FaultSim: A Fast, Configurable Memory-Reliability Simulator for Conventional and 3D-Stacked Systems

PRASHANT J. NAIR, School of Electrical and Computer Engineering, Georgia Institute of Technology
DAVID A. ROBERTS, AMD Research, Advanced Micro Devices Inc.
MOINUDDIN K. QURESHI, School of Electrical and Computer Engineering, Georgia Institute of Technology

As memory systems scale, maintaining their Reliability Availability and Serviceability (RAS) is becoming more complex. To make matters worse, recent studies of DRAM failures in data centers and supercomputer environments have highlighted that large-granularity failures are common in DRAM chips. Furthermore, the move toward 3D-stacked memories can make the system vulnerable to newer failure modes, such as those occurring from faults in Through-Silicon Vias (TSVs). To architect future systems and to use emerging technology, system designers will need to employ strong error correction and repair techniques. Unfortunately, evaluating the relative effectiveness of these reliability mechanisms is often difficult and is traditionally done with analytical models, which are both error prone and time-consuming to develop. To this end, this article proposes FaultSim, a fast configurable memory-reliability simulation tool for 2D and 3D-stacked memory systems. FaultSim employs Monte Carlo simulations, which are driven by real-world failure statistics. We discuss the novel algorithms and data structures used in FaultSim to accelerate the evaluation of different resilience schemes. We implement BCH-1 (SECDED) and ChipKill codes using FaultSim and validate against an analytical model. FaultSim implements BCH-1 and ChipKill codes with a deviation of only 0.032% and 8.41% from the analytical model. FaultSim can simulate 1 million Monte Carlo trials (each for a period of 7 years) of BCH-1 and ChipKill codes in only 34 seconds and 33 seconds, respectively.

CCS Concepts: • **Computing methodologies → Discrete-event simulation; Simulation tools;** • **Computer systems organization → Reliability;** *Processors and memory architectures*; • **Hardware → Semiconductor memory;** 3D integrated circuits;

Additional Key Words and Phrases: Error correcting codes, monte carlo simulation, stacked memory, reliability, through silicon vias

ACM Reference Format:
Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. 2015. FaultSim: A fast, configurable memory-reliability simulator for conventional and 3D-stacked systems. ACM Trans. Archit. Code Optim. 12, 4, Article 44 (December 2015), 24 pages.
DOI: http://dx.doi.org/10.1145/2831234

1. INTRODUCTION

Fast and accurate simulation tools are vital for computer architects to analyze a problem and to compare the effectiveness of different solutions. Such tools become even more critical when the community is trying to address a new set of constraints in

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.
Authors' addresses: P. J. Nair and M. K. Qureshi, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332; emails: pnair6@gatech.edu, moin@ece.gatech.edu; D. A. Roberts, 1 AMD Place, Sunnyvale, California 94088; email: David.Roberts@amd.com.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2015 ACM 1544-3566/2015/12-ART44 $15.00
DOI: http://dx.doi.org/10.1145/2831234

designing the system or trying to incorporate an emerging technology that has widely different properties than traditional technologies. For example, *Wattch* [Brooks et al. 2000] was instrumental in driving power-related architecture research when power became the primary constraint in system design. Similarly, *CACTI* [Muralimanohar et al. 2007] equipped architects to study the timing and area of different cache organizations without relying on circuit designers. As the era of multicore systems dawned, *McPAT* [Li et al. 2009] enabled an integrated power, area, and timing modeling framework for multicore and many-core architectures. The multicore era also saw the advent of many core performance simulators such as *Graphite* and *Sniper* [Miller et al. 2010; Carlson et al. 2011]. In this article, we assert that memory reliability is becoming a growing concern, and that there is a pressing need for a toolset that can compare the effectiveness of different memory reliability solutions quickly and accurately.

Memory reliability has always been a major concern for HPC systems as the large number of memory components in such systems can lead to frequent errors [Bergman et al. 2008]. With technology scaling, the error rates of memory modules are likely to increase significantly [Nair et al. 2013; Son et al. 2015], which makes maintaining reliability a concern even for small- and medium-scale systems [Li et al. 2011; Kim et al. 2007; Thomasian and Menon 1997; Chung 2013]. Furthermore, emerging devices such as 3D-stacked memories present new points of failure such as Through-Silicon Vias (TSVs) [Jiang et al. 2012]. Reliability estimation is also essential to investigate applications for new memory technologies such as PCM, ReRAM, and STT-MRAM [Qureshi 2011]. To aid system reliability modeling, several studies have collected failure data for large systems [Schroeder et al. 2009; Schroeder and Gibson 2010]. They found that individual DRAM chips exhibit both large-granularity failures (such as bank failures) and bit failures at different failure rates [Sridharan and Liberty 2012; Sridharan et al. 2013, 2015]. A system designer may provision the memories with one of several mitigation techniques such as error correction codes, spare memories, RAID, Chipkill, scrubbing, and so forth. Given that the interaction between fault models and the mitigation techniques is quite complex, it is not straightforward to estimate the effectiveness of any ECC technique even when they are applied alone.

Designers can estimate memory system reliability by applying analytical models [Jian et al. 2013]. Unfortunately, developing an analytical model is quite time-consuming and relies on several simplifying assumptions to make the model tractable. Furthermore, analytical models are often prone to errors, and any changes in the memory system may require a new model. To make matters worse, modern memory systems may perform periodic memory scrubbing, which becomes quite complex to incorporate into an analytical model [Awasthi et al. 2012]. Therefore, the realm of analytical models has focused on only a few schemes such as SECDED or Chipkill [Jian et al. 2013]. Extending these models to incorporate newer failure modes (such as those arising from 3D memories) or to an arbitrary mitigation technique is quite challenging and cumbersome, if not impractical.

An alternative method to assess reliability of the system is via Monte Carlo simulation [Kamat and Riley 1975]. In Monte Carlo simulations, to determine the probability of a memory device failure at a certain point in time, the device lifetime is divided into equal-sized intervals and faults are injected with a precomputed probability. Error correction is then invoked periodically (at the scrubbing interval or on reads) on the simulated system to determine whether an error can be detected, corrected, or repaired using the underlying ECC scheme [Silicon Power 2010; Chen and Hsiao 1984; Chien 1964; Fujiwara and Pradhan 1989]. By performing a large number of trials, the outcome can be expected to converge to the system failure probability. Running repeated trials for the entire system lifetime tends to be a compute-intensive process, spanning several hours or days. A recent work, MEMRES, uses motivations from a prior workshop version of this article to model application and link effects. However, we are not

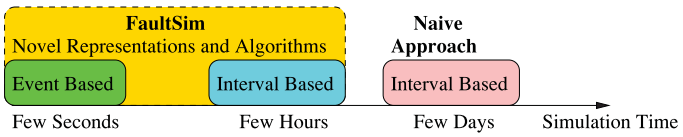

Fig. 1. FaultSim uses novel data representations and algorithms that provide very high simulation speed and accuracy.

aware of any existing publications detailing the data structures and algorithms that help in mitigating the simulation time of this process to a few seconds [Wang et al. 2015; Roberts and Nair 2014]. The large simulation times of a Monte Carlo simulator prevents quick design space exploration. To this end, this article proposes Fault Simulator (FAULTSIM), a fast Monte Carlo simulator that uses novel fault representations and reliability estimation techniques to reduce the simulation time by three to four orders of magnitude.

The key parameters in memory reliability simulation are the fault rates and fault granularity for every memory device. Representing and tracking these multigranularity faults in a simulator is quite challenging. FaultSim uses space-efficient representations for tracking faults at different granularities and quickly evaluating the interaction between such faults. These representations also enable new algorithms for failure detection. These optimizations reduce the simulation time of FaultSim to a few hours, as shown in Figure 1.

To reduce simulation time even further, we employ the observation that real-world devices show modest fault rates. At these fault rates, only a few faults occur in the system lifetime. We leverage this insight and propose a novel *Event-Based* fault injection framework. Rather than the traditional method of computing faults at every time interval (which we call *Interval-Based* simulation) and advancing the time by a constant value throughout the system lifetime, our Event-Based approach determines the time between faults and advances time from one fault to the next. This enables almost 5,000× lower simulation time compared to the interval-based approach.

We discuss the number of Monte Carlo trials required to meet the desired precision bounds. We validated FaultSim with analytical models and found the accuracy of BCH-1 and Chipkill codes to be within 0.032% and 8.41% of the analytical model. We discuss how FaultSim can be used to evaluate different schemes such as ECC, Chipkill, Sparing, and TSV sparing (for 3D memories) while incorporating memory scrubbing. This article demonstrates the ability of FaultSim to explore the reliability design space rapidly and faithfully. Furthermore, for the benefit of the community, FaultSim is open sourced and is available for download at the developers' websites.

2. BACKGROUND AND MOTIVATION

2.1. Need for Memory Resilience Studies

Memory reliability has always been a concern for large-scale HPC systems and is identified as one of the key challenges in the design of Exascale supercomputers [Bergman et al. 2008]. Similarly, high-availability servers guard against memory failures by designing memory systems that can support Chipkill or a RAID-like architecture that can tolerate memory channel failure [Emery 2013]. With technology scaling, memory cells become inherently less reliable, which means future memory systems will need to pay even more attention to memory reliability [Nair et al. 2013; Son et al. 2015]. Furthermore, to mitigate bandwidth challenges, DRAM memories are moving toward 3D die-stacking technology, which uses TSVs to enable high bandwidth. Unfortunately, such memories are susceptible to TSV failures, which can cause large-granularity failures, ranging from a column failure to a bank/rank failure. Therefore, mitigating

Table I. DRAM Failures per Billion Device Hours (FIT)
[Sridharan and Liberty 2012]

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
Multibank	0.3	1.4
Multirank	0.9	2.8

reliability challenges of future memory systems is becoming an important area of research in the architecture community.

2.2. Design-Time Faults Versus Runtime Faults

Broadly, as memory systems scale to lower nodes, two major classes of faults are seen to occur in DRAM systems. The first class of faults is called design-time faults. These faults are manifested during the manufacture time of DRAM and primarily occur due to technology scaling. Consequently, design-time faults tend to be denoted with the metric Bit-Error-Rate (BER) and can be assumed to remain constant with time. For instance, two state-of-the-art article, *CiDRA* and *ArchShield,* model this phenomenon by preselecting bits from the memory chip [Son et al. 2015; Nair et al. 2013].

In contrast, the second class of faults, called runtime faults, occur while the DRAM device remains in the field and gets accrued over time. Runtime faults have both a space (Bank, Rank, etc.) and a time component. For instance, a DRAM DIMM system in the field may start developing faults of different granularities after months or years, denoted by the metric Failures-in-Time (FIT) rate [Sridharan and Liberty 2012; Sridharan et al. 2013, 2015].

The effects of runtime faults cannot be captured by design-time fault simulators. For instance, a word can develop two transient bit faults in its lifetime. In such cases, the first fault may occur in the second year and the second fault may occur in the third year. Design-time fault simulators will detect these as two bit faults, and if the ECC cannot handle two bit faults, it will lead to a system failure. In contrast, a runtime fault simulator such as FaultSim models scrubbing, and these faults get scrubbed and removed. Compared to design-time fault simulations, this article aims to provide fundamental principles for efficient runtime multigranularity fault simulations. The techniques proposed in this article are thus orthogonal to design-time fault simulations and can be used alongside these simulators.

2.3. Common ECC Schemes Used for Mitigating Runtime Faults

High-reliability memories are often designed with ECC DIMMs that support Single Error Correction and Double Error Detection (SECDED) codes, which we refer to as *BCH-1* codes. While these modules are capable of correcting one error at any point in time, they are unable to handle large-granularity failures, such as a chip failure or a row failure. One may think that the rate of large-granularity failure is negligibly small; however, recent field studies [Sridharan and Liberty 2012; Sridharan et al. 2013, 2015; Schroeder et al. 2009; Schroeder and Gibson 2010] show that large-granularity failures are almost as common as bit failures in DRAM memories. Table I depicts the various chip failure probabilities from one of the recent field studies [Sridharan and Liberty 2012].

While single-bit failures can be tolerated with ECC DIMMs, tolerating large-granularity failures often requires more complex schemes such as Chipkill, sparing, and RAID. As these schemes typically incur significant area and power overheads, a system designer would want to know the effectiveness of different schemes at reducing the system failure rate in order to choose the most efficient schemes that can deliver the system reliability requirements. Unfortunately, evaluating the efficacy of different reliability schemes is an arduous task, and there are no publicly available evaluation tools that can perform such evaluations quickly and accurately.

2.4. Analytical Models for Runtime Faults: Uses and Shortcomings

For a relatively simple system such as that with BCH-1 code, we may be able to develop a straightforward analytical model for time-based faults and estimate the reduction in the probability of system failure due to BCH-1. Unfortunately, scaling such analytical models to more complex schemes such as Chipkill, sparing, and RAID tends to be an arduous task. To make matters worse, these mitigation schemes often appear in combination with other mitigation schemes (such as scrubbing). Furthermore, the effectiveness of these solutions also depends on different memory parameters such as row size, bank size, and number of DIMMs in the systems. Incorporating a large number of system parameters in an analytical model becomes impractical, so often there are several simplifying assumptions made to keep the model tractable. Even with such simplifications, the models tend to be quite complex. For example, the analytical model for Chipkill proposed by Jian et al. [2013] spans several pages. The main challenge with analytical models is not only that they tend to be quite time-consuming to develop but also that every new scheme or combination of multiple existing schemes requires development of a new model, with even more simplifications and potential for inaccuracies. Ideally, we want an evaluation framework that can easily incorporate the details of different system components and can evaluate a wide array of mitigation techniques accurately and quickly. One approach to do this is to use Monte Carlo simulations.

2.5. Monte Carlo Simulation for Runtime Faults: Overview and Challenges

Monte Carlo simulation is a stochastic technique that uses random sampling to evaluate the failure probability of the system. The various components of the system (such as rows, banks, and channels) can be modeled and faults can be injected in this system depending on the failure rate (FIT) of each component. Each trial can be run for a desired system lifetime (say, 7 years) or until the first uncorrectable or undetectable error is encountered. These trials can be repeated a large number of times to get an accurate estimate of the system failure probability. Such an evaluation framework is general enough to support any arbitrary error mitigation policies and system components. Compared to Monte Carlo simulations for design-time faults, Monte Carlo simulations for runtime faults have the following three key differences:

—Runtime faults occur at different rates in time and at multiple granularities. The Monte Carlo simulator must iterate over time considering all fault granularities and rates.
—Time-based events such as scrubbing cycles occur periodically. The Monte Carlo simulator must operate at a time granularity that captures these events in detail.
—At higher fault rates, multigranularity faults will require storing large fault information at multiple instances of time. Thus, these time-dependent studies must also have space-efficient representation of faults.

Subsequently, during a Monte Carlo simulation, the underlying ECC scheme in the memory system is consulted whenever faults are inserted. If the errors are

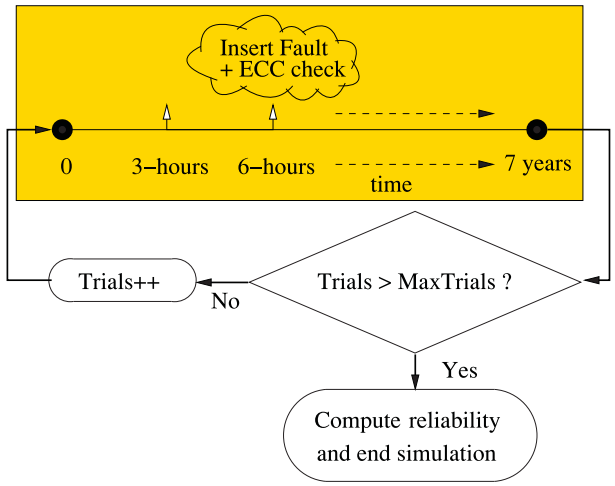

Fig. 2. Operation of FaultSim. FaultSim divides time into intervals at which faults get injected and ECC checks are performed.

uncorrectable or undetectable, the simulation is terminated and the error is reported.[1] To reduce simulation time, it is important for the Monte Carlo simulator to quickly determine if the fault(s) can be corrected or detected by ECC. While Monte Carlo simulators have been used in the past for validating analytical models [Jian et al. 2013], we are not aware of any prior work detailing the data structures and techniques to reduce the simulation time of time-dependent memory fault evaluations.

The goal of this article is to outline a set of principles and to describe a tool that can perform evaluations of complex error mitigation schemes for both 2D and 3D memory systems with negligible space overheads and with simulation time that is several orders of magnitude faster than a traditional Monte Carlo-based simulation. To this end, we present *FaultSim*, a fast and accurate memory-data resilience simulator.

3. AN OVERVIEW OF FAULTSIM

Figure 2 shows the basic FaultSim operation. The system lifetime (say, 7 years) is divided into smaller time intervals (say, 3 hours) during which faults are inserted and ECC checked. This process is repeated over a large number of trials before the simulation ends.

The time intervals are kept relatively small to increase accuracy. A smaller interval also allows simulating memory scrubbing. The timing model of FaultSim takes the memory organization, fault model, ECC, and scrubbing scheme options as parameters. These blocks provide the core functionalities for a reliability simulator and they form the FaultSim core, as shown in Figure 3. The FaultSim core determines how the faults interact with the ECC schemes for a memory organization. We describe each of these components briefly.

The memory organization and interconnection graph block allow the user to specify the parameters of the memory system including chips per rank and an option to enable 3D-stacked memory. For complex memory systems, FaultSim has a graph option to

[1]We do not distinguish between faults and errors in this article. We terminate the simulation as soon as an uncorrectable or undetectable fault is encountered, and the assumption is that this fault would have eventually led to an error during the remaining lifetime of the system. If the application is tolerant to data errors, then the exact timing of a fault being converted into an error can be delayed.

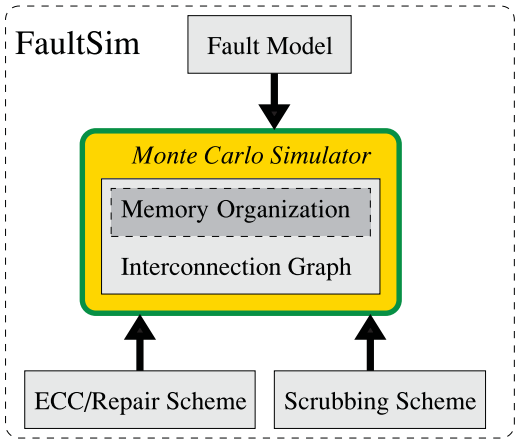

Fig. 3. Top-level block of FaultSim has interactions between the memory system, fault model, ECC, and scrubbing schemes.

represent the interconnect between multiple memory devices and a host processor. During initialization, FaultSim loads the configuration parameters and constructs "Fault Domains" that represent dies and "Group Domains" that represent channels. Multiple Group Domains can be instantiated and interconnected as a memory system.

The fault model can be selected from a number of alternatives and specifies the statistical distributions governing when the various fault granularities occur in the memory devices. Since devices are composed of banks, rows, columns, words, and bits, field studies have pointed out that device failures occur at various granularities. Further, these device failures can be transient or permanent. For instance, a faulty chip may have a transient word fault alongside a permanent bank fault. We use "failures per billion hours" (FIT rates) from field studies to generate fault distributions for known failures. FaultSim also has an option to consider a uniform FIT rate for all granularities, making it oblivious of field studies. The fault probabilities may be constants when considering a fixed failure rate, or they may vary over time, for example, to model the wear-out of nonvolatile memories [McCool 2012]. The FIT rates for Fault Domains are specified during initialization.

An ECC and/or repair scheme is selected for each Fault Domain, along with a scrubbing interval. It can also support complex parity-based ECC such as RAID and Three-Dimensional Parity [Nair et al. 2014; Thomasian and Menon 1997]. The scrubbing operation removes all correctable transient faults at each scrubbing interval. To avoid repeated permanent errors, repair schemes such as memory page decommissioning can be modeled. Such schemes are invoked when errors are corrected or during memory scrubbing. The core Monte Carlo simulator calls the ECC and repair schemes when any Fault Domain develops a fault.

The simulation of FaultSim consists of three phases. The first phase is the initialization phase, which sets values for failure rates and also describes the memory system organization. The second phase is the instantiation phase, which creates the Fault Domains and Group Domains. During the third phase, the timing-interval-based Monte Carlo simulator operates on the memory system.

FaultSim is written in C++ and simulation parameters can be input at the command line or using a config file. The components of FaultSim can be classified broadly into four categories: first, the fault injection framework that determines when a particular faulty type is injected in the system; second, the data structures and algorithms used

to represent faults and determine correctability or detectability based on the memory system organization; third, modeling of different mitigation techniques; and fourth, techniques aimed to explicitly reduce the simulation time (such as Event-Based fault injection and limiting the number of Monte Carlo trials for the desired precision). The following sections describe each of these four topics in detail.

4. FAULT INJECTION: INTERVAL BASED

The interval-based simulation divides the lifetime of the system into equal-sized intervals for every Monte Carlo trial. At the end of each time interval, random number generators are consulted to decide the insertion of faults. For accuracy, the time interval is very small compared to the lifetime of the system. A large number of intervals ensures that the random number generators are consulted a sufficient number of times during one trial. This process is then repeated for a very large number of trials and the results are averaged out.

FaultSim uses failure in billion hours (FIT rates) to denote the fault rates of its devices. These FIT rates are used as parameters to a simulated exponential distribution in time (t) [Zacks 1992]. A device may have multiple types of faults with different FIT rates (λ). Due to this, the random number generator is consulted for every fault type at the end of every interval. For any fault type, the probability density function $p_{pdf}(t)$ for an exponential distribution is described by Equation (1):[2]

$$p_{pdf}(t) = \lambda \cdot e^{-\lambda \cdot t}. \tag{1}$$

The probability of inserting a fault at the end of time interval T is a cumulative distribution ($P_{fault}(T)$) of Equation (1) and is described by Equation (2):

$$P_{fault}(T) = \int_0^T p_{pdf}(t)dt = \int_0^T \lambda \cdot e^{-\lambda \cdot t}dt. \tag{2}$$

Since $\int e^{-\lambda \cdot t}dt = \frac{e^{-\lambda \cdot t}}{-\lambda}$, Equation (2) becomes Equation (3):

$$P_{fault}(T) = 1 - e^{-\lambda \cdot T}. \tag{3}$$

$P_{fault}(T)$ is computed for every type of fault during the initialization of FaultSim. At the end of time interval T, a uniform random generator *rand(U(0,1))* is consulted and the random number is compared with $P_{fault}(T)$. The *decision* to insert faults is made using Equation (4):

$$decision = \begin{cases} \text{Insert Fault}, & \textbf{if } rand(U(0,1)) \leq P_{fault}(T) \\ \text{No Fault}, & \text{otherwise.} \end{cases} \tag{4}$$

FaultSim operates on a large number of time intervals to provide accuracy. For instance, consider a memory system operating for 7 years. For reasonable accuracy, faults are inserted at every 3-hour interval.[3] This translates to approximately 20,000 intervals in 7 years for every trial.

[2]Wear-out failures follow a Weibull distribution [McCool 2012]. A Weibull distribution is characterized by a change in λ over time. Rather than modeling this complex behavior, we can approximate the Weibull distribution as piecewise exponential distribution.

[3]In the limiting case, if the faults were inserted every 12 hours (43,200 seconds), then there is only a 63.2% (i.e., $[1 - \frac{1}{43200})^{\frac{43200}{4}}]$) confidence that the fault will be inserted before scrubbing. However, if faults are inserted at 3 hours and scrubbed every 12 hours, then there is a 98.2% (i.e., $[1 - \frac{1}{43200})^{\frac{43200}{4}}]$) confidence that faults are inserted before scrubbing.

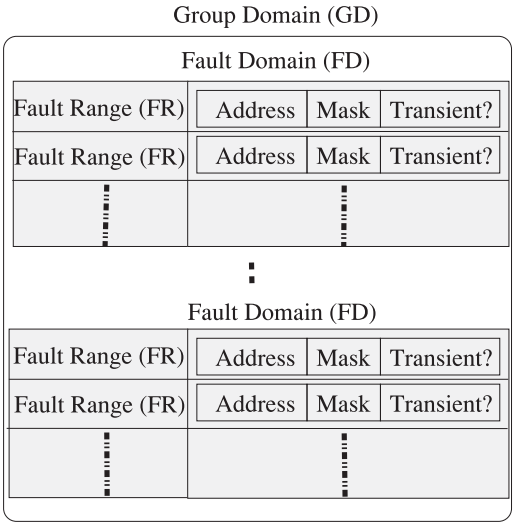

Fig. 4. A memory block is composed of Fault Domains (physical memory die) and Fault Ranges (faults per die).

5. ALGORITHMS AND DATA STRUCTURES FOR EFFICIENT TRACKING OF MULTIGRANULARITY FAULTS

The simulation core engine inserts new faults into chips according to their failure probabilities. Even at low FIT rates (such as those shown in Table I), the effect of fault representations can aggravate the complexity of computation. For instance, a faulty chip, if represented by billions of faulty bits, will increase the complexity of computation. Furthermore, at high FIT rates (say, >100,000), used for evaluating areas such as approximate computing with faulty memories, row hammering, coding theory-assisted read optimization, and optimizing DRAM refresh operations [Kim et al. 2014, 2015; Nair et al. 2015; Qureshi et al. 2015; Chou et al. 2015], an efficient fault representation can help in data management. Thus, faults must be represented efficiently to reduce space and time to evaluate ECC schemes. The faults are compared across chips to check for those which affect the same codewords (fault intersection) during the ECC check.

5.1. Fault Representation

The FaultSim memory representation is shown in Figure 4. 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 fault. Each FD contains a list of FRs that accrue over time.

To demonstrate fault representation inside an FD, consider the example of a hypothetical memory die that contains 64 bits in two banks (bank 0 and bank 1). Let us assume that such a hypothetical die develops three faults, a column fault A and row fault B in bank 0 and a row fault C in bank 1. Figure 5 shows how column (fault A) and row faults (faults B and C) are represented.

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. In this example, an 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 (bank, row, column,

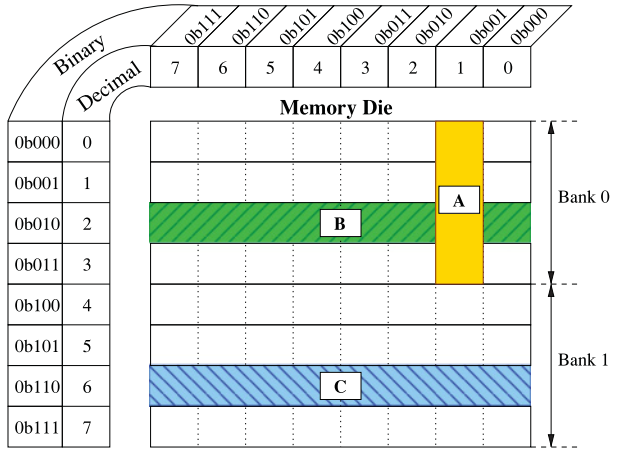

Fig. 5. A memory die (Fault Domain) can contain column (A) and row (B and C) faults in different memory banks.

Table II. Fault Range with Its Address and Mask

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

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 or symbols in an ECC codeword, we can conservatively determine whether the error count exceeds the capability of the detection or correction code, determined by the minimum Hamming distance. 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 that must be scanned). Due to this, all fault granularities (bit, word, column, bank, rank) are represented by a single FR for any failure rate model.

5.2. Fault Intersection

Faults accumulate in the memory FDs as FRs. The basic operation *intersect(X,Y)* determines if the FRs share any of their addresses within a chip. FaultSim traverses the FDs and checks the number of errors to be corrected or detected depending on the correction and detection capability of ECC. Equation (5) describes the function *intersect(X,Y)* as bitwise operations, where n is the number of bits in the address:

$$\forall_{i \in 0..n-1} : (X.mask_i + Y.mask_i) + \left(\overline{X.addr_i \oplus Y.addr_i}\right). \tag{5}$$

Tables II and III illustrate the use of Equation (5) 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, in case only a specific address bit value is faulty.

Table III. 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

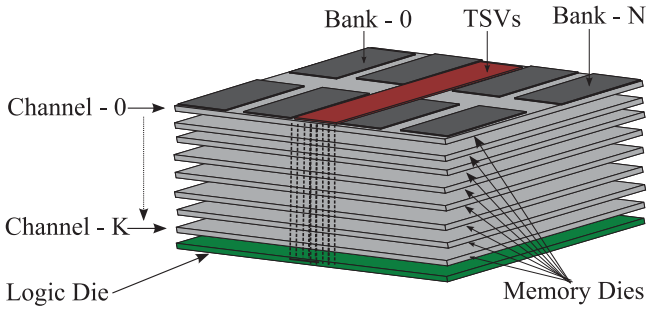

Fig. 6. A high-bandwidth stacked memory (HBM) with TSVs connecting all dies to the logic layer.

5.3. Modeling TSV Faults for 3D Memory Systems

In a 3D-stacked memory, TSVs act like conduits for addresses, data, command, and power to every level in the stack.[4] Figure 6 shows a stacked memory[5] that is organized similarly to the high-bandwidth memory standards (HBM) [Standard 2013]. In HBM, every memory die acts as an independent channel(s), consisting of banks that are connected to the logic die using TSVs.

In 3D memory, data, address, command, or power TSVs can be faulty. We describe how to model each such fault.

5.3.1. Address TSV Faults. An address TSV in HBM will addresses a cache line in a bank that belongs to a channel. The address TSV is connected to row and column decoders in a bank to select the appropriate content in the row. A faulty address TSV will make a single bit in the address erroneous. Due to this, half of the memory will be incorrectly addressed. So, a faulty address TSV can be modeled as a half-bank fault.

5.3.2. Data TSV Faults. A data TSV in HBM transfers data from one or more banks in a channel. A single 64B cache line is transferred in two DDR cycles. Thus, the number of data TSVs can be limited to the size of the cache line (512 bits) divided by the number of flits (4). Hence, a channel in HBM consists of 128 data TSVs. A faulty data TSV causes 4 bits in a cache line to fail for the given channel. Therefore, data TSV failures can be modeled as multicolumn failures.

5.3.3. Command TSV Faults. Command TSVs transfer the memory commands to the appropriate channel. These commands are used to read, write, activate, precharge,

[4]TSVs in stacked memories can be envisioned similarly to metal and IO pad connections in 2D DIMMs with one key difference. In this article, we assume that the TSV technology usually does not discriminate between address and data TSV lengths and widths and tends to be uniform. Metal connections, on the other hand, are routed in different layers and can have different characteristics and may result in different fault models. Furthermore, just changing the layout of TSVs enables various data organizations in the address and data space.

[5]Memory stack may also be organized as per-hybrid memory cube or tezzaron models [Pawlowski 2011; Tezzaron Corp. 2010]. The fundamental principles for modeling address and data faults will not change with a change in organization. These organizations only provide different address, data, command, and power TSV mappings.

Table IV. Distribution of Faults Encountered in a 7-Year period

No. Faults Encountered	1× FIT Rate	10× FIT Rate
0	92.9%	48.3%
1	6.7%	35.1%
2	0.2%	12.9%
3+	0.2%	3.7%

and refresh the memory system. A faulty command TSV usually results in jumbled DRAM operations and results in functional failures. These are usually catastrophic failures and can be denoted as channel (multibank) faults.

5.3.4. Power TSV Fault. Power TSVs supply power to the memory channel and contain voltage and ground lanes. Any failure in power TSVs will result in lower power, possibly resulting in a system shutdown. Such failures are fatal and can be modeled using channel (multibank) faults.

6. RAPID SIMULATION VIA EVENT-BASED FAULT INJECTION AND BOUNDING MAXIMUM TRIALS

Thus far, we have assumed the notion of interval-based fault injection, as it is the intuitive way in which a typical fault simulator would get built. For each interval, we would consult a random number generator to determine if a fault must be injected or not for each component. For such a Monte Carlo simulation to evaluate a system lifetime of 7 years, the total number of calls to the random number generator is equal to 20,000 (3-hour intervals each) times the number of components. We found that this was the key reason for the simulation to take several hours. Fortunately, real-word devices have modest FIT rates and a memory system experiences only a few faults during its lifetime. To corroborate this hypothesis, we measured the average number of faults (of any type) that got injected in a system with 18 chips (two DIMMs each with nine chips). We performed this study both at a nominal FIT rate and at a 10× higher FIT rate. Table IV shows the distribution of number of faults in this system, for the nominal FIT rate (1×) and higher FIT rate (10×).

For a nominal FIT rate, in 99.8% of the cases only fewer than three faults get injected. In fact, even if the FIT rate was 10× higher, still in more than 96% of the cases fewer than three faults get injected. Therefore, it is wasteful to consult the random number generator several tens of thousands of times to get these three faults. If instead of asking the question "Is there a fault in this interval?" we asked, "What is the time duration between consecutive faults?", then we can improve simulation time by several orders of magnitude. Fortunately, the duration between faults can be estimated easily: for events that occur independently with a constant average rate, the distance between events tend to be exponentially distributed [Balakrishnan 1996]. Therefore, instead of consulting the uniform random number generator many times (20K), we can instead consult the exponential random number generator only a few times (less than three in the common case). This is the key insight behind the *Event-Based Fault Injection* framework, as shown in Figure 7.

6.1. Incorporating Event-Based Fault Injection

The distance in time between faults is derived from the expression $P_{fault}(t) = 1 - e^{-\lambda \cdot t}$ (i.e., Equation (3)) and solving for t. The time distance for a fault is depicted by Equation (6):

$$t = -\frac{\log(1 - P_{fault}(t))}{\lambda}. \tag{6}$$

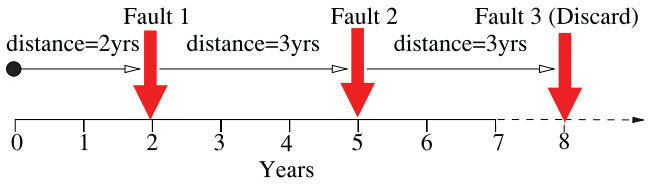

Fig. 7. Event-Based fault injection estimates the distance in time between faults (faults appearing at >7 years get ignored).

Table V. Simulation Time Comparison (for 1 Million Trials)

	Simulation Time (Wall Clock)	
ECC Scheme	Interval Based	Event Based
BCH-1	49.5 hours	34 seconds
Chipkill	49.2 hours	33 seconds

Since $P_{fault}(t)$ can have any value between 0 and 1, we compute the distance in time for faults during a trial as shown in Equation (7):

$$t = -\frac{\log(1 - U(0,1))}{\lambda}. \tag{7}$$

For each component in the system, we first obtain the timestamp of all the faults (if any) encountered by that component in the 7-year period, depending on the failure rate of the component. Then, we merge the timestamps of all the faults of all the components (as there are fewer than three faults in total in the common case, this step incurs negligible time). Once the global list of timestamps of all the faults is available, the simulation time is advanced from 0 to the time of the first fault, to the time of the second fault, and so on, until the 7-year time limit is reached. Given that most trials have fewer than three faults, the memory system usually performs fewer than three evaluations for a 7-year simulation, instead of approximately 20,000 steps in the interval-based fault injection model, resulting in a significant reduction in simulation time. We validated that the results obtained from the Event-Based model are virtually identical (within 0.18%) to the results obtained from the Interval-Based model.

6.2. Simulation Time: Interval Based Versus Event Based

We use the "wall clock" simulation time to provide a fair comparison for Interval-Based and Event-Based simulation engines. All of our experiments are performed on an AMD Opteron® Processor 6276 operating at 2.3GHz. Table V shows the time to complete 1 million Monte Carlo trials, each for a system lifetime of 7 years, for Interval-Based and Event-Based fault injection. We compare two ECC techniques: BCH-1 and Chipkill. Event-Based simulation is approximately 5,000× faster than Interval-Based simulation for both BCH-1 and Chipkill. Event-Based simulation benefits from having only a few (usually less than three) events against approximately 20,000 intervals for Interval-Based simulation. The simulation time for BCH-1 is slightly higher than Chipkill, because unlike Chipkill, the intersection algorithm of BCH-1 loops multiple times over the same FDs.

6.3. Determining the Number of Monte Carlo Trials

Every trial in FaultSim can have only two outcomes: system operational or system failure. In a trial when the system works, the system ECC corrects all the faults and this trial occurs with a probability q. During a trial when the system fails, the system ECC cannot correct the errors with a probability p. Events with two exclusive outcomes

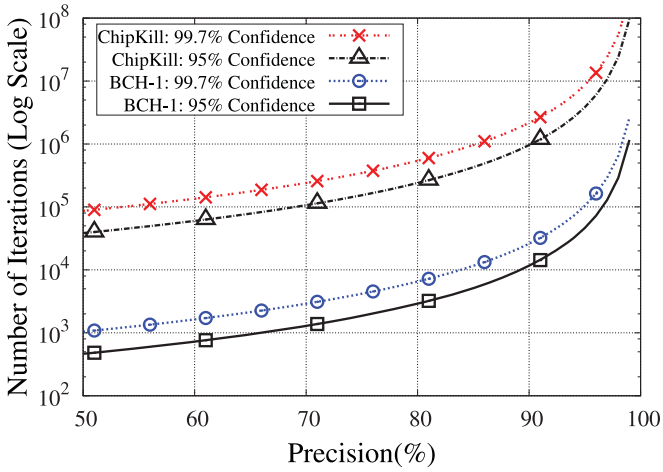

Fig. 8. Number of Monte Carlo trials required to achieve a desired precision for BCH-1 and Chipkill.

depict Bernoulli trials and follow a binomial distribution. Therefore, for n trials, the mean number of failures (μ) and their deviation (σ) in FaultSim can be determined by Equation (8):

$$\begin{aligned} \mu &= n \cdot p \\ \sigma &= \sqrt{n \cdot p \cdot q}. \end{aligned} \tag{8}$$

Since $q = 1 - p$ and since p is very small for real-world failures, Equation (8) degenerates to Equation (9):

$$\sigma \approx \sqrt{n \cdot p} \approx \sqrt{\mu}. \tag{9}$$

The value of μ can be estimated to obtain a desired level of precision. For example, for a given value of n, μ is 400 failures. Then, we can conclude that subsequent runs with the same n will lie between 360 failures and 440 failures with 95% confidence (i.e., $\mu \pm 2\sigma$), or between 340 failures and 460 failures with 99.7% confidence (i.e $\mu \pm 3\sigma$). Thus, the experiment with 400 total system failures can be deemed to have 90% precision with 95% confidence, or 85% precision with 99.7% confidence. FaultSim can be simulated either for a fixed number of trials irrespective of the precision or until a given precision bound is reached.

Figure 8 shows the number of trials required for BCH-1 and Chipkill codes for providing a given level of precision at different confidence levels. At higher precision, the number of trials increases rapidly and is influenced by the underlying ECC scheme. Our analysis enables the user to tailor precision to his or her own compute resources and application. For example, one can run only 100K trials for BCH-1 but 1 million for Chipkill for the same precision levels.

7. MODELING DIFFERENT REPAIR ALGORITHMS

The ECC and repair algorithms determine the resilience of the memory system to faults. We illustrate the algorithms for single-error-correcting BCH-1 [Chen and Hsiao 1984] and Chipkill [Dell 1997] in the following subsections.[6] Additional repair schemes can be added to FaultSim easily by creating a new C++ class.

[6]We have also implemented RAID-like schemes and sparing schemes. However, due to space limitations, we only discuss the two schemes that we validate analytically. Other schemes, or combinations of schemes, can easily be incorporated in FaultSim as well.

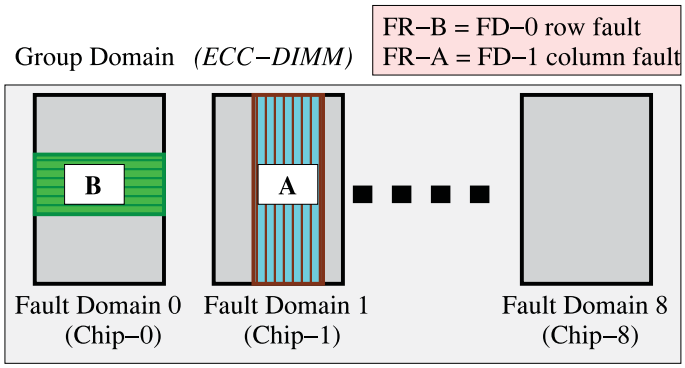

Fig. 9. A memory die (Fault Domain) can contain column (A) and row (B) faults in different memory banks.

7.1. Error Correction and Detection in FaultSim

Figure 9 shows the ECC-DIMM organization (memory block) with chip-0 (FD-0) having a faulty row (FR-B) and chip-1 (FD-1) having a faulty column (FR-A).

The *intersect* function compares FRs across chips and will show that FR-A and FR-B have overlapping addresses. The *intersect* function finds only those FRs that overlap and provides a very fast comparison. A codeword read from an ECC-DIMM is distributed across DRAM dies and has the same address in each die. Therefore, any codeword read from the region of intersection of FR-A and FR-B will show bit errors spanning two devices.

7.2. Efficiency in Space

A common task of the ECC algorithm is to read fault locations from the FDs. To maximize performance, these FDs must store faults with minimum memory overhead. A naive approach is to represent multigranularity faults using an array of bits to depict rank, bank, row, and column faults. Unfortunately, rows, columns, banks, and ranks contain many bits (several thousands to several billion), and this implementation would be inefficient in space. For instance, in an ECC-DIMM with 8Gb dies and eight banks per die, a single faulty bank contains a billion faulty bits, resulting in unreasonable memory overheads. FaultSim addresses this inefficiency by representing a fault with only a single FR. This enables FaultSim to represent numerous multigranularity faults in multiple chips with negligible memory overhead.

7.3. Efficiency in Time

An important task performed by the ECC algorithm is the *intersect* function. To determine the number of faulty bits in a codeword using the naive algorithm, every bit in every codeword must be scanned and counted. The leads to a time complexity of $O(n)$, where n is the total number of faulty bits in a fault granularity (e.g., bank, row, column, bit, etc.). FaultSim performs considerably better than the naive scanning algorithm since it uses a fast $O(1)$ *intersect* function.

7.4. ECC Algorithms

Common DRAM ECC algorithms use parity to detect and correct errors. When a cache line is read, data and parity are fetched over several bus cycles, with one or more cycles forming a codeword. Each chip typically provides 8- or 4-bit chunks of the codeword. We briefly describe the ECC algorithms for two commercial schemes: BCH-1 and Chipkill.

7.4.1. BCH-1. Consider an ECC-DIMM with nine ($\times 8$) chips (single rank) and codewords spanning the width of the data bus (typically $64 + 8 = 72$ bits). Algorithm 2

(shown in the Appendix) considers every FR in each chip as a reference (FR0). The algorithm then counts faulty bits in codewords covered by the same FR. Additionally, it also counts 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. To avoid double-counting intersecting faults within the nested loops, care is taken that the bit intersection count (*n_intersections*) is reset for each reference FR0.

If the user is interested in error detection capability, the simulation must continue after finding uncorrectable errors, until the first undetectable error is found.

7.4.2. Chipkill. Symbol-based ECC schemes such as Chipkill can be implemented using Algorithm 1 (shown in the Appendix). The variant of Chipkill we consider does not count individual bits and instead counts 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.

7.5. Scrubbing and Repair

FaultSim simulates periodic scrubbing and removes correctable transient faults by deleting such FRs from FDs. By allocating spare capacity, FaultSim can model repair schemes that use remapping or spare resources [Nair et al. 2013]. In such repair schemes, during the scrubbing interval, all correctable data errors get remapped into a spare region.

8. VALIDATION WITH ANALYTICAL MODELS

It is possible to derive approximate analytical failure models' BCH-1 code and Chipkill using FIT data from field studies. However, the use of different probabilities for various fault granularities complicates the analysis.

8.1. BCH-1 Code Analytical Model

BCH-1 code can correct single-bit errors and detect double-bit errors. Fortunately, memory systems have large capacities relative to the failure rates and the BCH-1 code tolerates all isolated 1-bit errors. Due to this, there is a low probability of single-bit faults accumulating in the same codeword. Therefore, we can ignore the impact of single-bit faults. However, any multibit fault results in uncorrectable errors. Let the probability of failure of a chip due to multibit faults be $P_{Fail-MultiBit}$. The probability that the n-chip system experiences a multibit fault in a system with BCH-1 code ($P_{SysFail-BCH1}$) is given by Equation (10):

$$P_{SysFail-BCH1} \approx 1 - \binom{n}{0} P^0_{Fail-MultiBit} \times (1 - P_{Fail-MultiBit})^n. \tag{10}$$

8.2. Chipkill Analytical Model

Chipkill can tolerate failure of one chip. Chipkill can also tolerate bank, column, row, and bit faults in different codewords across multiple chips. Field studies show that large-granularity faults are nearly as likely as small-granularity faults. Large-granularity faults that occur with a relatively high failure probability are likely to coincide with other faults. To a first order, this implies that the large-granularity faults such as rank, multibank, and single-bank faults determine Chipkill resilience. To a first order, the failures in Chipkill occur due to two phenomena. First, in a memory system with a multibank or multirank faulty chip there occurs another faulty chip with any other failure. Second, if a single-bank faulty chip has addresses that coincide with any chip that has a small-granularity fault.

8.2.1. Influence of Multibank and Multirank Faults. For an n-chip system, let probability of any faulty chip be $P_{Fail-Any}$, and multibank or rank faults be $P_{Fail-Multi}$. Let the probability of chip failure due to a single-bank fault be $P_{Fail-OneBank}$. The probability of chip failure due to nonmultibank or rank faults (i.e., bit, word, column, row, or bank) is given by $P_{Fail-NonMulti}$. The probability of precisely one chip with a multibank or rank fault is given by Equation (11):

$$P_{OneMultiBank} = \binom{n}{1} P^1_{Fail-Multi} \times (1 - P_{Fail-Multi})^{n-1}. \tag{11}$$

The probability of system failure ($P_{SysFail(1)}$) when one multibank or a rank fault ($P_{OneMultiBank}$) occurs along with with any other faulty chips is given by Equation (12):

$$P_{SysFail(1)} = P_{OneMultiBank} \times \left[1 - \binom{n-1}{0} P^0_{Fail-Any} \times (1 - P_{Fail-Any})^{n-1}\right]. \tag{12}$$

8.2.2. Influence of Single-Bank Faults. The probability of failure when one single-bank fault ($P_{OneChipBank}$) occurs in one chip is given by Equation (13):

$$P_{OneChipBank} = \binom{n}{1} P^1_{Fail-OneBank} \times (1 - P_{Fail-OneBank})^{n-1}. \tag{13}$$

The probability of system failure ($P_{SysFail(2)}$) when one single-bank fault occurs along with any fault contained within a single bank on another chip ($P_{Fail-NonMulti}$) is represented by Equation (14). Note that we divide by eight to account for eight banks in the chip:

$$P_{SysFail(2)} = (1/8) \times P_{OneChipBank} \times \left[1 - \binom{n-1}{0} P^0_{Fail-NonMulti} \times (1 - P_{Fail-NonMulti})^{n-1}\right]. \tag{14}$$

8.2.3. Total System Failure Probability. The probability that the system fails for Chipkill ($P_{SysFail-Chipkill}$) is the sum of Equations (12) and (14) and is given by Equation (15):

$$P_{SysFail-Chipkill} \approx P_{SysFail(1)} + P_{SysFail(2)}. \tag{15}$$

8.3. Validation

FaultSim uses FIT rates for DRAM chips to insert faults into the memory system (from Table I). The error correction capability of BCH-1 code and Chipkill is compared over a 7-year lifetime. We simulate an 18-chip one-rank system with a bus width of 4 bits per chip. To maintain accuracy, we ran these simulations for 2.5 million iterations. The time-interval-based simulation has a 3-hour time step (interval). Memory scrubbing was disabled in FaultSim as the analytical model does not model memory scrubbing.

Figure 10 shows that FaultSim reports a system failure probability of 3.45×10^{-2} and 4.15×10^{-4} after 7 years for BCH-1 and Chipkill, respectively. The analytical model for BCH-1 and Chipkill has a difference of only 0.032% and 8.41% versus the simulation results. Note that the analytical model is approximate, so the deviation does not necessarily indicate an inaccuracy in simulation results. Furthermore, for the Chipkill system, the system failure rate is fairly low (415 parts per million), so an 8.41% deviation results in a very small change in absolute numbers (35 parts per million).

9. RESULTS AND ANALYSIS

9.1. Sensitivity to Number of Devices

As memory systems scale, they are likely to be interconnected using complex networks that have multiple devices. Intuitively, a memory system with more devices will exhibit

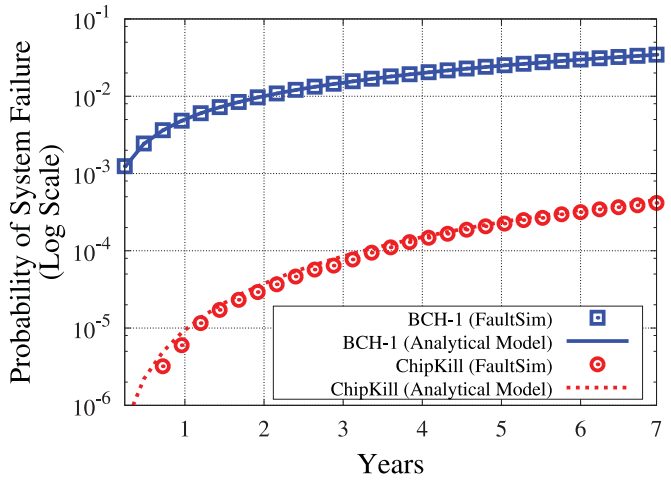

Fig. 10. Probability of memory system failure with elapsed time for BCH-1 code and Chipkill shows that curves from "Analytical Models" and FaultSim closely follow each other.

Table VI. Variation in Number of Faults with Number of Chips

Total Faults	18 Chips (2 DIMMs)	36 Chips (4 DIMMs)	72 Chips (8 DIMMs)	144 Chips (16 DIMMs)
No Fault	92.98%	86.45%	74.68%	55.79%
1 Fault	6.77%	12.59%	21.79%	32.62%
2 Faults	0.24%	0.92%	3.2%	9.45%
3+ Faults	0.01%	0.05%	0.32%	2.15%

Table VII. FIT Rate Versus Sim. Time: 1 Million Systems (7 Years Each)

		Simulation Time (Wall Clock)			
		1× FIT	2× FIT	4× FIT	8× FIT
BCH-1	Interval Based	49.5 hrs	47.9 hrs	47.1 hrs	43.6 hrs
	Event Based	34s	33.9s	34.1s	34.1s
Chipkill	Interval Based	49.2 hrs	48.6 hrs	48.5 hrs	48.4 hrs
	Event Based	33s	32.9s	33s	33.1s

a larger number of faults. Table VI shows the variation in the probability of faults with an increase in the number of chips in a memory system. An increase in memory chips results in a nonlinear scaling in the number of systems with one or two faults. However, even for a modestly large system, the episode of three or more faults occurring during the system lifetime remains uncommon (2.15%).

9.2. Sensitivity to FIT Rates

As memory technologies scale to lower nanometer nodes, the fault rate tends to increase [Nair et al. 2013]. Factors such as altitude and temperature also influence the failure rates [Sridharan et al. 2013; White et al. 2011]. Ideally, FaultSim must perform evaluations for higher device failure rates without incurring any additional simulation time. Table VII shows the sensitivity simulation time to FIT rate. As FIT rates increase, faults start appearing earlier in the memory system lifetime and this leads to uncorrectable errors, which causes Interval-Based simulation to terminate early and its simulation time to be reduced. As Event-Based simulation performs ECC checks

Table VIII. Reliability Analysis of Scrubbing

ECC Scheme	Probability of System Failure	
	With Scrubbing	Without Scrubbing
BCH-1	3.43×10^{-2}	3.45×10^{-2}
Chipkill	3.22×10^{-4}	4.15×10^{-4}

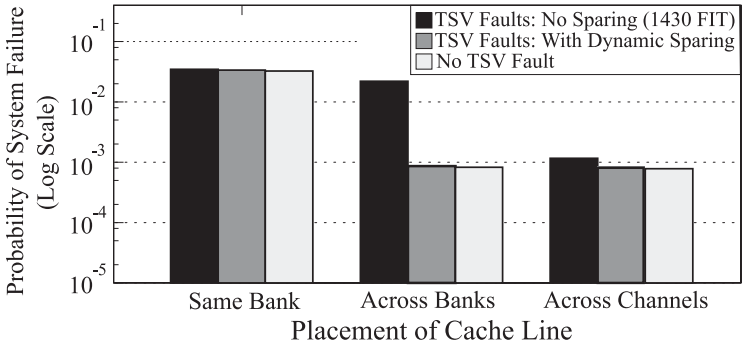

Fig. 11. Using FaultSim for analyzing the impact of TSV faults and dynamic TSV sparing in 3D memories.

after the faults in a system lifetime are injected, Event-Based simulation time is almost insensitive to FIT rates.

9.3. Effect of Memory Scrubbing

High-reliability systems also employ periodic memory scrubbing. FaultSim can incorporate scrubbing with any memory reliability scheme. Table VIII shows the resilience from scrubbing when compared to a baseline scheme that does not employ scrubbing. Scrubbing is invoked once every 12 hours. Since BCH-1 code is vulnerable to frequently occurring large-granularity failures, scrubbing shows negligible benefits for BCH-1 code. On the contrary, Chipkill is robust against large and small-granularity faults and scrubbing helps remove some correctable transient failures.

9.4. Effect of Dynamic TSV Sparing for 3D Memories

FaultSim can also be used to analyze newer failures modes, such as TSV faults in 3D-stacked DRAM. As TSV faults can cause large-granularity failures, the impact depends on the policy for placement of cache line, whether the line is the same bank, across banks, or across channels. Faulty TSVs can be mitigated by using spare TSVs. We use FaultSim to analyze the impact of TSV failures and TSV sparing on system reliability assuming a TSV failure rate of 1,430 FIT [Nair et al. 2014]. Figure 11 shows the probability of system failure for different data striping policies for three systems: no TSV faults, TSV faults but no TSV sparing, and dynamic TSV sparing. By performing dynamic sparing, the chances of incurring uncorrectable failures due to TSVs are reduced significantly.

10. RELATED WORK

Several studies in academia have looked at improving the reliability of SRAM, DRAM Cache, DRAM-based memories, and Flash memories [Kim et al. 2007; Nair et al. 2013; Udipi et al. 2012; Sim et al. 2013; Chen and Zhang 2014; Cai et al. 2015]. A recent study has investigated the impact of TSV and large-granularity faults in stacked memory systems [Nair et al. 2014]. Although these techniques present stronger RAS

schemes and architectural techniques, they do not talk about how to build an integrated framework for estimating RAS.

Field studies on supercomputing and server clusters help obtain real-world data. Some studies on DRAM-based main memory systems have investigated data errors [Schroeder et al. 2009; Schroeder and Gibson 2010]. However, contrary to reporting fault rates, these studies report data error rates, which depend on the application that the system executes and its memory mapping. For instance, a memory system with a single bit with permanent fault can result in billions of errors if the bit remains uncorrected and if the application frequently accesses the faulty memory bit. Similarly, systems can also report billions of errors if the OS naively maps pages into such faulty locations without decommissioning the region. However, to evaluate reliability, fault statistics provide a clear metric when compared to error statistics. To address this, Sridharan and Liberty [2012] and Sridharan et al. [2013] present a clearer distinction between errors and faults and report memory faults and their positional effects by studying supercomputer clusters. Although these studies present detailed failure data, they do not use this data to suggest quick reliability exploration techniques. Commercial solutions like Chipkill present specific results for certain FIT rates; however, they do not estimate memory reliability as these systems scale [Dell 1997]. In an attempt to estimate reliability, recent studies have investigated integrating field data into analytical models [Jian et al. 2013]. Although this is a significant step forward, complex analytical models are required to investigate memory reliability. Studies that use analytical methods for estimating faults use Monte Carlo techniques to verify their model [Jian et al. 2013; DeBardeleben 2013]. However, none of these studies present techniques to simplify and reduce the simulation time for Monte Carlo-based simulations.

FPGA-based fault-tree analysis has been proposed for circuit-level fault simulations. This analysis constructs a time-to-failure tree for the system based on the failure times of individual components. FPGAs are used to accelerate the Monte Carlo process to simulate individual components. We believe that the fundamentals of FaultSim in accelerating Monte Carlo simulation can enable making this event-based simulation and reduce the simulation time [Ejlali and Miremadi 2004]. Crashtest tests logic resiliency and evaluates how the design reacts to faults [Pellegrini et al. 2008]. Along similar lines, GangES and Relyzer provide frameworks to evaluate errors in execution states [Hari et al. 2014; Sastry Hari et al. 2013]. From a different perspective, the performance of parallel runs in the Monte Carlo simulator can be improved by running them on GPGPUs [Braun et al. 2012]. FaultSim improves the performance of a single run and these techniques are orthogonal to the techniques employed by FaultSim.

11. SUMMARY

Memory reliability is a growing concern for all systems, ranging from HPC to servers to commodity systems. The increase in failure rate with smaller technology nodes, and the newer failure modes from emerging technologies (such as 3D DRAM) have made the investigation of efficient solutions to enhance memory reliability a key area of research. While simple mitigation schemes can be evaluated using analytical models, such models become quite complex for more advanced schemes and are not easily extendable to new schemes, or even a combination of schemes. A tool set that can quickly and accurately estimate the effectiveness of different reliability solutions will accelerate research in this area. To that end, this article makes the following contributions:

(1) We present FaultSim, a fast and accurate Monte Carlo simulator. We describe novel data structures for efficient tracking of faults that can occur at multiple granularities.

(2) We propose an Event-Based Fault Injection framework, which improves the simulation speed by a factor of 5,000× compared to Interval-Based simulation.
(3) We also discuss how to set the number of trials based on precision bounds. The simulation can be run for a fixed number of iterations to be within bounds.
(4) We show how FaultSim can be used to model a variety of fault mitigation schemes such as BCH-1, Chipkill, TSV Sparing for 3D Memories, and Scrubbing.

We successfully validate FaultSim using analytical models. As emerging memories come to market and knowledge of failures is gained through field studies, FaultSim will aid in rapid evaluation of evolving failure rates and resilience schemes. We continue to develop this tool via implementation of recently published fault models. This tool is now available for researchers as an open-source framework using a BSD3 licence agreement.

APPENDIX: PSEUDO-CODE FOR REPAIR SCHEMES

ALGORITHM 1: 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
```

ALGORITHM 2: 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
```

ALGORITHM 3: Pseudo-Code for Dynamic TSV Sparing

```
spareTSV=inactive
while simulation==active do
    if ECC Check == TSV Fault then
        if spareTSV == active then
            simulation=inactive
        end
    end
    spareTSV=active
end
```

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. We are grateful to Gabriel Loh of AMD Research and Prasun Gera of Georgia Tech for their comments on a prior version of this work. We also thank all members of our research group, CARAT Lab, at Georgia Tech and AMD Research for their insightful feedback. This work was supported in part by the Center for Future Architectures Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

REFERENCES

M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan. 2012. Efficient scrub mechanisms for error-prone emerging memories. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA'12)*. IEEE Computer Society, Washington, DC, 1–12.

K. Balakrishnan. 1996. *Exponential Distribution: Theory, Methods and Applications*. Taylor & Francis.

K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. 2008. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. Technical Report.

C. Braun, S. Holst, H. Wunderlich, J. M. Castillo, and J. Gross. 2012. Acceleration of monte-carlo molecular simulations on hybrid computing architectures. In *Proceedings of the 2012 IEEE 30th International Conference on Computer Design (ICCD'12)*. 207–212.

D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00)*.

Y. Cai, Y. Luo, S. Ghose, and O. Mutlu. 2015. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. 438–449.

T. E. Carlson, W. Heirman, and L. Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. 1–12.

C. L. Chen and M. Y. Hsiao. 1984. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal* 28, 2 (March 1984), 124–134.

L. Chen and Z. Zhang. 2014. MemGuard: A low cost and energy efficient design to support and enhance memory system reliability. In *Proceedings of the 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. 49–60.

R. T. Chien. 1964. Cyclic decoding procedures for Bose- Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, 10, 357–363.

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

H. H. Chung, C. 2013. Partial parity cache and data cache management method to improve the performance of an SSD-based RAID. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 99, 1470–1480.

N. DeBardeleben. 2013. Reliability models for double chipkill detect/correct memory systems. In *Los Alamos National Laboratory Associate Directorate for Theory, Simulation, and Computation (ADTSC'13) LAUR 13-20839*.

T. J. Dell. 1997. *A White Paper on the Benefits of ChipkillCorrect ECC for PC Server Main Memory*. Technical Report 11/19/97. IBM.

A. Ejlali and S. G. Miremadi. 2004. FPGA-based monte carlo simulation for fault tree analysis. *Microelectronics Reliability* 44, 6 (2004), 1017–1028.

H. Emery. 2013. *The IBM zEnterprise EC12 (zEC12) System: Processor, Memory and System Structure Enhancements*. Technical Report.

E. Fujiwara and D. K. Pradhan. 1989. *Error-Control Coding in Computers Systmes*. Prentice-Hall.

S. K. S. Hari, R. Venkatagiri, S. V. Adve, and H. Naeimi. 2014. GangES: Gang error simulation for hardware resiliency evaluation. In *Proceedings of the 2014 ACM / IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. 61–72.

X. Jian, N. Debardeleben, S. Blanchard, V. Sridharan, and R. Kumar. 2013. Analyzing reliability of memory subsystems with double chipkill detect/correct. In *Proceedings of the 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'13)*.

L. Jiang, Q. Xu, and B. Eklow. 2012. On effective TSV repair for 3D-stacked ICs. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'12)*. 793–798.

S. J. Kamat and M. W. Riley. 1975. Determination of reliability using event-based monte carlo simulation. *IEEE Transactions on Reliability,* R-24, 1 (April 1975), 73–75.

D.-H. Kim, P. J. Nair, and M. K. Qureshi. 2015. Architectural support for mitigating row hammering in DRAM memories. *Computer Architecture Letters* 14, 1 (Jan. 2015), 9–12.

J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. 2007. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40'07)*. IEEE Computer Society, 197–209.

Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceedings of the 41st Annual International Symposium on Computer Architecuture (ISCA'14)*. IEEE, 361–372.

S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009 (MICRO-42'09)*. 469–480.

S. Li, K. Chen, M.-Y. Hsieh, N. Muralimanohar, C. D. Kersey, J. B. Brockman, A. F. Rodrigues, and N. P. Jouppi. 2011. System implications of memory reliability in exascale computing. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. 1–12.

J. I. McCool. 2012. *Using the Weibull Distribution: Reliability, Modeling and Inference*. Wiley.

J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *Proceedings of the 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA'10)*. 1–12.

N. Muralimanohar, R. Balasubramonian, and N. Jouppi. 2007. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40'07)*.

P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi. 2015. Reducing read latency of phase change memory via early read and Turbo Read. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 309–319.

P. J. Nair, D.-H. Kim, and M. K. Qureshi. 2013. 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)*. ACM, New York, NY, 72–83.

P. J. Nair, D. A. Roberts, and M. K. Qureshi. 2014. Citadel: Efficiently protecting stacked memory from large granularity failures. In *MICRO'14*.

J. T. Pawlowski. 2011. Hybrid memory cube (HMC). In *HOT-CHIPS 23*.

A. Pellegrini, K. Constantinides, Dan Zhang, S. Sudhakar, V. Bertacco, and T. Austin. 2008. CrashTest: A fast high-fidelity FPGA-based resiliency analysis framework. In *Proceedings of the IEEE International Conference on Computer Design, 2008 (ICCD'08)*. 363–370.

M. K. Qureshi. 2011. Pay-As-You-Go: Low-overhead hard-error correction for phase change memories. In *MICRO*. 318–328.

M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu. 2015. AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. 427–437.

D. A. Roberts and P. J. Nair. 2014. FaultSim: A fast, configurable memory-resilience simulator. In *The Memory Forum: In conjunction with ISCA-41*.

S. K. Sastry Hari, S. V. Adve, H. Naeimi, and P. Ramachandran. 2013. Relyzer: Application resiliency analyzer for transient faults. *IEEE Micro* 33, 3 (May 2013), 58–66.

B. Schroeder and G. A. Gibson. 2010. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing,* 7, 4 (2010), 337–350.

B. Schroeder, E. Pinheiro, and W.-D. Weber. 2009. DRAM errors in the wild: A large-scale field study. *SIGMETRICS Performance Evaluation Review* 37, 1 (June 2009), 193–204.

Silicon Power 2010. *DDR3 ECC Unbuffered DIMM Spec Sheet*. Silicon Power.

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

Y. H. Son, S. Lee, S. O, S. Kwon, N. S. Kim, and J. H. Ahn. 2015. CiDRA: A cache-inspired DRAM resilience architecture. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 502–513.

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. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)*. ACM, New York, NY, 297–310.

V. Sridharan and D. Liberty. 2012. A study of DRAM failures in the field. In *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*. 1–11.

V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. 2013. Feng shui of supercomputer memory: Positional effects in DRAM and SRAM faults. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*. Article 22, 22:1–22:11 pages.

JEDEC Standard. 2013. High bandwidth memory (HBM) DRAM. In *JESD235*.

Tezzaron Corp. 2010. *Octopus 8-Port DRAM for Die-Stack Applications: TSC100801/2/4*. Tezzaron Corp.

A. Thomasian and J. Menon. 1997. RAID5 performance with distributed sparing. *IEEE Transactions on Parallel and Distributed Systems,* 8, 6 (1997), 640–657.

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 *Proceedings of the 2012 39th Annual International Symposium on Computer Architecture (ISCA'12)*. 285–296.

S. Wang, H. (C.) Hu, H. Zheng, and P. Gupta. 2015. MEMRES: A fast memory system reliability simulator. In *SELSE 2015*.

M. White, J. Qin, and J. B. Bernstein. 2011. A study of scaling effects on DRAM reliability. In *Proceedings of the 2011 Annual Reliability and Maintainability Symposium (RAMS'11)*. 1–6.

S. Zacks. 1992. *Introduction to Reliability Analysis*. Springer Texts in Statistics.

Received March 2015; revised August 2015; accepted September 2015