Balancing Reliability, Cost, and Performance Tradeoffs with FreeFault

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

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

***Abstract*—Memory errors have been a major source of system failures and fault rates may rise even further as memory continues to scale. This increasing fault rate, especially when combined with advent of integrated on-package memories, may exceed the capabilities of traditional fault tolerance mechanisms or significantly increase their overhead. In this paper, we present FreeFault as a hardware-only, transparent, and nearly-free resilience mechanism that is implemented entirely within a processor and can tolerate the majority of DRAM faults. FreeFault repurposes portions of the last-level cache for storing retired memory regions and augments a hardware memory scrubber to monitor memory health and aid retirement decisions. Because it relies on existing structures (cache associativity) for retirement/remapping type repair, FreeFault has essentially no hardware overhead. Because it requires a very modest portion of the cache (as small as 8KB) to cover a large fraction of DRAM faults, FreeFault has almost no impact on performance. We explain how FreeFault adds an attractive layer in an overall resilience scheme of highly-reliable and highly-available systems by delaying, and even entirely avoiding, calling upon software to make tradeoff decisions between memory capacity, performance, and reliability.**

I. Introduction

Memory reliability has been a major design constraint for mission-critical and large-scale systems for many years. Despite this history, two important trends highlight the need for continued innovation in addressing this important problem. The first trend is the increasing severity of memory faults and error rates, with the rates of memory errors that affect a large number of bits and permanent faults being on par with the rates of single-bit errors and transient faults, respectively [55, 56, 22, 53]. The second trend, which follows from the first, is that once a permanent fault occurs, a decision needs to be made about trading off cost, performance, and reliability. The two main reasons a tradeoff is necessary are that: (1) a permanent fault is likely to result in numerous erroneous accesses, each requiring possibly high correction overhead; and (2) 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 – stronger codes can tolerate more errors, but have higher overhead.

The straightforward solution to addressing this issue of repeated costly corrections and reduced coverage is to replace faulty memory devices, however, doing so is expensive, especially when the faulty memory device is integrated with a processor [23]. An economical alternative is to retire and possibly remap just the faulty memory regions. There are three broad categories for retirement techniques (we refine this classification and provide more detail in Section VI): (1) retire memory and reduce available memory capacity (e.g., node-level, channel-level, or OS frame level), which is costly in lost resources and difficult and complicated to implement in some systems [12]; (2) retire a faulty chip in a memory module and either change protection level or compensate protection by coupling memory channels, which then impacts performance and power [9, 18, 25, 28]; and (3) remap faulty addresses to alternative memory locations, which currently requires redundant memory and adds latency and complexity for remapping hardware [58, 45, 44, 38].

In summary, previously-proposed retirement techniques are able to better balance reliability and cost than naive solutions, but are not ideal. These existing techniques either require sophisticated software support, impact capacity, reliability, and/or performance, or introduce additional storage and hardware structures.

We introduce *FreeFault* , which is a hardware-only microarchitecture that can retire memory at a very fine granularity with no software support, without impacting reliability, and with minimal impact on cost, performance, and energy efficiency. Instead of adding hardware, FreeFault uses the existing last-level cache (LLC) and binds a small amount of retired memory to locked lines in the LLC. In this way, we achieve near-free repair because we repurpose existing microarchitecture mechanisms while introducing no additional memory access latency for retired memory. Our evaluation demonstrates that the vast majority of faults can be repaired with only a small fraction of LLC capacity sacrificed for remapping retired memory leading to negligible performance impact. Hence, trading off existing SRAM for faulty DRAM is an attractive additional mechanism in the memory resilience toolkit; FreeFault complements other techniques with negligible hardware cost.

An important advantage of FreeFault is that it requires no software intervention and can be dynamically applied to a running system. To enable this truly hardware-only operation, we propose and evaluate a hardware-only mechanism for deciding when and which memory to retire. Given the low

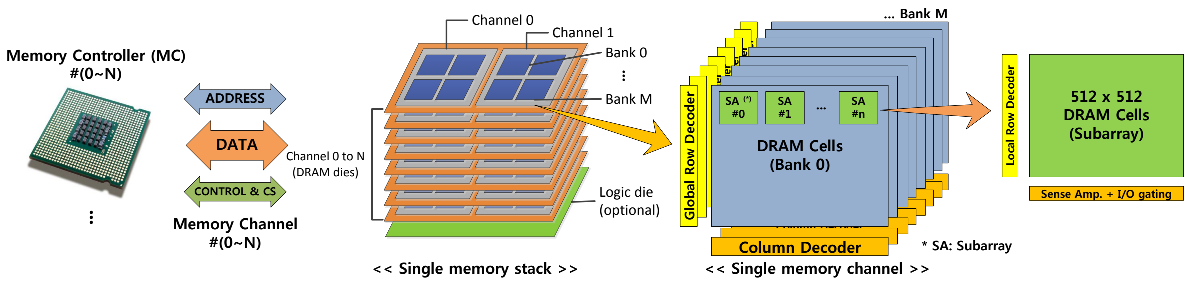


Fig. 1: A simplified DRAM system structure. 3D stacked memory [23, 19] is shown as an example, but currently popular DIMMs have an almost same structure while replacing a stack with a memory module grouped multiple DRAM devices horizontally.

performance impact, no reliability impact, and hardware-only fine-grained repair, we discuss how FreeFault fits within a broader resilience and high-availability scheme and serves to handle the vast majority of faults transparently and cheaply before requiring higher-level mechanisms to intervene.

To restate the main contributions of FreeFault:

- FreeFault is able to transparently hide (without impacting reliability or performance) the majority of expected memory faults in both integrated on-package and traditional off-package memory (8 8-layer stacks per package and 8 8-chip DIMMs per socket). About 60% of sockets with faults can be transparently repaired with no measurable impact on performance.
- FreeFault is able to tradeoff minor performance degradation for increased repair coverage. Nearly 90% of on-package faults and 85% of off-package faults can be hidden entirely in the microarchitecture of a running system with only a 1% average performance degradation (8.2% maximum) observed in our experiments.
- FreeFault can be implemented with negligible changes to the microarchitecture (depending on how coherence is implemented) and be dynamically applied to a running system. This is in contrast to alternatives that add hardware, increase memory latency, and/or require significant changes to system software and system operation.
- FreeFault may include a low-cost microarchitectural mechanism for identifying and tracking retirement candidates with no software support. This structure also leverages the LLC and adds no additional storage on the processor; it may be assisted by a small amount (< 68KB) of BIOS-managed FLASH memory.
- FreeFault complements other resilience scheme and reduces the reliance on frequent ECC corrections that reduce performance, pushes out the need for making reliability and capacity tradeoffs required when retiring memory devices or channels, and eliminates the requirements for software modifications of frame-level retirement.

The rest of this paper is organized as follows: Section II provides background on memory system organization and ECC protection; Section III details the FreeFault architecture and associated tradeoffs; Section IV describes our evaluation methodology, including DRAM fault modeling; Section V presents our evaluation results; Section VI discusses current retirement schemes and other related work; and Section VII concludes the paper.

II. Background

A. DRAM Fundamentals

Figure 1 shows a simplified view of a typical DRAM system organization [31] with an emphasis toward 3D stacked memories [31, 23, 19]. The processor connects to memory via, possibly multiple, *channels*, each of which is attached to a dedicated memory controller. Each channel typically includes independent control and data paths, with control information consisting of select, command, address, and mask signals. Each channel consists of multiple memory *devices* (DRAM chips or dies), which are grouped within each channel into *ranks*. All ranks share the control and data buses of the channel, such that only single rank is selected for control and data transfer at any given time. Different commands can be sent to the different ranks by interleaving them in time. All devices within a rank are controlled in unison with each device sending or receiving data on its own dedicated data path within the rank.

Internally, each DRAM device contains multiple independently-controlled *banks*. Banks share the device pins and are controlled and access the data path in a time interleaved manner. Physically, each bank is comprised of shared peripheral circuits and an array of bits, which is divided into a large number of *subarrays* (e.g., $\sim 1,024$ subarrays in a 2Gb DRAM device). Each subarray is a collection of, typically, 512×512 bits, which made *rows* and *columns* respectively, with a set of wordline and bitline drivers and sense amplifiers [6, 37]. The subarrays are needed to constrain the capacity of the wires and thus improve efficiency and latency. The address decoders within each bank occupy significant area and are hierarchical; part of the address decoding is done at the bank level with additional decoding performed at the subarray level. To minimize internal routing, each subarray is connected to an independent set of device (chip) output data pins, and multiple of those pins are used for each transfer – the device interface width, which is typically 4–128 bits wide. Within a bank, a column is the minimum addressable granularity in each DRAM device and is equal to the interface width multiplied by the DRAM *burst length*; because the channel links run at a higher frequency than the DRAM core, a single column specifies a sequence of transfers across the interface. A row is a sequence of columns within a bank.

Many different module designs and interfaces are possible, including currently common dual in-line memory modules (DIMMs) of DDR3 [32] or directly-soldered GDDR5 or LPDDR2 devices [35, 33] and evolving designs, such as

WideIO, HMC, and HBM as shown in Figure 1 [34, 19, 23]. From the perspective of FreeFault, all of these designs are equivalent because all inherently use the same device organization, and FreeFault manages memory vulnerability based on the microarchitectural property, a LLC line size.

B. Memory Faults and Errors

A typical DRAM fault affects a particular structure within DRAM and a single fault may therefore affect a potentially large number of DRAM locations. Prior work has explored the relation between faults and the memory locations they affect, 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 [55, 56, 36]. For the purposes of this paper, it is enough to understand two aspects related to the structures affected by a fault. The first aspect is that each fault impacts either one cacheline-sized contiguous region of physical memory addresses, a small number of such regions, or is likely to affect a significantly large memory region. The second aspect is that some faults lead to errors that can be corrected by the processor's ECC mechanisms (Section II-C) while others may be uncorrectable or even escape detection. We discuss these two aspects further in Section III-B.

A particular fault can also be classified based on whether it persists across multiple accesses or not and the likelihood of it being *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 impact operation during a specific access. Faults that do not persist are known as *soft faults* (also referred to as transient faults); a soft fault in DRAM is active exactly once and can be repaired with ECC protection. Faults that persist and may be active at any time after they occur are known as *hard faults* (also referred to as persistent 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-intermittent faults are faults that are always present (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. There is little data and analysis of the relative rates at which hard-intermittent and hard-permanent faults occur and also of the rate at which hard-intermittent faults are activated. The reason is that the large-scale field studies required for this analysis are rarely conducted or published and those that have been recently published used systems with aggressive coarse-grained retirement policies, removing faults from further measurement [55, 56].

The analysis of Sridharan and Liberty [55] shows that the total rate of hard faults is roughly 50 FIT (soft faults are less frequent and result in an error rate of roughly 20 FIT). Hence, a new hard-intermittent or hard-permanent fault occurring in a specific DRAM chip in DDR2 or DDR3 technology is roughly once every $2,500$ years of operation of a single DRAM device. While this rate sounds small, a large system may contain millions of DRAM devices and a new fault may occur once every 5 hours or so. Their analysis also indicates that the activation rate of hard-intermittent faults varies over a wide

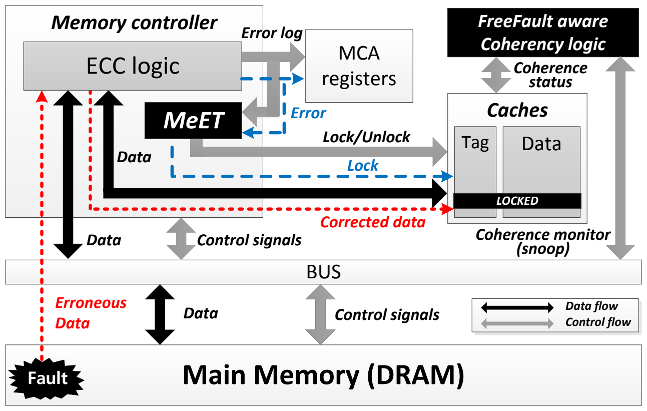


Fig. 2: FreeFault augments memory controller with MeET, coherency logic, and scrubber (not shown in this figure). MeET collects error events by referencing the existing machine check architecture (MCA) registers to decide DRAM data block retirement. MeET also can invoke hardware scrubber to determine a fault mode. Modified cache coherence logic prevents from evicting locked cacheline, and enables valid data always resides in at least one of LLCs in a system (mostly in a locked cacheline, but not necessarily). Dashed lines show the data/control flow when the loaded data has correctable errors.

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 of one activation per 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 note that a hard-permanent error may lead to a very high error rate if it affects a region or component of DRAM that is accessed frequently [53].

C. Memory Protection and ECC

Resilience is of greater concern as technology scales and system component-count grows. Unprotected memories are one of the major sources of system failure and large-scale systems therefore utilize strong ECC mechanisms such as *Redundant Bit-Steering*, *Chipkill-correct* (IBM) [18, 25, 10, 24], *Extended ECC* (Oracle) [46], *Chipspare* (HP) [20], and *Single Device Data Correction (SDDC)* or *Single/Double Device Data Correction (SDDC/DDDC)* (Intel/AMD) [28, 1]. However, accessing memory data that requires correction with those strong ECCs incurs power and latency overhead [2, 40]. Furthermore, it is even harder to implement such ECC techniques in recently-introduced wide I/O memories including 3D stacked memories because of higher cost to add redundant devices, wider interfaces, and new fault modes.

III. FreeFault Memory Repair

FreeFault follows the straightforward design and operation of a retire/remap repair mechanism, but uniquely utilizes microarchitectural features that already exist in current high-end CPUs. Figure 2 illustrates how the main components of FreeFault are organized and used to repair faulty memory (DRAM) locations:[1]

(1) First, each memory access determines whether it should use DRAM or has been retired and remapped to alternative storage.

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

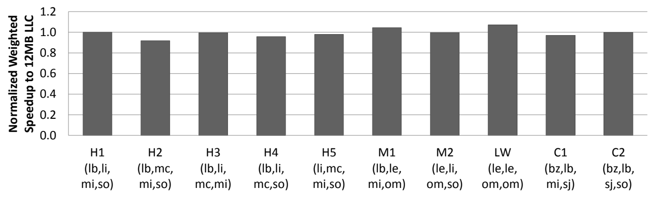


Fig. 3: Performance vs. LLC size on real systems: two Nehalem processors, an Intel Xeon E5520 [26] and an E5620 [27], which have 8MB and 12MB LLCs respectively. Both processors have 4 cores, running at 2.26/2.4GHz of baseline frequency and 2.4/2.53GHz of boosted frequency for the E5520 and the E5620 respectively. Considering E5620 runs in 5–6% higher frequency, performance impact from reduced LLC capacity by 33% is very small. Workload notations follow Table IV but with 4 concurrent benchmarks (one per core).

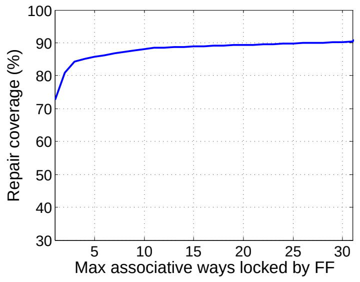


Fig. 4: Fault coverage vs. the maximum number of ways locked in any set in the cache. The number of ways locked varies across sets (and runs). The x axis represents the number in the set with the most locked ways for achieving the repair coverage reported on the y axis. The results are for the system described in Section IV with 1x FIT rate after 6 years of operation; with full repair of the 128GB on-package DRAM, 73% of faults are repaired with no more than a single way locked in any set. (FF: FreeFault)

(2) Second, retired addresses query a remapping table to locate their data for completing the memory requests from the processor. Alternatively, addresses that are not retired proceed to request data directly from DRAM. Note that in the FreeFault design, these first two steps are combined as part of a regular LLC lookup.
(3) Third, in addition to the regular error-handling flow of the processor (e.g., correction with an ECC code), requests from DRAM that report an error are registered with a mechanism that identifies candidates for repair.
(4) Fourth, once a DRAM location is determined to have a permanent fault, it is either retired and remapped to alternative storage or the fault is exposed to system software for other types of more coarse-grained retirement or repair.

Figure 2 also shows how FreeFault realizes this flow with a very low-cost microarchitecture. The alternative storage, retirement check, and remapping are all performed within an essentially unmodified LLC of a processor cache hierarchy (III-A). A retired DRAM location is simply always an LLC hit whereas an unretired DRAM location uses the cache normally and may either hit in the cache or miss and access DRAM. We propose to implement a hardware memory error tracker (MeET) that can track memory errors and identify DRAM regions that are candidates for replacement without relying on system software (Section III-B). At the end of this section we show that the hardware overheads of FreeFault are negligible and evaluate FreeFault's impact on performance and reliability in later sections.

In this work, we argue that counter-intuitively, trading off a fraction of the LLC for avoiding DRAM faults provides a beneficial overall tradeoff. This is because FreeFault enhances resilience with minimum cost by utilizing already available SRAM (LLC) resources and microarchitecture components instead of adding hardware. Because FreeFault helps lower the strength of required ECC, it also provides an interesting tradeoff between reduced LLC capacity and achievable performance benefit. We do not claim that this solution is appropriate in all scenarios and that reducing LLC capacity by a significant amount always has a small impact on performance. However, our experiments confirm results reported by others that large LLCs are not utilized very efficiently and performance is not very sensitive to small relative capacity reduction [4, 62] (Figure 3).

More importantly, as discussed in Section V, we demonstrate that the vast majority of processors that actually experience faults only require a small number of LLC cache lines for repair and the impact on performance with highly-associative LLCs is quite small.

A. *FreeFault Cache Hierarchy*

FreeFault leverages the associative mechanisms of the cache hierarchy, which already map DRAM addresses to alternative storage for performance, for the resilience mechanism of retirement-based repair. When a DRAM region is identified for retirement (Section III-B), FreeFault locks the cache lines associated with the retired physical addresses in the LLC. Caches of high-end processors typically already support locking lines within the cache hierarchy using cache-control instructions in the ISA [29]. FreeFault can even potentially use these same instructions, or simple variants of them, from within firmware or microcode. The modifications needed for FreeFault are associated with preventing the eviction of FreeFault-locked lines, the handling of DMA operations, uncached accesses, and other aspects of coherence, and maintaining the retirement information across reboots. Note that FreeFault can be used online without software support and no system-software or application modifications are necessary.

Locked cache lines. A DRAM region that has been retired with FreeFault must *never* be evicted from the cache hierarchy (with the exception of shifting to another coarser-grained resilience technique, which must involve software). Therefore, care must be taken to: (1) not allow any software to unlock a block that represents retired DRAM because FreeFault locking impacts correctness and is not a performance optimization; (2) not evict locked lines for coherence and DMA accesses (discussed in the next subsections).

The first constraint of not allowing software eviction can be easily achieved by adding a state to each cacheline tag that indicates whether FreeFault or software locked the line. This requires at most one additional bit in each tag and can also be achieved with a small bloom filter that tracks cachelines locked by FreeFault. We propose to use an even lower-cost mechanism by designating a subset of cache ways to not allow software locking/unlocking. The cache ways would not be dedicated to retirement, which significantly decreases cache capacity; rather the way would be dedicated to locking control by FreeFault to satisfy the strong locking requirement. We expect retired lines to be balanced well across sets because only a fraction of the LLC is used by FreeFault (see Section V), because independent fault locations are random, and because XOR-based address hashing is usually used in the memory system [63, 16] (Figure 4).

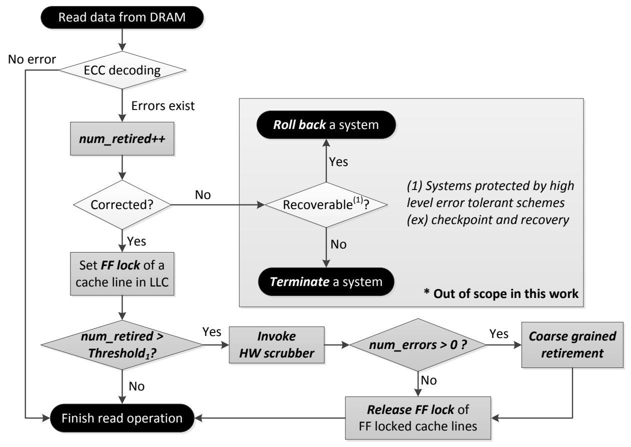
Coherence. Hardware cache coherence may need to be modified in a very small way. FreeFault requires a retired line to always be in *some* cache, thus the primary objective is not to allow eviction of FreeFault-locked cachelines. In a single socket multi-processor in which all cores share a single LLC, this is trivial. Multi-socket multiprocessors, however, require a minor modification to the coherence protocol. Similarly to previously proposed coherence protocols supporting locked cachelines [49, 8, 17], FreeFault adds two coherence states, which are *Locked Invalid* and *Locked Valid*. These additional states are represented with the existing coherence state bits and the additional *lock* flag bit.

When a processor (CPU_A) has a *Locked Valid* cacheline corresponding to faulty memory under its control (accessed through its socket) receives an invalidation signal (or snoop request) from another processor (CPU_B), CPU_A invalidates the data but maintains the locked flag, thus putting the cacheline in the *Locked invalid* state (instead of the conventional *Invalid* state). The replacement policy is then modified to not evict locked lines even when they are invalid. Data can now be supplied by the cache of CPU_B or CPU_A recaptures the data into its cache on a writeback from CPU_B. Thus, the retired memory address is always serviced within the cache hierarchy.

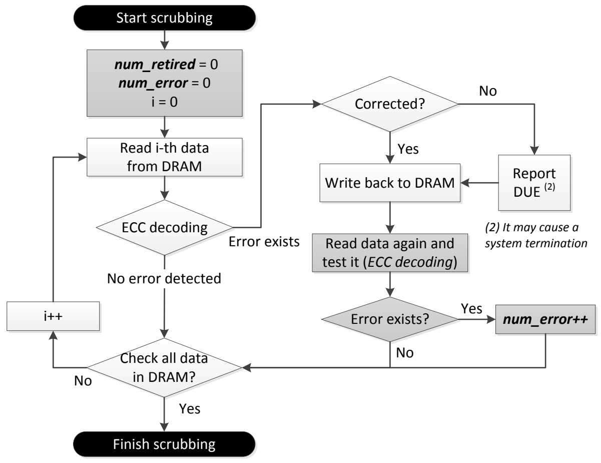
DMA and Uncached Accesses. DMA read and write requests must be satisfied by the locked line in the cache rather than memory. Because DMA engines also use the data bus/interconnect to exchange data between different modules, the LLC monitors the data bus/interconnect and provides the data if it has the up-to-date or locked data (either by snooping at the cache or the directory slice). This is similar to the cacheline invalidation for external memory update mechanism presented by Rahman et al. [49]. Depending on the processor implementation, this may in fact already be how DMA requests are handled. Even if DMAs are handled through invalidation, the modification to this snooping mechanism are straightforward.

In some systems, it is possible to make uncached accesses that bypass the cache hierarchy. While uncached accesses may have multiple uses, they are fundamentally similar to DMA accesses in that they originate from clients on the bus/interconnection network. Hence, uncached accesses can be handled using the same mechanisms and protocols as described above for DMA accesses.

Maintaining Retirement Information. Ideally, the information on which memory regions require locking will be stored in non-volatile memory available to the FreeFault firmware. When a DRAM region is first retired, FreeFault would register the region to a retirement table in the non-volatile memory. Based on the analysis presented later, we expect this table to have at most 16K entries, each storing a retired physical cache line address (roughly 34 bits per entry in today's systems). Current firmware typically contains more FLASH memory in the chipset than the required 68KB. For example, a typical *Unified Extensible Firmware Interface (UEFI)* chipset has 8MB of FLASH [60].


Read data from DRAM
No error
ECC decoding
Errors exist
num_retired++
Corrected?
No
Yes
Set FF lock of a cache line in LLC
Roll back a system
Yes
Recoverable(1)?
No
Terminate a system
(1) Systems protected by high level error tolerant schemes (ex) checkpoint and recovery
* Out of scope in this work
num_retired > Threshold1?
Yes
Invoke HW scrubber
num_errors > 0 ?
Yes
Coarse grained retirement
No
No
Release FF lock of FF locked cache lines
Finish read operation

(a) MeET data retirement flow



(b) FreeFault aware hardware scrubber working flow

Fig. 5: FreeFault operation flow. When ECC detects errors, the cacheline associated with the error is marked as *FreeFault (FF)* locked. If error events happens more than some threshold count within a single scrubbing period, MeET invokes the hardware scrubber to detect if a coarse-grained fault occurred. After scrubbing, *FF* locked lines may be unlocked if MeET decides a soft fault was likely.

If non-volatile storage is not available, it is possible to allow cache lines to experience an error and be dynamically "re-retired" by FreeFault. This re-retirement may be done during the boot sequence or even at runtime. During boot, a memory test can be run. If such a test is too expensive, faults can be dynamically identified as errors reoccur. Although these approaches avoid non-volatile storage, there is a risk that an uncorrectable error that could have been prevented with retirement from a persistent table during boot will occur.

B. *Memory Error Tracking*

A key aspect of retirement-based repair is identifying DRAM regions that are candidates for retirement. We design a hardware memory error tracker (MeET) based on the insights about memory faults summarized in Section II-B. Specifically, we account for the fact that per-DRAM chip fault rates are very low, the fact that most faults affect a fairly small number of cacheline-sized regions of DRAM, and the fact that many hard-intermittent errors are difficult to differentiate from soft errors. While we can design fairly complex hardware mechanisms that can differentiate between soft and hard-intermittent errors with high likelihood, we argue below that such a distinction will rarely, if at all, impact the benefits of FreeFault. Instead, the goals of MeET are to enable policies for managing reliability/performance tradeoffs with a very rough estimate of fault type. Figure 5 illustrates how MeET achieves this goal.

As with the retirement/repair mechanism of FreeFault, MeET utilizes and slightly augments structures that already exist within the memory hierarchy, specifically, the LLC and the hardware *memory scrubber* [51]. The scrubber periodically (typically once every few hours) cycles through all of memory performing ECC corrections of errors caused by soft faults and possibly by infrequent hard-intermittent faults. MeET augments the scrubber to monitor the severity of faults in DRAM locations retired with FreeFault and uses the LLC to track fault state.

MeET interacts with the processor's reliability infrastructure to identify erroneous transfers from DRAM. When an error is detected, MeET decides whether to retire and repair the corresponding DRAM location based on the state and number of previously retired DRAM locations. The baseline policy is to aggressively retire DRAM as long as performance impact is small. This is because, when an error is first detected, it is impossible to determine whether it is a result of a soft fault or an infrequently-activated hard-intermittent fault. Because the fault rate per memory module is so low, MeET must only worry about coarse-grained faults that impact a large number of DRAM locations that span many cachelines.

There are two main tradeoffs associated with reducing the capacity needed for FreeFault repair. The first has to do with whether faults are soft and will not reoccur, or hard and require mitigation. The second aspect is whether ECC protection should be relied upon for correction, and retirement only performed for faults potentially leading to uncorrectable errors.

While coarse-grained hard faults are indeed problematic, we wish to avoid our aggressive retirement policy from requiring software intervention when a coarse-grained *soft* fault occurs. In such cases, retired lines can be reused and aggressive retirement was unnecessary. We observe that all retirement events associated with a soft fault must occur within a single scrubbing interval. This is because scrubbing essentially repairs all soft faults.[2] Therefore, we track the number of retirement events within a single scrubbing period. Once this number exceeds a *coarse-grained threshold* we seek to determine if a coarse-grained soft fault occurred. To do this, we invoke the hardware scrubber immediately in a mode that attempts to do a quick read-correct-write-read test. If no errors are reported during this test, the fault was very likely a soft fault. We then unretire the DRAM locations that have been retired in the now-previous scrubbing interval.

In keeping with FreeFault low-cost approach, we propose to identify those lines retired in the current/previous scrub interval using the same approach we use to differentiate between FreeFault-locked lines and software-locked lines. We reserve two ways for locking only by FreeFault (disallow software locking) instead of just one. When a line is first locked, it is placed into Way_1 indicating it was locked in the current scrubbing interval. After MeET confirms retirement during scrubbing, retired lines are moved to Way_0.

Because it is possible the errors were due to a hard-intermittent fault, we also record the coarse-grained fault event with a sticky bit associated with the memory module. If a second coarse-grained event is observed in the same module we conservatively assume that both events were due to the same coarse-grained fault. Using the Birthday Paradox, we estimate the rate of two coarse-grained soft faults occurring at the same module to be less than once every 1 year per million DRAM devices. Note that invoking the scrubber in the proposed way will introduce a multi-second disruption to memory traffic on the memory channel, but such coarse-grained faults are rare and are expected to occur not more than once every several weeks on even the largest installations that have millions of memory devices.

Higher-level support for FreeFault. As FreeFault retires cachelines, performance degradation may become unacceptable. While we discuss the microarchitecture aspects of FreeFault, FreeFault is not intended as hardware-only solution for all fault types. Instead, we offer a new tradeoff opportunity to reduce the need for software approaches (or retirement) for the vast majority of processors. We argue that FreeFault can be used along with SW approach to enhance efficiency.

While coarse-grained events may lead to many retirements within a single scrubbing interval, coarse-grained hard-intermittent faults may also create new cacheline-sized retirement over a long period of time. This accumulating effect may also lead to performance degradation if FreeFault locks a large enough capacity of the LLC. In such cases another potential tradeoff is between reliability and coverage. FreeFault may be configured to only retire DRAM locations once they lead to an uncorrected error. Thus, less severe faults will not consume LLC capacity and will instead rely on ECC for correction. However, this may increase the number of ECC correction events reducing performance in a different way or degrade reliability because protection capability is compromised by the faulty component. Thus, this configuration must be done carefully and only for installations that can tolerate this loss of reliability and that may require coarser-grained retirement because of performance degradation.

IV. Methodology

The primary objective of FreeFault is to provide very low-overhead resilience while simultaneously maintaining performance, DRAM capacity, and DRAM reliability. We therefore evaluate the impact of FreeFault on three factors:

DRAM Capacity. We evaluate the impact of FreeFault repair by estimating what fraction of nodes with DRAM faults can be repaired with different LLC volumes locked for repair.

DRAM Reliability. To represent the reliability tradeoff, we evaluate the impact on DRAM capacity of repairing any fault or just those faults that cannot be corrected by ECC; the impact on performance and power can then be derived.

Performance/Power. We evaluate performance by measuring the throughput of an 8-core simulated processor under different assumptions of how much LLC capacity is locked for FreeFault

[2]Some rare soft faults may affect DRAM control state that persists across scrub periods, however, we assume such faults are very rare and that it is therefore acceptable to conservatively treat them as hard faults.

repair. We also evaluate DRAM power, which may increase if more LLC cache misses occur.

A. Capacity and Reliability

We use a Monte Carlo fault-injection simulator of a processing node to estimate the impact on DRAM capacity and reliability. We use a set of independent fault processes, with each fault type on each module and device following an independent *Poisson process*. We detail the fault processes and their associated rates below. As faults accumulate over a period of time (we simulate 1-year and 6-year scenarios), a greater fraction of SRAM capacity must be dedicated for FreeFault repair as more DRAM requires retirement. We report the expected fraction of faulty nodes that can be handled with varying FreeFault LLC usage. Note that all the figures are normalized to just those nodes that experienced any faults. We assume 8 stacks (8 dies or devices per stack) of on-package memory and 8 DIMMs (each has 8 or more DRAM devices depends on ECC support) of off-package memory and present results of the two configurations separately. We run 200 million Monte Carlo experiments per scenario.

Reliability. For evaluating reliability, we explore two different retirement policies: retire on any fault and retire only on faults that can lead to uncorrectable errors. Our baseline ECC is capable of correcting errors that impact an entire device in the off-package scenario (i.e., chipkill [10]) and an entire sub-array in the on-package scenario (i.e., subarray-kill ECC [15]). For each module in the system at each point in time in simulation, we decide which memory regions may result in uncorrectable errors and report only those as requiring retirement. Note that this methodology is likely conservative, because many such faults are unlikely to interact in a way that causes uncorrectable errors.

Fault Model. For off-package memory, we use the fault model presented by Sridharan and Liberty [55], which is based on a large-scale empirical study of DDR2 memories. We do not use the model for DDR3 presented more recently [56] because the earlier model provides additional detail that we use to determine potentially uncorrectable errors. We also provide results based on a $10\times$ higher fault rate, which we use as a proxy for potentially much higher future error rates. While there is no model that we are aware of for future DRAM technology fault rates, there is consensus that DRAM fault rates are much more strongly related to the number of DRAM dies (chips/devices) than to the number of DRAM bits [55, 56]. Table I summarizes the fault rate used in this work.

We are also unaware of a detailed model of faults in on-package memories. We believe the best estimate is the same model described above because DRAM faults have been shown to be strongly correlated to DRAM-die counts and not strongly correlated with DRAM technology nodes. To account for uncertainty, we use the $10\times$ greater fault-rate model as well. While better models should be developed, we believe our conclusions are unlikely to fundamentally change with greater model fidelity.

B. Performance and Power Simulation

To understand the expected performance impact of using the LLC for FreeFault repair, we measure execution

Fault mode	Fault rate (1x)	Fault rate (10x)
Single-bit	18.6	186
Single-row	8.2	82
Single-column	5.6	56
Single-bank	10	100
Multiple-bank	1.4	14

TABLE I: Permanent fault rate of each DRAM device (FIT/device) [55]

Processor	8-core, single-threaded, 4GHz, x86 ISA, 4-way out-of-order
L1 I-caches	32KB, private, 8-way, 64B cache line, 1-cycle, 5 fetch cycle
L1 D-caches	32KB, private, 8-way, 64B cache line, 3-cycle
L2 caches	128KB, private for instruction and data, 8-way 64B cache line, 8-cycle
L3 caches	8MB shared, 32-way, 64B cache line, 30-cycle
Memory controller	FR-FCFS scheduling [50], open page policy channel/rank/bank interleaving bank XOR hashing [63]
Main memory	2 channels, 2 ranks / channel, 8 banks / rank All parameters from the Micron DDR3-1600 datasheet [43]

TABLE II: Simulated system parameters

throughput with MacSim [21], a cycle-based x86 processor simulator. While FreeFault targets both on-package and off-package memory, the impact of lower LLC capacity is likely to be greater with slower off-package memory. We therefore configure the simulator with a dual DDR3 memory channels. Table II summarizes the baseline system parameters for our performance evaluation. We explicitly mention changes to the baseline configuration when reporting sensitivity study results. We reduce cache capacity by randomly locking up to one way in each LLC set, or same number of ways per set when requires reducing more than one way per set. As explained in Section III, the existing XOR-based physical address interleaving balances faults across sets and our random selection is realistic [16]. While we did not run a large number of Monte Carlo experiments, we did verify that our random selection did not result in outliers with respect to access frequency.

Performance. We use Weighted Speedup (WS) [54] as defined by Equation (1) as our performance metric. IPC_{orig}^{alone} is the IPC when an application is run alone in the baseline system with no LLC retirement. $IPC_{new}^{reduced}$ is the IPC when running with other applications sharing LLC with various LLC capacity reduction to locate retired data. We use *misses per kilo-instruction* (MPKI) and DRAM power to analyze impact on the memory hierarchy.

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

Power. We estimate DRAM power with the number of different DRAM operations (activate, precharge, read, and write) performed and the energy associated with each operation as detailed by Micron [42]. Note that DRAM power is sufficient for our evaluation because, as we show in our results, the performance impact of FreeFault is very small.

Workloads. For our workloads we use both multi-threaded HPC-oriented benchmarks [5, 11, 13, 39] and multi-programmed SPEC CPU2006-based workloads [57]. Because we are concerned with the interactions with memory, we choose mostly memory intensive benchmarks among the SPEC CPU2006 benchmark suite. We also run compute-intensive

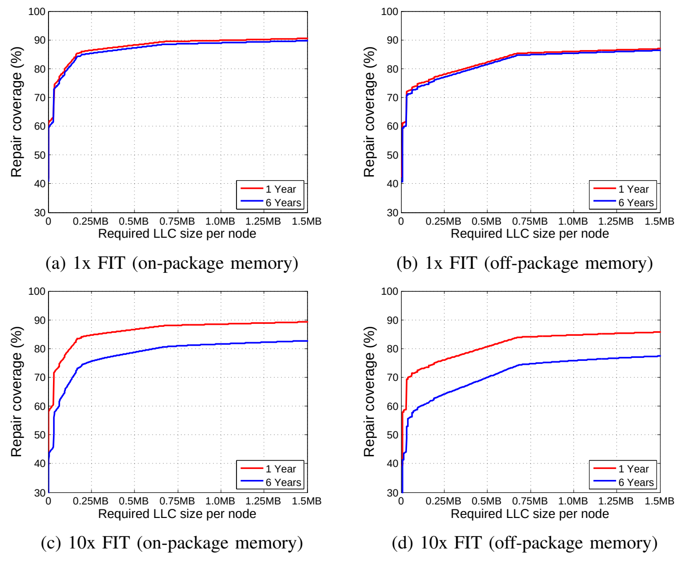
Workload	Benchmark	Input	Description
NAS parallel bench	BT	C	Block Tri-diagonal solver
	CG	C	Conjugate Gradient
	DC	A	Data Cube
	EP	C	Embarrassingly Parallel
	FT	B	Discrete 3D FFT
	IS	C	Integer Sort
	LU	C	Lower-Upper Gauss-Seidel solver
	MG	B	Multi-Grid on a sequence of meshes
	SP	C	Scalar Penta-diagonal solver
	UA	C	Unstructured Adaptive mesh
Co-design	LULESH	30^3	Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics

TABLE III: Multi-threaded workloads

Workload		Benchmarks
HIGH1	H1	lb, mi, so, li, lb, mi, so, li
HIGH2	H2	lb, mi, so, mc, lb, mi, so, mc
HIGH3	H3	lb, mi, mc, li, lb, mi, mc, li
HIGH4	H4	lb, mc, so, li, lb, mc, so, li
HIGH5	H5	mc, mi, so, li, mc, mi, so, li
MIX1	M1	lb, mi, so, li, mc, om, le, om
MIX2	M2	lb, mi, so, li, mc, om, le, le
LOW	L1	le, om, le, om, le, om, le, om
COMPUTE	C1	lb, mi, so, li, mc, le, sj, bz

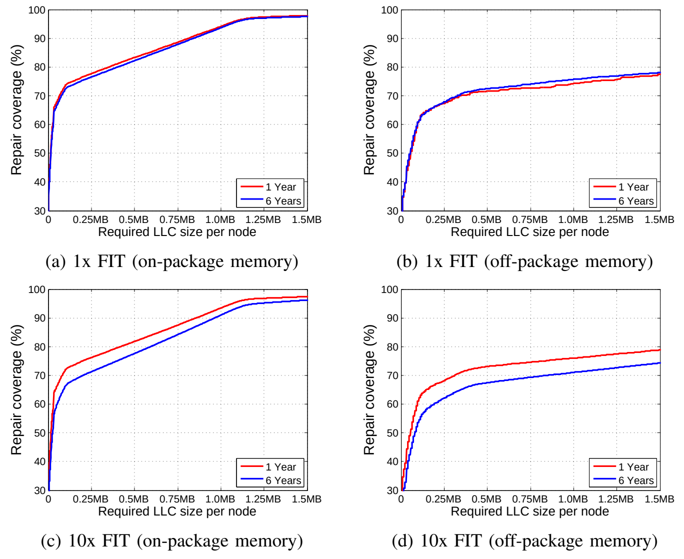
	Description
lb	470.lbm
mi	433.milc
so	450.soplex
li	462.libquantum
mc	429.mcf
le	437.leslie3d
om	471.omnetpp
sj	458.sjeng
bz	401.bzip2

TABLE IV: SPEC CPU2006 multi-programmed workloads composition



(a) 1x FIT (on-package memory)
(b) 1x FIT (off-package memory)
(c) 10x FIT (on-package memory)
(d) 10x FIT (off-package memory)

Fig. 6: Cumulative repair coverage vs. required LLC capacity for full repair with no impact on reliability or regular ECC corrections. The fractions of nodes having any retired data with on-/off-package DRAM are 14% and 77% for 1x and 10x baseline FIT respectively.



(a) 1x FIT (on-package memory)
(b) 1x FIT (off-package memory)
(c) 10x FIT (on-package memory)
(d) 10x FIT (off-package memory)

Fig. 7: Cumulative repair coverage vs. required LLC capacity for repairing only faults that are not correctable by ECC. Relying on ECC lowers the fraction of nodes having any retired data to 6% (1x FIT) and 45% (10x FIT) for on-package memory, and 0.1% (1x FIT) and 1% (10x FIT) for off-package memory.

workloads and memory-intensive workloads simultaneously to measure the impact of FreeFault in various scenarios. Table III and Table IV summarize the benchmarks we use. In Table IV, *HIGHn*, *MIXn*, and *LOW* have only memory intensive workloads while *COMPUTE* includes compute-intensive workloads (*bzip2* and *sjeng*). Benchmarks in *HIGHn* exhibit a larger number of LLC misses, those in *LOW* exhibit a moderate LLC miss rate and still stress memory to a degree, and those in *MIXn* are a mix of HIGH and LOW benchmarks.

In each simulation, every application begins executing at a SimPoint [47], 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.

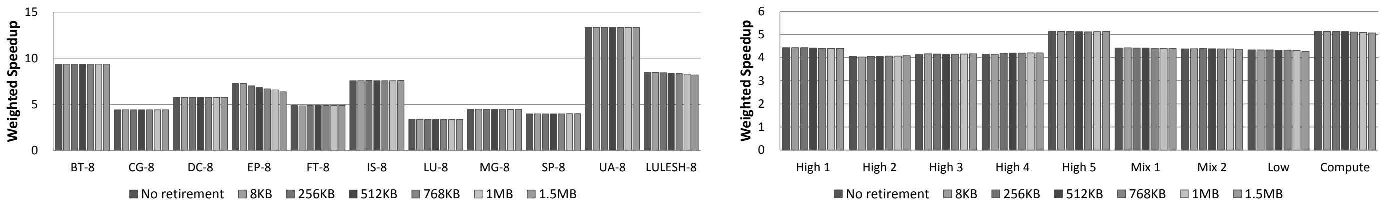
V. Evaluation Results

As explained earlier, we focus our evaluation on the potential impact on performance and power of LLC capacity loss due to FreeFault repair, and the required capacity for maintaining certain levels of DRAM reliability and repair coverage.

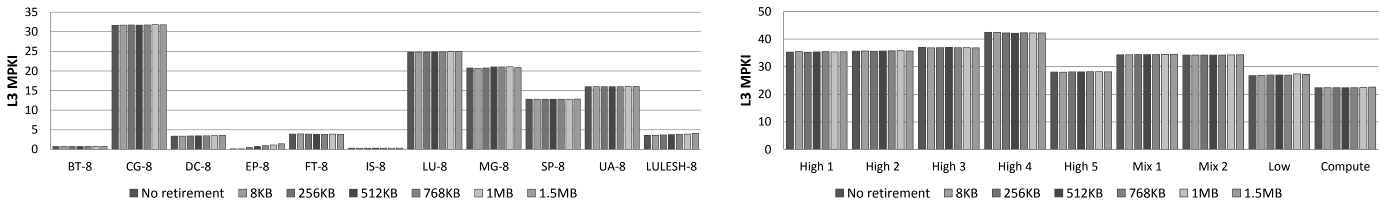
A. *Capacity and Reliability*

Figure 6 shows what fraction of nodes that experienced DRAM faults (14% of nodes with the baseline fault rate) can be fully repaired as greater LLC capacity is dedicated for FreeFault repair. In this experiment, any fault leads to retirement and we do not rely on ECC corrections except for initial repair. With even just a small $\sim$ 8KB capacity hit to the LLC, FreeFault is able to fully repair roughly 60% (60% for on-package, 58% for off-package) of faulty nodes assuming the baseline fault rates (Figure 6a and Figure 6b). Importantly, these repaired nodes have no degradation in reliability and no degradation in performance due to ECC corrections. As we show later, with such a small LLC capacity requirement, performance and power are also not impacted. With greater LLC capacity of up to $\sim$ 768KB, 89% of nodes with on-package faults and 85% of nodes with off-package faults can be fully repaired. Even at that level of LLC capacity decrease for FreeFault locking, performance impact is small.

Nodes that cannot be repaired (11–15% of nodes based on above estimation) suffered coarse-grained failures that require many MB of capacity to repair and are not suitable for FreeFault, requiring higher-level mitigation. The trends are similar for the $10\times$ error rate experiments with slightly lower



(a) Performance comparison (8 threads, 8MB LLC, 64 bytes cache line)



(b) MPKI at LLC (8 threads, 8MB LLC, 64 bytes cache line)

Fig. 8: Performance comparison with various LLC capacity dedicated to FreeFault in a system with the configuration shown in Table II. (a) Performance is measured in Weighted Speedup. (b) Cache performance is measured with LLC MPKI.

repair coverage (Figure 6c and Figure 6d). Note that the graphs are normalized to the number of nodes that experienced any fault, which are roughly 77% of nodes with the $10\times$ fault rates after 6 years of operation.

As explained in Section III, FreeFault can improve its repair coverage by relying on ECC to handle those faults that result in correctable errors. We show the impact on repair coverage in Figure 7. At the baseline fault rate, 97%/78% of nodes can be restored with roughly 1.2MB of the LLC dedicated to FreeFault for nodes with faults in on-/off-package memory, respectively. With the accelerated $10\times$ fault rate, the trends are similar with slightly lower coverage of 95%/72% for nodes with faults on/off-package respectively. The higher fault rate also shows the impact of fault accumulation as a greater number of nodes suffer coarse-grained faults after 6 years compared to at 1 year. Note that the underlying number of nodes (sockets) that require repair is much smaller when relying on ECC as discussed in Section III-B, which explains the difference in coverage compared to the scenario where reliability is not impacted.

B. Performance and Power

Repurposing a portion of the LLC for FreeFault increases the LLC miss rate, which in turn, increases DRAM accesses and degrades performance and power efficiency. We show the impact of this effect on system throughput (weighted speedup) and LLC miss rates (MPKI) in Figure 8. There are two main takeaways from our experiments. First, nearly all workloads are very insensitive to small – mild reductions in LLC capacity. Only two benchmarks showed performance variation of $> 2\%$ (EP and LULESH). The EP NAS parallel benchmark is most impacted by cache reduction because, by chance, its working set fits tightly within the baseline cache size chosen. EP also suffers from significantly greater MPKI as more capacity is devoted to repair, but its overall MPKI is not high and the impact on DRAM behavior is small overall. Even EP does not suffer any measurable degradation when only 8KB are used for FreeFault, although this point is already enough to fully repair roughly 60% of nodes and restore with ECC corrections almost 42% of nodes. Thus, FreeFault delivers on our goal of no-compromise repair for a majority of faults.

Figure 9 shows the impact of FreeFault on DRAM power, which corresponds almost directly to MPKI. The reason is that performance impact is so small that the overall behavior of the application is unchanged. Furthermore, the only significantly-impacted application, EP, does not consume any active DRAM

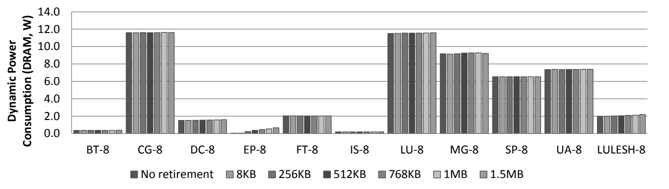
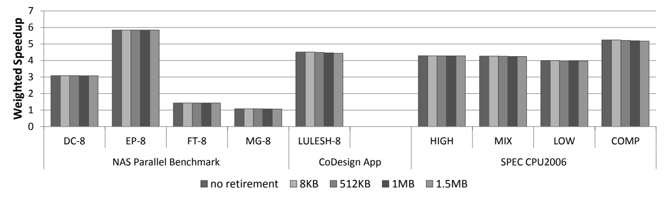
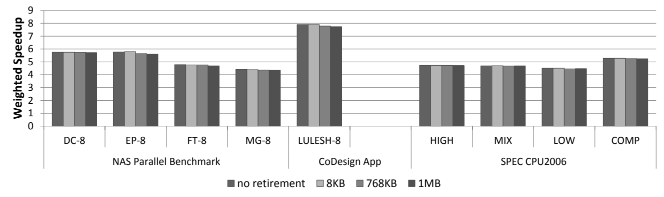


Fig. 9: DRAM dynamic power consumption with various amount of retirement blocks while running multi-threaded applications (NPB and Lulesh) in a machine with 8MB LLC.

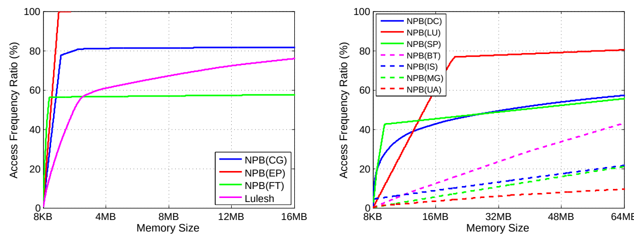


(a) Weighted speedup with 16MB LLC

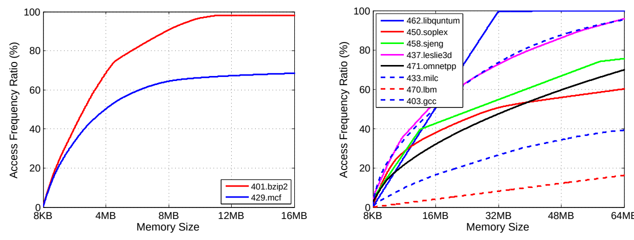


(b) Weighed speedup with 4MB LLC

Fig. 10: Performance impact with 16MB and 4MB LLC. Both cases show consistent results with our baseline, 8MB LLC, and FreeFault is applicable to various sized LLC having higher set associativity. For clarity, we exclude NPB workloads not showing noticeable performance difference with reduced LLC by 1.5MB and 1MB for each, and present averaged WS for SPEC CPU2006.



(a) NAS Parallel Benchmark (NPB) and Co-design Application (Lulesh)



(b) SPEC CPU2006

Fig. 11: Memory access frequencies per 8KB block are measured with PIN-tool [41]. Note that a graph on the right has 4 times bigger address range. The applications shown in the left side presents very high data access locality, so when those blocks are suffered from permanent faults the performance impact becomes significant even if they are correctable. Workloads in the right graph have relatively lower data locality, but in many cases, more than 50% of DRAM accesses are serviced by less than 64 MB memory region. Simple retirement on those blocks greatly reduces performance degradation due to data correction by ECC.

power and is therefore a non-factor in this analysis. We do not show overall system power because it is flat and energy efficiency correlates perfectly with performance.

The results are even better with a larger LLC. Figure 10a shows that with a larger LLC, all applications are essentially not impacted even when 1.5MB of the LLC is used for retirement – a point that can fully repair 90% of nodes and restore with ECC over 98% and 78% of nodes with on-/off-package memory faults, respectively. Figure 10b shows FreeFault works even with a smaller, 4MB LLC because associativity is still high and FreeFault consumes a small part of the LLC. Note that with the smaller case, the impact of FreeFault on EP is small because its working set no longer fits in the baseline cache and its absolute performance is lower (5.8 instead of 7.3).

Impact of ECC Corrections. While relying on ECC to correct the errors of some faults improves repair coverage, corrections increase memory latency and may decrease performance. Because the details of how chipkill correction is performed are not discussed in detail by vendors, it is unclear what this penalty is. Some research indicates that correction may take tens of cycles. Thus, if corrections are frequent, the small performance degradation because of FreeFault impact on LLC capacity may be dwarfed by that of ECC corrections. Figure 11 plots the cumulative distribution of memory accesses services by a certain (possibly non-contiguous) portion of DRAM. Many benchmarks show that even a small $<$ 2MB region of memory may be responsible for a majority of accesses. Thus in some unfortunate situations, ECC corrections that could be avoided with FreeFault repair may be triggered on the majority of accesses. Thus, without FreeFault, such nodes would be retired because of poor performance. Of course, software can address such anomalies by remapping virtual memory pages, but FreeFault achieves this entirely in the microarchitecture.

VI. Related Work

With hard faults being more frequent than transient faults [55, 56] and leading to far more frequent error events [22, 53], mechanisms have been developed to augment ECC for their protection. While some techniques involve fault-specific adaptation of the ECC code [3], a more general approach is to retire faulty components and potentially replace them. Such retire/replace techniques generally fall into one of six main categories (from least to most intrusive in terms of hardware design): (1) retire entire nodes in a large system (software); (2) retire memory ranks or channels (hardware); (3) retire memory frames at OS page granularity (software) [18, 25, 59]; (4) retire individual chips in a rank, which necessitates a reduction in ECC coverage of additional faults (hardware) [18, 25]; (5) compensate for reduced ECC by increasing access granularity with memory-channel coupling (hardware) [25]; and (6) fine-grained retirement with remapping to redundant storage (hardware) [45, 58, 18, 25, 38].

Retiring entire nodes is simple and can be effective provided the fault rate leading to such retirement is low enough and that no critical data is lost when retirement occurs. The problem with this approach is that the fault rate may be quite high if fault rates increase over time (Section IV) and that it is not suitable when specific node availability requirements are strict. Retirement of memory channels or ranks is also quite expensive in that it reduces memory capacity, which often has a severe adverse impact on performance and power efficiency. When cost is of secondary concern, channel mirroring can be used to completely mask faults and even allow hot-replacement of faulty memory for maintaining a desired reliability level [18, 25]. Of course, with mirroring, half the memory capacity and possibly also half the memory bandwidth is sacrificed [28].

More targeted retirement can be performed in systems with page-based virtual memory support. With virtual memory, the OS can be modified to remove faulty memory frames from the frame free list and thus prevent application data from ever using faulty memory regions [18, 25, 59]. While conceptually a simple and effective technique, successful and efficient implementations are not straightforward because physical to DRAM address mapping may increase the footprint of some DRAM faults to many OS-page granularity memory frames [63] and because some OS components and peripheral devices do not fully utilize virtual memory facilities [12] (IBM's AIX, in particular, limits its retiring feature to a specific memory address space [18, 25]).

For example, when a single column in a 512×512 sized subarray experiences faults, it may requires retiring up to 2MB or 1GB memory in 4KB and 2MB OS page respectively (512 pages with 4KB OS page, 2–512 pages with 2MB OS page depending on interleaving and number of ranks/banks/channels) while each OS page includes only a few defective bit cells. Additionally, recent work related to very high-capacity memory systems has proposed avoiding page-based memory management altogether due to translation inefficiency and replacing it with direct segments [7]. Such a system may be suitable for applications designed to fit into its physical memory size, but precludes page-based OS-managed retirement while FreeFault remains effective.

An interesting alternative to retiring nodes or reducing capacity is to retire the faulty DRAM device itself. Because ECC protection requires redundant devices, part of that redundancy can be repurposed for remapping data from the faulty DRAM device. Examples of this technique include the bit-steering approach of IBM's Memory ProteXion scheme of X-Series servers [9, 18, 25] and one way by which Intel's SDDC and DDDC schemes tolerate chip failures [28]. The downside of this repair approach is that resilience against further faults is significantly degraded because ECC redundancy is reduced. To compensate for this reduction, it is possible to change the ECC code used to a wider codeword and maintain protection level, but this requires coupling memory channels and reduces performance and power efficiency [14].

Finally, at the finest granularity hardware can introduce redundant storage for remapping faulty DRAM at a fine granularity. A classic example of this approach is row and column sparing within memory arrays [37], although such techniques are often only available during manufacture/test/integration time and may not be available for use in the field by customers. As an alternative, structures external to the arrays themselves have been suggested for use in repair. *Fault Caches* have been proposed within the DRAM die or package to be used as spare locations for memory retirement and repair [58]. A fault cache can effectively tolerate DRAM faults, but adds substantial cost because secondary storage and remapping components are necessary at each of the numerous memory devices in the system. A different implementation of a broadly similar idea has been proposed as part of the ArchShield architecture, which uses a fraction of DRAM storage to hold both indirection information and spare storage for remapping [44]. Also, in the same vein, ideas for tolerating wearout faults in non-volatile memories often utilize fine-grained spare storage and remapping [52, 30, 61, 48].

We note that while not discussed in prior work, it is also possible to include a small fault cache within each memory controller. Such a design still suffers from adding hardware and complexity that is very rarely exercised. Contrary to all these previous approaches, FreeFault supports fine-grained retirements with minimal impact on complexity (see Section III-A), performance, and power, without compromising reliability, and with no reliance on or interruption to running software.

VII. Conclusion

To conclude, we presented and demonstrated the potential of FreeFault as another layer in the memory resilience scheme of a highly-reliable and highly-available system. We show how FreeFault can be implemented with only minor modifications to current designs, with the most significant modifications amounting to ensuring DMA and other uncached accesses are snooped by the coherence mechanisms of the LLC, while providing repair capabilities entirely within the microarchitecture.

Our results show that FreeFault can be used to fully repair roughly 60% of all faulty nodes assuming current fault rates with only an 8KB reduction in LLC capacity. In some scenarios, a larger fraction of the LLC may be locked without impacting performance and this extra capacity (up to 768KB total) improves fault coverage to over 85% in general and roughly 90% of on-package faults. This is done with essentially no impact on performance and absolutely no impact on reliability and DRAM capacity. Furthermore, by sacrificing some reliability and relying on ECC corrections (new faults may lead to undetected errors), fault-repair coverage is increased to 97% and 78% of nodes with faults in on- and off-package memory, respectively. Again, this is done transparently and before software must be called upon to perform coarser-grained tradeoffs between DRAM capacity, performance, and reliability.

Thus, we reach the surprising conclusion that *FreeFault demonstrates the counter-intuitive property of benefiting from trading off a small amount of on-chip SRAM for repairing DRAM*.

Acknowledgment

The authors acknowledge the Texas Advanced Computing Center for providing HPC resources and the support of the Department of Energy under Award #B599861 and the National Science Foundation under Grant #0954107, which partially funded this research.

References

[1] Advanced Micro Devices (AMD), Inc., "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors," Jan 2013.

[2] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future Scaling of Processor-Memmory Interfaces," in *Proc. the Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2009.

[3] F. Aichelmann, "Fault-Tolerant Design Techniques for Semiconductor Memory Applications," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 177–183, 1984.

[4] 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, pp. 310–321.

[5] 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.

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

[7] 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, pp. 237–248.

[8] S. K. Chan, J. A. Gerardi, and B. L. McGilvray, "Cache locking controls in a multiprocessor," Apr. 23 1985, US Patent 4,513,367.

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

[10] T. J. Dell, "A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," IBM Microelectronics Division, Nov. 1997.

[11] 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.

[12] 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, p. 11.

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

[14] 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.

[15] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H. M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, "Exploring DRAM

organizations for energy-efficient and resilient exascale memories," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 23.

[16] 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, pp. 76–83.

[17] D. W. Green, "Cache with finely granular locked-down regions," Mar. 28 2000, US Patent 6,044,478.

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

[19] HMC, "Hybrid Memory Cube Specification 1.0," Hybrid Memory Cube Consortium, 2013.

[20] HP, "HP Integrity rx2800 i2 Server: Achieve high performance with the mission-critical rackmount server," http://h20338.www2.hp.com/enterprise/downloads/4AA0-7916ENW.pdf, 2011.

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

[22] 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," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 111–122, 2012.

[23] Hynix, "Blazing A Trail to High Performance Graphics," Hynix Semiconductor, Inc., 2011.

[24] IBM, "Enhancing IBM Netfinity Server Reliability," 1999.

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

[26] Intel, "Intel Xeon Processor E5520 (8M Cache, 2.26 GHz, 5.86 GT/s Intel QPI)," http://http://ark.intel.com/products/40200/Intel-Xeon-Processor-E5520-8M-Cache-2_26-GHz-5_86-GTs-Intel-QPI, 2009.

[27] Intel, "Intel Xeon Processor E5620 (12M Cache, 2.40 GHz, 5.86 GT/s Intel QPI)," http://http://ark.intel.com/products/47925/Intel-Xeon-Processor-E5620-(12M-Cache-2_40-GHz-5_86-GTs-Intel-QPI), 2010.

[28] 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.

[29] Intel, "Intel 64 and IA-32 Architectures Software Developers Manual Volume 1: Basic Architecture," http://download.intel.com/design/processor/manuals/253665.pdf, 2011.

[30] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: building reliable systems from nanoscale resistive memories," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1. ACM, 2010, pp. 3–14.

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

[32] JEDEC, "JESD 79-3f DDR3 SDRAM standard," 2010.

[33] JEDEC, "JESD 209-2e LPDDR2," 2011.

[34] JEDEC, "JESD 229 Wide I/O SDR," 2011.

[35] JEDEC, "JESD 212a GDDR5 SGRAM," 2013.

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

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

[38] B. Kleveland, J. Choi, J. Kumala, P. Adam, P. Chen, R. Chopra, A. Cruz, R. David, A. Dixit, S. Doluca, M. Hendrickson, B. Lee, M. Liu, M. J. Miller, M. Morrison, B. C. Na, J. Patel, D. Sikdar, M. Sporer, C. Szeto, A. Tsao, J. Wang, D. Yau, and W. Yu, "Early detection and repair of VRT and aging DRAM bits by margined in-field BIST," in *VLSI Circuits Digest of Technical Papers, 2014 Symposium on*. IEEE, 2014, pp. 1–2.

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

[40] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments," in *Proc. the 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[41] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *ACM Sigplan Notices*, vol. 40, no. 6, pp. 190–200, 2005.

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

[43] Micron, *Micron 2 Gb* $\times 4$, $\times 8$, $\times 16$, *DDR3 SDRAM: MT41J512M4, MT41J256M4, and MT41J128M16*, Micron Corp., 2011.

[44] 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, pp. 72–83.

[45] Y. Nitta, N. Sakashita, K. Shimomura, F. Okuda, H. Shimano, S. Yamakawa, A. Furukawa, K. Kise, H. Watanabe, Y. Toyoda, T. Fukada, M. Hasegawa, M. Tsukude, K. Arimoto, S. Baba, Y. Tomita, S. Komori, K. Kyuma, and H. Abe, "A 1.6 GB/s data-rate 1 Gb synchronous DRAM with hierarchical square-shaped memory block and distributed bank architecture," in *Solid-State Circuits Conference, 1996. Digest of Technical Papers. 42nd ISSCC., 1996 IEEE International*. IEEE, 1996, pp. 376–377.

[46] Oracle, "Maximizing Application Reliability and Availability with SPARC T5 Servers," http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/o13-027-t5-ras-1924294.pdf, 2013.

[47] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1. ACM, 2003, pp. 318–319.

[48] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 318–328.

[49] S. R. Rahman, D. F. Greenberg, K. C. Stacer, K. M. Bruce, M. B. Smittle, M. D. Snyder, and G. L. Whisenhunt, "Cache locking device and methods thereof," Nov. 2 2010, US Patent 7,827,360.

[50] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*. IEEE, 2000, pp. 128–138.

[51] A. M. Saleh, J. J. Serrano, and J. H. Patel, "Reliability of scrubbing recovery-techniques for memory systems," *Reliability, IEEE Transactions on*, vol. 39, no. 1, pp. 114–122, 1990.

[52] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 141–152.

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

[54] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous mutlithreading processor," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 234–244, 2000.

[55] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–11.

[56] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: positional effects in DRAM and SRAM faults," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 22.

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

[58] 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," *Solid-State Circuits, IEEE Journal of*, vol. 27, no. 11, pp. 1525–1533, 1992.

[59] 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 *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. IEEE, 2006, pp. 365–370.

[60] Unified Extensible Firmware Interface Forum, "UEFI Specification Version 2.4 (Errata B)," http://www.uefi.org/sites/default/files/resources/2_4_Errata_A.pdf, 2014.

[61] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE, 2011, pp. 466–477.

[62] D. Zhan, H. Jiang, and S. C. Seth, "STEM: Spatiotemporal management of capacity for intra-core last level caches," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*. IEEE, 2010, pp. 163–174.

[63] 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, pp. 32–41.