RATT-ECC: Rate Adaptive Two-Tiered Error Correction Codes for Reliable 3D Die-Stacked Memory

HSING-MIN CHEN and CAROLE-JEAN WU, Arizona State University
TREVOR MUDGE, University of Michigan
CHAITALI CHAKRABARTI, Arizona State University

This article proposes 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. The tier-1 code is a strong symbol-based code that can correct errors due to small granularity faults and detect errors caused by large granularity faults; the tier-2 code is an XOR-based code that corrects errors detected by the tier-1 code. 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 reliability and timing performance.

CCS Concepts: • **Computer systems organization** → **Reliability**; • **Hardware** → *3D integrated circuits*; *Dynamic memory*; *Transient errors and upsets;*

Additional Key Words and Phrases: 3D memory, reliability, performance, error control coding

ACM Reference Format:
Hsing-Min Chen, Carole-Jean Wu, Trevor Mudge, and Chaitali Chakrabarti. 2016. RATT-ECC: Rate adaptive two-tiered error correction codes for reliable 3D die-stacked memory. ACM Trans. Archit. Code Optim. 13, 3, Article 24 (September 2016), 24 pages.
DOI: http://dx.doi.org/10.1145/2957758

1. INTRODUCTION

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 compared to 2D DRAM [Loh 2008; Meng et al. 2011]. There are several prototypes for die-stacked DRAM, including high bandwidth memory (HBM) [JEDEC HBM 2013], hybrid memory cube (HMC) [Jeddeloh and Keeth 2012] and Octopus from Tezzaron [2014]. In this article, we consider an HBM-like structure with 8 data dies and 1 error correction code (ECC) die stacked on top of the logic die; stacking is achieved by the use of through silicon vias (TSVs).

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 Hwang et al. [2012], Sridharan and Liberty [2012], and Sridharan et al. [2013, 2015]. The 2D DRAM errors that were due to transient and permanent single-bit, column, row and bank failures are expected

This work was supported in part by the DARPA PERFECT program, NSF CNS 18183, ARM and DOE.
Authors' addresses: H.-M. Chen, C.-J. Wu, and C. Chakrabarti, School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, 85287-5706 USA; emails: {hchen136, carole-jean.wu, chaitali}@asu.edu; Trevor Mudge, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48109-2121 USA; email: tnm@umich.edu.

DOI: http://dx.doi.org/10.1145/2957758

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), a data block from a lower-level cache is stored in a single bank and not striped across multiple chips, as in 2D DRAM. While this results in lower power and higher performance, it makes the design of a reliable memory 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 [Locklear 2000], virtualized ECC [Yoon and Erez 2011], LOT-ECC [Udipi et al. 2012], and Multi-ECC [Jian et al. 2013], were all based on data being striped across different chips. Hence, these schemes cannot be extended to handle 3D DRAM directly. Furthermore, in large-sized DRAMs, refresh power accounts for a significant portion of the power consumption [Liu et al. 2012]. Reducing the refresh power by increasing the refresh interval indiscriminately makes the memory even more unreliable.

In recent years, several schemes have been proposed for improving the reliability of 3D DRAM. These include Subarraykill [Giridhar et al. 2013], Resilient-DRAM-Caches [Jaewoong et al. 2013a], E-RAS [Jeon et al. 2014], Citadel-1 [Nair et al. 2014], and Citadel-2 [Nair et al. 2016]. Subarraykill [Giridhar et al. 2013] proposes an ECC scheme for the specialized Tezzaron Octopus structure, which allows the data to be striped across several subarrays. The ECC design in Jaewoong et al. [2013a] focuses on the protection of the last-level cache; the single-bit error correction code and small size of cyclic redundancy code (CRC) [Lin and Costello 2004] are not strong enough for 3D DRAM memory directly. E-RAS [Jeon et al. 2014] and Citadel [Nair et al. 2014, 2016] are contemporary systems that have been designed for HBM-like memory. Both use two-tiered codes, in which the first tier is used only to detect [Nair et al. 2014], or detect as well as correct [Jeon et al. 2014; Nair et al. 2016]; the second tier is used to correct errors detected, but not corrected, by the first tier. Finally, while there are no schemes to explicitly reduce the refresh power of 3D DRAM, the existing 2D DRAM techniques—such as RAIDR [Liu et al. 2012], which uses different refresh intervals for different rows, or Hi-ECC [Wilkerson et al. 2010], which uses a strong ECC to correct errors due to retention failures—can be extended to 3D DRAM.

In this article, we propose a rate-adaptive, two-tiered error-correction scheme (RATT-ECC) that provides very high reliability for an HBM-like 3D DRAM system. The tier-1 code is based on a Reed-Solomon (RS) code [Lin and Costello 2004] that provides 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 is used to correct one symbol error and detect five symbol errors. Since the undetected error probability of tier-1 code is as low as $2.4 \cdot 10^{-10}$, 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 failures in data TSVs can be easily corrected. Furthermore, RATT-ECC has a mechanism to free up banks in the ECC die so that they can be used as spares for 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. Finally, use of a low latency tier-1 code that can correct small granularity faults allows RATT-ECC to operate at large refresh intervals, thereby reducing refresh power.

Overall, this article makes the following key contributions:

—We design RATT-ECC, a rate-adaptive two-tiered ECC scheme with a total storage overhead of 12.5%, to provide strong reliability for an HBM-like 3D DRAM system. It

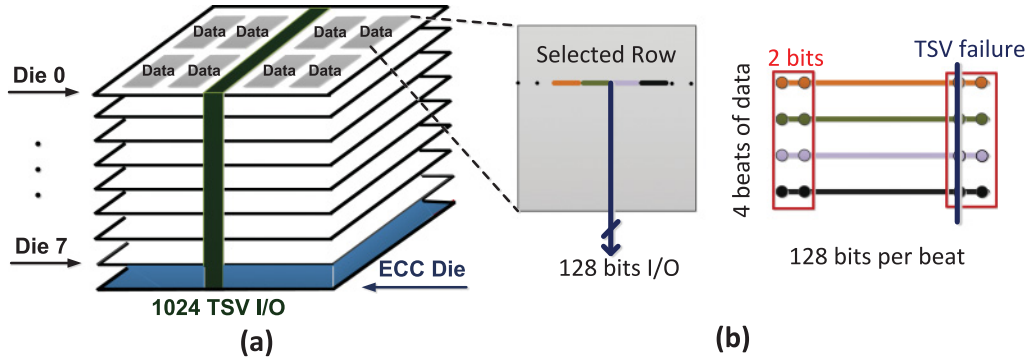

Fig. 1. (a) 3D HBM-like structure with 8 data dies and 1 ECC die; each die has 8 banks. (b) Data of size 512b or 64 symbols are read from one bank in 4 beats (128b per beat); the red box represents an 8b symbol and a TSV failure results in 1 symbol error.

corrects all errors due to small granularity faults and reduces the errors due to large granularity faults significantly. Overall, this scheme reduces the raw failure-in-time (FIT) rate by more than 10^{10}x.

—RATT-ECC uses a strong symbol-based code with small latency in tier-1 (0.52ns for syndrome calculation in 28nm node) to provide very high reliability with minimal performance degradation.

—The rate-adaptive feature of tier-1 code and tier-2 code can be used to dynamically free 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 can be used to handle permanent failures in the data TSVs with no performance overhead.

—The low-latency tier-1 code can also be used to correct additional errors that are introduced by increasing the refresh interval with minimal performance degradation. Simulation-based evaluation with SPEC2006 benchmarks [SPEC2006 2011] demonstrates that, if the refresh interval increases from 64ms to 256ms, the refresh power reduces by 75% while the total power consumption is reduced by 7.1%, 15.1%, 17.6%, and 21.6% when the DRAM size is 8GB, 16GB, 32GB, and 64GB, respectively.

The rest of the article is organized as follows: Section 2 describes the 3D DRAM memory system followed by error characteristics and presents a brief review of existing schemes. Section 3 gives a high-level overview of our proposed ECC scheme and Section 4 presents the detailed design. The simulation infrastructure, hardware overhead, and timing performance are analyzed in Section 5, and the reliability part is analyzed in Section 6. We present our conclusions in Section 7.

2. BACKGROUND AND RELATED WORK

2.1. 3D DRAM Architecture

In this article, we present an ECC-based scheme for improving the reliability of an HBM-like memory. Figure 1(a) shows the internal stack organization of such a memory. There are eight dies to store data; each die has eight banks. We also assume that there is an additional ECC die to store the ECC bits [JEDEC HBM 2013]. The parity check symbols for tier-1 and tier-2 code are stored in the ECC die, making the storage overhead of ECC 12.5%. Each die is assigned an independent channel. There is one controller for each channel and the memory controller (MC), which is housed in the logic die, coordinates the activities of the nine channel controllers.

In each memory request, 512b of data is accessed from a certain bank of a single data die. The data is read over four beats with 128b being read in each beat (see Figure 1(b)). In the HBM standard [JEDEC HBM 2013], each channel has a data width of 128b; thus, there are 128 data TSV lines per channel. Since there are eight channels for the eight data dies, there are a total of 1024 data TSV lines. Eight data bits form a symbol; thus, a single TSV failure leads to only one symbol error, as shown in Figure 1(b).

2.2. Error Characteristics

DRAM errors can be broadly classified into soft errors and hard errors [Sridharan and Liberty 2012]. Soft errors are caused by transient faults that 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. Note that a single fault can result in multiple error instances [Sridharan and Liberty 2012].

DRAM errors can also be classified into those that are due to small granularity faults such as single bit or single column that account for 62.8% to 84.8% of all faults, and large granularity faults such as row and bank failures [Sridharan and Liberty 2012; Sridharan et al. 2013, 2015]. 3D DRAM also has errors due to TSV failures [Nair et al. 2014; Jeon et al. 2014], which fall under small granularity faults. In this article, we analyze the capability of the memory 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 utilize the error-recoding mechanism of machine-check architecture (MCA) [AMD 2011]. The MCA has registers to store the error address, time, and type (corrected or uncorrected) of error. Error events are recorded both during memory scrubbing and during normal read operation [Intel 2011]. Now, 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 MC. If a TSV is marked as faulty, we utilize the erasure correction capability of the tier-1 code (Section 4.4); if the data banks are marked as faulty, we use the rate-adaptive feature of the tier-1 and tier-2 codes to free up banks in the ECC die (Sections 4.2 and 4.3).

2.3. Related Work in Reliability

This section contains a brief description of prior approaches on improving the reliability of 3D Die-stacked DRAM [Nair et al. 2014; Jeon et al. 2014; Giridhar et al. 2013; Jaewoong et al. 2013b]. Of these schemes, E-RAS [Jeon et al. 2014] and Citadel [Nair et al. 2014] also focus on an HBM-like structure and are closest to our proposed scheme.

Subarraykill Giridhar et al. [2013] correct errors due to a subarray failure in the Tezzaron Octopus structure. A data block 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 Jaewoong et al. [2013b] use 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 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.

E-RAS Jeon et al. [2014] propose 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 an 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-1 Nair et al. [2014] also use two-tier ECC protection to handle errors due to small- and large-granularity faults. For the first tier, Citadel-1 uses CRC-32 to provide strong error detection capability. After errors are detected, it relies on multiple levels of parity (3DP) to recover errors. It identifies TSV faults through additional computation and repairs them at runtime through TSV swapping. In addition, it provides for two spare banks to handle permanent bank failures. While Citadel-1 is optimized to handle errors due to permanent faults (TSV, row, bank), it has a very large overhead for correcting errors due to transient faults, including those that affect only a single bit. Recently, Citadel-2 [Nair et al. 2016] was proposed, which uses SEC along with CRC-30 to reduce the decoding latency of single-bit errors. Both Citadel-1 and Citadel-2 have a storage overhead of 14%.

2.4. Related Work in DRAM Refresh

While refresh operations are essential for DRAM operation, they degrade timing performance, energy consumption, and I/O throughput; the degradation increases with the increase in DRAM size. According to Liu et al. [2012], for a large 64GB DRAM device, refresh operations account for 50% of the DRAM power, while degrading memory throughput by 50%. Several techniques have been proposed to reduce the refresh power of 2D DRAM by selectively increasing the refresh interval from the default 64ms that is specified by JEDEC [2010]. The technique in Kim and Papaefthymiou [2003] selects a subset of DRAM blocks to be refreshed using a larger refresh interval. The interval is chosen based on the retention capability of the cells in that block. RAIDR [Liu et al. 2012] and REFLEX [Bhati et al. 2015] also adjust the refresh intervals of DRAM blocks based on their retention capability. RAPID [Venkatesan et al. 2006] is an OS-based solution that allocates pages to the DRAM rows with longer retention time.

Another approach is to extend the refresh interval and handle the extra errors due to retention failures by using a strong ECC code. Hi-ECC [Wilkerson et al. 2010] is one such approach. To reduce storage overhead, it increases the cache-line size to 1KB, resulting in significant bandwidth overhead. Another scheme, Morphable ECC [Chou et al. 2015], uses a strong ECC code with long latency in idle mode to compensate for errors due to long refresh intervals and a weak code during active mode. Our approach to reducing refresh power is also through use of strong ECC code. However, our design reduces refresh power when the computer system is in active mode.

3. 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. A high-level view of RATT-ECC is shown in Figure 2. RATT-ECC is a two-tiered scheme, like E-RAS [Jeon et al. 2014], Citadel-1 [Nair et al. 2014], and Citadel-2 [Nair et al. 2016]. For each memory-read request, along with the data, the corresponding tier-1 ECC parity symbols are read out and used to perform error correction and detection. The tier-1 code is a strong symbol-based code that can correct all small-granularity faults and can detect large-granularity faults with high probability. The tier-2 code is an 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 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.

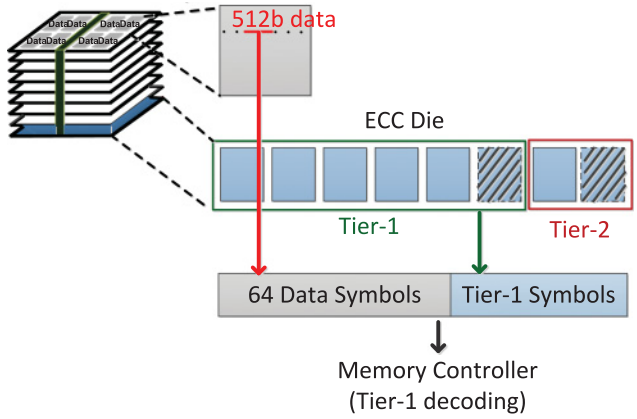

Fig. 2. RATT-ECC overview. Tier-1 codeword is formed by concatenating 512b or 64 symbols (that are read from one of 64 data banks) with tier-1 ECC symbols (that are read from one of banks in the ECC die). Tier-1 is used for correcting errors due to small-granularity faults. Tier-2 is used for correcting errors due to large-granularity faults. Tier-1 and Tier-2 can each provide one spare bank (shaded) to handle permanent bank failures.

3.1. Strategies for Handling 3D DRAM Faults

***Transient small-granularity faults*:** We use a strong tier-1 code that can correct errors due to transient small-granularity faults, such as single-bit, single-column and single-TSV faults, without triggering tier-2, and which has strong detection capability to avoid error leaks.

***Permanent small-granularity faults*:** If the small-granularity faults, such as TSV faults, have been marked as permanent by the MCA, we can treat the errors due to these faults as erasures and use erasure correction to correct them.

***Transient large-granularity faults*:** We rely on the strong detection capability of tier-1 code and the correction capability of tier-2 code to recover from errors due to transient large-granularity faults such as row or bank failures. The strong detection capability of tier-1 is required to avoid error leaks since tier-2 code only performs correction (no detection).

Permanent large-granularity faults: Since correcting errors due to permanent large-granularity faults by tier-2 code has a large timing overhead, we propose the use of spare rows/banks. To handle the case in which there are a few row failures in a single bank, we allocate a few spare rows per bank and house them in the MC. When there is a permanent bank failure, we propose modifying tier-1 or tier-2 code so that fewer banks are required to store ECC bits and these banks can now serve as spare banks.

3.2. 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. This would imply use of a weaker tier-1 code, however, resulting in weaker correction and detection capabilities, which is undesirable. Thus, 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 RS code [Lin and Costello 2004]. If we use RS (72,64) code over GF(2^8), the 12.5%

storage overhead budget would go toward 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 hurts timing performance; thus, we consider only 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 [Lin and Costello 2004]. For example, the code rate for RS (70,64) code is $\frac{64}{70}$.

As shown in the Appendix, 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) 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 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 less memory access without affecting the reliability. Also, the embedded structure is used to implement a rate-adaptive tier-1 code that helps provide a spare bank when needed.

4. 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 4.1), (ii) one data bank is marked as faulty (Section 4.2), (iii) two data banks are marked as faulty (Section 4.3), and (iv) permanent data TSV failures (Section 4.4).

4.1. Scenario 1: No Banks are Marked as Faulty

Here, none of the data banks is marked as faulty; thus, all eight ECC banks can be used by the tier-1 and tier-2 codes. Since we choose RS (70,64) code over GF(2^8) as the tier-1 code, six ECC banks are reserved for storing parity 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 [Lin and Costello 2004]: (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 I lists the detectable and correctable error (DCE) rate, detectable but uncorrectable error (DUE) rate and silent data corruption (SDC) rate [Rao and Fujiwara 1989; Sullivan et al. 2015] for different types of failures for the four cases. We utilize the formula given in Kasami and Lin [1984] to derive the probability of undetected error or SDC rate of RS (70,64) code.

Table I. 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%

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. Furthermore, case (ii) can correct single symbol errors, which is important since small-granularity faults account for a large fraction of all faults (84.8% in 2D DRAM [Sridharan et al. 2015]).

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 be detected even by tier-1 code. Thus, we propose distributing the six symbols of tier-1 code across multiple banks to support power-of-two address mapping and higher reliability.

We can distribute the six symbols in one of the following ways: (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; 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; 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

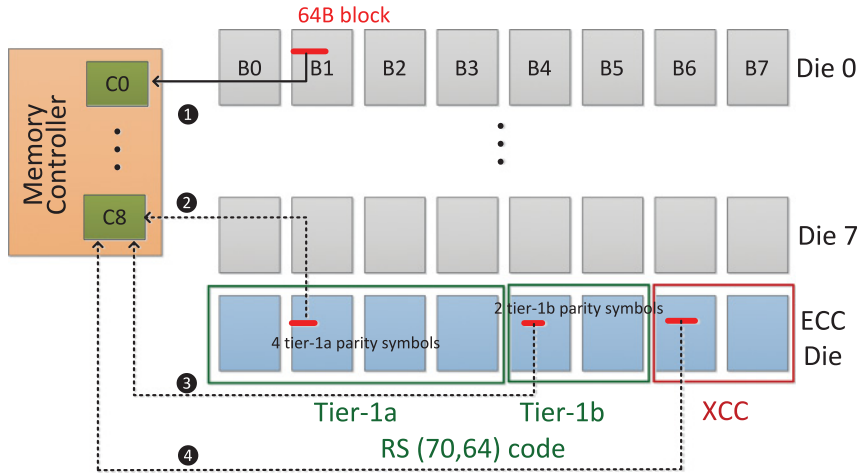

Fig. 3. Scenario 1: High-level diagram illustrating access of data block, tier-1 code, and tier-2 code during read.

lowest number of accesses 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.

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 reading 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); thus, 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 Nair et al. [2014] or Jeon et al. [2014] also utilize ECC caches. We cache tier-1a, tier-1b and tier-2 parity symbols in the ECC cache; however, we have to check only tier-1a for each memory read request in the error-free case.

READ: The sequence of operations during read have been described in Figures 3 and 4. First, a 64B block is read from one of the data dies (step 1). The MC checks whether tier-1a parity symbols are in the ECC cache or not. If it is not cached, the MC reads the corresponding ECC symbols of tier-1a from the ECC bank (step 2) and calculates the syndrome corresponding to the 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 (step 3). The parity symbols of tier-1a and tier-1b and the 64 symbols from the data cache line form a codeword of RS (70,64), which can correct a 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). When tier-2 is launched, 32 data blocks (31 data blocks + 1 ECC data block) have to be read from 32

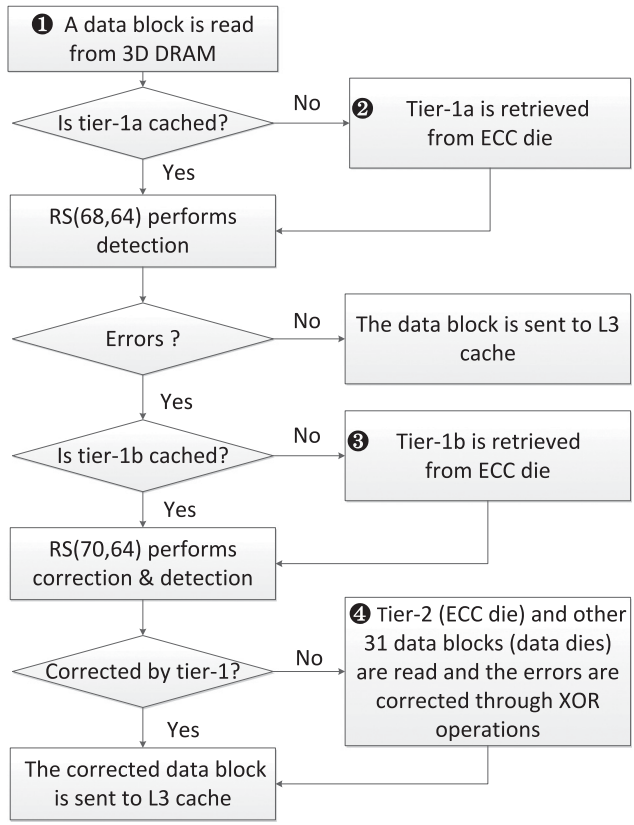

Fig. 4. Scenario 1: Sequence of operations corresponding to a read request.

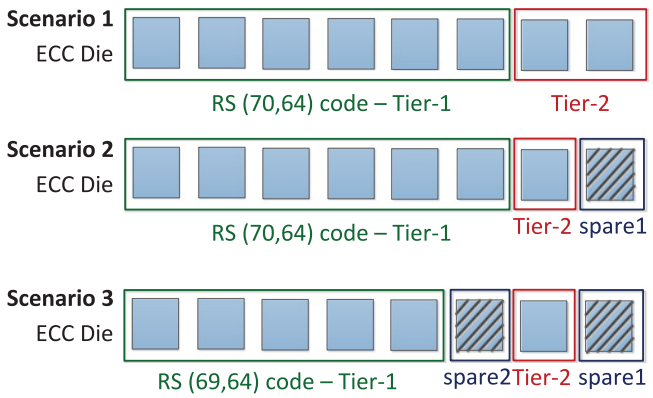

Fig. 5. Three scenarios of RATT-ECC.

banks (31 data banks + 1 ECC bank). Thus, at most 4 data blocks have to be read from each die, resulting in 4 sequential reads per channel.

WRITE: When a dirty cache line is evicted from L3 cache to 3D DRAM, it is written back to one of the data dies, and the corresponding tier-1a, tier-1b, and tier-2 ECC symbols are written back from the ECC cache to the ECC die. The MC needs to issue 3 sequential operations to update the ECC symbols. Although we have one extra write compared to Nair et al. [2014] and Jeon et al. [2014] for updating tier-1 parity, the extra write operations can be well hidden and our simulation results (see Section 5) show that the overhead of this extra write is quite small.

4.2. 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. We propose using 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); thus, 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; 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 changes the error correction latency for only large-granularity faults. We therefore 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 in 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. To recover from errors in a 64B data block, 64 data blocks (63 data blocks + 1 ECC data block) have to be read from 64 banks (63 data banks + 1 ECC bank); thus, there are 8 sequential read accesses per channel.

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

4.3. Scenario 3: 2 Banks are Marked as Faulty

If there are two bank failures, we use RS (69,64) code that 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 correction 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, which cannot be detected and corrected by tier-1 code. Thus, 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

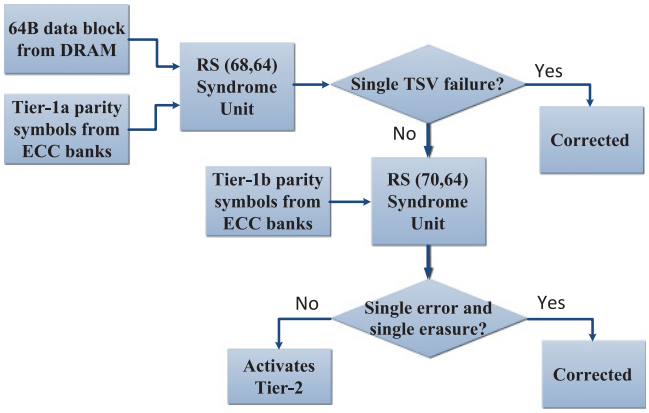

Fig. 6. Decoding flowchart of erasure and error correction of RS (70,64) code to handle permanent TSV data failures.

tier-1a across all three scenarios. Second, the tier-1a continues to have high detection probability. Finally, for Scenario 3, we need to store only the tier-1b parity symbols corresponding to RS (69,64). These fit into one ECC bank, freeing up the second bank. This implies that the tier-1b parity symbols of RS (69,64) that were distributed across two ECC banks now have to be mapped onto one ECC bank.

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

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

4.4. Handling Permanent Data TSV Failures

In the organization described in Figure 1, a single data TSV failure leads to only one symbol error, which can easily be corrected by tier-1 code. However, both tier-1a and tier-1b parity symbols are needed to perform correction; in the worst case, this leads to two additional memory accesses. Now, if the TSV failure is marked as a permanent failure by the MC, the corresponding symbol can be treated as an erasure. Since the error location of an erasure is known (by definition), correcting erasures is a lot simpler compared to correcting random errors.

Figure 6 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). After the syndrome vector is generated, the RS (68,64) decoder checks whether the syndrome vector is a single-erasure event. In the case of an erasure, it corrects it. When there are more errors, two tier-1b symbols are read out from an ECC bank and a 70 (=68+2) symbol codeword is sent to the RS (70,64) decoder, which then checks whether it is a single-error and single-erasure event. If so, it corrects the errors; otherwise, it launches tier-2. The decoding procedures of single-erasure correction of RS (68,64) and single-erasure and single-error correction of RS (70,64) are shown in the Appendix. Note that our tier-1 has been designed to correct single erasure (due to TSV fault) along with a single error (which could be caused by TSV fault) because single-bit errors have a high probability of occurrence. Thus, RATT-ECC can handle errors due to two data TSV faults (one of which is marked faulty) with no performance overhead. In case there are faults in the address TSV lines, we could utilize the method in Jaewoong et al. [2013b], which folds the row index into the

Table II. Simulation Configuration

Processors	
Core	8 cores, 3.2GHz 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 8MB, 24 cycles; 64B cache line
3D Stacked DRAM	
Size	9GB, 8 data layers + 1 ECC layer
Bus frequency	800MHz (DDR3 1.6GHz), 128b per channel
Channels	8 + 1 channels per stack
Memory controller	9 channel controllers
Banks	8 banks per channel
Data block size	64B
Scheduling policy	FR-FCFS
Row buffer	2KB
tRAS-tCAS-tRP	27-9-9

data and computes the exclusive-OR of the data with several copies of the row index to detect row decoder failures.

5. EVALUATION

Section 5.1 describes the methodology for our experimentation. We analyze the hardware overhead of RATT-ECC in Section 5.2. The timing performance with ECC cache is elaborated in Section 5.3. The refresh power reduction achieved by increasing the refresh interval is described in Section 5.4.

5.1. Simulation Infrastructure and Workloads

To evaluate the timing performance of our proposed ECC scheme, we utilize macsim [HPArch 2009], a cycle-level x86 simulator. We model an eight-core processor, each core having a private 32KB(I) cache + a 32KB(D) L1 cache and a unified 256KB L2 cache. The size of the shared L3 cache is 8MB. The memory system uses 3D-stacked DRAM with 8 data dies + 1 ECC die (1GB per die). There are 9 channel controllers with each controller controlling an independent channel. The detailed configuration of our system is described in Table II.

We use 17 SPEC CPU2006 benchmarks and simulate a representative set of 250M instructions identified with SimPoint [Perelman et al. 2003]. 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 III summarizes the workload combinations.

For power-consumption simulation, we use a cycle-accurate simulator, DRAMSim2 [Rosenfeld et al. 2011] to obtain the DRAM power consumption. We simulate the 9-channel structure (8 channels for 8 data dies + 1 channel for 1 ECC die). Since there is no power model for HBM-like 3D DRAM, we utilize the DDR3 power model (1GB to 8GB) provided by Micron [JEDEC 2010] for each of the dies.

We extract the memory traces from macsim for the 17 SPEC CPU2006 benchmarks and feed the memory traces to DRAMSim2. We also add the additional memory accesses to the ECC die for the case in which there is an ECC cache miss or extra correction is needed due to refresh errors.

Table III. Simulation Workloads

Mix	Workload	MPKI	Group
WL-1	8 x astar	13.51	M
WL-2	8 x bwaves	30.8	H
WL-3	8 x bzip2	2.15	M
WL-4	8 x dealII	0.68	L
WL-5	8 x gromacs	0.68	L
WL-6	8 x h264ref	1.24	M
WL-7	8 x lbm	43.34	H
WL-8	8 x libquantum	23.05	HM
WL-9	8 x mcf	83.85	H
WL-10	8 x namd	0.1	L
WL-11	8 x omnetpp	22.23	HM
WL-12	8 x perlbench	0.06	L
WL-13	8 x povray	0.02	L
WL-14	8 x sjeng	0.33	L
WL-15	8 x soplex	27.73	H
WL-16	8 x tonto	0.02	L
WL-17	8 x zeusmp	6.16	M

Table IV. Synthesis Results for 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 and Single-Error Correction	5,658um^2	1.01ns	0.0271 mW (Static) 21.44 mW (Dynamic)

5.2. Hardware Overhead of RATT-ECC

The overhead of the 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 [Rao and Fujiwara 1989] that is used for E-RAS and RS(70,64) code using 28nm industry library. Syndrome and error-correction units for RS(69,64) and RS(68,64) are derived from the RS(70,64) unit. Basically, small additional logic is used to support decoding of the punctured codes (RS(69,64) and RS(68,64)). Table IV 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. If there is

a faulty TSV line, the MC launches single-erasure correction, which has a latency of 0.33ns or single-erasure and single-error correction, which has a latency of 1.01ns.

Considering that an average latency of a memory read is around 30ns to 40ns [Meng et al. 2011] 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 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 sent to the RS(70,64) decoder. When there is a faulty bank marked by the MCA (Scenario 2), data in two ECC banks of tier-2 are XORed and stored in one ECC bank. This step can be done in parallel with storing corrected data in the spare bank. However, when the second spare bank is required (Scenario 3), we move from RS(70,64) to RS(69,64) to free up a bank and only the tier-1b parity symbols that are common to RS(69,64) and RS(70,64) have to be stored. The tier-1b parity symbols of RS(69,64) that were originally stored in the bank marked *spare* have to be routed to the other tier-1b bank. Finally, to handle remapping of faulty banks, we utilize the bank remap table in the MC, which associates a faulty bank address with the spare bank address.

We assume that there are up to 4 row failures in each bank [Nair et al. 2014]. Since each row is of size 2KB, and there are 64 data banks, the MC has a memory of size $4 \times 64 \times 2$ KB $= 512$ KB to store spare rows. To handle remapping of rows, we utilize a row remap table in the MC, which associates a faulty row address with the spare row address.

***ECC cache*:** We consider a simple direct mapped ECC cache with sizes ranging from 256KB to 2MB; the effect of the cache size is analyzed in Section 5.3. The ECC cache size could be reduced with a set associative cache organization.

5.3. Timing Performance Analysis

5.3.1. 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.

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. Since the ECC cache is direct mapped, it takes a few cycles to determine whether it is a hit or a miss. If it is a hit, we assume that there is no delay; if there is a miss, a read request is launched by the MC to retrieve the data from the tier-1a ECC banks. Every ECC cache miss costs one read to the ECC die. This corresponds to tRC = tRAS + tRP (worst case), in which tRAS is 27 cycles and tRP is 9 cycles in our simulation configuration; thus, for a memory frequency of 800MHz, a read operation takes 45ns. For each write-back request, the MC has to update the data block in the data bank along with the tier-1a, tier-1b, and tier-2 parity symbols in the ECC banks. This takes 3 sequential writes to the ECC die, which corresponds to 135ns in our simulation.

5.3.2. Timing Performance of RATT-ECC. We assume that the system is error-free. Also, we simulate only for Scenario 1 (no banks are marked faulty) since it is projected to have the worst timing performance. The ECC cache-line size is also set to 64B, which is consistent with the rest of the memory system. Figure 7 gives the hit rate for different

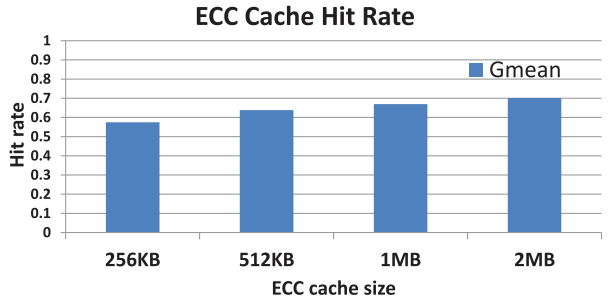

Fig. 7. Hit rate of different-sized ECC caches.

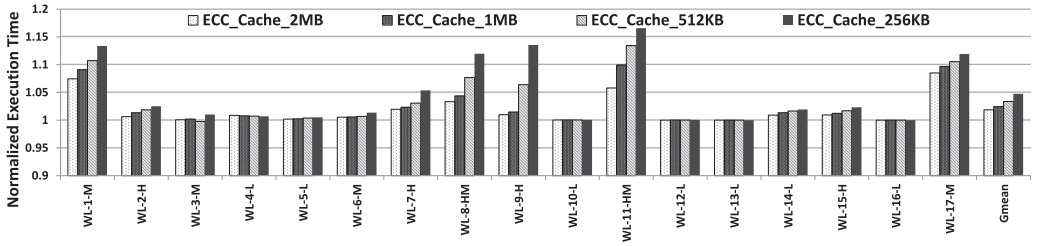

Fig. 8. Normalized execution time for different ECC cache sizes.

ECC cache sizes. When ECC cache size is larger than 1MB, the hit rate is stable at 70%. Figure 8 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 that 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 the 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 of the ECC cache covers 32 data blocks across 32 banks. In Scenario 2, the 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 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. Thus, Scenario 1 will have the worst timing performance, followed by Scenario 2 and Scenario 3.

In the error-free case, RATT-ECC and the competing schemes have similar timing performance when the ECC cache is used. All schemes utilize ECC caches to reduce the extra reads/writes in the ECC die; thus, there is no performance difference. However, if there are errors, RATT-ECC, E-RAS [Jeon et al. 2014], and Citadel-2 [Nair et al. 2016] have comparable performance that is better than Citadel-1 [Nair et al. 2014] since Citadel-1 takes 0.7s to correct transient errors, including single-bit errors.

If there is no ECC cache, our simulation results show that there is a 14% performance degradation. This is because each tier-1a miss during a read costs 45ns and the writes to tier-1a, tier-1b, and tier-2 banks have to be done sequentially and cost a total of $45 \times 3 = 135$ns.

Table V. Number of DRAM Cell Retention Errors as a Function of Refresh Interval in a 60nm Process

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

Note: The default refresh interval is 64ms based on Kim and Lee [2009].

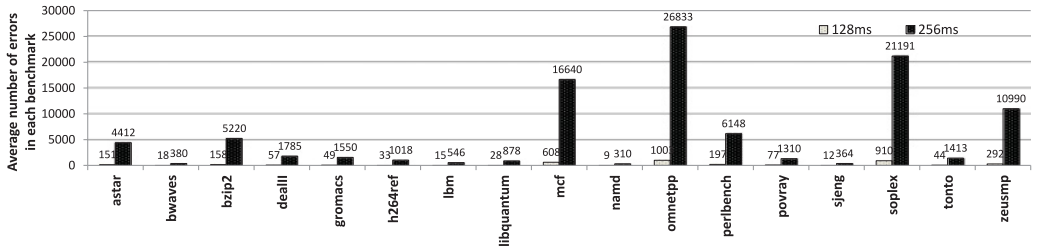

Fig. 9. Number of additional error corrections for 8GB DRAM when the refresh interval is increased.

5.4. Refresh Power Evaluation

Current 3D DRAM memories, such as HBM, are 8GB [JEDEC HBM 2013] and are projected to become larger in the near future. As the 3D DRAM size increases, the refresh power accounts for a larger part of 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 the increase in refresh interval from Kim and Lee [2009]. Table V shows the significant increase in the number of single-bit errors (retention failures) as the refresh interval increases for different memory sizes. According to Kim and Lee [2009], weak cells caused by increasing the refresh interval are randomly distributed throughout the DRAM array. Thus, for each memory read access, there is at most one bit error (among 512b). 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; thus, 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 V. 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 9.

When the memory read access contains data from the leaky cell (single-bit error), the MC 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 IV), respectively.

Figure 10 shows the power reduction of the 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%

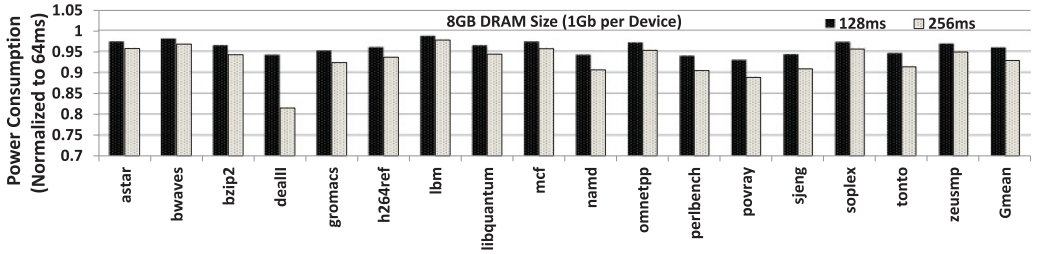

Fig. 10. Power reduction of 8GB memory.

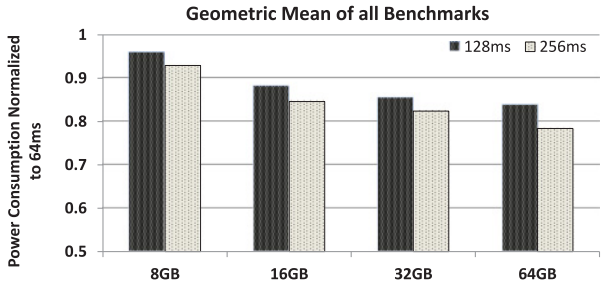

Fig. 11. Power reduction of 8GB to 64GB memory.

and minimum 2%) and 7.1% (maximum 11.2% and minimum 3.2%) when the refresh interval is 128ms and 256ms, respectively.

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 11. For memory size of 16GB, 32GB, and 64GB, the power consumption reduces by an average of 11.9%, 14.6%, and 16.3%, respectively, if the refresh interval is 128ms, and reduces by an average of 15.1%, 17.6%, and 21.6%, respectively, 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 [Jeon et al. 2014], Citadel-1 [Nair et al. 2014], and Citadel-2 [Nair et al. 2016] for larger refresh intervals. When the refresh interval is increased, RATT-ECC, E-RAS, and Citadel-2 have very little overhead in timing performance, since they can all correct the additional single-bit errors using the tier-1 code. However, Citadel-1 takes 0.7s to correct every single-bit error, which increases its total execution time and power consumption significantly.

6. RELIABILITY

In this section, we analyze the reliability of RATT-ECC and compare it with respect to the existing methods. We consider several failure modes. A single-bit failure leads to only a single-bit error in a data block. A single-column failure also leads to a single-bit error in a data block (since data is read out along a row). A single-TSV failure leads to 4b errors in a data block but, due to the access alignment, a TSV failure leads to only one symbol error. A single-row or bank failure leads to multiple bit errors (up to 64 symbol errors) in a single data block.

We begin by analyzing the performance of the tier-1 codes using the three metrics: DCE, DUE, and SDC. RS(70,64), used in Scenarios 1 and 2, has strong correction and

Table VI. Comparison of the Error-Handling Performance of Tier-1 Codes Used in the Existing Schemes

	E-RAS	Citadel		RATT-ECC	
Error Type	4 x Rotational code [Jeon et al. 2014]	CRC-32 [Nair et al. 2014]	CRC-30 & SEC [Nair et al. 2016]	1 x RS (69,64)	1 x RS (70,64)
Single-bit failure	DCE: 100% DUE: 0% SDC: 0%	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: 100% DUE: 0% SDC: 0%	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: 100% DUE: 0% SDC: 0%	DCE: 0% DUE: 100% SDC: 0%	DCE: 0% DUE: 100% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%	DCE: 100% DUE: 0% SDC: 0%
Row or Bank failure	DCE: 0% DUE: 97% SDC: 3%	DCE: 0% DUE: $1 - 2.3 \cdot 10^{-10}$ SDC: $2.3 \cdot 10^{-10}$	DCE: 0 DUE: $1 - 10^{-9}$ SDC: 10^{-9}	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: $2.4 \cdot 10^{-10}$

detection capabilities. Although we read only 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, the other two ECC symbols are read and the 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, used in Scenario 3, can correct one symbol error and detect four symbol errors, and has slightly higher SDC ($5.9 \cdot 10^{-8}$). The DCE, DUE, and SDC rates of RS(70,64) (RATT-ECC Scenarios 1 and 2) and RS(69,64) (RATT-ECC Scenario 3) are summarized in Table VI.

The capability of RATT-ECC to handle errors due to different types of faults is as follows:

***Transient small-granularity faults*:** RATT-ECC can correct all errors due to transient small-granularity faults without activating tier-2.

***Permanent small-granularity faults*:** RATT-ECC uses erasure-correction capability of the RS code to directly correct errors due to permanent data TSV faults by only using tier-1a. The latency of single-erasure correction is only 0.33ns (see Table IV). Since single errors have a high probability of occurrence, RATT-ECC also supports single-erasure and single-error correction. The additional error can be due to a single-bit, single-column, or single-TSV failure, and our design can correct this additional error without activating tier-2.

***Transient large-granularity faults*:** The strong tier-1 code detects almost all errors due to large-granularity faults with a very small SDC rate. These errors are corrected by tier-2 code.

***Permanent large-granularity faults*:** Large-granularity faults tend to have a bimodal distribution as indicated by Nair et al. [2014]. 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. Row failures are handled by utilizing spare rows that are stored in the MC. The rates of tier-1 and tier-2 codes are used to free up ECC banks to be used as spare banks. Thus, RATT-ECC allows for ECC banks to be used as spares only when needed.

FIT Analysis: Real-world field data from Sridharan and Liberty [2012] provides DRAM FIT rate for a 1GB 2D DRAM device. Since field data are not available for 3D DRAM, we scaled the error rates by 10× to account for higher error rates in 3D DRAM, as in Jeon et al. [2014]. The TSV failure rate is also borrowed from Jeon et al. [2014],

Table VII. FIT Analysis

Failure Mode	Raw FIT Trans.(Perm.)	Resultant FIT E-RAS [Jeon et al. 2014]	Citadel-1 [Nair et al. 2014]	Citadel-2 [Nair et al. 2016]	RATT-ECC
Single-bit	142(186)	0(0)	0*(0)	0(0)	0(0)
Single-column	14(56)	0(0)	0*(0)	0(0)	0(0)
Single-row	2(82)	0.06*(2.46*)	$4.6 \cdot 10^{-10}$*(0)	$2 \cdot 10^{-9}$*(0)	$4.8 \cdot 10^{-10}$*(0)
Single-bank	20(142)	0.6*(4.26*)	$4.6 \cdot 10^{-9}$*(0)	$2 \cdot 10^{-7}$*(0)	$4.8 \cdot 10^{-9}$*(0)
Single-TSV	20(21)	0(0)	0*(0)	0*(0)	0(0)
Summary	685	7.38	$5.06 \cdot 10^{-9}$	$2.02 \cdot 10^{-7}$	$5.28 \cdot 10^{-9}$
*means that the correction should be performed by tier-2.					

and we assume that the transient and permanent TSV faults are each 50%. Table VII presents the final FIT rate of the competing schemes along with the breakdown of transient/permanent FIT rates.

Comparison with existing schemes: Since E-RAS [Jeon et al. 2014] uses a rotational code (in Rao and Fujiwara [1989]) to correct errors due to small-granularity faults and to detect errors due to large-granularity faults, its SDC rate for row or bank failures is quite large. While the CRC-32-based scheme in Nair et al. [2014] has very good detection capability, it cannot correct any errors without launching tier-2. The more recent scheme in Nair et al. [2016] can correct single-bit errors and still has good detection capability since it uses CRC-30. Table VI compares the error-handling performance of all the schemes for errors due to each type of failure.

Overall, all schemes can completely remove the errors due to small-granularity faults. For errors due to large-granularity faults, RATT-ECC and Citadel-1 [Nair et al. 2014] can reduce the raw FIT rates to around $5 \cdot 10^{-9}$, which improves the FIT rate by more than 10^{10}x compared to the baseline (no ECC) scheme. Since RATT-ECC and Citadel both provide for spare rows and banks, we assume that all permanent rows or bank faults can be removed. Citadel-2 [Nair et al. 2016] has a slightly larger FIT rate of $2 \cdot 10^{-7}$ compared to Nair et al. [2014] since it uses CRC-30. E-RAS [Jeon et al. 2014] decreases the overall raw FIT rate to only 7.38 because of use of a weaker code in tier-1.

7. CONCLUSION

In this article, we presented RATT-ECC, a two-tiered error-correction scheme that provides very high reliability with minimal performance degradation for HBM-like 3D DRAM memory systems. Unlike existing schemes that either use weaker codes or have large decoding latency, we use RS(70,64), a strong tier-1 code with small decoding latency, that can correct all errors due to small-granularity faults and reliably detect errors due to large-granularity faults (with SDC $2.4 \cdot 10^{-10}$). Being able to correct errors due to small-granularity faults (caused by an increase in refresh interval) with low decoding latency also allows RATT-ECC to reduce refresh power with minimal performance degradation. Finally, RATT-ECC does not earmark banks as spare banks as in existing schemes; instead, it allows two banks that are used for parity storage during normal operation to be used as spares. The dynamic sparing is implemented by utilizing the rate-adaptive feature of the RS and XCC codes. Thus, RATT-ECC is a low-cost solution to significantly improve the reliability of 3D die-stacked DRAM with minimal loss in performance through strong error correction, detection, and dynamic sparing.

APPENDIX

Embedded RS code structure: We show how the codewords of the three RS codes—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 [Lin and Costello 2004]. After shortening, the parity check matrices of RS(69,64) and RS(70,64) can be derived from that of RS(68,64) [Joiner and Komo 1996]. The parity check matrix of RS(68,64), H, is given by

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

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

$$H^{'} = \left(\begin{array}{c|c} & 1 \\ H_{4\times 68} & \vdots \\ & 1 \\ \hline 0 & 1 \end{array}\right).$$

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

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

Assume that 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 Joiner and Komo [1996]. $G^{'}$ is given by

$$G^{'} = (\, G \;\; g^{'} \,),$$

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

$$G^{''} = (\, G^{'} \;\; g^{''} \,),$$

where $g^{'}$ and $g^{''}$ are given in Joiner and Komo [1996].

For encoding, the codeword of RS(70,64) code can be expressed in the form $\vec{c}^{''} = (c_0, c_1, \ldots, c_{67}, c_{68}, c_{69})$, 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 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 $\vec{s} = (s_1, s_2, s_3, s_4)$, can be obtained by removing s_0 and s_5 from $\vec{s}^{''}$. Thus, the syndrome calculation 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, that is, 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 here, multiple generator matrices or parity check matrices would have to be used to implement the rate-adaptive RS codes, resulting in a significant increase in ECC hardware overhead.

Single-erasure correction: To correct one symbol error due to permanent TSV failure, we use the single-erasure correction mode of RS(68,64) code. Here, the faulty TSV symbol, which is marked as an erasure symbol, is replaced with the zero symbol in the received codeword and fed to the syndrome calculation unit. The syndrome vector of RS (68,64), $\vec{s}$, 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 $\vec{s} = e_i h_i$ for column h_i in H. If it holds, the decoder recovers the erasure value e_i in address i of the codeword. Otherwise, tier-1b symbols are read and the single-erasure and single-error correction unit of RS(70,64) is activated.

Single-erasure correction and single-error correction: 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, $\vec{s''}$, is a linear combination of h_i and h_j, where j is from 0 to 69 and $i \neq j$. The hardware consists of 70 subdecoders, with the j^{th} decoder having h_j embedded in it. The MC feeds column h_i to all but the i^{th} subdecoder. If the i^{th} and j^{th} columns of the parity check matrix are $h_i = (h_{i0}, h_{i1}, h_{i2}, h_{i3}, h_{i4}, h_{i5})^T$ and $h_j = (h_{j0}, h_{j1}, h_{j2}, h_{j3}, h_{j4}, h_{j5})^T$, 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 $\tilde{s_2} = e_i \cdot h_{i2} + e_j \cdot h_{j2}$, $\tilde{s_3} = e_i \cdot h_{i3} + e_j \cdot h_{j3}$, $\tilde{s_4} = e_i \cdot h_{i4} + e_j \cdot h_{j4}$ and $\tilde{s_5} = e_i \cdot h_{i5} + e_j \cdot h_{j5}$. If $\tilde{s_2} = s_2$, $\tilde{s_3} = s_3$, $\tilde{s_4} = s_4$, and $\tilde{s_5} = s_5$ all hold, the decoder declares that the error is located in location j of the received codeword and corrects it. Otherwise, it declares that there are more errors.

REFERENCES

AMD. 2011. AMD64 Architecture Programmer's Manual Volume 2: System Programming, May, 2013. http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf.

I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob. 2015. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 235–246.

C. Chou, P. Nair, and M. K. Qureshi. 2015. Reducing refresh power in mobile devices with morphable ECC. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. 355–366.

B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H.-M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw. 2013. Exploring DRAM organizations for energy-efficient and resilient exascale memories. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*.

HPArch. 2009. Macsim simulator. (2009). https://code.google.com/p/macsim/downloads/list.

A. Hwang, I. Stefanovici, and B. Schroeder. 2012. Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design. *SIGARCH Computer Architecture News* 111–122.

Intel. 2011. Intel xeon processor e7 family: Reliability, availability and serviceability: Advanced data integrity and resiliency support for mission-critical deployment. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-e7-family-ras-server-paper.pdf.

S. Jaewoong, Gabriel H. Loh, Vilas Sridharan, and M. O'Connor. 2013a. Resilient die-stacked DRAM caches. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 416–427.

S. Jaewoong, G. H. Loh, V. Sridharan, and M. O'Connor. 2013b. Resilient die-stacked DRAM caches. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. 416–427.

J. Jeddeloh and B. Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Symposium on VLSI Technology (VLSIT'12)*. 87–88.

JEDEC. 2010. DDR3 SDRAM specification. http://www.jedec.org/sites/default/files/docs/JESD79-3E.pdf, July 2010.

JEDEC HBM. 2013. High bandwidth memory (HBM) DRAM, JESD235. https://www.jedec.org/sites/default/files/docs/JESD235A.pdf, Nov. 2015.

H. Jeon, G. Loh, and M. Annavaram. 2014. Efficient RAS support for die-stacked DRAM. In *IEEE International Test Conference (ITC'14)*. 1–10.

X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar. 2013. 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 (SC'13)*.

L. L. Joiner and J. J. Komo. 1996. Time domain decoding of extended Reed-Solomon codes. In *Proceedings of the IEEE Southeastcon'96. Bringing Together Education, Science and Technology.* 238–241.

T. Kasami and S. Lin. 1984. On the probability of undetected error for the maximum distance separable codes. *IEEE Transactions on Communications* 32, 9, 998–1006.

J. Kim and M. C. Papaefthymiou. 2003. Block-based multiperiod dynamic memory design for low data-retention power. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11, 6, 1006–1018.

K. Kim and J. Lee. 2009. A new investigation of data retention time in truly nanoscaled DRAMs. *IEEE Electron Device Letters* 30, 8, 846–848.

S. Lin and D. J. Costello. 2004. *Error Control Coding* (2nd ed.). Pearson, New York.

J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. 2012. RAIDR: Retention-aware intelligent DRAM refresh. In *39th Annual International Symposium on Computer Architecture (ISCA'12)*. 1–12.

D. Locklear. 2000. Chipkill correct memory architecture. *Dell Enterprise System Group.* http://www.ece.umd.edu/courses/enee759h.S2003/references/chipkill.pdf, Aug. 2000.

G. H. Loh. 2008. 3d-stacked memory architectures for multi-core processors. In *35th International Symposium on Computer Architecture (ISCA'08).* 453–464.

J. Meng, D. Rossell, and A. K. Coskun. 2011. 3D systems with on-chip DRAM for enabling low-power high-performance computing. In *IEEE High Performance Embedded Computing, HPEC*.

P. Nair, D. Roberts, and M. Qureshi. 2014. Citadel: Efficiently protecting stacked memory from large granularity failures. In *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. 51–62.

P. J. Nair, D. Roberts, and M. Qureshi. 2016. Citadel: Efficiently protecting stacked memory from TSV and large granularity failures. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4, 49:1–49:24.

E. Perelman, G. Hamerly, M. Biesbrouck, T. Sherwood, and B. Calder. 2003. Using SI'mPoint for accurate and efficient simulation. In *Proceedings of the 2003 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'03)*.

T. R. Rao and E. Fujiwara. 1989. *Error-Control Coding for Computer Systems*. Prentice-Hall, Upper Saddle River, NJ.

P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters* 10, 1, 16–19.

SPEC2006. 2011. SPEC CPU2006 benchmark suite. Retrieved July 23, 2016 from http://www.spec.org/cpu2006/.

V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. 2015. Memory errors in modern systems: The good, the bad, and the ugly. *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)* 297–310.

V. Sridharan and D. Liberty. 2012. A field study of DRAM errors. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*.

V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. 2013. Feng Shui of supercomputer memory: Positional effects in DRAM and SRAM faults. *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13).*

M. Sullivan, J. Kim, and M. Erez. 2015. Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'15)*.

Tezzaron. 2014. Octopus. Retrieved July 23, 2016 from http://www.tezzaron.com/.

A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi. 2012. LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems. In *International Symposium on Computer Architecture (ISCA'12)*. 285–296.

R. K. Venkatesan, S. Herr, and E. Rotenberg. 2006. Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM. In *12th International Symposium on High-Performance Computer Architecture (HPCA'06)*. 155–165.

C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. 2010. Reducing cache power with low-cost, multi-bit error-correcting codes. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*.

D. Yoon and M. Erez. 2011. Virtualized ECC: Flexible reliability in main memory. *MICRO* 31, 1, 11–19.

Received March 2016; revised May 2016; accepted June 2016