luo, and T. Wang, "Mars: A MapReduce Framework on Graphics Processors," in *Proc. of the 17th IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 260–269.

[79] NVIDIA, "CUDA C/C++ SDK code samples v4.0," May 2011.

[80] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-Random Number Generator," in *ACM Transaction on Modeling and Computer Simulation*, Jan. 1998, vol. 8, pp. 3–30.

[81] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2010, pp. 1–11.

[82] M. Ghosh and H.-H.S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2007, pp. 134–145.

[83] S. Fenn, M. Benaissa, and D. Taylor, "Decoding Double-Error-Correcting Reed-Solomon Codes," in *IEE Proceedings on Communications*, 1995, vol. 142, pp. 345–348.

[84] R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[85] L.L. Joiner and J.J. Komo, "Time Domain Decoding of Extended Reed-Solomon codes," in *Proceedings of the IEEE Bringing Together Education, Science and Technology*, Apr. 1996, pp. 238–241.

APPENDIX A

DECODING ALGORITHMS FOR E-ECC

This following presents the decoding algorithms of the proposed E-ECC codes. ***(144,128) rotational code:*** This is a (36,32) code over GF(2^4) that has a minimum distance of 4 and supports the following cases: (i) single error correction and double error detection, (ii) single erasure correction, (iii) single erasure and single error correction, (iv) double erasure correction and (v) double erasure and single error detection.

The first step is syndrome calculation, where the syndrome vector $S = (s_0, s_1, s_2, s_3)^T$ is calculated by multiplying the parity check matrix H with the codeword. The parity check matrix of the rotational (144,128) code is of size 4×36, where each entry is a 4 bit symbol, and is given in [12]. Case (i) is the traditional single symbol correction and double symbol detection. The decoder compares the syndrome vector with all columns of the parity check matrix to determine the error location and the corresponding error value. Let the jth column of the parity check matrix, H, be $h_j = (h_{j0}, h_{j1}, h_{j2}, h_{j3})^T$, $j = 0, 1, ..., 35$. If $S = e_j h_j$ holds, then the error occurred in position j and has a value e_j. If there is no j for which $S = e_j h_j$, then the decoder declares it as a double error event.

Cases (ii) to (v) involve correction of erasure symbols. Note that since the location of the faulty chip is known, the corresponding symbols are marked as erasure symbols. Each erasure symbol is replaced with the zero symbol in the received codeword and then the syndrome vector is generated. For case (ii), the syndrome vector is compared with the column in the parity check matrix corresponding to the erasure address. If the erasure address is i, the decoder checks if $S = e_i h_i$ for column i in H. If it holds, the decoder recovers the erasure value e_i in address i of codeword. Otherwise, the single erasure and single error correction (case (iii)) unit is activated.

For case (iii), assume that the erasure address is i and the erasure value is e_i; similarly, the error address is j and error value is e_j, where $i \neq j$. The decoder needs to check whether the syndrome vector, S, is a linear combination of h_i and h_j, where j is from 0 to 35 and $i \neq j$. The hardware consists of 36 sub-decoders, where the j^{th} decoder has h_j embedded in it. The MC feeds column h_i to all but the i^{th} sub-decoder. If the i^{th} and j^{th} columns of the parity check matrix are $h_i = (h_{i0}, h_{i1}, h_{i2}, h_{i3})^T$ and $h_j = (h_{j0}, h_{j1}, h_{j2}, h_{j3})^T$, then $s_0 = e_i \cdot h_{i0} + e_j \cdot h_{j0}$, $s_1 = e_i \cdot h_{i1} + e_j \cdot h_{j1}$ equations are used to obtain e_i and e_j. The decoded e_i and e_j values are substituted back to calculate $\tilde{s_2} = e_i \cdot h_{i2} + e_j \cdot h_{j2}$ and $\tilde{s_3} = e_i \cdot h_{i3} + e_j \cdot h_{j3}$. If $\tilde{s_2} = s_2$ and $\tilde{s_3} = s_3$ both hold, the decoder declares that the error is located in location j of the received codeword and corrects it.

For case (iv), the MC sends two erasure addresses (assume i and j) to the decoder and the decoder extracts the two columns, h_i and h_j, corresponding to the two erasure addresses. It uses these two columns to check whether the syndrome is a linear combination of h_i and h_j. If the condition holds, the decoder recovers the two erased symbols. Otherwise, it declares that there are two erasures and one error, which is case (v).

RS (36,32) over GF(2^8): This code has minimum distance of 5 and supports the following cases: (i) single error correction, (ii) double error correction, (iii) single erasure correction, (iv) single erasure and single error correction, (v) double erasure correction, (vi) double erasure and single error correction. The parity check matrix, H, of this code is given as follows

$$H = \begin{pmatrix} 1 & 1 & 1 & ... & 1 & ... & 1 \\ 1 & \alpha & \alpha^2 & ... & \alpha^i & ... & \alpha^{35} \\ 1 & \alpha^2 & \alpha^4 & ... & \alpha^{2i} & ... & \alpha^{70} \\ 1 & \alpha^3 & \alpha^6 & ... & \alpha^{3i} & ... & \alpha^{105} \end{pmatrix},$$

where α is a primitive element of GF(2^8). For case (i) and case (ii), the decoder implements single error and double error correction. The decoder can classify the two cases by using the syndrome vector [55]. Let the syndrome vector be $S = (s_0, s_1, s_2, s_3)^T$. The condition $\frac{s_3}{s_2} = \frac{s_2}{s_1} = \frac{s_1}{s_0}$ corresponds to a single error event; otherwise, it is a double error event. We implement the double error correction based on the method used in [83, 84]. To solve the error locator polynomial, we use a deterministic way to solve the roots rather than using the Chien search method. The error locator polynomial can be simplified to $y^2 + y + K$ [83, 84]. To solve y, a deterministic way by using linearlized polynomials is shown in [84].

Case (iii) is trivial and easier than case (i). Suppose h_i is the i^{th} column of H and the erasure position is at i. The decoder compares $S = e_i \cdot h_i$ or not. If it holds, the decoder declares it is single erasure event. If it does not hold, it activates case (iv). Suppose the erasure address is i (known) and the error address is j (unknown). The decoder checks whether the syndrome vector, S, is a linear combination of h_i and h_j. Let $S = e_i \cdot h_i + e_j \cdot h_j$, where $h_i = (1, \alpha^i, \alpha^{2i}, \alpha^{3i})^T$ and $h_j = (1, \alpha^j, \alpha^{2j}, \alpha^{3j})^T$, then $\alpha^j = \frac{s_1 \cdot \alpha^i + s_2}{s_0 \cdot \alpha^i + s_1}$, $e_j = \frac{s_0 \cdot \alpha^i + s_1}{\alpha^i + \alpha^j}$ and $e_i = s_0 + e_j$. The decoder first finds the error address j, then derives e_i and e_j.

In Scheme 2, if two chips are marked faulty, then there are two erasures (case (v)). Assume the positions of two erasure symbols are i and j, where i and j are from 0 to 35 and $i \neq j$. The relation between syndrome vector and the columns corresponding to the erasure positions in H is as follows:

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = e_i \begin{pmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \alpha^{3i} \end{pmatrix} + e_j \begin{pmatrix} 1 \\ \alpha^j \\ \alpha^{2j} \\ \alpha^{3j} \end{pmatrix},$$

where e_i and e_j are the erasure values for positions i^{th} and j^{th}. The value e_j is calculated by $e_j = \frac{s_0 \alpha^i + s_1}{\alpha^i + \alpha^j}$ and e_i is obtained by $e_i = s_0 + e_j$. The decoded e_i and e_j values are used to calculate $\tilde{s_2}$ and $\tilde{s_3}$, namely, $\tilde{s_2} = e_i \cdot \alpha^{2i} + e_j \cdot \alpha^{2j}$ and $\tilde{s_3} = e_i \cdot \alpha^{3i} + e_j \cdot \alpha^{3j}$. If $\tilde{s_2} = s_2$ and $\tilde{s_3} = s_3$ hold, the decoder declares a double erasure event and corrects these two erased symbols (case (v)). Otherwise, the decoder activates the double erasure and single error correction unit corresponding to case (vi).

In case (vi), let the address of the error symbol be k, the error value be e_k and the addresses of the erasure symbols are i and j, where $i \neq j$, $j \neq k$ and $i \neq k$. The relation between syndrome vector and double erasure and one error is given as follows:

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = e_i \begin{pmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \alpha^{3i} \end{pmatrix} + e_j \begin{pmatrix} 1 \\ \alpha^j \\ \alpha^{2j} \\ \alpha^{3j} \end{pmatrix} + e_k \begin{pmatrix} 1 \\ \alpha^k \\ \alpha^{2k} \\ \alpha^{3k} \end{pmatrix}$$

The error address α^k is derived by $\alpha^k = \frac{s_1\alpha^{i+j}+s_2\alpha^j+s_2\alpha^i+s_3}{s_0\alpha^{i+j}+s_1\alpha^j+s_1\alpha^i+s_2}$. The error value e_k is obtained by $\frac{s_0\alpha^{i+j}+s_1\alpha^j+s_1\alpha^i+s_2}{(\alpha^k+\alpha^j)(\alpha^i+\alpha^j)}$ The erasure value e_j is obtained by $e_j = \frac{s_0\alpha^j+e_k\alpha^j+e_k\alpha^k+s_1}{\alpha^i+\alpha^j}$. Finally, e_i is obtained by $e_i = s_0 + e_j + e_k$.

In Schemes 3 and 4, if a single chip is marked faulty, there are two erasures but these erasures are in consecutive locations. So if the position of the first erasure is $2i$ the position of the second erasure is $2i+1$, where $i = 0, 1, ..., 17$. The procedure for case (v) and case (vi) in Schemes 3 and 4 are very similar to these in Scheme 2 and are not described here.

The RS (20,16) code over GF(2^8) has smaller parity check matrix but it has the same decoding algorithm as the RS (36,32) code. Hence, we skip the description of its decoding units of RS (20,16) here.

APPENDIX B

DECODING ALGORITHMS FOR RATT-ECC

In this appendix, we show how the codewords of the three RS codes, namely, RS (70,64), RS (69,64) and RS (68,64) are related. Specifically the codeword of RS (69,64) can be obtained by removing the last symbol of the codeword of RS (70,64). Similarly, the codeword of RS (68,64) can be obtained by getting rid of the last two symbols of the codeword of RS (70,64). We utilize this embedded code structure to implement the rate-adaptive tier-1 code.

The RS(68,64) code can be obtained by shortening the RS (255,251) code [5]. After shortening, the parity check matrices of RS (69,64) and RS (70,64) can be derived from that of RS (68,64) [85]. The parity check matrix of RS (68,64), H, is given by

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & ... & \alpha^{67} \\ 1 & \alpha^2 & \alpha^4 & ... & \alpha^{134} \\ 1 & \alpha^3 & \alpha^6 & ... & \alpha^{201} \\ 1 & \alpha^4 & \alpha^8 & ... & \alpha^{13} \end{pmatrix},$$

and the parity check matrix, $H^{'}$, for RS (69,64) is given by

$$H^{'} = \left(\begin{array}{ccc:c} 1 & \cdots & 1 & 1 \\ \hdashline & H_{4\times 68} & & 0 \end{array}\right).$$

The parity check matrix for RS (70,64) code can be obtained from $H^{'}$ as

$$H^{''} = \left(\begin{array}{ccccc:c} & & H^{'}_{5\times 69} & & & 0 \\ \hdashline 1 & \alpha^5 & \cdots & \alpha^{80} & 0 & 1 \end{array}\right).$$

Assume the generator matrix of RS (68,64) code is G, then the generator matrices $G^{'}$ and $G^{''}$ of RS (69,64) and RS (70,64) can be derived from [85]. $G^{'}$ is given by

$$G^{'} = \begin{pmatrix} G & g^{'} \end{pmatrix},$$

and $G^{''}$ is given by

$$G^{''} = \begin{pmatrix} G^{'} & g^{''} \end{pmatrix},$$

where $g^{'}$ and $g^{''}$ are given in [85].

For encoding, the codeword of RS (70,64) code can be expressed in the form $\vec{c}^{''} = (c_0, c_1, ..., c_{67}, c_{68}, c_{69})$, which is generated by using $G^{''}$. The codeword, $\vec{c}^{'}$, of RS (69,64) is simply obtained by puncturing c_{69} in $\vec{c}^{''}$. Similarly, the codeword, $\vec{c}$, of RS (68,64) is obtained by puncturing c_{68} and c_{69} in $\vec{c}^{''}$. This demonstrates that all three codes can share the same encoder.

For decoding RS (70,64), the syndrome vector $\vec{s} = (s_0, s_1, s_2, s_3, s_4, s_5)$ is obtained by using computing $\vec{c}^{''} \cdot (H^{''})^T$. Because of the structure of $H^{''}$, the syndrome vector of RS (69,64) code can be obtained by removing s_5 from the syndrome vector of RS (70,64). Similarly, the syndrome vector of RS (68,64) code which is (s_1, s_2, s_3, s_4) can be obtained by removing s_0 and s_5 from $\vec{s}$. Thus the syndrome caculation unit of RS (70,64) can be used to calculate the syndrome of RS (69,64) and RS (68,64). Similarly, the decoding units of RS (70,64) can be reused for RS (69,64). For example, for single error correction, RS (70,64) decoder checks whether $\frac{s_1}{s_0} = \frac{s_2}{s_1} = \frac{s_3}{s_2} = \frac{s_4}{s_3} = \frac{s_5}{s_4}$ and the controller unit just turns off the last condition i.e., it only checks $\frac{s_1}{s_0} = \frac{s_2}{s_1} = \frac{s_3}{s_2} = \frac{s_4}{s_3}$.

Note that if we do not use the embedded RS codes as given above, multiple generator matrices or parity check matrices would have to be used to implement the rate-adaptive RS codes, resulting in significant increase in ECC hardware overhead.