2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

RelaxFault Memory Repair

Dong Wan Kim
Electrical and Computer Engineering Dept.
The University of Texas at Austin
wannikim@utexas.edu

Mattan Erez
Electrical and Computer Engineering Dept.
The University of Texas at Austin
mattan.erez@mail.utexas.edu

ABSTRACT

Memory system reliability is a serious concern in many systems today, and is becoming more worrisome as technology scales and system size grows. Stronger fault tolerance capability is therefore desirable, but often comes at high cost. In this paper, we propose a low-cost, fault-aware, hardware-only resilience mechanism, *RelaxFault* , that repairs the vast majority of memory faults using a small amount of the LLC to remap faulty memory locations. RelaxFault requires less than 100KiB of LLC capacity, has near-zero impact on performance and power. By repairing faults, RelaxFault relaxes the requirement for high fault tolerance of other mechanisms, such as ECC. A better tradeoff between resilience and overhead is made by exploiting an understanding of memory system architecture and fault characteristics. We show that RelaxFault provides better repair capability than prior work of similar cost, improves memory reliability to a greater extent, and significantly reduces the number of maintenance events and memory module replacements. We also propose a more refined memory fault model than prior work and demonstrate its importance.

1. INTRODUCTION

Memory reliability has been a major design constraint for mission-critical and large-scale systems for many years. Continued innovation is still necessary because fault rates, and the errors they lead to, grow with system size [1, 2, 3, 4, 5] and because some faults become more likely as fabrication technology advances [6, 7, 8, 9, 10]. Furthermore, recent field studies have shown that more severe permanent/intermittent and multi-bit faults are roughly as frequent as single-bit and transient ones [1, 2, 3, 4, 5]. Therefore, strong *error checking and correcting* (ECC) schemes that can correct multi-bit errors have been developed and are in use. However, using ECC to correct the numerous recurring errors from permanent faults forces trading off cost, performance, and reliability. Firstly, a permanent fault is likely to result in numerous erroneous accesses, each requiring possibly high correction overhead. Secondly, once redundancy is used for correction, further errors may go uncorrected leading to data loss, or worse, go undetected and result in *silent data corruption* (SDC). Stronger ECC can then be used to tolerate more errors, but at higher overhead.

A good way to relax this tradeoff is to repair faulty memory and eliminate the errors from the permanent faults. We discuss various repair mechanisms in Section 6, but specifically focus on the recently introduced FreeFault mechanism [11]. FreeFault is appealing because it can repair the most common DRAM faults without requiring changes in main memory interfaces, modules, and devices and without requiring additional storage resources. FreeFault achieves this by redirecting memory requests that target memory addresses with permanent faults to only access a processor's *last level cache* (LLC). As long as the number of remapped addresses is small, FreeFault incurs negligible performance overhead because only a few cachelines are dynamically assigned for DRAM repair. However, FreeFault consumes far more cache resources than necessary, diminishing its effectiveness. In this paper we introduce *RelaxFault* , which exploits knowledge about how common DRAM faults map to physical addresses to significantly increase the range of faults that can be effectively repaired.

The main insight behind RelaxFault is that the vast majority of DRAM faults affect only a small number of bits in a few (typically just one) rows *or* columns of, typically a single DRAM device. However, the mapping of physical addresses to DRAM locations, which aims to maximize performance, spreads this small number of bits over many cache lines. FreeFault utilizes cacheline locking to force accesses to faulty memory regions to only touch the cache and avoid the faulty DRAM. Thus, the spreading effect causes FreeFault to lock many cachelines for repair, which reduces hit rate, degrades performance and power efficiency, and limits *repair coverage*—the fraction of faults that can be effectively repaired. With RelaxFault, we introduce a new access mode to the last level cache that is used exclusively for repair. This mode uses a different mapping of physical addresses to cache locations that utilizes the correlations of fault locations to coalesce many failed bits into a small number of cachelines.

We evaluate the effectiveness and overheads of RelaxFault and show that assuming current observed fault rates, RelaxFault repairs 90% of nodes with any permanent memory faults using at most 1 way in any LLC set and less than 82KiB of the last level cache capacity (less than 1% of the capacity of most current high-end processors). Even with a 10× fault rate, RelaxFault still repairs 84% of nodes while requiring less than 93KiB capacity. To operate, RelaxFault requires just 16KiB of new storage to maintain its state and a small amount of logic. Thus, RelaxFault has essentially

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.62

no impact on performance, but significant impact on reliability: RelaxFault reduces the rate of *detected uncorrectable errors* (DUEs) and SDCs by 52% and 41%, respectively. RelaxFault also significantly improves system availability and repairs 87% of the memory modules transparently, without system maintenance and module replacement downtimes. RelaxFault repairs more than one third of the nodes that cannot be repaired by the prior FreeFault mechanism [11]. If up to 4 ways at most in any set may be used, then with just over 256KiB cache capacity, RelaxFault repair coverage reaches 97%; FreeFault would need more than 1MiB of LLC capacity to achieve similar coverage with the most affected sets losing up to 16 ways.

To the best of our knowledge, all related prior work is based on fault models that assume every memory device exhibits precisely the average overall fault rate. As faults accumulate, ECC may not be able to correct or even detect some errors, leading to failures. We observe that the uniform fault-rate model fails to predict the actual failure rate measured on large scale systems [1, 5, 12]. We remedy this by introducing a refined fault-injection methodology that considers module-to-module and device-to-device variations and can match the failure rates observed in production systems. While our model is simple, we believe it provides both more realistic scenarios for evaluating reliability mechanisms and a solid start for even better models; such models can only be developed with access to detailed error logs, which are unfortunately not currently public.

To summarize, our main contributions are:

- We highlight the importance of understanding the physical layout aspects of faults and demonstrate the benefits of tailoring repair-remapping schemes to account for fault location correlations.
- We describe RelaxFault and the microarchitectural modifications required to effectively repair 90% of faulty nodes with only 82KiB of LLC capacity, no more than 1 way per LLC set, with just 16KiB of added metadata on chip, and with local changes that are limited to the LLC and its interface with the memory controller.
- We develop a refined fault injection methodology that accounts for reliability variations and that more closely matches observed memory failure rates in the field; we show that this affects multi-device fault statistics, which enables us to better study DUEs, SDCs, and *dual in-line memory module* (DIMM) replacements.
- We conclude that very low-cost DRAM repair reduces DUE and SDC rates by 52% and 41%, respectively, as well as saving 87% of memory modules that would otherwise need replacement.

The rest of this paper is organized as follows: Section 2 provides background on memory system organization and brief summary of the existing DRAM failure studies; Section 3 details the RelaxFault architecture and associated tradeoffs; Section 4 describes our evaluation methodology, including DRAM fault modeling; Section 5 presents our evaluation results; Section 6 discusses current retirement schemes and other related work; and Section 7 concludes the paper.

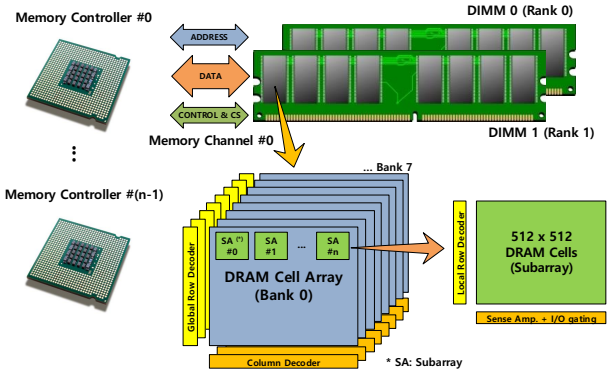


Figure 1: A simplified DRAM system (DIMM) structure.

2. BACKGROUND

DRAM Organization. Figure 1 illustrates a simplified organization of a typical DIMM based DRAM system [13, 14, 15]. Each *channel* consists of one or more *ranks*. Each rank is a group of multiple memory *devices* that are controlled in unison; each device sends or receives data on its own dedicated data path within a rank. Each DRAM device contains multiple, typically 8 or 16 independently controlled *banks*. Each bank consists of a large number of 2D arrays of data bits called *subarrays (or tiles)* where each bit is accessed with *row* and *column* indices. Banks share the device pins and peripheral circuits. To minimize internal routing, each subarray is connected to an independent set of device output data pins (DQs). Many different module designs and interfaces are possible, including currently common DIMMs of DDR3/4 [16, 17] or directly-soldered GDDR5 or LPDDR3/4 devices [18, 19, 20] and evolving designs such as WideIO2 [21], HMC [22], and HBM1/2 [23, 24]. From the perspective of RelaxFault, all of these designs are almost equivalent because all inherently use the same device organization.

Memory Faults and Errors. A DRAM fault typically affects a particular structure within a device, and a single fault may affect a large number of DRAM locations. Prior DRAM failure field studies have explored the relation between faults and the affected memory regions, which may be sets of bits, rows, or columns within a subarray or bank, sets of subarrays within a bank or device, sets of banks within a device or rank, and so forth [1, 2, 25, 5].

Each fault can also be classified based on whether it persists across multiple accesses or not and how likely it is to be *active* on any given access; an active fault is a fault that affects the operation of the faulty component, whereas an inactive fault may exist but did not change output during a specific access. Faults that do not persist are known as *soft faults* or *transient faults*; a soft fault in DRAM is active only once and can be repaired with ECC when errors are correctable. Faults that persist and may be active at any time after they occur are known as *hard faults* or *permanent faults*. Hard faults may be further classified as *hard-intermittent* (also referred to as intermittent faults) or *hard-permanent* (also referred to as just hard or just permanent faults). Hard-

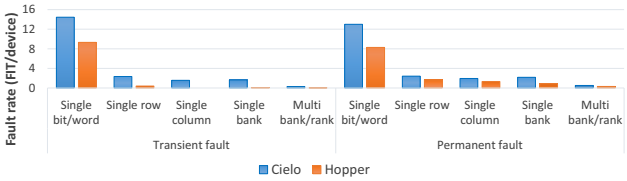


Figure 2: Fault rate of DDR3-based large scale systems [2, 5] (FIT/device). Faults are mostly occurred within a single bank and faults affecting multiple rows/columns are likely permanent.

intermittent faults are the faults that always exist after they occur, but are not always active; only impacting the operation of the faulty DRAM some of the time. Hard-permanent faults, on the other hand, impact the DRAM for the vast majority of accesses to a faulty structure. It is, unfortunately, not well known, or published to public at least, the relative rates at which hard-intermittent and hard-permanent faults occur and also the rate at which hard-intermittent faults are activated. Recently published field studies used systems with aggressive coarse-grained retirement policies, and thus removing faults from further measurement [1, 2, 5, 12].

DRAM Failure Field Study. Figure 2 represents FIT rate of two DDR3-based supercomputer systems, which are Cielo at Los Alamos National Laboratory [2, 5] and Hopper at the NERSC center at Lawrence Berkeley National Laboratory [5] respectively. These studies show that the total rate of hard faults is roughly $13-20$ FIT, and soft fault rate is similar or even smaller than that, which is roughly $10-20$ FIT. Hence, a new hard-intermittent or hard-permanent fault in a specific DDR3 DRAM device occurs roughly once every $5,700$ years of operation of a single DRAM device. While this rate sounds small, a modern large system already contains millions of DRAM devices, for example 3.6 millions of DRAM devices in Blue Waters [12], so a new fault may occur once every 3 hours or so. Studies also indicate that the activation rate of hard-intermittent faults varies over a wide range, with some faults likely to be activated only roughly once per month whereas others may occur once per hour or even more frequently. Note that even a low rate such as once a month is still many orders of magnitude greater than the expected rate of a new fault occurring on the same device with an existing hard-intermittent fault. Also a hard-permanent fault likely leads to a very high error rate if it affects a DRAM region that is accessed frequently [4].

3. RelaxFault MEMORY REPAIR

Figure 3 shows a block diagram of a system with RelaxFault. RelaxFault is a low-cost hardware-only microarchitectural fault tolerance mechanism that supports fine-grained repair—as fine as the size of a transfer block from a single memory device. RelaxFault relies on understanding how faults manifest in DRAM[1] for a unique remapping of fault addresses into the last level cache and can repair the vast majority of memory faults with negligible performance, power, and area overheads.

[1]For clarity we use the term DRAM to refer to memory that may require repair, however RelaxFault can be used with other memory types.

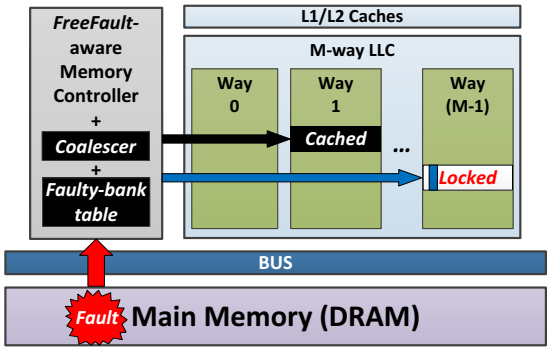


Figure 3: RelaxFault augments a FreeFault-aware memory controller with a data coalescer and a faulty-bank table. The coalescer aggregates retired data from faulty memory regions, and compacts it into a single cacheline. The faulty-bank table tracks which DRAM banks have faulty regions. The blue-colored box in LLC way $M-1$ highlights the sub-block corresponding to the requested data in the remapped cacheline.

RelaxFault shares several features with FreeFault [11]. Both mechanisms are hardware-only and use hardware to identify and track memory faults. Both also utilize lines in the LLC to store remapped data in faulty memory addresses for repair. However, while FreeFault requires a full cacheline for any memory block that has a fault, RelaxFault uses a sophisticated, yet simple to implement, address remapping scheme to dramatically reduce the number of LLC lines used for memory repair. This enables greater repair coverage and lower performance overheads. In the rest of this section we describe the modifications required to the LLC and memory controller, the RelaxFault mapping scheme, and expected overheads.

3.1 RelaxFault Cache Hierarchy

RelaxFault augments the FreeFault-aware memory controller[2] with two additional components: *coalescer* to compact repaired data into few LLC lines, and *faulty-bank table* to track the repaired faulty modules in a bank granularity.

LLC Access. With RelaxFault, the LLC holds both regular data that is accessed with physical addresses and remapped data that is accessed with the RelaxFault addressing scheme. To ensure the two types of data can be distinguished, we propose to extend the LLC tag by one bit. This bit matches the type of access requested, as shown in Figure 4. We evaluate the overhead of adding this RelaxFault flag bit in Section 3.3 and demonstrate that it is very low. However, alternative structures that do not require redesigning the tag array are possible.

Figure 5 shows how RelaxFault tests if the requested data is remapped in LLC. The faulty-bank table is a small 2D direct-access memory array with as many rows as the maximum number of modules in a socket (typically 8 [29]) and 1 bit per bank in each row. [3] Every LLC miss first examines the faulty-bank table with the module and bank IDs to

[2]RelaxFault supports multi-socket cache coherence and DMA, including I/O operations, in the same way as presented in [11].

[3]Table size grows as the number of memory modules in a socket increases, but is still always insignificant overall. For example, just 64 16-bit entries are needed for a node with 2TiB DDR4 memory.

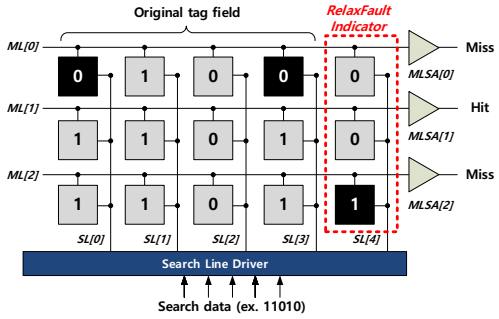


Figure 4: LLC tag array (CAM) implementation [26, 27, 28] to support RelaxFault repair (for clarity, we show a 3-way 5-bit wide tag array). The tag array is augmented with a 1-bit column, which indicates RelaxFault locked lines. The RelaxFault locked lines are misses with normal LLC access as shown in this figure, and normal LLC lines are misses when accessing RelaxFault locked LLC lines. (ML: Match Line, SL: Search Line, MLSA: Match Line Sense Amplifier)

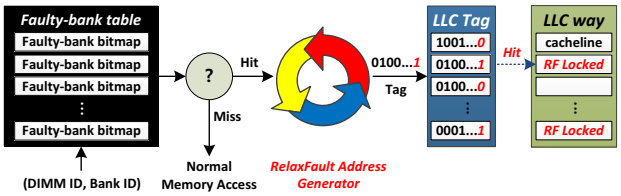
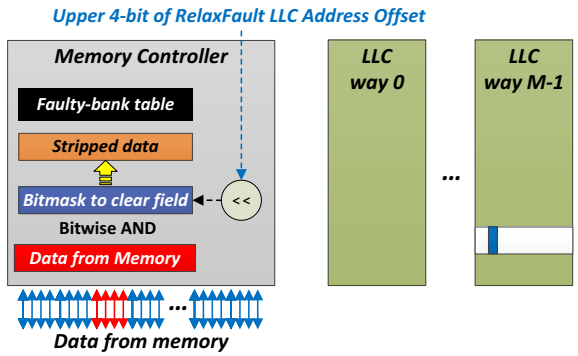


Figure 5: Working flow of testing if the requested data is remapped (repaired). When a tested bank bit in the faulty-bank table is set (hit), the address of the corresponding remapped cacheline is computed and the LLC is looked up. The LSB of the LLC tag represents the RelaxFault indicator shown in Figure 4.

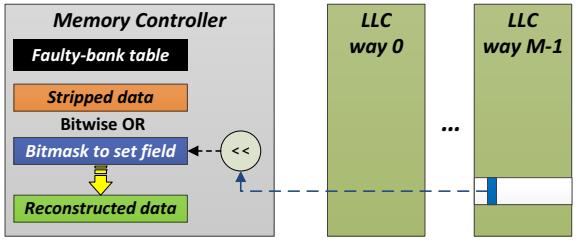
check if any locations from that module are repaired by RelaxFault. This simple filter avoids further RelaxFault tests in the vast majority of cases. If the module does have repaired locations, the corresponding remapped LLC line is located with an address computed by the address mapping discussed in Section 3.2. In the paragraphs below we discuss how the LLC is accessed, how LLC locations are allocated, and how data is read and written from and to repaired locations.

Faulty Memory Region Repair Allocation. When a memory fault is first discovered, the RelaxFault metadata is updated to perform repair. First, the RelaxFault remapping address is computed as discussed in Section 3.2. A cacheline in the LLC set corresponding to that address is evicted. That evicted line is then set to RelaxFault mode and is locked in the cache. The faulty-bank table is updated and the cache filled in the following way (assuming ECC can correct any errors).

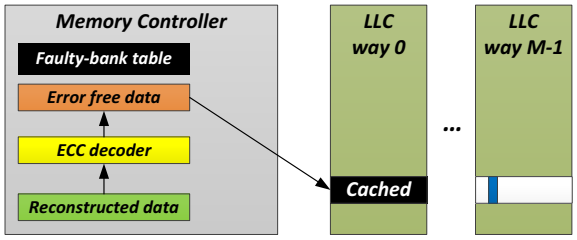
RelaxFault remaps data only from the faulty memory device. Therefore, each cacheline can store multiple remapped sub-blocks, for example, 16 of 4-byte data. RelaxFault fills all the sub-blocks into the remapped cacheline upon the first access to any data that needs to be remapped to the cacheline. To do so, the memory controller generates memory requests to those blocks and issues them back-to-back to exploit the DRAM row buffer. Note that this is only a one time cost. When the memory controller receives the requested data from memory, the sub-block from the faulty device is written to the remapped cacheline. From that time onward,



(a) Clear data received from a faulty device.



(b) Reconstruct data with cached remapped data.

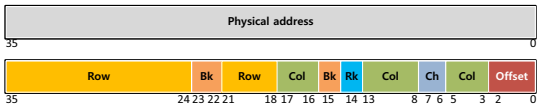


(c) Write data to caches.

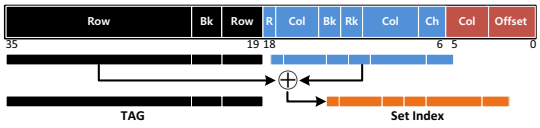
Figure 6: Memory access operation with RelaxFault repair support. The blue-colored box in LLC way $M-1$ highlights the sub-block corresponding to the requested data in the remapped cacheline.

all access to these memory locations will be proceed with RelaxFault repair, even if not all the memory locations are faulty.

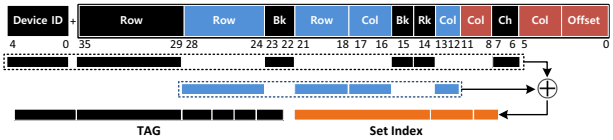
LLC Fills. When accessing data from one or more faulty DRAM devices that are already repaired by RelaxFault, data from RelaxFault and data from memory should be combined. Figure 6 illustrates the LLC fill process. On every LLC miss, a memory controller tests if the requested data *may* include remapped locations by examining faulty-bank table. If yes, the memory controller performs LLC lookup with the address translated by the RelaxFault-specific mapping scheme. If this data misses, as it should in most cases, memory access proceeds as usual. If a location is remapped, the RelaxFault line is in the LLC. The memory controller then reads the remapped data from the LLC into a buffer. At the same time the memory controller also reads the memory block from DRAM. When both the DRAM and RelaxFault cache accesses return data, the MSBs of the RelaxFault LLC address offset (bits $8-11$ in Figure 7c) are used to generate a mask for combining the data, which is then passed to the ECC unit and back to the LLC as filled data. Once cached,



(a) DRAM address mapping



(b) LLC address mapping for normal data access



(c) LLC address mapping to access remapped data by RelaxFault

Figure 7: Physical-to-cache/-DRAM address mapping with RelaxFault repair. Black colored blocks represent tag field, blue colored blocks represent non-hashed LLC set index field, and orange colored blocks represent a XOR-based hashed set index [30]. This mapping is for a system with 4 channels that each channel has 2×4 8GiB DIMMs. Each DIMM has 8 banks internally, and LLC capacity is 8MiB and is configured to 16-way and 64B line so the number of LLC sets are $8,192$.

further hits to that address do not require any RelaxFault intervention.

LLC Writebacks. When dirty data is evicted from the LLC and the evicted line includes repaired locations, the repaired locations *must* be updated with the written back data. As with a fill, faulty-bank table is queried and a masked write is performed to the RelaxFault cacheline.

Memory Controller Accesses LLC. In essence, the RelaxFault memory controller does not replace the LLC controller, but is rather an additional client of the LLC; it can: (1) query the LLC tag array to determine whether a DRAM location is faulty, (2) read and write LLC data to replace faulty DRAM locations, and (3) perform these operations in parallel to DRAM access.

3.2 Address Mapping

Modern memory controllers translate physical addresses to DRAM locations in a manner that attempts to maintain DRAM row locality while spreading accesses equally across banks to minimize conflicts. Figure 7a illustrates one such address mapping scheme, which was used in Intel's Nehalem processor [31]. We use this bit-swizzling mapping in our examples, but the techniques we describe apply equally easily to more sophisticated hash-based mapping schemes as well [32]. Because DRAM locations appear "randomized" the canonical contiguous cache tag, index, and offset physical address to cacheline mapping scheme (Figure 7b) results in most multi-bit DRAM faults being spread across numerous cache lines; each cache line contains at most 8 bits from any particular DRAM tile in current memory systems, all in a single row. While physical address to cache location mapping is likely to include some hashing as well [30, 33], this does not help the fault-spreading effect.

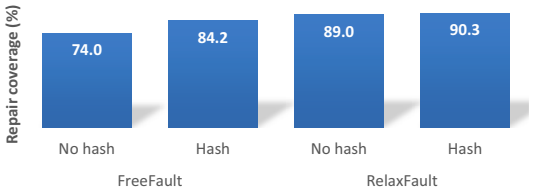


Figure 8: Cumulative repair coverage of RelaxFault and FreeFault with and without XOR-based LLC set index hashing [30] when using at most 1-way in any LLC set to repair memory. A node has 8 8GiB DIMMs and 8MiB 16-way LLC

We propose a new mapping scheme specifically for RelaxFault repair (Figure 7c), which places as many bits as possible from a single DRAM tile into a cacheline. We do this by treating each column address as referring to the data from a single memory device. We extend the cacheline offset field by 4 bits in our example system because there are 16 data devices per rank and each cacheline corresponds to a full rank access. We then differentiate cachelines that remap different devices by adding the device ID to other fields of the cache address (e.g., the tag). Note that because an ECC DIMM has 1 or 2 redundant devices, the device ID requires 5 bits rather than 4. We can, however, repurpose the valid bit (or any other state bit) and add it to the device ID. A specific memory-fault location is uniquely represented with a tag and set index. Each LLC set can store multiple remapped cachelines and each of them is easily identified by its tag value.

Coalescing remapped faulty bits in this way significantly reduces the number of cachelines dedicated for repair. This leads to much better repair coverage and lower performance and power impact. Multi-bit faults are likely to affect just one or a few full rows, a few rows in one column, or very coarse-grained structures like banks where the faults are spread out in those banks. Faults in one or a few full rows are more frequent than the other faults [1, 2, 5]. We demonstrate this in Section 5 where we compare RelaxFault to the previously-published FreeFault scheme.

Cache Set Index Hashing. Most architectural simulators use the classic canonical cache addressing scheme without XOR hashing. We evaluated the impact of adding XOR-based cache set hashing [30, 33] on the repair coverage of RelaxFault and FreeFault (Figure 7b). Before applying a hash, the RelaxFault mapping scheme already swizzles physical address bits in a way that matches the DRAM address mapping to distribute common remapped faults (which are typically limited to a single bank in a single device) across sets. Figure 8 presents the results of our evaluation. First, we observe that RelaxFault provides much better coverage than FreeFault, which we discuss in detail in Section 5. Second, spreading common fault remapping addresses by design, as we do with RelaxFault mapping is effective; the coverage with and without cache set address hashing is roughly the same. However, the fault-unaware baseline mapping of FreeFault is far more sensitive to set conflicts and XOR hashing improves coverage from 74% to 84%. This verifies the importance of hashing that was mentioned by Kim and Erez [11]. We therefore generally apply set address hashing when evaluating the repair mechanisms in detail later in this paper.

	Size (bytes)	Description
Faulty-bank table	8	1 byte per DIMM in a node
Data coalescer	128	Pre-computed bitmasks
LLC tag extension	16,384	1 bit per LLC tag
Total	16,520	

Table 1: RelaxFault storage overhead assuming 8MiB 16-way LLC with 64B block and 8 DDR3 DIMMs per node.

3.3 Overhead Estimation

Storage Overhead. While RelaxFault does not require redundant storage for remapping data, it requires dedicated storage for managing 16KiB of metadata as summarized in Table 1. We also measured the area overhead of adding the 1-bit flag to each LLC tag with CACTI 6.5 [34] for a 8MiB 16-way LLC with 64B cacheline (as our evaluation setup shown in Section 4). We did not observe any noticeable change in size.

Energy Overhead. The 8-byte faulty-bank table lookup energy is negligible and the tag lookup requires just 9pJ (1MiB 16-way LLC bank); we did not observe any noticeable change in access latency with the augmented tag array. In total, the metadata access energy, in the worst case, is less than 1.5% of the LLC access (0.641nJ) and less than 0.03% of the energy required to service the miss from DRAM (36nJ, DDR3) [35]. In the vast majority of cases (fault free accesses) just the negligible table lookup is required. The additional static energy is also negligible as RelaxFault adds 1 bit to every LLC tag and just 1 bit per DRAM bank in the faulty-bank table.

The overall memory access latency does not change because: (1) metadata and repair data are accessed in parallel to the DRAM access for both reads and writebacks, and (2) merging received data from DRAM with the remapped data from the LLC is done with one bitwise AND and one bitwise OR operations (Figure 6).

4. METHODOLOGY

The major benefit from RelaxFault repair is to provide very low-cost memory permanent fault tolerance without impacting performance and DRAM capacity. We therefore evaluate the impact of RelaxFault on three factors:

Capacity and Reliability. To represent the reliability tradeoff, we evaluate the impact RelaxFault repair has on system DUE and SDC rate. To that, we first estimate what fraction of nodes with any kinds of DRAM faults can be fully repaired by RelaxFault and how much LLC capacity is sacrificed for repair.

System Availability. We evaluate how RelaxFault repair avoids system interruptions by estimating how many DIMM replacements can be eliminated through fine-grained repair.

Performance and Power. We evaluate performance impact by measuring the throughput of an 8-core simulated processor under different amounts of LLC capacity used for repair. We also evaluate DRAM power, which may increase as more LLC cache misses may occur as a result of RelaxFault lines.

4.1 Capacity and Reliability

We use a Monte Carlo DRAM fault-injection simulator to inject faults into DRAM. Each injected fault disables one or more DRAM cells, depending on the type of fault simulated. As done in prior work [11, 36, 37, 38, 39], we simulate multiple independent Poisson processes to model different fault types, based on the types and rates observed in the field [1, 2, 5]. When faults that affect a large number of cells occur and as faults accumulate over a 6-year period, a growing fraction of the LLC of a node may be needed for RelaxFault repair and an increasing number of nodes experience some permanent fault. We report the expected fraction of nodes that experience at least one permanent fault that can be repaired by RelaxFault under varying RelaxFault LLC usage limits. We assume 8 DIMMs per node and a 16-way 8MiB LLC with 64B cache lines. Each DIMM is comprised of 18 $\times$4 DRAM devices to support chipkill-level ECC. We run 16.4 billion Monte Carlo trials for each experimental scenario. A node is considered faulty if it experiences at least one permanent fault in any of its DRAM devices. However, not all faulty nodes will manifest a DUE or an SDC because of the chipkill ECC and repair mechanisms, as discussed below.

4.1.1 Reliability and Availability Models

Every time a new fault is injected, we test whether it could lead to an SDC or DUE as done in [36]. In our model, which is based on analyzed field data, the rate of DUEs and SDCs in previously non-faulty nodes is negligible because with chipkill-level ECC, a DUE or an SDC can only occur when two devices in the same DIMM are faulty, and simultaneous random faults are extremely rare. However, once a permanent fault occurs, future faults may lead to a failure. This is the reason why repair mechanisms are so important – once a permanent fault is detected and repaired, the risk of DUEs and SDCs is significantly reduced.

We evaluate two major repair policies in this paper. The baseline policy is to use no repair until a DIMM fails (exhibits a DUE) and then repair the DIMM by replacement. The second policy is to apply RelaxFault after each permanent fault is observed to completely repair a DIMM when possible. We also evaluate a more aggressive policy in which a DIMM is replaced once it has any permanent faults, even before a DUE is observed or an SDC occurs. Under this policy, once a significant number of correctable errors are reported, a DIMM is replaced because it may reduce performance due to corrections and increase the probability of failure [12, 40]. Prior work on FreeFault [11] also considered using fine-grained repair only for those faults not corrected by ECC; for brevity, we do not discuss this option in this paper although it is equally applicable to RelaxFault as to FreeFault.

As appropriate, we report the following metrics for each policy: the fraction of faulty nodes that can be repaired with RelaxFault, the expected number of DUEs in a large system with 16K nodes, the expected number of SDCs in that system, and the expected number of replaced DIMMs. The number of replaced DIMMs serves as our availability metric because replacing a DIMM requires maintenance downtime (unless very expensive DIMM sparing is used).

Fault mode	Transient fault	Permanent fault
Single bit	14.5	13.0
Single row	2.3	2.4
Single column	1.6	1.9
Single bank	1.6	2.2
Multiple banks	0.1	0.3
Multiple ranks	0.2	0.2

Table 2: Fault rate of studied DDR3-based system (FIT/device) [2, 5]. As we also measured system reliability with scaled FIT rate, we also use 10 times FIT of them.

4.1.2 *Fault Model*

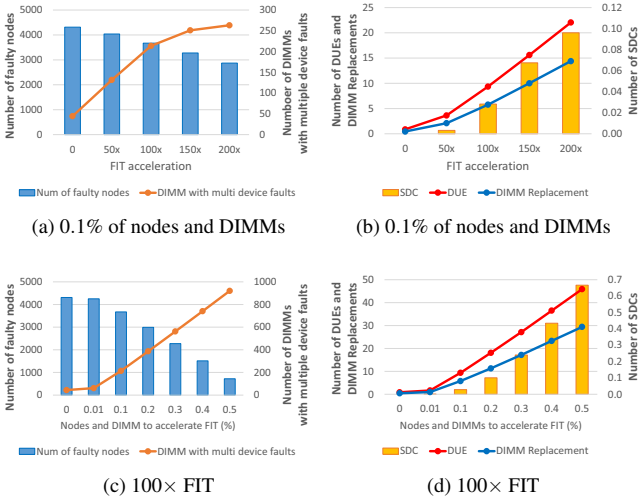
We develop a new fault model that augments prior DRAM fault models [1, 2, 11, 36] with device-to-device and module-to-module variations. The basis of the model is prior work that analyzed the faults observed in large-scale field studies, categorized the faults into fault modes or processes, and provided expected rates for those faults [1, 2, 5]. The modes and rates we use as our basis are summarized in Table 2 and were discussed in Section 2 and closely resemble those reported for the Cielo system [2, 5]. We only present results based on the rates of the Cielo system and we confirmed (but do not show) that applying rates from other reported systems has little impact on RelaxFault results.

It is unrealistic to expect that every DRAM device and every DRAM module exhibits identical statistics because both devices and modules are subject to both process and operating variations. To model device-to-device variation we randomly select a different rate for each Poisson fault process and each device. The rate is a Lognormal random variable with a mean equal to the reported expected rate shown in Table 2 and a variance that is $\frac{1}{4}$ of the mean (we experimented with different coefficient of variance values and the results are not sensitive to this parameter).

The motivation for our module-to-module variation model stems from observations pointing to some correlations between faults within modules or nodes and the appearance of DUEs without fault accumulation [4, 3, 1]. Such correlated faults can occur because some modules may be of lower quality, be composed of lower quality devices, or experience more detrimental environmental conditions (such as higher temperature and humidity and larger temperature swings). In our model, we randomly select a small fraction of nodes and modules and then (uniformly) accelerate the fault rates of those modules. We maintain the same overall fault statistics by lowering the rates of all other devices to compensate for the ones we accelerated. This is shown in Equation (1) where N stands for nodes, D stands for modules (DIMMs), and P_N and P_D represent the fractions of fault-accelerated nodes and modules respectively.

$$FIT = P_N \times FIT_{acc,N} + P_D \times FIT_{acc,D} + (1 - P_N - P_D) \times FIT_{adj,REST} \quad (1)$$

To decide on what fractions of nodes to accelerate and to what degree, we performed a sensitivity study and show the results in Figure 9. Figure 9a shows that if the fraction of accelerated nodes is fixed, the number of DIMMs that can have an SDC or a DUE rises roughly linearly until an acceleration factor of 100 and then slows down. The number of faulty nodes trends down because the more some modules are accelerated, the lower the fault rate of other nodes



(a) 0.1% of nodes and DIMMs (b) 0.1% of nodes and DIMMs

(c) 100× FIT (d) 100× FIT

Figure 9: Dynamic FIT adjustment sensitivity in 6 years. Each system has 16,384 nodes that each has 8 DIMMs, and a DIMM is replaced only when a permanent fault generates a DUE. When accelerating FIT of a fixed percentage of nodes and DIMMs (0.1%) more, (a) number of nodes with faulty DIMMs, and (b) the number of DUEs, SDCs, and DIMM replacements (b) are measured. When changing ratio of FIT accelerated nodes and DIMMs with fixed amount of acceleration (100×), (c) number of nodes with faulty DIMMs, and (d) the number of DUEs, SDCs, and DIMM replacements are measured.

becomes in order to maintain a constant average rate. Figure 9b shows that the expected number of SDCs, DUEs, and replaced DIMMs grows roughly linearly with acceleration rate. Because the knee of the curves is roughly at 100× acceleration, we choose this value for our model. The effect of increasing the fraction of accelerated nodes is roughly linear on all parameters as shown in Figure 9c and Figure 9d. We arbitrarily chose a fraction of 0.1%, which when using the 100× acceleration factor results in a 20% rate reduction for non-accelerated devices. Note that the expected number of DUEs is larger than the expected number of replacements because transient faults may cause DUEs but do not require replacement in this evaluation.

We compare our results both with this model and when using a model that does not include variation. In each graph, the left-most point (0-acceleration) corresponds to the previous model that is based purely on the fault rates reported in [2, 5]. With such a model, the expected number of DUEs is just 1 out of 131,072 DIMMs over 6 years of operation. This is far too low a rate compared to the observed rate of DIMM replacement (and DUEs) in real systems (e.g., study of Blue Waters [12]). With our proposed model, the number of DUEs is much more realistic. Our model also reasonably matches the relative probability of DUEs given existing faults presented in [1]. We believe this model offers a good first step toward a more refined model, but leave that to future work that can focus entirely on analyzing data from large-scale field studies.

4.2 Performance and Power Simulation

To understand the expected performance impact of using the LLC for RelaxFault repair, we measure execution throughput with MacSim [43], a cycle-based x86 proces-

Processor	8-core, 4GHz, x86 ISA, 4-way OOO
L1 I-caches	32KiB, private, 8-way, 64B cache line, 1-cycle,
L1 D-caches	32KiB, private, 8-way, 64B cache line, 3-cycle
L2 caches	128KiB, private, 8-way, 64B cache line, 8-cycle
L3 caches	8MiB shared, 16-way, 64B cache line, 30-cycle
Memory controller	FR-FCFS scheduling [41], open page policy channel/rank/bank interleaving, bank XOR hashing [32]
Main memory	2 channels, 2 ranks / channel, 8 banks / rank All parameters from the Micron DDR3-1600 [42]

Table 3: Simulated system parameters

Workload	Benchmark	Description
NAS parallel bench	CG (C)	Conjugate Gradient
	DC (A)	Data Cube
	LU (C)	Lower-Upper Gauss-Seidel solver
	SP (C)	Scalar Penta-diagonal solver
	UA (C)	Unstructured Adaptive mesh
Co-design	LULESH (30^3)	Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics
SPEC CPU 2006	MEM	429.mcf, 433.milc, 450.soplex, 462.libquantum, 470.lbm, 437.leslie3d, 471.omnetpp
	COMP	429.mcf, 433.milc, 450.soplex, 462.libquantum, 470.lbm, 401.bzip2, 458.sjeng

Table 4: Workloads description. Input sets are specified in parentheses.

sor simulator. We configure the simulator with dual DDR3 memory channels. Table 3 summarizes the system parameters for our performance evaluation. In this evaluation, we reduce cache capacity by randomly locking up to one way in each LLC set if a required size is specified, or a fixed number of ways per set when specified in *N-way*. As explained in Section 3, our proposed address mapping scheme distributes remapped data well across LLC sets. With RelaxFault, there is never a need for more than 4 ways in any set, and typically just one or zero ways are locked, so our methodology is realistic and even pessimistic.

Performance. We use Weighted Speedup (WS) [44] as defined by Equation (2) for our performance metric. IPC_{orig}^{alone} is the IPC when an application is run alone in the evaluated system with no LLC retirement. $IPC_{new}^{reduced}$ is the IPC when running with other applications if any sharing LLC with various LLC capacity reduction to locate retired data.

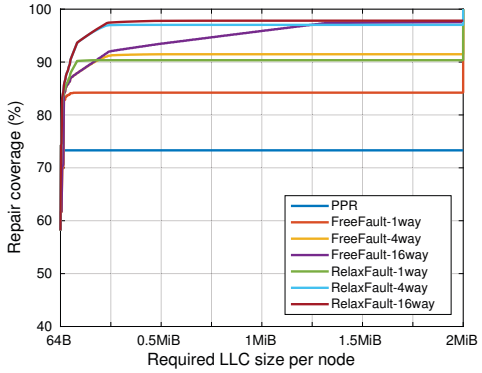
$$WS = \sum_{i=0}^{N-1} \frac{IPC_{i,new}^{reduced}}{IPC_{i,org}^{alone}} \quad (2)$$

Power. We estimate DRAM power[4] with the number of different DRAM operations (activate, precharge, read, and write) performed and the energy associated with each operation as detailed by Micron [45]. Note that DRAM power is sufficient for our evaluation because, as we show in our results, the performance impact of RelaxFault is very small.

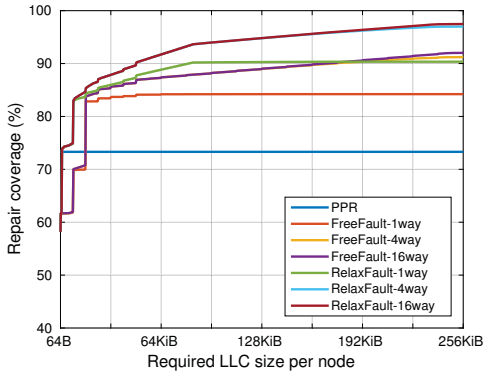
Workloads. For workloads we use multi-threaded HPC-oriented benchmarks [46, 47, 48, 49], and multi-programmed SPEC CPU2006 workloads [50]. Because we are interested in the interactions with memory, we choose mostly memory-intensive benchmarks from each benchmark suite. In SPEC CPU2006, we also run compute-intensive workloads and memory-intensive workloads simultaneously to measure the impact of RelaxFault in various scenarios. Table 4 summarize the benchmarks we use. In SPEC CPU2006 benchmark, *MEM* has only memory-intensive workloads while *COMP* includes compute-intensive workloads. We ran various combinations of benchmarks in each category, but we represent the results as harmonic mean because each result shows similar behavior.

In each simulation, every application begins executing at a SimPoint [51] and simulation of all applications continues until the slowest application (or core/thread in multi-threaded workloads) completes 200M instructions. We collect statistics from each application or core only for its first 200M instructions, but keep executing all applications or cores to correctly simulate contention for shared resources.

[4]As discussed in Section 3.3, repair increases the LLC energy by a very small amount, but additional LLC misses caused by reduced LLC capacity can be significant. Therefore, we estimate the energy overhead of RelaxFault by counting the additional DRAM accesses.



(a) Memory fault repair coverage.



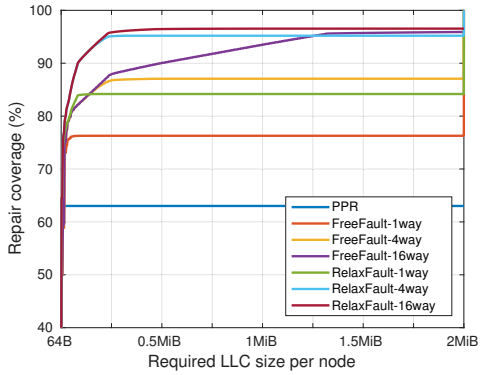
(b) Memory fault repair coverage (zoomed).

Figure 10: Cumulative repair coverage vs. required LLC capacity for full repair with no impact on reliability or underlying ECC corrections for baseline FIT rate. The fraction of nodes having any retired data is 12% **after** 6 **years.**

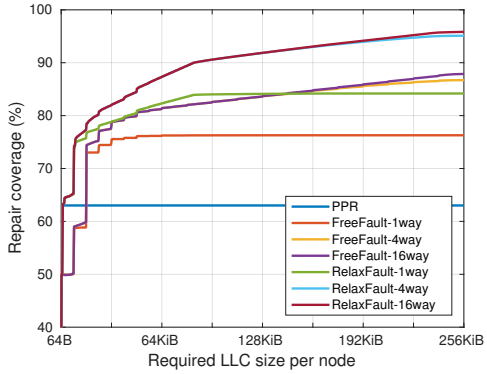
5. EVALUATION RESULTS

5.1 Reliability and Availability

Figure 10 depicts the percentage of nodes that have at least one permanent fault which can be seamlessly repaired by retiring faulty memory regions with RelaxFault, the previously published FreeFault, and the *post-package repair*



(a) Memory fault repair coverage

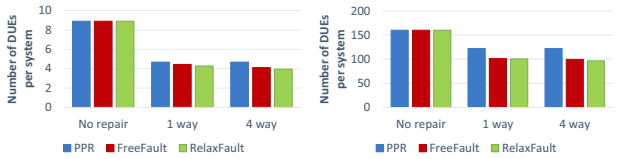


(b) Memory fault repair coverage (zoomed)

Figure 11: Cumulative repair coverage vs. required LLC capacity for full repair with no impact on reliability or regular ECC corrections for $10\times$ FIT. The fractions of nodes having any retired data is 71% in 6 years.

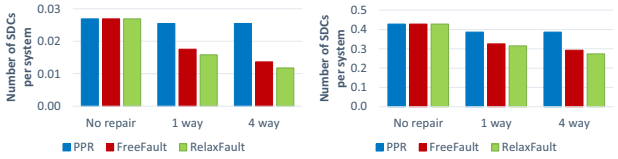
(PPR) mechanism introduced in the JEDEC DDR4 specification [17]; PPR allows up to one row per bank group to be repaired (see more in Section 6). When limiting the number of LLC ways that RelaxFault and FreeFault use to repair memory to at most 1 of 16 ways in any set, RelaxFault achieves 90% repair coverage compared to FreeFault that repairs up to 84% with cacheline address hashing or 74% without hashing. PPR also achieves roughly 73% repair coverage. RelaxFault is superior because it requires fewer cache lines and distributes those lines better across sets such that the one way per set limit does not affect RelaxFault coverage. In fact, RelaxFault with one way or fewer is equivalent to the coverage of FreeFault with up to 4 ways and even exceeds FreeFault that requires up to all 16 ways in some sets when total LLC capacity used is limited to under 128KiB. Thus, RelaxFault provides both higher coverage and has a smaller (negligible) impact on performance.

If RelaxFault is allowed to occupy up to 4 ways at most in some set, then repair coverage increases to almost 97%, still requiring about 256KiB of LLC capacity. FreeFault, in contrast, requires up to 16 sets to be used and over 1MiB of the cache. The 3% of cases, which cannot be repaired by RelaxFault and FreeFault in our evaluation are those that correspond to massive faults that affect entire banks in DRAM, though it is likely that such faults are still covered well by



(a) $1\times$ FIT (b) $10\times$ FIT

Figure 12: Expected number of DUEs when using either RelaxFault, FreeFault, or PPR in a system with 16,384 nodes (8 ×4 DIMMs per node) over 6 years.



(a) $1\times$ FIT (b) $10\times$ FIT

Figure 13: Expected number of SDCs when using either RelaxFault, FreeFault, or PPR in a system with 16,384 nodes (8 ×4 DIMMs per node) over 6 years.

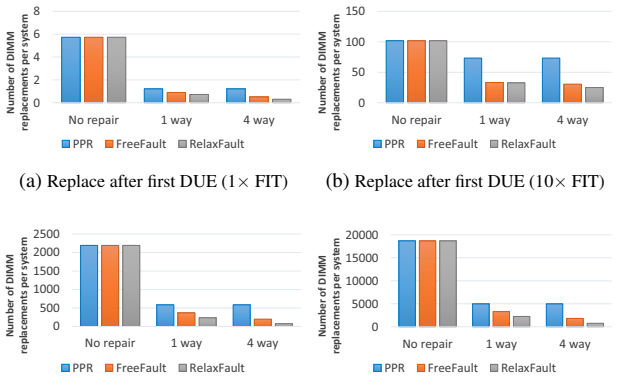
chipkill-level ECC, as we discuss in the next subsection.

We also perform the same evaluation with a $10\times$ higher fault rate (Figure 11) to gauge the effectiveness of RelaxFault if future technologies are less reliable. Despite the much higher fault rate, RelaxFault is almost as effective. Repair coverage with at most one way per set drops from 90% to 84%, but if up to 4 ways can be used then repair coverage is still above 95% while overall capacity used is still less than 256KiB. The trends with FreeFault are similar, but PPR drops to a much lower coverage of 63%.

5.1.1 DUEs and SDCs

Large scale and mission-critical systems typically employ strong error checking and protection schemes (e.g., ECC) that can tolerate a single faulty DRAM device per rank and provide high detection coverage against silent data corruption (SDC) by reporting nearly all errors originating from more than a single device as detectable uncorrectable errors (DUEs). However, once a permanent fault exists in a rank, there is a risk of SDC and the likelihood of observing a DUE goes up significantly. Repair can help curb this risk. We evaluate the expected number of DUEs and SDCs in the system following the methodology described by Kim et al. [36].

Figure 12 shows the expected number of DUEs with the various repair mechanisms with both the nominal and $10\times$ higher fault rates. All three repair mechanisms reduce the DUE rate by about 50% at the nominal fault rate and RelaxFault is most effective at 52%. The $10\times$ higher fault rate makes a very large difference in expected DUE count. At the higher rate, RelaxFault and FreeFault show significantly larger reduction than PPR with RelaxFault decreasing expected DUE count by 37%. Interestingly, DUE reduction is not sensitive to the number of ways allowed for repair. The reason is that for a DUE to occur, a single ECC codeword must include more than one faulty symbol. With current codes, this means that two different devices must exhibit faults within a single 64B memory block. This is only likely



(a) Replace after first DUE (1× FIT) (b) Replace after first DUE (10× FIT)

(c) Replace after frequent errors (1× FIT) (d) Replace after frequent errors (10× FIT)

Figure 14: Expected number of DIMM replacements when using RelaxFault, FreeFault, or PPR in a system with 16,384 nodes (8 ×4 DIMMs per node) over 6 years; a DIMM is replaced when either a DUE occurs ($Repl_A$) or the number of errors exceeds a threshold ($Repl_B$).

if at least one of the devices has a "coarse-grained" fault that affects a very large number of bits. Such faults cannot be repaired by any of the fine-grained repair mechanisms, regardless of how many ways are allowed for use. DUEs are reduced because most faults affect only a single bit or a small number of bits. These faults can be repaired even with only a single way.

Figure 13 shows similar results for expected SDC count (the expected count is less than 1 because SDCs are very rare). The exception is that PPR is ineffective at reducing SDCs despite its success at reducing DUEs. The reason is that while a common cause of a DUE is at least one device with a small number of faulty bits that PPR can repair, SDCs are more common when one device has multiple fine-grained faults that exceed the capabilities of PPR but not those of RelaxFault and FreeFault.

5.1.2 *Availability*

Fine-grained repair can be used as an alternative to replacing DIMMs. As mentioned earlier, ECC alone cannot guarantee error-free operation unless faulty memory is repaired or retired. Replacing faulty DIMMs often requires system interruption for maintenance. While it is possible to have seamless repair [52, 53], the high costs and overheads associated with sparing modules and mirroring channels is too high for most systems. We evaluate two repair by replacement policies. The first replaces a DIMM immediately after a non-transient DUE is observed ($Repl_A$) [1, 4]. The second replaces a DIMM after the number of corrected errors from that DIMM exceeds a given threshold within a certain time window ($Repl_B$) [12, 40]. The motivation for this aggressive replacement policy is that frequent errors indicate a permanent fault has occurred, at which point reliability is degraded.

Fine-grained repair masks out the data from permanently faulty regions, eliminating the possibility of many errors and decreasing replacement rate. Figure 14 shows the expected number of DIMM replacements with both policies with and without memory repair mechanisms. There are five main takeaways from the results. First, all three fine-grained repair mechanisms significantly reduce the replacement rate. However, RelaxFault is particularly effective and if allowed to use at most 4 cache ways in any set, reduces the replacement rate by more than 10×, compared to about 4× if using only PPR. Second, increasing the resources available for repair (RelaxFault vs. FreeFault and 4- vs. 1-way) has a big impact on the fraction of DUEs reduced because more it repairs less likely multiple erroneous symbols are in a codeword. Third, because fine-grained repair is effective, it increases maintenance window flexibility and improves availability. Fourth, the aggressive replacement policy is very aggressive. It results in almost 350× more replacements than if replacing only after a DUE is observed. Fifth, referring to the reliability results discussed above, this aggressive repair is not well justified with current fault rates given the very low SDC rates and quite-low DUE rate.

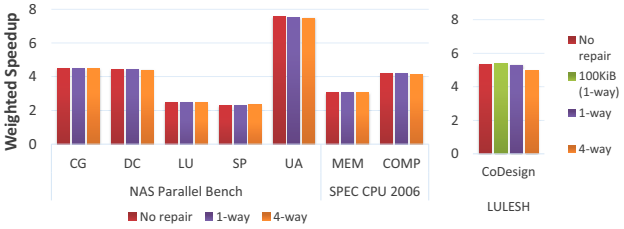


Figure 15: Performance comparison with various LLC capacity dedicated to RelaxFault in a system with the configuration shown in Table 3. Performance is measured in Weighted Speedup.

5.2 Performance and Power

RelaxFault utilizes the LLC for storing remapped memory blocks for repair and may increase LLC misses and thus impact performance and power consumption. We measure the system throughput (weighted speedup) and the power overhead associated with extra misses as different LLC capacities are dedicated for RelaxFault repair (Figure 15). In a real system, only a small number of lines will mostly be used by RelaxFault. However, to avoid costly Monte Carlo simulation, we instead make entire ways unavailable for regular data and assume they are dedicated to RelaxFault. We find that LULESH is the only benchmark we evaluated that shows any perceptible sensitivity to even removing four ways of the LLC (7% ↓).[5] We study LULESH in more detail and performed a Monte Carlo experiment with 32 trials; each trial was a simulation of LULESH with 100KiB randomly assigned to RelaxFault (we did not observe more than one used way in any set). Figure 15 shows the worst of those 25 trials and the performance impact is negligible.

Figure 16 illustrates the impact of RelaxFault on DRAM dynamic power consumption, with and without RelaxFault repair. Unsurprisingly, the power results track the performance results: 4-way RelaxFault repair increases DRAM dynamic power consumption in some applications (LULESH and NPB DC), but 100KiB RelaxFault repair has nearly zero impact on power consumption. Note that Figure 16 only shows DRAM dynamic power, which is roughly

[5]We confirm the observations of prior work, which demonstrate that applications are not sensitive to a small reduction in capacity of a highly-associative LLC [11, 54, 55].

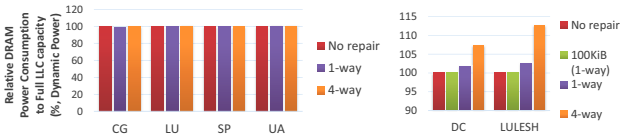


Figure 16: Relative DRAM dynamic power consumption to full LLC capacity (no repair with RelaxFault) of multi-threaded workloads in a system with the configuration shown in Table 3.

50% of total DRAM power consumption [35] and is also a fraction of system power [56]. Combined with performance results, energy consumption is also mostly unaffected by 1-way RelaxFault repair.

6. RELATED WORK

As permanent faults are more frequent than transient faults [1, 2, 5] and permanent faults lead to more frequent error events [3, 4], retiring or even replacing faulty components becomes an interesting addition to ECC. The simplest type of retirement is to retire an entire node, and this approach can be feasible only when faults lead to this is low enough, which, unfortunately, would not be true in large scale systems as shown in several system failure field studies [1, 2, 5, 3, 4, 12, 40]. This approach may not even be possible as some systems rely on the availability of certain specific nodes. Retiring memory channel or rank is also possible but it is still expensive because of reduced capacity, which often severely affects performance and power in HPC systems in particular. Interesting DIMM replacement approach is channel mirroring enabling hot-replacement of faulty memory module, but it is possible at the cost of reduced capacity and possibly bandwidth by a half [52, 53].

OS or a device driver also can retire faulty memory regions by using page-based virtual memory support, which unmaps affected memory frames from the free frame list, and examples are IBM's AIX OS [52, 53], Oracle's Solaris OS [57], and Nvidia's GPU memory management [58, 59]. This approach is conceptually simple, but due to mismatch between physical address and DRAM address retiring small physical memory region would require retiring a large number of frames that each has only a small faulty region [32], and using huge page also makes this approach more expensive. Some OS components and peripheral devices do not use virtual memory facilities [60], so this approach is sometimes unavailable. For example, IBM's AIX limits address space supported by frame retirement [52, 53]. Additionally, recent work related to very high-capacity memory management proposed to avoid page-based memory management due to its higher cost but using direct segmentation [61], which make page-based retirement approach unfeasible. Such a system often runs application that designed to fit into physical memory size, so in that case, reducing DRAM capacity becomes critical while RelaxFault remains effective.

Instead of retiring node or rank, a faulty DRAM device itself would be retired without capacity reduction by exploiting that ECC protection uses redundant devices. Bit-steering approach of IBM's Memory ProteXion scheme of X-Series server [62, 52, 53] and recently proposed *Bamboo ECC* [36] that use remained bit in a redundant device is one example, and another example is Intel's *Double Device Data Correction* (DDDC) that tolerates a device failure with lowering ECC capability to *Single Device Data Correction* (SDDC) level [63]. The downside of these approaches is to degrade resilience against further faults, and sometimes requires coupling multiple channels [63, 64].

The fine grained retirement is supported by adding redundant storage to remap data from faulty region. Row and column sparing with memory array [65] is a classic example which is only available during manufacture/test/integration phase. Recent LPDDR4 and DDR4 standards begin to support in-field row sparing using the eFuse technology while its capability is limited to 1 per bank (LPDDR4) or bank group (DDR4), and once used repair becomes permanent. As an alternative, external structure can be added to DRAM die or package as a spare location to remap data from faulty memory blocks [66]. Recently proposed ArchShield [6] uses similar approach, but using DRAM array itself as a remapping storage with reduced effective DRAM capacity. These approaches often statically add redundant storage, so considering low fault rate per device the overhead becomes severe.

Lastly, recently proposed FreeFault enables fine grained retirement without adding redundant storage, but with microarchitectural structure [11]. When necessary, FreeFault repurposes a small fraction of the processor's LLC to remap data from a faulty DRAM region, improving reliability with nearly zero impact on performance. Because the processor already queries its LLC before accessing DRAM, the mechanism is transparent during execution. However, as discussed in Section 5 repair coverage of FreeFault highly depends on address mapping scheme, and rather expensive LLC resources are often wasted, which may have diminishing return. RelaxFault improves this inefficiency with only a minor modification of repair mechanism, and is independent to underlying LLC address mapping scheme to achieve higher recovery with even less LLC capacity reduction and virtually zero performance impact.

7. CONCLUSION

We present a low-cost microarchitectural memory repair mechanism, *RelaxFault* , which effectively relaxes the requirements on other fault tolerant mechanisms to improve both reliability and performance. We show that RelaxFault can eliminate the vast majority of permanent faults and is more effective than other low-cost fine-grained repair schemes. RelaxFault relies on a novel addressing scheme to remap faulty memory bits into a small fraction of the processor LLC. We conclude that because it coalesces faults from a single device into a single cacheline, its impact on performance, power, and cost is negligible. Our evaluation introduces a new fault-injection methodology that results in faults that better predict system failure rates, and we use it to show the benefits RelaxFault provides, especially for curbing module replacement and maintenance to boost system availability.

Acknowledgment

The authors acknowledge the Texas Advanced Computing Center for providing HPC resources and the support of the

Department of Energy under contract LLNS B609478 and the National Science Foundation under Grant #0954107, which partially funded this research.

References

[1] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, IEEE, 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 the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, ACM, 2013.

[3] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic Rays Don'T Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, ACM, 2012.

[4] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-scale Field Study," in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, ACM, 2009.

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

[6] 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*, ACM, 2013.

[7] S. Hong, "Memory technology trend and future challenges," in *2010 International Electron Devices Meeting*, 2010.

[8] K. Kim, "Future memory technology: challenges and opportunities," in *International Symposium on VLSI Technology, Systems and Applications*, IEEE, 2008.

[9] Tech Insights, "Technology Roadmap of DRAM for Three Major manufacturers: Samsung, SK-Hynix and Micron." `http://www.techinsights.com/uploadedFiles/Public_Website/Content_-_Primary/Marketing/2013/DRAM_Roadmap/Report/TechInsights-DRAM-ROADMAP-052013-LONG-version.pdf`, 2013.

[10] J. H. Yoon, H. C. Hunter, and G. A. Tressler, "Flash & DRAM Si Scaling Challenges, Emerging Non-Volatile Memory Technology Enablement - Implications to Enterprise Storage and Server Compute systems," *Flash Memory Summit*, 2013.

[11] D. W. Kim and M. Erez, "Balancing reliability, cost, and performance tradeoffs with FreeFault," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture*, IEEE, 2015.

[12] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of Blue Waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2014.

[13] B. L. Jacob, "Synchronous DRAM Architectures, Organizations, and Alternative Technologies," *University of Maryland*, 2002.

[14] R. J. Baker, *CMOS: Circuit design, layout, and simulation*, vol. 18. Wiley-IEEE Press, 2011.

[15] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*, vol. 13. Wiley-IEEE Press, 2007.

[16] JEDEC, "JESD79-3F DDR3 SDRAM Standard," 2010.

[17] JEDEC, "JESD79-4 DDR4 SDRAM," 2012.

[18] JEDEC, "JESD212A GDDR5 SGRAM," 2013.

[19] JEDEC, "JESD209-3 Low Power Double Data Rate 3 (LPDDR2)," 2012.

[20] JEDEC, "JESD209-4 Low Power Double Data Rate 4 (LPDDR4)," 2014.

[21] JEDEC, "JESD229-2 Wide I/O 2(WideIO2)," 2014.

[22] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1," 2014.

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

[24] JEDEC, "JESD235A High Bandwidth Memory (HBM) DRAM," 2015.

[25] X. Jian, S. Blanchard, N. Debardeleben, V. Sridharan, and R. Kumar, "Reliability Models for Double Chipkill Detect/Correct Memory Systems," in *SELSE*, 2013.

[26] K. J. Schultz and P. G. Gulak, "Throttled-buffer ATM switch output control circuitry with CAM-based multicast support," in *1997 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, IEEE, 1997.

[27] K. J. Schultz and G. Gulak, "CAM-based single-chip shared buffer ATM switch," in *Communications, 1994. ICC'94, SUPERCOMM/ICC'94, Conference Record,'Serving Humanity Through Communications.'IEEE International Conference on*, IEEE, 1994.

[28] K. Pagiamtzis and A. Sheikholeslami, "Using cache to reduce power in content-addressable memories (CAMs)," in *Proceedings of the IEEE 2005 Custom Integrated Circuits Conference*, IEEE, 2005.

[29] Intel, "Intel Xeon Processor E7 Familiy." `http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family.html`, 2015.

[30] A. González, M. Valero, N. Topham, and J. M. Parcerisa, "Eliminating cache conflict misses through XOR-based placement functions," in *Proceedings of the 11th international conference on Supercomputing*, ACM, 1997.

[31] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, ACM, 2012.

[32] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, ACM, 2000.

[33] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters," in *Research in Attacks, Intrusions, and Defenses*, pp. 48–65, Springer, 2015.

[34] S. J. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[35] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards Energy-proportional Datacenter Memory with Mobile DRAM," in *Proceedings of the 39th Annual International Symposium on*

Computer Architecture, 2012.

[36] J. Kim, M. Sullivan, and M. Erez, "Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture*, IEEE, 2015.

[37] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2012.

[38] J. A. Maestro and P. Reviriego, "Reliability of single-error correction protected memories," *IEEE Transactions on Reliability*, vol. 58, no. 1, pp. 193–201, 2009.

[39] A. Bacchini, M. Rovatti, G. Furano, and M. Ottavi, "Characterization of data retention faults in DRAM devices," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, IEEE, 2014.

[40] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2015.

[41] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, IEEE, 2000.

[42] Micron Corp., *Micron 4 Gb* ×4*,* ×8*,* ×16*, DDR3 SDRAM: MT41J1GM4, MT41J512M8, and MT41J256M16*, 2011.

[43] HPArch, "MacSim." `http://code.google.com/p/macsim/`.

[44] A. Snavely and D. M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, ACM, 2000.

[45] Micron, "Calculating Memory System Power for DDR3," Tech. Rep. TN-41-01, Micron Technology, 2007.

[46] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS parallel benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[47] H. Feng, R. F. Van der Wijngaart, R. Biswas, and C. Mavriplis, "Unstructured Adaptive (UA) NAS Parallel Benchmark, Version 1.0," *NASA Technical Report NAS-04*, vol. 6, 2004.

[48] M. A. Frumkin and L. Shabanov, "Arithmetic data cube as a data intensive benchmark," *National Aeronautics and Space Administration*, 2003.

[49] Lawrence Livermore National Lab, "Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory," Tech. Rep. LLNL-TR-490254.

[50] Standard Performance Evaluation Corporation, "SPEC CPU 2006." `http://www.spec.org/cpu2006/`, 2006.

[51] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for Accurate and Efficient Simulation," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, ACM, 2003.

[52] D. Henderson, B. Warner, and J. Mitchell, "IBM Power Systems: Designed for Availability," Tech. Rep. POW03020-USEN-01, 2009.

[53] IBM, "IBM System x3850 X6 and x3950 X6 Planning and Implementation Guide." `http://www.redbooks.ibm.com/redbooks/pdfs/sg248208.pdf`, 2014.

[54] J. Albericio, P. Ibáñez, V. Viñals, and J. M. Llabería, "The reuse cache: downsizing the shared last-level cache," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2013.

[55] D. Zhan, H. Jiang, and S. C. Seth, "STEM: Spatiotemporal management of capacity for intra-core last level caches," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 2010.

[56] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[57] D. Tang, P. Carruthers, Z. Totari, and M. W. Shapiro, "Assessment of the effect of memory page retirement on system RAS against hardware faults," in *International Conference on Dependable Systems and Networks (DSN'06)*, IEEE, 2006.

[58] Nvidia, "Dynamic Page Retirement." `http://docs.nvidia.com/deploy/dynamic-page-retirement/index.html`, 2015.

[59] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell, "Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, ACM, 2015.

[60] K. B. Ferreira, K. Pedretti, R. Brightwell, P. G. Bridges, D. Fiala, and F. Mueller, "Evaluating operating system vulnerability to memory errors," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ACM, 2012.

[61] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ACM, 2013.

[62] M. T. Chapman, "Introducing IBM Enterprise X-Architecture Technology." IBM Corporation White Paper, August 2001.

[63] Intel, "Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability." `http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-e7-family-ras-server-paper.pdf`, 2011.

[64] FUJITSU, "FUJITSU Server PRIMERGY & PRIMEQUEST Memory performance of Xeon E7-8800 / 4800 v2 (Ivy Bridge-EX) based systems." `http://globalsp.ts.fujitsu.com/dmsp/Publications/public/wp-ivy-bridge-ex-memory-performance-ww-en.pdf`, 2014.

[65] H. L. Kalter, C. H. Stapper, J. E. Barth Jr, J. DiLorenzo, C. E. Drake, J. Fifield, G. Kelley, S. C. Lewis, W. B. Van Der Hoeven, J. Yankosky, *et al.*, "A 50-ns 16-Mb DRAM with a 10-ns data rate and on-chip ECC," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1118–1128, 1990.

[66] A. Tanabe, T. Takeshima, H. Koike, Y. Aimoto, M. Takada, T. Ishijima, N. Kasai, H. Hada, K. Shibahara, T. Kunio, T. Tanigawa, T. Saeki, M. Sakao, H. Miyamoto, H. Nozue, S. Ohya, T. Murotani, K. Koyama, and T. Okuda, "A 30-ns 64-Mb DRAM with built-in self-test and self-repair function," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1525–1533, 1992.