FAULTSIM: A fast, configurable memory-resilience simulator

David A. Roberts* and Prashant J. Nair†
AMD Research - Advanced Micro Devices, Inc.*, Georgia Institute of Technology†
Email: david.roberts@amd.com*, pnair6@gatech.edu†

Abstract—Recent studies of DRAM failures in data centers and supercomputer environments have highlighted non-uniform failure modes in DRAM chips. Failures fall into different classes depending on the source of the failure (e.g., an I/O pin, rank, bank, row, column, or bit). These failures will be common in future memory technologies. To mitigate them, memory systems employ complex error correcting codes and fault repair mechanisms. One way to evaluate the relative potency of these mechanisms is with analytical models, which are time-consuming to derive. Therefore, we propose FaultSim, a configurable memory-reliability simulation tool for 2D and 3D-stacked memories. FaultSim uses Monte Carlo methods, real-world failure statistics and novel algorithms to accelerate evaluation of different resilience schemes. Using multi-granularity failure rates from field studies with BCH-1 (SECDED) and ChipKill codes, simulated results are within 0.41% and 1.13% of an approximate analytical model, respectively.

1 INTRODUCTION

Memory-reliability modeling has been an important part of system design for on-chip SRAM as well as off-chip memories such as DRAM and storage devices including Flash memory. With the introduction of emerging storage-class non-volatile memories (NVM) such as phase-change (PCM), spin-torque magnetic (STT-MRAM) and resistive (RRAM), new memory-reliability models will be required to design efficient resilience schemes such as error-correcting codes (ECC) and repair techniques that maximize protection for a given hardware cost. Because different memories have different failure modes, customized resilience schemes have been proposed that are based on the latest failure pattern data for each type of device. Failure data typically is obtained by manufacturers who may use accelerated testing via high temperatures or high-radiation environments. They also have access to manufacturing test data for thousands of devices. In addition, field studies [1], [2], [3], [4] have recorded memory-error locations for installations containing thousands of chips during several years of operation. These studies allow the capture of time-dependent failure rates and wear-out effects. Finally, failure models may be derived based on detailed physical models of memory cells, interconnect, and packaging. However, because some factors influencing faults are environmental, such models may fail to capture real-world behavior.

Faults fall into two basic categories: permanent and transient. Permanent faults arise from manufacturing defects or degradation over time (e.g., wear-out in NVMs). Transient faults have many possible causes, including energetic particle strikes (e.g., from cosmic rays) and power-supply noise. The faults are exhibited as errors observed by the device consuming the data. Once transient errors are corrected, the data can be written back into memory and accessed as usual. Read requests to corrected permanent faults require repeated error correction unless the memory is replaced with non-faulty cells (repair), resulting in performance and energy costs.

Given a fault model for a particular set of memory devices and technologies, we need to determine the improvement in resilience obtained by using data-encoding and repair techniques. These usually involve a combination of ECC and redundant memory cells that can be used to replace cells suffering permanent faults. For DRAM main memories, a number of spare rows are available on each device to repair permanently faulty rows identified at manufacturing test. During operation, additional ECC is added to protect against soft errors and emerging permanent faults. Examples of modern DRAM ECC includes ECC-DIMM and ChipKill. ECC-DIMM adds an additional memory chip for every eight data chips, providing the ability to correct single-bit errors within each word [5]. ChipKill adds two or more chips to each DRAM rank to allow continued operation even when an entire chip fails [6]. Calculation of the reliability of these schemes becomes complicated (although possible) when using real-world, multi-granularity fault models [7]. When the ability to perform scrubbing and on-line repair is added to the resilience scheme, it becomes even more time-consuming to derive an analytical model.

Earlier papers [7] avoid this difficulty by using statistical modeling approaches. Monte Carlo methods are a class of algorithms that use repeated random sampling to obtain the distribution of an unknown probabilistic entity[8]. In our case, we want to determine the probability of a memory device failure at a certain point in time under different resilience schemes. Prior approaches [7] use the same basic method of dividing device lifetime into equal-sized intervals and injecting faults into a memory array according to their probability of occurrence during the small interval. Error correction then is invoked periodically (at the scrubbing interval or on reads) on the simulated array to determine whether an error can be detected, corrected, and/or repaired, or is uncorrectable. By running thousands of experiments, the outcome will converge on an expected error probability. However, we are not aware of any publications detailing the data structures and algorithms

used in this compute-intensive process. In this paper, we describe novel algorithms and data structures that provide simulation performance that is higher than a naive approach by orders of magnitude. We also demonstrate the power of such a tool to explore the reliability design space rapidly.

2 Fault Models and ECC

This section describes common fault models and DRAM ECC schemes, with particular emphasis on how detection and correction abilities can be determined algorithmically.

As mentioned earlier, several methods of modeling faults are available. By default, we use field data-based failure rate models defined as failure-in-time (FIT) rates, which represent failures per billion hours. These parameters can be obtained from studies by Sridharan et al. [1], [2]. FaultSim can incorporate new models flexibly as desired.

The error-detection and -correction capabilities of a code can be defined by the code's minimum Hamming distance [9]. Given a code with Hamming distance d, then $d-1$ errors are guaranteed to be detected and $\lfloor d/2 \rfloor$ errors can be corrected. However, it is possible for a given code to detect more errors. For speed, FaultSim conservatively uses the minimum distance to determine whether an error is detectable, correctable, or uncorrectable.

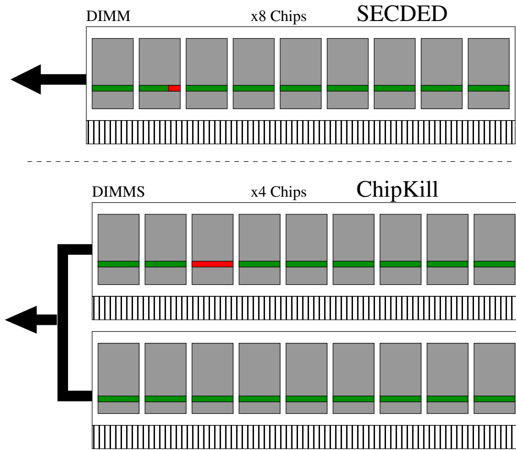

Fig. 1: ECC-DIMM and ChipKill conceptual block diagrams. The green block represents the rows being read. The red block indicates the size of a failed block of bits that can be corrected by each scheme. The arrows indicate the set of chips that are read or written in parallel.

Figure 1 illustrates two of the most common DRAM ECC schemes. ECC-DIMM adds an additional 8-bit check symbol to each 64-bit transaction on the memory bus, via a ninth 8-bit wide DRAM chip, forming a rank. Using a Hamming code capable of single-error correction, double-error detection (SECDED), any one bit in a bus cycle may be corrected. This condition can be tested by counting the number of unique faulty bits occupying the same bus cycle as other faulty bit ranges. The maximum number of faulty bits occupying the same bus cycle can be no greater than 1 for correction.

The ChipKill scheme requires two additional chips (providing two symbols) and can correct a single-symbol error in each codeword, including the case of a complete single-chip failure. The correction capability of a ChipKill system can be determined as with ECC-DIMM, except that only a single faulty symbol can exist per codeword, rather than a single bit.

3 Simulator

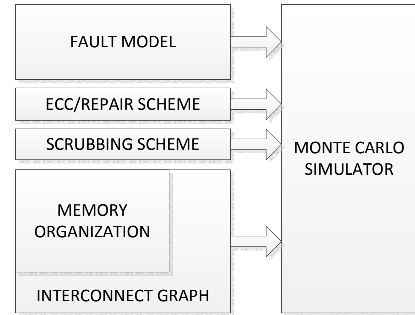

Fig. 2: FaultSim top-level block diagram

FaultSim consists of several interacting components illustrated in Figure 2. The fault model specifies the probability of inserting faults of each granularity at a particular time step. The probabilities may be constants when considering a fixed failure rate, or they may vary over time e.g. to model the wear-out of non-volatile memories. The memory organization allows the user to specify the capacity and physical organization of the memory system including chips per rank and an option to enable 3D-stacked memory. An optional graph representing the interconnect between multiple memory devices and a host processor may also be specified. An ECC / repair scheme is selected for the simulation run, along with a scrubbing interval. Finally, the core Monte Carlo simulator is started once all objects have been specified. Although the simulator is not parallelized, it is simple to execute multiple instances of the simulator and combine the results to improve accuracy. Currently, the user must specify the number of trials to achieve their desired level of accuracy.

4 Algorithms and Data Structures

The core simulation engine repeatedly performs two steps. First, every chip randomly inserts new faults according to the failure probability within one time interval. Second, any associated ECC schemes are invoked on the current list of faults to look for correctable or detectable errors. When ECC fails, the simulation is terminated, the faults are cleared, and the next simulation begins. Given enough simulations, the distribution of failure times can be generated.

4.1 Fault representation

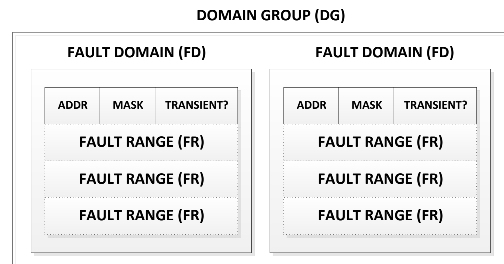

Fig. 3: FaultSim core data structures

Figure 3 shows the core data objects involved in FaultSim. The memory is represented as a collection of fault domains (FDs) and fault ranges (FRs). An FD represents a physical memory die, while an FR represents a range of the address space that exhibits a permanent or transient error. Each FD contains a list of FRs as they accumulate over time.

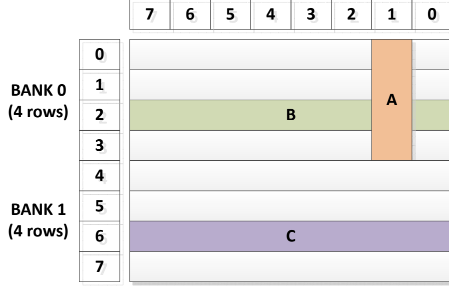

Fig. 4: Column fault example

Figure 4 illustrates how three faults are represented as FRs A, B, and C. Both memory banks consists of four rows with eight columns each. For fault A, column 1 (the second column, assuming indexes starting from 0) in bank 0 is faulty. For fault B, row 2 of bank 0 and for fault C, row 3 of bank 1 are faulty.

Each fault is stored as a single FR object. A FR contains an address (ADDR) and mask (MASK) field, each of which is 64 bits wide, along with a bit indicating transient or permanent. Bit positions in these fields represent bits in the physical device address map (in descending order from MSB) of (rank, bank, row, column, bit). Any MASK bit set to 1 indicates that all addresses (with a 0 or 1) in this bit position are faulty. The ADDR bits in a position in which MASK is 0 specify a fixed part of the fault address. In this way, MASK acts as a wildcard so that ranges of faulty bits can be represented in a compact form that can be compared with other FRs. By counting the number of faulty bits in an ECC codeword, we can determine whether the error count exceeds the capability of the detection or correction code. Restricting the analysis to the bits in the FRs greatly reduces the search space for the analysis (as contrasted with a naive method in which faults are recorded as an array of bits). When using the failure rate model of [1], all of the fault classes (bit, word, column, bank, rank) can be represented by a single FR. However, when modeling through-silicon vias (TSVs) for 3D-stacked die, we need to insert multiple FRs because the TSVs span multiple die.

As time progresses, faults accumulate in the memory FDs. These are recorded as lists of FR objects contained within each FD. The basic operation applied to the data structure is *intersect(X,Y)* (i.e., a Boolean function that determines whether two FRs (X and Y) overlap). Armed with this function, high-level models can traverse the FR structures to determine which errors can be detected or corrected.

TABLE 1: Fault Range example

FR	mask	addr
A	011000	000001
B	000111	010000
C	000111	110000

TABLE 2: Fault Range intersection example

XY	$X.mask + Y.mask$	$\overline{X.addr \oplus Y.addr}$	intersects
AB	011111	101110	1
AC	011111	001110	0
BC	000111	011111	0

The function *intersect(X,Y)* is described by the bitwise operations in Equation 1, where n is the number of bits in the address;

$$\forall_{i \in 0..n-1} : (X.mask_i + Y.mask_i) + \left(\overline{X.addr_i \oplus Y.addr_i}\right) \quad (1)$$

Tables 1 and 2 illustrate the use of this equation in determining intersection. The term involving the masks determines that FRs X and Y could intersect because at least one FR contains all possible addresses determined by that bit position. However, to be certain of an overlap, the addresses of X and Y must match precisely where MASK is zero (the second term in Equation 1), in case only a specific address bit value is faulty. If both of these conditions are met for all address bits, the FRs overlap.

4.2 3D-stacked memories

There are several proposed 3D-stacked memory architectures in existence [10], [11], [12], but they typically use through-silicon vias (TSVs) to carry power, clock, command, address and data signals between die. FaultSim was extended to model an array of data TSVs with a fixed failure probability. In the event of TSV failure during a time step, multiple bit error FRs are inserted into the address space where they are accessed by that specific TSV.

4.3 Memory Networks

By defining a graph connecting multiple memory devices and assigning failure probabilities to each inter-memory link, we can incorporate multi-memory networks (such as those supported by the 3D-stacked Hybrid Memory Cube [12]). We define a "host" node representing the consumer of the data. Using a reachability algorithm at each time step interval, we record a system failure when any memory becomes unreachable from the host due to link failure.

5 Example ECC and Repair Algorithms

By themselves, the *intersect* function and data structures cannot determine whether errors can be detected or corrected. Therefore we apply a second level of algorithms that utilize them. We illustrate the algorithms for single-error correcting BCH [13] and ChipKill [6] in the following sub-sections. These well-known schemes were chosen to validate the simulator. Note that many more ECC and repair schemes can be explored using FaultSim simply by creating a new C++ class, and we have successfully applied it to RAID-like [14] schemes in 3D-stacked memories.

As a baseline, we consider a naive simulation approach in which faulty data bit locations are recorded in an array of bits the same size as the simulated memory. To determine the number of faulty bits or symbols in a codeword (and, hence, correctability), every bit in every codeword must be scanned at each time step. The leads to a time complexity of $O(n)$,

where n is the total number of bits in the memory. In our example, the system is composed of 18 1-Gbit chips for a total of $18Gbits$ storage. We can do considerably better than the naive scanning algorithm by using the $O(1)$ *intersect* function over the set of fault ranges, which for realistic failure rates will be very few (0.036 single-bit fault per 7 year period for an 18-chip system, using data from [1]). In FaultSim, one fault range can represent an entire chip's worth of faulty bits.

The ECC algorithms count the number of faulty bits or symbols in each codeword covered by an FR. ECC algorithms terminate the simulation after recording the time of failure, if the number of faulty bits or symbols per codeword exceed the number corrected or detected by the specified ECC.

5.1 BCH Code

This description assumes a single rank of chips in an ECC-DIMM with codewords spanning the width of the data bus (typically 64+8=72 bits). For BCH codes, the algorithm (1) considers every FR in each chip as a reference (FR0) and count faulty bits in codewords covered by that same FR and FRs in any other chips (FR1) that intersect with codewords covered by the first FR. Bit counting is achieved by manipulating the *mask* and *addr* bits of a temporary FR to test each bit position on the bus individually. Because the bit intersection count ($n_intersections$) is reset for each reference FR0, intersecting faults are not double-counted within the nested loops.

Algorithm 1: BCH ECC algorithm for a single Monte Carlo run. N is the number of chips in the rank

```
for FR0 in FR[0..N-1] do
    FR_temp = FR0
    n_intersections = 0
    CLEAR lower 2 bits of FR_temp.addr
    CLEAR lower 2 bits of FR_temp.mask
    for bit_addr in (0..3) do
        for FR1 in FR[0..N-1] do
            if FR_temp intersects FR1 then
                n_intersections++
            end
        end
        FR_temp.addr++
    end
    if n_intersections > correctable errors then
        terminate simulation;
    end
end
```

If we are interested in error-detection capability, the simulation is not terminated on the first uncorrectable error; instead, it continues until the first undetectable error.

5.2 ChipKill

For ChipKill, we perform a similar algorithm (2) to BCH except that instead of counting individual bits, we count 8-bit symbols in overlapping address ranges between chips. Symbol-level comparison is easily achieved by setting the least significant mask bits to span the symbol size.

Algorithm 2: ChipKill ECC algorithm for a single Monte Carlo run. N is the number of chips in a rank

```
for FR0 in FR[0..N-1] do
    FR_temp = FR0
    n_intersections = 0
    SET lower 3 bits of FR_temp.mask
    for FR1 in FR[0..N-1] do
        if FR_temp intersects FR1 then
            n_intersections++
        end
    end
    if n_intersections > correctable errors then
        terminate simulation;
    end
end
```

5.3 Scrubbing and Repair

FaultSim simulates the practice of periodic scrubbing that is prevalent in server systems. During the scrubbing cycle, all correctable transient faults in memory can be repaired by removing the appropriate FRs from the list in the FD. By allocating spare capacity, FaultSim also can model repair schemes that use remapping or spare resources [15]. In such repair schemes, during the scrubbing interval, all correctable data errors are remapped into a spare region using an address remap table. This table must be consulted by the ECC-evaluation algorithm before the data structure is addressed.

6 Results

In this section, we derive an approximate analytical model for memory system failure probability under ECC-DIMM and ChipKill. We compare this with the statistical results from FaultSim to determine its accuracy.

6.1 Analysis

For ECC-DIMM and ChipKill, it is possible to derive approximate analytical failure models. The use of different probabilities for various fault granularities from field study data complicates the analysis, which will only become more complex as more is learned about failure modes from detailed field studies.

BCH-1 code can correct single bit errors and detect double bit errors (SECDED). Since BCH1 codes can tolerate all isolated 1-bit errors and the memories have large capacities relative to the failure rates, there is a low probability of single-bit faults accumulating in the same codeword. Therefore, we will ignore the impact of single bit faults. Any multi-bit fault results in uncorrectable errors. Let the probability of failure of a chip due to multi-bit faults be$P_{Fail-MultiBit}$. The probability that the n-chip system experiences a multi-bit fault in a system with BCH1 code ($P_{SysFail-BCH1}$) is (2).

$$P_{SysFail-BCH1} \approxeq 1-\binom{n}{0} P^{0}_{Fail-MultiBit} \times (1-P_{Fail-MultiBit})^{n} \quad (2)$$

ChipKill ECC uses 8-bit symbols in our case, and can tolerate any size of fault in any one chip. These systems can also tolerate bank, column, row and bit faults in different codewords across multiple chips. Occasionally, faulty symbols can overlap in the same address region in other chips, resulting in more than one bad symbol per codeword. This implies that the larger classes of faults such as rank, multi-bank and single bank faults determine ChipKill resilience to a first order, since they have relatively high failure probability and are likely to coincide with other faults. For an n-chip system, let the probability of a chip failure due to any fault type be $P_{Fail-Any}$, and multi-bank or rank faults be represented as $P_{Fail-Multi}$. Let the probability of chip failure due to a single bank fault in a chip be $P_{Fail-OneBank}$. The total probability of chip failure due to non-multibank faults (i.e. bit, word, column, row or bank) is represented by $P_{Fail-NonMulti}$. The probability of precisely one chip having a multi-bank or rank fault is given in Equation (3).

$$P_{OneMultiBank} = \binom{n}{1} P^1_{Fail-Multi} \times (1 - P_{Fail-Multi})^{n-1} \quad (3)$$

The probability of system failure when one multi-bank or rank fault ($P_{OneMultiBank}$) occurs, simultaneous with any other fault on one or more other chips, is represented by (4).

$$P_{SysFail(1)} = P_{OneMultiBank} \times \left[1 - \binom{n-1}{0} P^0_{Fail-Any} \times (1 - P_{Fail-Any})^{n-1}\right] \quad (4)$$

The probability of system failure when one single-bank fault ($P_{OneChipBank}$) occurs in one chip is given in Equation (5).

$$P_{OneChipBank} = \binom{n}{1} P^1_{Fail-OneBank} \times (1 - P_{Fail-OneBank})^{n-1} \quad (5)$$

The probability of system failure when one single-bank fault occurs, simultaneous with one or more faults contained within a single bank on another chip ($P_{Fail-NonMulti}$), is represented by (6). Note that since there are 8 banks in the chip, we divide by 8 to account for the fact that only bank-aligned faults across chips cause uncorrectable errors, and there are 8 banks per chip.

$$P_{SysFail(2)} = (1/8) \times P_{OneChipBank} \times \left[1 - \binom{n-1}{0} P^0_{Fail-NonMulti} \times (1 - P_{Fail-NonMulti})^{n-1}\right] \quad (6)$$

We have now accounted for those non-multibank faults which align with the first bank fault with probability 1/8. We must now add the probability of a bank fault intersecting with one or more multi-bank or rank faults which always intersect with the bank fault's codewords (Equation (7)).

$$P_{SysFail(3)} = P_{OneChipBank} \times \left[1 - \binom{n-1}{0} P^0_{Fail-Multi} \times (1 - P_{Fail-Multi})^{n-1}\right] \quad (7)$$

The probability that the system fails for ($P_{SysFail-Chipkill}$) due to the fault combinations from equations (4), (6) and (7) can be represented by (8).

$$P_{SysFail-Chipkill} \approxeq P_{SysFail(1)} + P_{SysFail(2)} + P_{SysFail(3)} \quad (8)$$

We use the values from Table 3 [1] to obtain the various chip failure probabilities.

6.2 Validation

We compare error correction capability of ECC-DIMM (BCH1) and ChipKill DIMM (SSCDSD) over a 7 year lifetime. For both types of ECC we use the same 18-chip 1-rank system for consistency (bus width of 4 bits per chip). The experiment is performed on AMD Opteron 8431 operating at 2.4 GHz. FaultSim ran 'ChipKill with Scrubbing' at 2.3 seven-year simulations per second and 'BCH1 with scrubbing' 2.6 seven-year simulations per second. The accelerated speed of Monte Carlo simulation is attributed to the data structures and organization of FaultSim. To maintain accuracy, we ran these simulations with a 3 hour time step and a total of 80,000 simulations for BCH1 and 240,000 for ChipKill.

TABLE 3: DRAM Failure Rates in failures per billion device hours (FIT) [1]

DRAM Chip Failure Mode	Fault Rate (FIT)	
	Transient	Permanent
Single bit	14.2	18.6
Single word	1.4	0.3
Single column	1.4	5.6
Single row	0.2	8.2
Single bank	0.8	10
Multi-bank	0.3	1.4
Multi-rank	0.9	2.8

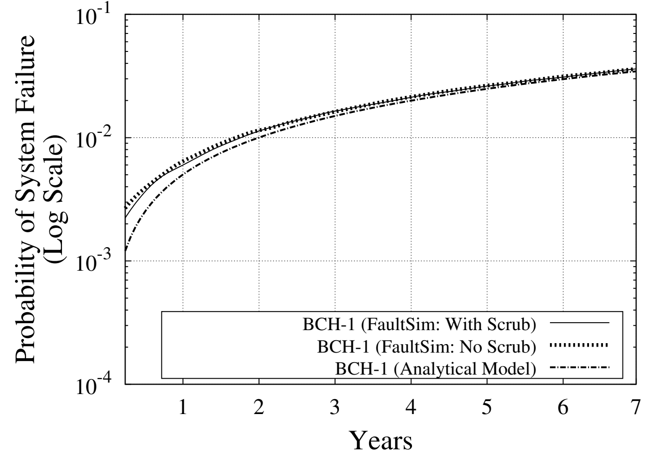

Fig. 5: Failure probability as a function of elapsed time for BCH-1 Code

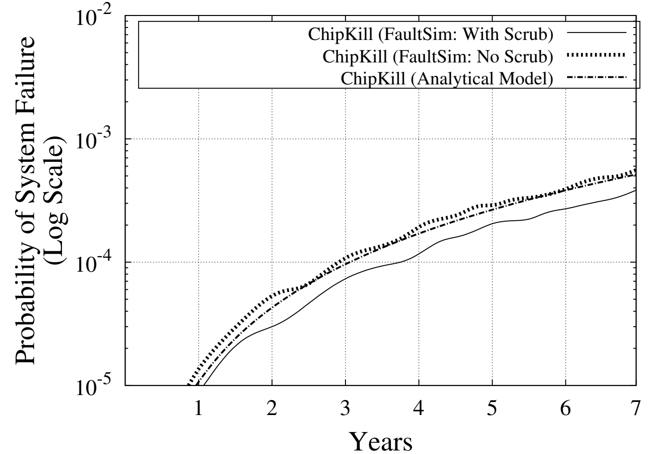

Fig. 6: Failure probability as a function of elapsed time for ChipKill

Figures 5 and 6 show that BCH-1 codes are more susceptible to failures versus Chipkill as they cannot tolerate

large granularity chip failures. The simulated results show $P_{SysFail-BCH1}$ and $P_{SysFail-ChipKill}$ over the 7 year period. After 7 years (without scrubbing) simulated probabilities are 0.035975 and 0.0005583 respectively. The analytical model for BCH-1 and ChipKill has a difference of only 0.41% and 1.13% versus the simulation results.

7 Conclusions and Future Work

In this paper, we presented a simulator and fast algorithms for memory-resilience analysis. As emerging memories come to market and knowledge of failures is gained through field studies, the need for rapid evaluation of evolving failure rates and resilience schemes grows significantly. We hope to continue development of this tool via parallelization and implementation of recently published fault models [2].

References

[1] 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*, pp. 1–11, 2012.

[2] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: Positional effects in dram and sram faults," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pp. 22:1–22:11, 2013.

[3] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: a large-scale field study," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 193–204, June 2009.

[4] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, 2010.

[5] Silicon Power, *DDR3 ECC Unbuffered DIMM Spec Sheet*, 2010.

[6] T. J. Dell, "A white paper on the benefits of chipkillcorrect ecc for pc server main memory," Tech. Rep. 11/19/97, IBM, 1997.

[7] X. Jian, N. Debardeleben, S. Blanchard, V. Sridharan, and R. Kumar, "Analyzing reliability of memory subsystems with double chipkill detect/correct," in *The 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2013.

[8] S. J. Kamat and M. W. Riley, "Determination of reliability using event-based monte carlo simulation," *Reliability, IEEE Transactions on*, vol. R-24, pp. 73–75, April 1975.

[9] R. W. HAMMING, "Error detecting and error correcting codes," *BELL SYSTEM TECHNICAL JOURNAL*, vol. 29, no. 2, pp. 147–160, 1950.

[10] J. Standard, "Wide-io dram," in *JESD229*, 2013.

[11] J. Standard, "High bandwidth memory (hbm) dram," in *JESD235*, 2013.

[12] H. M. C. Consortium, "Hybrid memory cube specification 1.0," 2013.

[13] C. Chen and M. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review," *IBM Journal*, vol. 28, pp. 124–134, March 1984.

[14] A. Thomasian and J. Menon, "Raid5 performance with distributed sparing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 6, pp. 640–657, 1997.

[15] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: architectural framework for assisting dram scaling by tolerating high error rates," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 72–83, ACM, 2013.