MEMRES: A Fast Memory System Reliability Simulator

Shaodi Wang*, Henry (Chaohong) Hu†, Hongzhong Zheng† and Puneet Gupta*
*Department of Electrical Engineering, University of California, Los Angeles
†Samsung Semiconductor, Milpitas, CA, USA

***Abstract*—With scaling technology, emerging non-volatile devices and data-intensive applications memory faults have become a major reliability concern for computing systems. With various hardware and software approaches proposed to address this issue, a comprehensive evaluation is required to understand the effectiveness of these solutions. Considering the complex nature of various memory faults as well as interactions between various correction mechanisms we propose MEMRES, a fast memory system reliability simulator. It enables memory fault simulation with ECC algorithms and modern memory reliability management, including memory page retirement, mirroring, scrubbing, and hardware replacement. MEMRES is computationally efficient in obtaining memory failure probabilities in presence of multiple failure mechanisms and complex correction scheme, allowing for optimization of memory reliability.**

I. Introduction

Increasing data-heavy applications motivate the need of a reliable memory system. However, growing memory density and volume have resulted in increasing memory fault rate [1]. Scaling technology and emerging non-volatile technologies exacerbate the problem [2] [3]. In addition, failure recovery affects system's serviceability, and faulty memory replacement increases costs in data center. Hence, improving the reliability of future memory system is essential. This requires deep understanding of failure mechanisms.

Experimental field studies find that memory failure is caused by a large variety of memory fault types [4], and hard failures are the dominating factor [1]. Various techniques (see Fig. 1) have been proposed to enhance system reliability and prolong lifetime. However, most previous memory reliability studies have relied on analytical models like [5]. These models are insufficient to handle a variety of memory fault types simultaneously and deal with reliability enhancement techniques for multiple reasons. Firstly, a new fault type may significantly change existing fault models [5] such as data-link error [6] [7]. Secondly, memory failure modeling is strongly dependent on reliability enhancement techniques. For instance, single-bit error is the most critical fault in a memory system without error-correcting code (ECC) but becomes less critical for a memory system with single-error correcting and double-error detecting (SECDED) ECC [8] [9]. Thirdly, sophisticated ECCs increase the difficulty for analytical model derivation, e.g., [5], for the dynamic random-access memory (DRAM) with a double-Chipkill [10] [11]. Fourthly, fault rate is strongly application dependent [1], which is hard to model. Fifthly, memory reliability management (see Fig. 1) further increases modeling difficulty. Therefore, efficient simulation methodologies that can overcome these difficulties must be introduced.

Various fault simulators have been well developed and used in industry and academia [12] [13]. However, they focus on simulating fault propagation and detection. This makes them unsuitable to analyze memory fault due to unacceptably long run time on circuit simulation (i.e., orders of magnitude slower than simulated time). In order to obtain failure probability of memory system, a large amount of Monte-Carlo simulations are required, increasing the difficulty of using slow simulators to analyze memory reliability problems. Recently, Faultsim [14], a fast Monte-Carlo DRAM fault simulator, was proposed, which took a few hours to obtain seven-year DRAM failure probability. Compared to analytical methodology, it saves effort on model derivation for various memory designs. However, Faultsim does not account for memory access behavior, which is crucial to modeling memory reliability management. Memory access patterns also affect simulation accuracy because failure behavior strongly depends on applications [1].

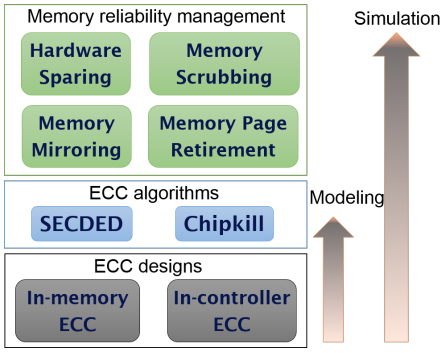


Fig. 1. The reliability enhancement techniques.

In this paper, we propose MEMRES, a comprehensive memory fault simulator. It performs long-term (i.e., over years) application-based fault simulation of memory system with modern reliability enhancement techniques. Our contributions are summarized as follows:

- We implement a comprehensive memory fault simulator with efficient data structures and accurate modeling. This simulator takes a few hours to obtain precise probability and reasons for memory system failure.
- Application-based memory access behavior is modeled and incorporated in MEMRES. The access behavior ignored by existing models and simulators has been demonstrated to significantly impact fault behavior. Considering the memory access model not only improves MEMRES's accuracy, but is also a key to model memory reliability management.
- Memory reliability management is modeled including memory scrubbing, row/column/rank sparing, memory page retirement, and memory mirroring. This enables system-level memory reliability evaluation.
- Multiple failure mechanisms including data-link error [6] [7] are modeled and demonstrated to affect the effectiveness of reliability enhancement techniques.

MEMRES modeling and implementation are described in Section II. MEMRES is validated in Section III. The examples of MEMRES simulation are shown in Section IV. In the end, we conclude our work in Section V.

II. MEMRES Framework

Fig. 2 illustrates the overview of MEMRES. MEMRES comprises of pre-sim processing and Monte-Carlo simulator.

The pre-sim processing is a one-time procedure for fault and memory access model fitting. It passes fitted parameters including access range (AR) and fault failure-in-time (FIT) to Monte-Carlo simulator. The AR is to model the memory access behavior of

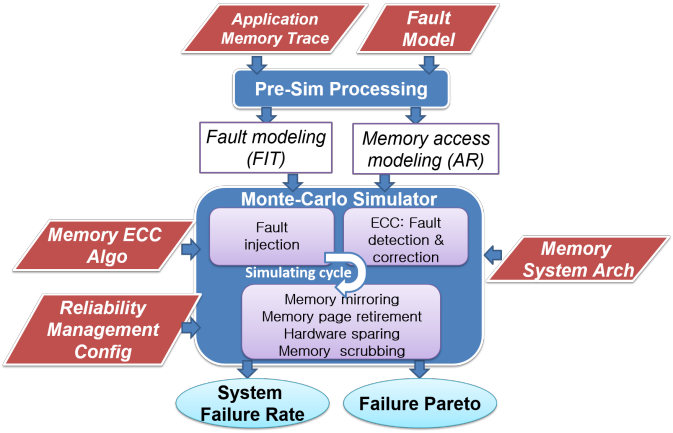


Fig. 2. The framework overview of MEMRES.

applications. The FIT (i.e., expected number of failures in a unit of device-time) models fault rate.

Inputs to Monte-Carlo simulator include memory architecture (see Fig. 3), ECC algorithm and configuration of memory reliability management. During the Monte-Carlo simulation, the simulator divides memory lifetime into short intervals and then simulates one interval after another. For each interval, a sequence of three steps is performed as illustrated in Fig. 2. During the initial step, probabilities of different fault types are computed through the use of a fault model, and faults are injected according to the probabilities. In the second step, if a fault is accessed, ECC is performed to detect and correct the error produced by the fault. In the final step, memory reliability management is used to remove the detected faults. MEMRES has two termination conditions. The appearance of uncorrected fault causes crash of simulated memory system and stops the simulation. If there is no such fault, the simulation continues till the end of preset time. Outputs of MEMRES include a memory system failure probability as a function of operating time; and failure pareto (i.e., a histogram showing the probability of system failure caused by different reasons).

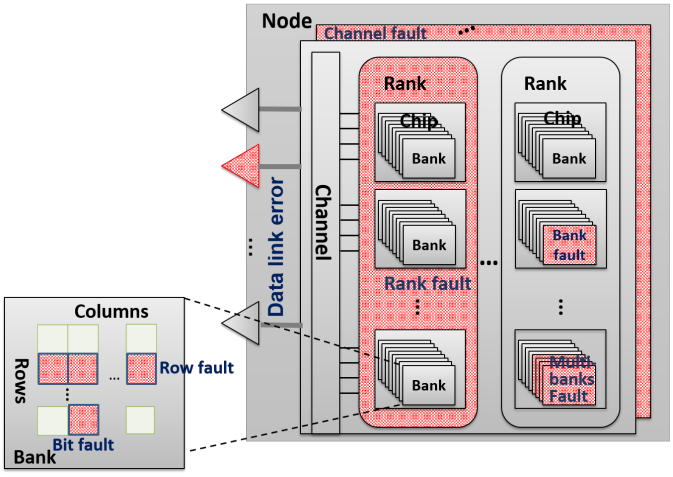


Fig. 3. Memory architecture and memory fault types. To read/write a word, one channel and rank are selected, and then all chips in the selected rank are read simultaneously. Eight banks are built in a chip. The data sharing same address (e.g., 4 bits or 8 bits) in chips across a rank are read together in a word (64 bits).

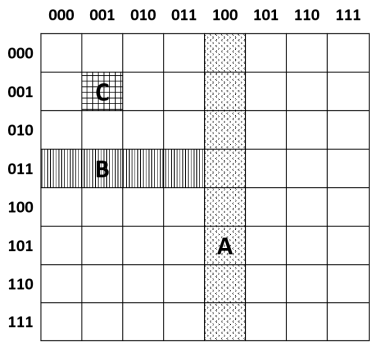
A. Data Structure

Many detected faulty bits are found to gather and form big faulty regions, e.g., a row, a column, and a bank [4]. A fault type is then defined according to the faulty region as can be seen in Fig. 3 which shows examples of different types of faults. Faultsim [14] uses a data structure of Mask and Address to represent large faults, where Mask specifies the fault size, and Address locates the fault in memory space. This data structure is not only fast in calculation but also occupies little memory. However, not all addresses covered by such representation are faulty. The percentage of faulty bits in different fault types varies from 20% to 90% [4]. In addition to Mask and Address, we use Cover-rate (ranging from 0 to 1) to represent the percentage of faulty bits in a fault type. The Mask, Address, and Cover-rate form one basic data structure, fault range (FR), in MEMRES. Fig. 4 shows examples of FRs A, B, and C.

As can be seen from Fig. 4, Address and Mask are sets of binary bits. They represent a region of device addresses (i.e., address of physical location in memory). The setting rule of Address and Mask is described as below. Initially, all bits are set to 0. If certain bit of all binary addresses covered by the representing FR is fully masked (i.e., changing this bit of the addresses does not move them out of the FR), this bit of Mask is set to 1, and that of Address is set to 0. Otherwise the bit of Mask is set to 0 and the bit of Address is set to the same number as covered addresses (i.e., addresses covered by a FR should have the same number in such bit). For example, in Fig. 4, A (a column fault) has the mask of "11100", where the"111" means that all row addresses are covered by the FR (i.e., the three bits cannot decide whether an address is or is not covered by the FR), and "000" points the FR to unique column address. If bits are set correctly, the Address should have the same binary representation as the smallest address covered by the FR. For example, in Fig. 4, the A's Address of "000100" locates the FR at the column 5, which equals to the device address of row 1 and column 5.

One FR can represent a fault with the size equaling to powers of two. It can represent all fault types in Fig. 3. A fault with size S not exactly equaling powers of two can be divided into no more than log_2^S parts such that each part can be represented by an FR. The FR helps MEMRES to save memory space by orders of magnitude compared to traditional simulators.

In order to catch application dependent fault behavior and model



Fault/Access Range Representation

	Mask		Address		Coverrate
	Row	Column	Row	Column	
A	111	000	000	100	0.75
B	000	011	011	000	0.5
C	000	000	001	001	1

Fig. 4. Examples of AR/FRs A, B, And C. A is a column FR/AR, B is a 4-bits FR/AR, and C is a single-bit FR/AR.

system-level reliability enhancement techniques, MEMRES uses AR to model memory access behavior. AR also comprises of Mask, Address, and Cover-rate. Mask specifies the size of memory access covered by an AR, Address locates the memory access region, and Cover-rate represents the percentage of accessed addresses in the AR.

B. Pre-sim Processing

In the pre-sim processing, memory traces are processed to be represented by ARs. At first, the whole memory access space is divided into multiple regions. In a simulation cycle. each region is covered by an AR. Cover-rate specifies the percentage of accessed addresses in the region. In this model, accesses in the region covered by an AR are assumed to be random implying that Cover-rate also represents the probability of an address being accessed. ARs differ from regions since memory access density varies from location. And for the same region, ARs also vary from simulation cycles for the reason that memory access load changes over time.

Although AR can be set to cover large memory space so that fewer ARs are used in simulation, the modeling accuracy is improved when small ARs are used. Smaller ARs more accurately catch access behaviors by holding more precise Cover-rate. Nevertheless, smaller ARs result in more ARs and increasing runtime. The expectation and variance of the number of cycles for a fault being accessed once in our modeling are $(1-c)^s/(1-(1-c)^s)$ and $(1-c)^s/(1-(1-c)^s)^2$ respectively, where c is the Cover-rate product of the fault's FR and the AR accessing the fault, and s is the size of the fault. As can be seen, when there is large fault or high Cover-rate in AR and FR, the error of expectation and variance is very limited even if the Cover-rate is not accurate. Therefore, for memories with intensive access or large amount of faults, large ARs can be used to speed up simulation without losing accuracy.

FIT is calculated for each AR to obtain accurate application dependent fault rate. Emerging memory may require specific fault modeling.

C. Basic Operations

Computation using FR/AR is efficient. Three basic bitwise operations in MEMRES are INTERSECT, MERGE, and REMOVE. They have the closure property (i.e., both the operands and results of these operations are FR/ARs). Most operations in MEMRES are implemented by them.

INTERSECT is the most used operation in MEMRES. It tests whether two faults from different chips can be accessed in a word (64 bits) and whether an AR accesses an FR. The answer is only true if two data structures intersect, meaning that the two FR/ARs (may come from different chips) share identical addresses. Fig. 5 (a) shows an example of calculating INTERSECT(A, B). When FR/ARs A and B intersect, the bit-wise formula shown in (1) results "1"s for all bits. And the FR/AR of intersection between A and B is obtained using (2).

$$(A_{Mask} + B_{Mask}) + \overline{A_{Address} \oplus B_{Address}} \quad (1)$$

$$INTERSECT_{Mask} = A_{Mask} \& B_{Mask} \quad (2)$$

$$INTERSECT_{Address} = A_{Address} + B_{Address}$$

$$INTERSECT_{Coverrate} = A_{Coverrate} \cdot B_{Coverrate}$$

MERGE is mainly used to combine two data structures, such as merging two faults to obtain the combined fault. In Fig. 5 (b), a MERGE operation of A and B is illustrated. Four steps are applied in this operation. Firstly, find an intersection between A and B. Secondly, create an FR/AR for the intersection with the Cover-rate calculated using (3). The third step entails removing the intersection from A and B. Procedure 1 details the removing step, which uses the least number of AR/FRs to cover the remaining parts of A and B. Finally, the intersection and remaining parts are stored together in a collection. In order to merge A to a collection of AR/FRs B, the MERGE is iteratively performed between A (or the remaining parts of A) and every AR/FR in B.

Procedure 1 Remove FR/AR B from PR/AR A

Input: PR/AR A and PR/AR B contained by A.
Output: The collection of the remaining FR/ARs in A after removing B

1: **for** i from the index of the MSB to the that of the LSB **do**
2: **if** $T_{Mask}(i)$ == 1 && $B_{Mask}(i)$ == 0 **then**
3: //split T into two halves $T0$ and $T1$
4: Copy T to $T0$ and $T1$
5: Set $T0_{Mask}(i)$ = 0 and $T1_{Mask}(i)$ = 0. Set $T0_{Address}(i)$ = 0 and $T1_{Address}(i)$ = 1
6: **if** $T0$ intersects B **then**
7: Add $T1$ into the collection R. Set T = $T0$
8: **else**
9: Add $T0$ into the collection R. Set T = $T1$
10: **end if**
11: **end if**
12: **if** T == B **then**
13: break loop
14: **end if**
15: **end for**
16: **return** R

$$Coverrate = 1 - (1 - A_{Coverrate}) \cdot (1 - B_{Coverrate}) \quad (3)$$

REMOVE is mainly used to clear faults or to block access to certain addresses. For instance, after a rank replacement (i.e. replacing a faulty rank with a spare one) all faults in the rank are cleared. Fig. 5 (c) illustrates removing B from A. Firstly, find the intersection between A and B. Secondly, Procedure 1 is used to remove the intersection from A. Thirdly, create an FR/AR for the intersection with Cover-rate calculated using (4). Finally, store the remaining parts of A and the intersection in a collection (if the intersection's Cover-rate is positive).

$$Coverrate = 1 - (1 - A_{Coverrate}) \,/\, (1 - B_{Coverrate}) \quad (4)$$

D. Fault Injection

Fault behavior is application dependent. In [4], large volume of memory faults appear at the beginning of usage, and then faults occur at a stable rate in the remaining lifetime. Different behavior is shown in [1] that fault appearing rate increases with time. Analytical models assume a constant FIT like [5] to simplify derivation meaning. In MEMRES, the fault model is not restricted to any above situations and can dynamically change as a function of time. At the beginning of every simulating cycle, probability of injecting $fault_i$ under memory access space AR_j is calculated in (5). Fault injection is performed based on the probability.

$$P\left(i, AR_j\right) = 1 - \exp\left(-FIT_{i,j} \cdot T_{cycle}\right) \quad (5)$$

where $FIT_{i,j}$ is the failure rate for $fault_i$ under AR_j, and T_{cycle} is the time interval of simulation. Extending the interval in a simulation decreases the total number of intervals resulting to speed up simulation. Nevertheless, high fault rate demands short simulation interval in order to maintain the accuracy of random fault injection. Because MEMRES injects a maximum of one fault for each type in an interval. The error of the single fault injection model used in MEMRES, the probability of appearance of multiple faults of one type, increases with interval length. For the fault rates reported in

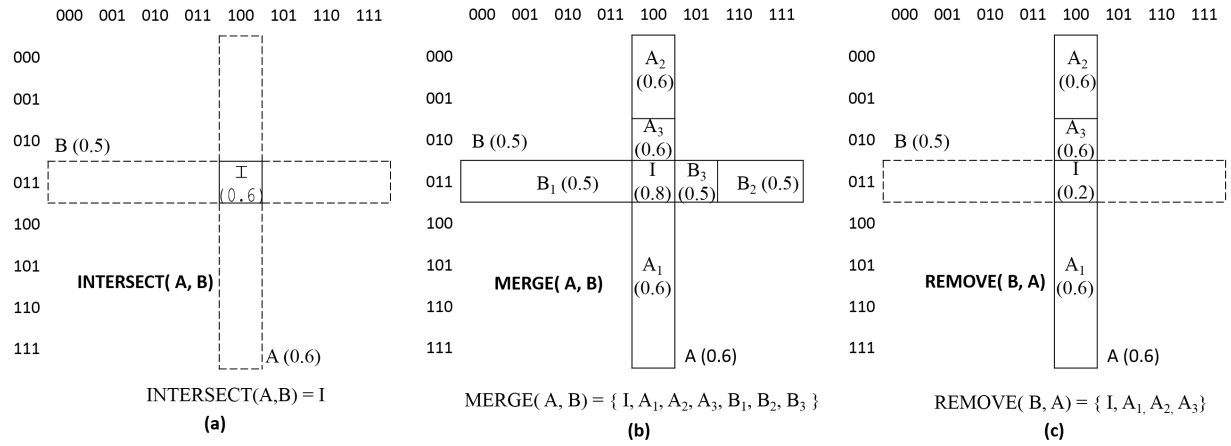


Fig. 5. The basic operations used in MEMRES: (a) INTERSECT, (b) MERGE, and (c) REMOVE. Cover-rates of A and B are 0.6 and 0.5 respectively.

[4], one hour length of a simulation interval is acceptable. If a fault is to injected to an AR, its injection location is randomly chosen in the AR.

Data link bus creates many errors due to bus write/read interference, clock jitter and voltage variation. Bit error rates (BER) of 10^{-14} and 10^{-16} are reported in [6] and [7] respectively. But single-bit data-link error can be easily corrected by ECC, and multiple-bits data-link error is very rare. Data-link error may affect memory system when it occurs simultaneously with other memory errors. To avoid useless data-link error injection to a memory with ECC, whether to inject data-link error is only checked when a memory fault is accessed. The injection probability is calculated as in (6).

$$P_{Bus} = 1 - (1 - BER)^{N_a \cdot N_b} \tag{6}$$

where BER is error rate of single-bit data link, N_a is the expected number of accesses of the memory fault in current simulation interval and N_b is the number of bits in the bus excluding faulty bits from memory (e.g., N_b is 63 when a single-bit memory fault is read by a 64-bits bus).

E. ECC Algorithms

SECDED and Chipkill are the most popular ECC algorithms in modern systems. SECDED can correct single-bit error and detect double-bit error. Chipkill can correct and detect one faulty symbol (i.e., bits from single chip). Stronger Chipkill may detect and correct two symbols [5]. In MEMRES, ECC algorithm is configurable with four parameters: maximum detectable faulty bits, maximum correctable faulty bits, maximum detectable faulty symbols, and maximum correctable faulty symbols. The four parameters cover the functionality of most ECC algorithms. Although stronger ECC leads to overhead of hardware, delay, and power inefficiency, this is outside the scope of this paper.

F. Memory Reliability Management

Memory faults are classified into two classes: transient and permanent. Transient faults cause soft error and disappear after being overwritten while permanent fault frequently produces hard errors and cannot be repaired. ECC can correct some memory errors, though faults, especially permanent faults, accumulate with time and may produce uncorrectable multiple-bits errors. To avoid fault accumulation, modern systems use memory reliability management to deactivate existing faults. These techniques need identified fault location, which can be identified by ECC from detected errors. MEMRES models the identification process. We use one collection of FRs called faulty addresses collection (FAC) to store all faulty addresses. It merges all existing faults. Errors are produced when the FAC is accessed by ARs (i.e., ARs intersect with FAC). Then detectable errors are saved in a collection called error record (ER). MEMRES analyzes ER so that addresses frequently producing errors are classified as permanent faults. Both FAC and ER are updated using the MERGE operation during every simulation interval.

Modeling of memory reliability managements is detailed below:

- Memory scrubbing periodically removes transient faults to avoid accumulation. In MEMRES, the scrubbing cycle is configurable. During the scrubbing, MEMRES removes all transient faults from FAC using the REMOVE operation.
- Hardware sparing is used to remove permanent faults. For example, rank sparing protects ranks with spare ones. The protected rank is replaced when detected permanent faults reach a configurable threshold. Then REMOVE is applied to clear faults belonging to the replaced hardware from MEMRES's database.
- Memory page retirement (MPR) retires faulty memory pages. MPR blocks these pages and moves data to other pages to avoid accessing these faults again. The page here uses physical address, but AR uses device addresses. So such retired page firstly needs to be mapped to physical addresses. Then they are removed from all memory access (ARs) using REMOVE. The maximum number of retired pages in a rank is configurable.
- Memory mirroring is done such that two memory spaces store the same data. The system only visits one space until errors are detected, and then it shifts to the other one. This protection is frequently used for critical data. In MEMRES, the mirrored space is configurable. When access to space A is changed to its mirror B, MEMRES performs three operations: deleting A from memory access (ARs) using REMOVE, changing the deleted ARs' Address to B, and adding them back to memory access (ARs) using MERGE.

III. Framework Validation

We cannot substantially validate our framework. Large scale experimentation is too expensive and impractical to most research groups given the desire to look at failures over an extended period of time. Existing simulators take unacceptable time to complete the validation task, and existing analytical models do not support memory reliability management. Nevertheless, in this section, we derive an analytical

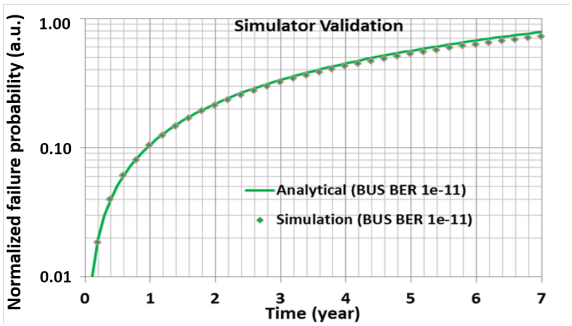


Fig. 6. The normalized failure probability for a 16GB main memory as a function of time.

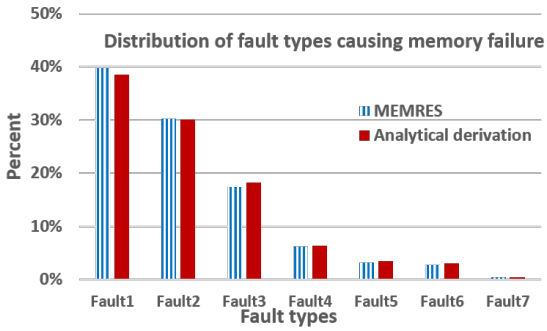


Fig. 7. The histogram of reasons for memory failure. We omit names of fault types for this validation.

model to compare its accuracy with MEMRES. High-level reliability managements are not validated.

Here we derive an analytical model with Cover-rate (Cover-rate is not included in existing models) for a memory with SECDED. SECDED cannot correct any multiple-bits errors, which may be caused by multiple-bits faults or multiple intersecting single-bit faults. We omit the derivation for the case of multiple-bits fault, which is a simple exponential distribution model. For multiple single-bit faults' intersection, we only consider the case of two single-bit faults' intersection, because the probability of intersections of over two faults is orders of magnitude less likely than that of two faults, which thereby contributes little to memory failure. The intersections of two single-bit faults are classified into two classes: two intersecting single-bit memory faults and one single-bit memory fault collided with a data-link error.

The analytical model for the intersection of two single-bit memory faults is shown as below:

$$P_{mm}(T) = \int_0^T P_1(t_1) \cdot \left(\int_0^{T-t_1} P_2(t_1+t_2) \cdot P_S(T-t_1-t_2) \right) \quad (7)$$

where $P_{mm}(T)$ is the probability of the intersection occurring within time T, $P_1(t_1)$ is the probability of the first fault appearing at time t_1, $P_2(t_1+t_2)$ is the probability of the second fault appearing at t_1+t_2, and $P_S(T-t_1-t_2)$ is the probability of the two faults intersecting and being accessed within $T-t_1-t_2$. $P(t_1)$ is detailed as $\exp(-\lambda_1 t_1)\cdot(1-\exp(-\lambda_1 dt_1))$, where λ_1 is its failure rate, the first term in parenthesis is the probability of the fault not appearing from the start to time t_1, and the following one is the probability of it appearing during (t_1, t_1+dt_1) that can be further simplified to $\lambda_1 dt_1$. Similarly, $P_2(t_1+t_2)$ can also be simplified. $P_S(T-t_1-t2)$ is $1-(1-C_I)^{N_a(T-t_1-t_2)}$, where I is the intersection of the two faults and the ARs accessing them, and C_I is the Cover-rate of I, and N_a is the number of access to I in unit time.

The model for a memory fault coupled with a with data link error is calculated below:

$$P_{ml}(T) = \int_0^T P_1(t_1)\, P_2(T-t_1) \quad (8)$$

where $P_1(t_1)$ has the same meaning as in (7), $P_2(T-t_1)$ is the probability of a data-link error occurring simultaneously with the memory error.

Fault FITs are obtained from [4]. As is illustrated in Fig. 6, MEMRES matches with the analytical model for a 16-GB memory with SECDED. The failure probability increases with time for the reason that memory faults accumulate. The slight mismatch ($< 10\%$) is observed in the region of high failure probability. This is because the analytical model ignores high order coupling effects between multiple faults. In reality and simulation, a memory fault causing memory failure results in system crash and masks other potential faults, while in analytical model the probability of each fault is individually calculated without considering interaction. In addition, error of ignoring the high order effects grows with increasing failure probability. However, high order effects are too complicated to be analytically modeled.

Fig. 7 shows the probability fraction of different fault types to cause memory failure. Small difference exists again between MEMRES and the analytical model. For the most critical Fault1, analytical model underestimates its contribution comparing to MEMRES, but reverse situations are shown for non-critical faults. Again the difference is present due to the high order effect ignored by the model. The critical fault is more likely to mask other faults in reality.

The run time complexity of MEMRES is $O\left(N \cdot \log(M)^3\right)$ where M is the size of memory system and N is the failure rate. The size and number of data structures scale with the address length, $\log(M)$. INTERSECT scales with the size of data structure, $\log(M)$. MERGE and REMOVE, the main operations, described in Procedure 1 scale with the number of data structure ($\log(M)$) and INTERSECT ($\log(M)$), which is $\log(M)^2$. The number of operations is proportion to the product of the number of injected faults and the number of data structures, which is $N \cdot \log(M)$. The complexity of total run time is the product of the number of operations and the operations' complexity, which is $O\left(N \cdot \log(M)^3\right)$. The memory consumption is decided by the number and size of data structures, which is $O\left(N \cdot \log(M)^2\right)$. In experiments, the wall time for the validation (100,000 7-years simulations) of Fig. 6 and Fig. 7 on a workstation is 26 minutes.

IV. Examples of MEMRES Usage

In Fig. 8, a memory with SECDED is simulated. As the data-link BER increases from 10^{-15} to 10^{-10}, more single-bit memory errors are collided with data-link errors resulting in more uncorrectable errors. As a result, the failure rate increases by 25% (from 0.8 to 1.0) demonstrating that ECC-correctable errors can still cause failure when high volume of data-link errors are produced. This increase may saturate with data-link BER after all single-bit memory errors have collided with data-link errors. To reduce such failures, data-link BER has to be controlled below 10^{-14}.

To explore the features of system-level reliability enhancement techniques, MEMRES is used to evaluate the reliability of a 128-GB memory with memory reliability managements. The input of failure rates is obtained and scaled from [4]. We separately apply three

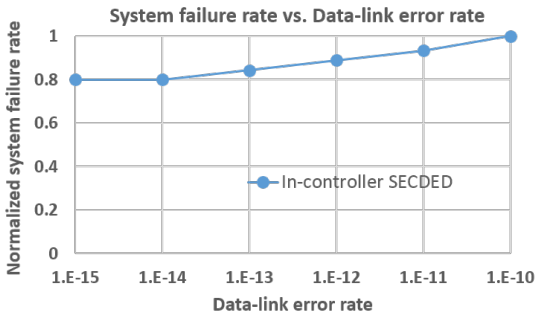


Fig. 8. Normalized memory failure rate vs. data link error rate for a memory system with SECDED.

TABLE I
SIMULATION CONFIGURATIONS OF MEMORY RELIABILITY MANAGEMENT

Configurations	Config. 1	Config. 2	Config. 3
SECDED	✓	✓	✓
Memory scrubbing	✓	✓	✓
Rank sparing		✓	✓
Memory page retirement			✓

combinations of memory reliability managements as is presented in Table I. SECDED and memory scrubbing with the cycle of 48 hours are efficient to clear transient faults. Hence, we apply them as the baseline in Config. 1. Rank sparing, added in Config. 2, protects half of the ranks. Memory page retirement, added in Config. 3, can retire a maximum of 6.25% of pages in each rank. Fig. 9 shows the normalized 7-years probability of system failure caused by different fault types. All probabilities are normalized to the highest rate of each fault type. From Config. 1 to Config. 2, the added rank sparing only reduces the probability of multi-banks fault, while other fault types show higher failure rate. This is because the threshold (percentage of faulty addresses in a rank) to trigger rank sparing is set to too high (10%). All fault types except multi-banks fault cannot trigger it. This also demonstrates that system-level reliability enhancement techniques should be carefully evaluated and calibrated. Since other faults are less masked by multi-banks fault, the probabilities of failure caused by them increase relatively (i.e., the high order effects discussed in Section III). From Config. 2 to Config. 3, the MPR efficiently reduces failure probability except for multi-banks fault and single-lane fault, of which sizes are too big to be handled by MPR. The failure probability increase for those two big fault types in Config. 3 is also due to the high order effects.

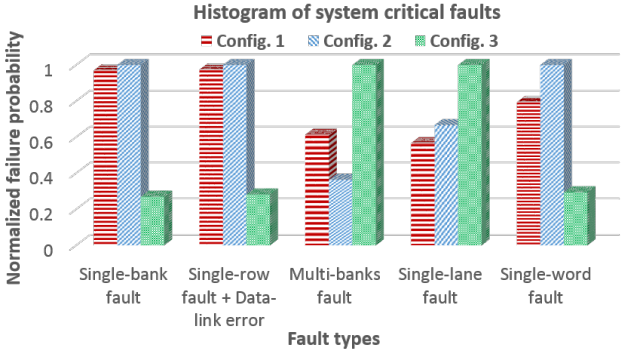


Fig. 9. The probability of critical fault types that cause system failure. The probability of each bar is normalized to the highest rate of its fault type.

V. CONCLUSION

The proposed MEMRES facilitates a fast and convenient way to assess the reliability of modern memory systems. It can perform application-based memory fault simulation with ECC and memory reliability management. The accuracy of MEMRES is validated by the comparison with the derived analytical model. The effectiveness of modern reliability enhancement techniques including ECC algorithms, ECC designs, and memory reliability can be calibrated and optimized for targeting application through MEMRES. In the future, MEMRES can assist the emerging memory system design by appropriately modeling fault behavior of non-volatile devices.

REFERENCES

[1] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 193–204.

[2] W. Zhao, Y. Zhang, T. Devolder, J.-O. Klein, D. Ravelosona, C. Chappert, and P. Mazoyer, "Failure and reliability analysis of stt-mram," *Microelectronics Reliability*, vol. 52, no. 9, pp. 1848–1852, 2012.

[3] J. Li, C. Augustine, S. Salahuddin, and K. Roy, "Modeling of failure probability and statistical design of spin-torque transfer magnetic random access memory (stt mram) array for yield enhancement," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*. IEEE, 2008, pp. 278–283.

[4] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[5] X. Jian, N. Debardeleben, S. Blanchard, V. Sridharan, and R. Kumar, "Analyzing reliability of memory sub-systems with double-chipkill detect/correct," in *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*, Dec 2013, pp. 88–97.

[6] N. Miura, K. Kasuga, M. Saito, and T. Kuroda, "Isscc 2010/session 24/dram & flash memories/24.3," 2010.

[7] N. Nguyen, Y. Frans, B. Leibowitz, S. Li, R. Navid, M. Aleksic, F. Lee, F. Quan, J. Zerbe, R. Perego *et al.*, "A 16-gb/s differential i/o cell with 380fs rj in an emulated 40nm dram process," in *VLSI Circuits, 2008 IEEE Symposium on*. IEEE, 2008, pp. 128–129.

[8] M. Blaum, R. Goodman, and R. McEliece, "The reliability of single-error protected computer memories," *Computers, IEEE Transactions on*, vol. 37, no. 1, pp. 114–119, 1988.

[9] W. Mikhail, R. Bartoldus, and R. Rutledge, "The reliability of memory with single-error correction," *IEEE Transactions on Computers*, vol. 31, no. 6, pp. 560–564, 1982.

[10] T. J. Dell, "A white paper on the benefits of chipkill-correct ecc for pc server main memory," *IBM Microelectronics Division*, pp. 1–23, 1997.

[11] ——, "A white paper on the benefits of chipkill-correct ecc for pc server main memory," *IBM Microelectronics Division*, pp. 1–23, 1997.

[12] C.-F. Wu, C.-T. Huang, and C.-W. Wu, "Ramses: a fast memory fault simulator," in *Defect and Fault Tolerance in VLSI Systems, 1999. DFT'99. International Symposium on*. IEEE, 1999, pp. 165–173.

[13] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "Proofs: A fast, memory-efficient sequential circuit fault simulator," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, no. 2, pp. 198–207, 1992.

[14] D. Roberts and P. Nair, "Faultsim: A fast, configurable memory-resilience simulator," The Memory Forum: In conjunction with ISCA-41, Tech. Rep., 2014.