FREE-p: Protecting Non-Volatile Memory against both Hard and Soft Errors

Doe Hyun Yoon[†]
doehyun.yoon@gmail.com

Naveen Muralimanohar[‡]
naveen.muralimanohar@hp.com

Jichuan Chang[‡]
jichuan.chang@hp.com

Parthasarathy Ranganathan[‡]
partha.ranganathan@hp.com

Norman P. Jouppi[‡]
norm.jouppi@hp.com

Mattan Erez[†]
mattan.erez@mail.utexas.edu

[†]*The University of Texas at Austin*
Electrical and Computer Engineering Dept.

[‡]*Hewlett-Packard Labs*
Intelligent Infrastructure Lab.

Abstract

Emerging non-volatile memories such as phase-change RAM (PCRAM) offer significant advantages but suffer from write endurance problems. However, prior solutions are oblivious to soft errors (recently raised as a potential issue even for PCRAM) and are incompatible with high-level fault tolerance techniques such as chipkill. To additionally address such failures requires unnecessarily high costs for techniques that focus singularly on wear-out tolerance.

In this paper, we propose fine-grained remapping with ECC and embedded pointers (FREE-p). FREE-p remaps fine-grained worn-out NVRAM blocks without requiring large dedicated storage. We discuss how FREE-p protects against both hard and soft errors and can be extended to chipkill. Further, FREE-p can be implemented purely in the memory controller, avoiding custom NVRAM devices. In addition to these benefits, FREE-p increases NVRAM lifetime by up to 26% over the state-of-the-art even with severe process variation while performance degradation is less than 2% for the initial 7 years.

1. Introduction

Non-volatile memory (NVRAM) technologies are emerging as a scalable substitute of DRAM as main memory. For example, phase-change memory (PCRAM) is almost as fast as DRAM (only 2-3× higher latency at the same bandwidth), provides larger capacity, and scales better. However, most NVRAM technologies, including PCRAM, have finite write endurance; memory cells wear out after a certain number of writes. Recent architectural research has focused on this write endurance issue but this prior work is incomplete for three important reasons: (1) it relies on integrating custom error-tolerance functionality within memory devices – an idea that the memory industry is historically loath to accept because of strong demand to optimize cost per bit; (2) it ignores soft errors (in both peripheral circuits and cells), which can cause errors in NVRAM as shown in recent studies; and (3) it requires extra storage to support chipkill that enables a memory DIMM to function even when a device fails. We propose *Fine-grained Remapping with ECC and Embedded-Pointers* (FREE-p) to address all three problems. Fine-grained remapping nearly eliminates storage overhead for avoiding wear-out errors. Our unique *error checking and correcting* (ECC) component can tolerate wear-out errors, soft errors, and device failures. The proposed mechanism shifts resiliency functions entirely to the memory controller, leaving NVRAM devices as simple and cheap as possible.

Prior research focused on protecting NVRAM only against wear-out failures, ignoring soft errors and device failures. The motivation has been that NVRAM cells are inherently robust against particle strikes. Unfortunately, recent work [5] identified new soft-error mechanisms that impact NVRAM, which we summarize in Section 2.1. Further, memory cells typically account for only 60% of the die area with the rest used for global and peripheral circuits, which are still susceptible to soft errors [32]. Recent architecture research on error-tolerance in NVRAM systems ignores this important source of errors. Simply augmenting existing mechanisms, such as dynamically replicating memory (DRM) [10] and error correcting pointers (ECP) [22], with the techniques for soft-error tolerance used in current DRAM-based systems requires too high an overhead. Moreover, existing NVRAM reliability solutions often require custom functionality embedded within NVRAM devices. ECP, for example, implements hard error detection/correction logic within an NVRAM device. Embedding reliability mechanisms

at the device increases the cost of memory and the protection level is fixed at design-time. It also protects only cell array; hence, it does not achieve end-to-end protection. As practiced in DRAM and FLASH, it is better to implement error detecting/correcting at the memory controller so that we can even detect and correct errors in wires, packaging, and periphery circuits in addition to errors in memory cells.

FREE-p is the first mechanism that is designed specifically to tolerate both soft and hard errors in NVRAM main memory systems without error tolerance functionality within the NVRAM devices themselves. FREE-p relies on a novel *fine-grained remapping* (FR) mechanism that has almost zero storage overhead initially and dynamically adapts to wear-out failures. The innovation is in utilizing the still-functional cells of worn-out memory blocks to store remapping information. We then integrate FR with specially designed ECC for detecting and correcting both hard and soft errors. Unlike prior work, our mechanism can be easily augmented to support chipkill-correct. FR, however, incurs a performance overhead. When accessing a remapped block, the memory controller first reads the original location, and then follows the pointer to the remapped location. This increases memory traffic and access latency, potentially degrading performance. To mitigate the negative impact of FR, we propose a set of optimization techniques to accelerate remapping, including simple caching and a more effective hash-based scheme.

We implement all the necessary functionality except the additional storage for ECC (limited to under 12.5% typical in current systems) at the memory controller, enabling end-to-end protection with simple (and expected commodity) NVRAM devices. Compared to the most efficient prior work, ECP, FREE-p achieves 7.5% and 26% longer lifetime at typical and high process variation. These advantages are in addition to tolerating soft errors and potential chip failures. The performance impact of the fine-grained remapping is negligible in the first 7 years of operation, less than 1.8% on average, and is around 10% on average even near end of life (8.8 years).

The rest of the paper is organized as follows: We briefly review failure mechanisms in PCRAM and related work in Section 2; we present FREE-p in Section 3; we evaluate the wear-out tolerance and performance overhead of FREE-p in Section 4; and Section 5 concludes the paper.

2. Background and related work

Our technique, FREE-p, is applicable to any non-volatile memory technology but we use PCRAM as an example technology in this paper. We first describe basics of PCRAM as well as failure mechanisms in Section 2.1, and discuss related work in Section 2.2.

2.1. Failures in phase-change memory

PCRAM operations. PCRAM is a non-volatile memory built out of Chalcogenide-based materials such as alloys of germanium, antimony, or tellurium (Ge2Sb2Te5, GeSb, Sb2Te3). Unlike DRAM and FLASH that record data through charge storage, PCRAM uses distinct phase-change material states (hence, resistances) to store values. Specifically, when a phase-change material is heated to a high temperature for an extended period of time, it crystallizes and reduces its resistance (SET operation). The SET operation is slow, and determines the write latency of PCRAM. To RESET a cell into a high resistance state, a current large enough to melt the phase-change material (almost double the SET current) is applied for a short period, and then abruptly cut-off. The abrupt current fall quenches the material into the amorphous phase, resulting in high resistance.

A promising feature of PCRAM is its capability to store multiple bits in a single cell, also referred to as Multi Level Cells (MLC). The pulse width of RESET for MLC is adjusted such that it partially crystallizes the phase-change material and modifies its resistance to an intermediate value between SET and RESET resistances. ITRS projects the availability of 4-bit MLC by 2012 [1]. Reading a cell simply involves sending a small current and measuring the voltage drop across the cell. As both crystalline and amorphous phases are relatively stable at normal operating temperature (more about this later), the cell can ideally retain the value for many years.

Hard Errors in PCRAM. While high operating temperatures, required for SET/RESET operations, help keep PCRAM cells stable at room temperature, they significantly impact the lifetime of PCRAM. After repeated high temperature RESET operations, the electrical path through the phase-change material begins to break and this permanently RESETs the cell into a high resistance state. Recent studies on PCRAM prototypes show that the number of writes to a PCRAM cell is limited to 10^8-10^{10} [5] while a DRAM or SRAM cell can support more than 10^{15} writes. This significant difference in endurance between PCRAM and volatile memories is considered a critical drawback that precludes PCRAM from becoming a universal memory. Going from cell to chip, PCRAM chip endurance (based on vendor specifications such as [16]) can further drop to 10^6 due to process variation and non-ideal wear-leveling.

Soft Errors in PCRAM. Although PCRAM is robust against particle-induced soft errors, there are several factors that can cause soft errors in PCRAM. Some common factors include write noise, resistance drift (or short-term drift), and spontaneous crystallization (or long-term drift). In particular, the short-term resistance drift is prone to causing soft errors in PCRAM but has received little attention from the architecture community.

Write noise (also called thermal crosstalk) refers to disturbances in a cell value due to repeated SET/RESET operations on a nearby cell. Studies on PCRAM reliability, however, show crosstalk is not an issue [17, 12]. The root cause of short- and long-term resistance drifts lies in the metastable nature of the amorphous phase. After the sudden cooling of a PCRAM cell that triggers the state change, the resistance of the cell continues to grow for a certain period of time before it starts reducing again. This phenomenon is referred to as short-term drift. Long-term drift is a result of slow crystallization of the phase-change material at room temperature, which degrades the cell resistance over time. As the rate of crystallization is directly proportional to temperature, it can take many days to produce a noticeable change in cell resistance at room temperature. Long-term drift can be easily addressed by periodically refreshing cells every several days. However, short-term drift can be problematic in PCRAM, especially for multi-level cell (MLC) PCRAM. The random nature of short-term drift due to process variation makes it difficult to guarantee correctness through periodic refreshing or scrubbing.

In addition to soft errors in the PCRAM cells themselves, the peripheral circuits such as decoders, sense-amps, and repeaters still use CMOS transistors that are susceptible to soft errors. With memories typically having area efficiency (ratio of area of memory cells to the total area) of less than 60% [32], the likelihood of a failure in peripheral circuits is non-trivial.

Chip Failures. In addition to soft and hard errors in memory arrays, a recent study shows memory chip failures, possibly due to packaging and global circuit issues, cause significant down time in datacenters [23]. Hence, business critical servers and datacenters demand chipkill-correct level reliability, where a DIMM is required to function even when an entire chip in it fails. There are various solutions for chipkill-correct [6, 3, 33, 27] and the industry is pursuing even stronger protection [8, 15], for example, double chipkill or soft error correction under chipkill. With memory manufacturers' relentless focus on cost per bit, PCRAM memories will likely require very robust fault tolerance techniques as well. For these reasons, in addition to wear-out protection, traditional coding techniques such as ECC and parity will be required to tolerate PCRAM chip failures.

2.2. Related work

Many techniques have been recently proposed to improve NVRAM endurance, focusing on write reduction and wear-leveling to increase lifetime (Section 2.2.1) and hard error detection/correction for graceful degradation (Section 2.2.2). We also discuss other prior work (Section 2.2.3).

2.2.1. Increasing NVRAM lifetime.

Avoiding unnecessary writes. Researchers have developed various techniques to avoid unnecessary writes. Lee et al. [13] proposed to only write back modified cache lines or words. Qureshi et al. [19] also explored writing back only modified data. A variety of fine-grained approaches (bit-level partial write) have also been proposed, including data comparison write (DCW) [31], Flip-N-Write [7], and many others [36, 34, 11], by utilizing read-before-write to detect modified data and potentially selectively invert bits.

Wear-leveling. Another approach to improve lifetime is by distributing writes equally to all cells in the device. This technique is known as wear-leveling and is commonly used with FLASH memory. Prior research on PCRAM wear-leveling includes row shift [36, 22], word shift [34], and randomized address mapping [19, 18, 24].

Our work focuses on soft and hard error tolerance with commodity NVRAM devices, and can be combined with any lifetime-improvement techniques.

2.2.2. Tolerating wear-out failures. We first present prior wear-out failure detection/correction schemes, and then explain the extra actions needed once such mechanisms become insufficient due to a large number of failures. We also describe one of the most advanced prior proposals, ECP, in detail.

Detecting/correcting wear-out failures. The most intuitive approach to detect wear-out failures is to use an ECC code. The complexity of ECC-based error correction, however, increases linearly with the correction capability [26], rendering general ECC unsuitable for NVRAM [10, 22].

An entirely different approach to detecting wear-out errors is to use verify-after-write, in which detection happens when writing to NVRAM. Verify-after-write first writes a data block to NVRAM and then immediately reads the value for comparison. A mismatch indicates a write failure due to wear-out. Verify-after-write incurs high traffic and performance

overheads if implemented at the memory controller and recent work advocates implementing this functionality within the NVRAM devices themselves [10, 22]. Because an NVRAM write is relatively slow, it is generally believed that the penalty of adding an internal read after the write is not significant. A caveat is that the verifying reads increase power consumption even when there is no wear-out failure.

After error detection, errors are corrected by using another, non-faulty set of cells to store the data. Two recent techniques have been suggested to accomplish this. Dynamically replicating memory (DRM) [10] replicates the write to a separate NVRAM page with disjoint failures, and future reads access both pages to retrieve the correct data. Later research [22], however, shows that even a simple ECC scheme with a single-bit error correcting and double-bit error detecting (SEC-DED) code outperforms DRM. A promising alternative, ECP [22], can tolerate wear-out failures without multiple accesses, but further modifies NVRAM devices.

ECP implements all error-correcting functionality within the NVRAM device: it uses verify-after-write to detect errors, a set of pointers to encode error locations, and additional storage cells to hold patched values. For a 512-bit data block, a 10-bit error correcting entry can tolerate one bit failure (9 bits to identify the error location and 1 bit to hold the patched value). The 6-bit tolerating ECP (ECP6) uses 6 such 10-bit entries, and has a 12% static overhead. Although correcting multi-bit errors with ECP is much simpler than with traditional error codes (e.g., BCH codes), ECP has several significant limitations.

- ECP can tolerate only wear-out failures. Augmenting ECP to achieve end-to-end reliability will result in high storage overhead.
- The hard error tolerance level with ECP is fixed and determined at device design time. Due to process variation and non-ideal wear-leveling efficiency[1], the overhead of ECP is unnecessarily large.
- ECP requires custom NVRAM devices. DRAM and FLASH manufactures are highly motivated to minimize cost per bit, and would prefer solutions without custom logic in the NVRAM devices for simplicity, flexibility, and cost advantages.

Recent work, SAFER [25], proposes a better mechanism compared to ECP to tolerate hard failures in PCRAM. While it is orthogonal to our work, similar to ECP, SAFER also requires a custom-designed PCRAM device.

[1] We use "normalized endurance" proposed by Qureshi et al. [18] as the metric for the efficiency of wear-leveling, which is 90%, at most, using one of the best practical wear-leveling mechanisms.

FREE-p departs from this recent work on tolerating NVRAM errors. We focus on end-to-end reliability rather than on wear-out errors, and also restrict ourselves to current best-practice constraints: (1) we keep storage devices optimized for cost per bit, and implement all resiliency functionality at the memory controller; and (2) we tolerate wear-out, soft errors, and potential device failures with less than 12.5% storage overhead.

2.2.3. Other related work. There has been other prior work that uses pointers and fine-grained data remapping. Wilkerson et al. presented a bit-fix technique [29] that stores pointers as meta-data similar to ECP [22]. Our technique differs in that a pointer is stored within a data block itself and used for fine-grained data remapping rather than error correction. Roberts et al. coupled two cache lines assuming one will function if the other fails [21] similar to the DRM approach [10]. Wu et al. presented fine-grained data relocation for migrating frequently accessed cache lines to faster storage. This technique uses cache "tags" for managing fine-grained data relocation. Instead, we use embedded pointers for fine-grained data remapping.

3. FREE-p implementation

This section describes our proposal for an end-to-end reliable NVRAM memory system with commodity devices and FREE-p. We start by describing fine-grained remapping (FR), our low-cost wear-out failure tolerance mechanism, in Section 3.1. In Section 3.2, we explain how to use FR to design an NVRAM-based memory system with all necessary reliability mechanisms implemented at the memory controller, leaving NVRAM devices simple. We identify overheads and propose a set of optimization techniques to mitigate the potential performance degradation in Section 3.3. We then discuss hard/soft error detection and correction in Section 3.4 followed by required operating system support in Section 3.5 and chipkill support in Section 3.6.

3.1. Fine-grained remapping

As described earlier, current wear-out failure tolerance techniques are designed to tolerate up to a certain number of errors in a memory block. When a block accumulates more wear-out failures than can be corrected, it gets disabled and remapped. The primary problem with the existing techniques is that the remapping and the disabling of data blocks happen at a coarse granularity of a device or OS page. For example, the FLASH controller disables device pages that can no

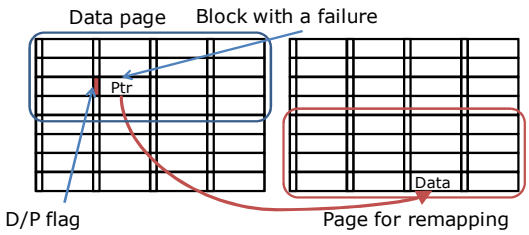

Figure 1. Example of fine-grained remapping with an embedded pointer.

longer be used [14]. Most recent work on NVRAM does bit-level error correction prior to remapping [22]. Although coarse-grained techniques are simple to implement (through OS virtual to physical mapping table), they are inefficient. A single 64B block with intolerable failures triggers the mapping out of an entire large page of data of 4kB or more.

In contrast to coarse-grained approaches, we propose fine-grained remapping (FR); FR maps out only a small block that contains wear-out failures, leaving the rest of the physical page usable. This both increases the lifetime of a page and makes NVRAM more tolerant to process variation. For instance, prior studies on process variation show strong spatial correlation between memory cells [9] [2]. Hence, it is likely that several consecutive bits in NVRAM can fail well in advance compared to the vast majority of other bits in a page. Without FR, a single cache line failure will deactivate an entire page, leading to poor efficiency. Further, FR adapts to imperfections in wear-leveling more efficiently.

Although it is desirable to keep the remap size as small as possible, we use the last level cache line size (e.g., 64B) as the granularity for remapping. While solving the problem of disabling coarse-grained blocks, FR requires a large book-keeping storage. Maintaining all of the remapping information in dedicated storage within the processor or in main memory is impractical. We make a key observation that even if a block is deemed dead, it still has many functional bits that can store useful information. We propose using the remaining working bits in a dead block to *embed a remapping pointer*, as shown in Figure 1. Thus, we use the mapped-out block itself, otherwise useless, as free storage for remapping information. In addition, we need dedicated storage to indicate whether a block has been remapped; only 1 bit per block. When we detect a wear-out failure, we remap the block to a fault-free location, mark that this block is remapped (using a 1-bit *Data/Pointer flag (D/P)* per data block), and write the embedded remapping pointer within the faulty block.

[2] For evaluating our proposals, we conservatively model only random variation.

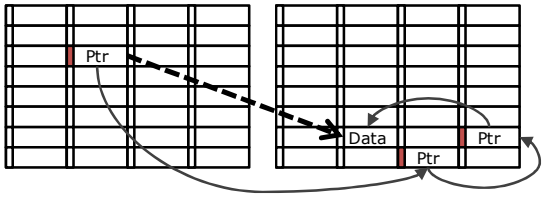

Figure 2. Chained remapping example (solid pointers) and limiting the number of hops (dashed pointer).

The operating system is responsible for identifying a remap region for the failed block (more on this in Section 3.5). In this way, the size of NVRAM degrades gracefully with age.

Because the embedded pointer is stored in essentially a faulty memory block, we must ensure that it can be correctly accessed. A pointer is much smaller than the data block it is remapping, and we can therefore use a very simple and strong error code. For example, we can even use a 7-modular-redundancy (7-MR) code, which replicates the 64-bit pointer 7 times. If the pointer cannot be correctly stored, even with this large redundancy, we map out the entire page as in current techniques. This, however, should be very rare considering the strength of the 7-MR code (the probability that 7-MR cannot tolerate 4-bit failures is roughly 9.04e-7, assuming uniform errors). We do not consider further wear-out failures on pointers and the D/P flag because we rarely re-write them.

A potential problem with FR is chained remapping when a remapped block is remapped again, as shown in Figure 2 (solid arrows). An access to this chained remapping will traverse a linked list, which may take a long time. We eliminate such chains by writing the final destination to the original faulty block when a remapped block is remapped again (dashed arrow in Figure 2).

3.2. NVRAM system organization

In this subsection, we show how to use FR and ECC to design a memory system with simple, expected commodity, NVRAM devices. We explicitly explore this as an alternative to the approach of prior research (e.g., ECP [22]) that recommends significant changes to the internal design and micro-architecture of NVRAM components.

Figure 3 illustrates a FREE-p memory controller and NVRAM organization. It is much like current DRAM systems: eight ×8 devices construct a 64-bit wide channel and a dedicated device stores the meta-data, which includes the D/P flag as well as ECC information. We assume a DDR3-like interface (burst 8) for NVRAM devices in this study. The main change

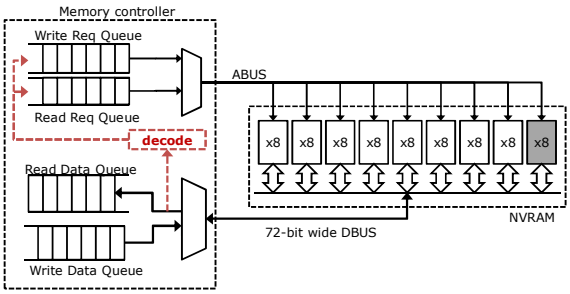

Figure 3. Memory controller and NVRAM organization. The gray device is for D/P flag and ECC.

to the memory controller is support for fine-grained remapping, detailed below for read and write operations.

Read operations. When we read data from NVRAM, we also read the D/P flag and identify whether a block is remapped or not. If the block is not remapped, the memory controller forwards the returned data to the upper level, e.g., cache controller, as in a standard memory controller (Figure 4(a)). Otherwise, the memory controller decodes the pointer, re-inserts the request onto the read queue with the remapped address, and schedules the forwarded read request (Figure 4(b)). Thus, reading remapped data requires no interaction with the processor core but incurs bandwidth and latency overheads for the second read. We propose mechanisms to mitigate these overheads in Section 3.3.

Write operations. Write operations with FR are trickier than reads. Because we do not rely on any functionality in NVRAM devices, the memory controller must make sure that the target block in NVRAM has not been previously remapped before issuing a write. Otherwise, we may accidentally overwrite the embedded pointer with new data (which will never succeed because of wear-out failures). Hence, we first read the data block from memory before writing data. If the block is not yet remapped, we write the new data to the block (Figure 5(a)). Otherwise, we write the new data to the remapped location using the pointer (Figure 5(b)).

Though a write-back is not on the critical processing path, always reading a block before a write is inefficient. With an order of magnitude difference between read and write latency in PCRAM, the overhead of performing a read before a write has minimal impact on performance (<4% performance penalty in our analysis). We can address even this small overhead by keeping track of all cache lines in a processor. Assuming a write-back write-allocate last-level cache (LLC), a data block is always fetched and cached before it is written out to NVRAM. We can keep track of the remapped status of blocks by

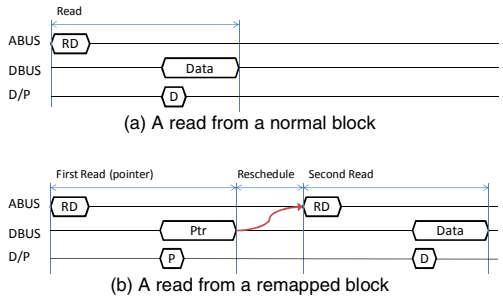

Figure 4. Read operation examples.

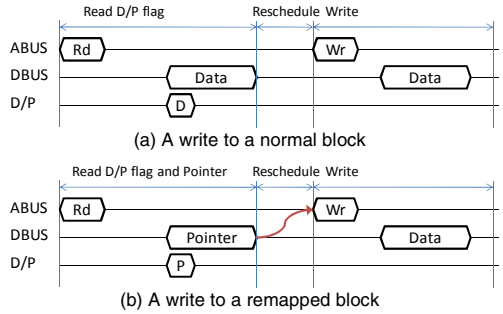

Figure 5. Write operation examples.

maintaining a single bit for each cache line that indicates whether that line is in a remapped block or not. This flag can either reside in the cache or in the memory controller, and its overhead is <0.2% (64kB for a 32MB LLC). We use this optimization when evaluating FREE-p (Section 4). Uncached and I/O operations always read before writing. However, their effect on performance is negligible.

3.3. Mitigating remapped access penalty

FR provides wear-out tolerance with nearly zero storage overhead but may degrade performance because of the additional traffic and latency for accessing remapped data. We consider two techniques that opportunistically avoid pointer fetching: simple caching of pointers and more efficient index caching with hash based remapping.

3.3.1. Remapped pointer cache. The most intuitive approach to mitigating the remapping penalty is to add a small cache to the memory controller that stores remapping pointers to avoid re-fetching them on a hit. Off-chip accesses often lack temporal locality so the remapping cache will work well only when the number of failed blocks is small. Thus, we expect the benefit of this cache to decrease as the memory ages. We use a 1024-set 4-way set-associative cache which requires around 64kB total in Section 4.

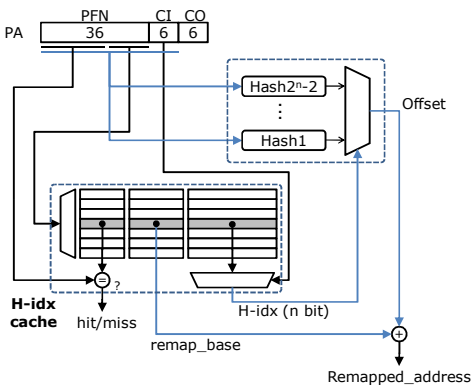

Figure 6. Index cache with hashing functions. PFN is physical frame number, CI is cache line index, and CO is cache line offset. We use a 4kB page and a 64B cache line; hence, each page has 64 lines.

3.3.2. Index cache with hashing. We propose storing the remapped addresses in a compressed form using hashing and pre-defined remapping (Figure 6). Instead of maintaining a remap table containing a 64-bit address of a remapped location, we associate each page with a *remap_base* and a *Hash index (H-idx)* vector. The remap base identifies a page that is being used to store remapped blocks, while H-idx is used to find a block within that page. We use the H-idx to select one of a set of hash functions, apply the hash to the block's original address, and use the result as an offset from the page's remap_base as the final remapped address. Thus we can compute a remapped location without accessing the embedded pointer. Additionally, all the remappings associated with an entire page (e.g., 4kB) can be stored efficiently.

While we can accelerate address remapping using this scheme, the embedded pointers are still required, because hash collisions may occur. A hash collision means that two blocks are remapped to a single location, which should not be allowed. The OS checks for collisions at the time a block must be remapped (this is a very rare event). When a collision is detected, the OS remaps the block to a different location, and the block is associated with an H-idx of all ones (i.e., 2^n-1 for an n-bit H-idx). The address remapping of such blocks cannot be accelerated and the embedded pointer is used as described in Section 3.2. Most blocks, of course, require no remapping, and this is represented with an H-idx value of zero. Note that the OS is responsible for selecting the remap_base address for a page and allocating regions for remapping targets to optimize the total memory capacity used for these remapping candidate regions and to minimize the likelihood of a hash collision.

H-idx cache. The H-idx cache is addressed by physical frame number (PFN). Though we illustrate it as a direct mapped cache in Figure 6, the cache can be fully- or set-associative. Each entry in the H-idx cache is composed of the tag (for the PFN), the remap_base address and the 64 H-idx's (one H-idx for each cache line). An H-idx cache entry is filled and evicted along with the processor's TLB so a 100% hit is guaranteed for reads. For write-backs, however, it is possible to miss the H-idx cache but it should be rare; a physical to ECC address translation architecture presented in [33] reports that the translation miss rate for write-backs is less than 0.05% in most memory-intensive applications.

In our evaluation (Section 4), we use 2 bits for each H-idx (n=2), hence, 24B per H-idx cache entry (8B for a remap_base and 16B for 64 H-idx's), and each remap region is 1MB. Although other configurations are possible, we found that a 2-bit H-idx is sufficient to identify more than 98% of the remapped blocks in most cases (see Figure 12(b)). To guarantee identifying all remapped blocks without fetching an embedded pointer, the H-idx cache should be at least as large as the TLB. Therefore, in the evaluation we use 512-set 4-way set-associative H-idx cache that requires 48kB data and 16kB for tag.

3.4. Detecting and correcting errors

The main objectives of our design are to implement error tolerance efficiently entirely within the memory controller and also to provide a mechanism for tolerating soft errors and device failures. We also restrict ourselves to a storage overhead of 12.5% or less to stay within best-practice constraints. We assume a DDR3-like interface with burst length of 8 and a 72-bit wide channel (64 bits for data and 8 bits for meta-data including D/P flag and ECC) as presented in Section 3.2. Hence, each 64B data block has 8 bytes of meta-data: 1 bit for the D/P flag and the remaining bits for ECC to tolerate wear-out and soft errors (chipkill support is discussed in Section 3.6). In an effort to extend the lifetime of a block as much as possible, we rely on ECC to tolerate initial wear-out failures. As we will explain later in the section, we remap blocks only when the number of errors exceeds 4-bits per cache line. We propose to use a 61-bit 6 bit-error correcting 7 bit-error detecting (6EC-7ED) BCH code. Note that the 6EC-7ED BCH code also protects the D/P flag.

The main design issue with the strong BCH code is its high complexity of *error correction* (encoding and detecting errors is straightforward to implement). We note that it takes a long time for a bit to wear out, but once it does the bit may always return an erroneous value (depending on a '0' or a '1'). Soft errors, on the

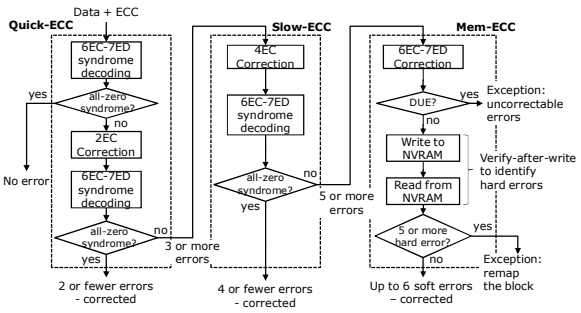

Figure 7. Error correction procedure using quick-ECC, slow-ECC, and mem-ECC. The illustrated quick- and slow-ECC are only examples, and we can design a better, and simpler, fast-path 2-bit/4-bit correction procedures.

other hand, are very rare. We utilize this asymmetry, and leverage the idea presented in Hi-ECC [28]. The key idea behind Hi-ECC is to employ low cost ECC for most blocks, which have little or no errors, and resort to high latency correction only for blocks with multi-bit errors. We extend this idea and implement three ECC logic paths: *quick-ECC*, *slow-ECC* and *mem-ECC*. Quick-ECC processes the 6EC-7ED BCH, detects up to 7 errors but corrects only up to 2 errors. Hence, it can handle 2 wear-out errors per block with no latency penalty. If quick-ECC identifies 3 bit errors or more, it invokes slow-ECC for correction. Slow-ECC can correct up to 4 bit errors and forwards the corrected data to the cache controller. If slow-ECC detects more than 4 bit errors, it invokes mem-ECC. Mem-ECC corrects up to 6 bit errors and also identifies which errors are wear-out errors by writing and re-reading data to/from NVRAM (verify-after-write through the memory controller). If a block has 5 or more worn-out bits, the memory controller raises an exception to the OS so that the OS can remap the failed block. Thus, mem-ECC will not be needed again to correct wear-out failures for that memory address. This procedure ensures that most blocks can be accessed with no penalty using quick-ECC, few blocks with slow-ECC, and in any case at least 2 soft bit errors can be corrected because remapping is triggered if more than 4 permanent errors exist. We estimate the latencies of quick-, slow-, and mem-ECC as 2, 4, and 183 memory cycles, where BCH coding/decoding latencies are from [26] and NVRAM parameters from [13, 10]. Note that the mem-ECC path latency is dominated by the verify-after-read operation: a row precharge (tRP: 120 cycles), a row activate (tRCD: 44 cycles), and a column read (tCL: 10 cycles). Mem-ECC is quite slow but is used extremely rarely, and will not impact overall performance. Mem-ECC is used only when several soft errors occur or once per block when wear-out accumulates to the point of remapping. Figure 7 depicts an example error correction procedure using quick-, slow-, and mem-ECC. The area overhead of the proposed tiered BCH encoder/decoder is only a fraction of the whole memory controller so we do not evaluate that in detail.

In summary, we can tolerate up to 4 worn-out bits per 64B block and up to 6 bit errors including soft and hard errors using the 6EC-7ED BCH code. Unlike prior research that uses coarse-grained remapping, FR maps out only the failed block and enhances storage efficiency. We leverage Hi-ECC to reduce the common case error correcting overheads. Importantly, FREE-p detects wear-out failures at the memory controller, obviating verify-after-write within NVRAM devices.

3.5. OS support

In the proposed FREE-p technique, the role of the OS is to allocate pages to map failed blocks and keep track of free spots in the allocated pages. Every time a block fails, the operating system will try to remap an empty spot in the remap page for the failed block. In the absence of empty slots, it allocates a fresh page from the free page list and adjusts the effective memory size available for workloads accordingly. With the index cache presented in Section 3.3.2, the role of the OS is more prominent. The OS must be aware of hashing functions and remapping regions used in the index cache. The OS is also responsible for managing H-idx information of physical pages; the cost is just 24B per 4kB page (less than 0.6%) for an H-idx of size 2 bits. A variation-aware OS mechanism suggested by Zhang and Li [34] may further improve the efficiency, but we leave this to future work.

3.6. Chipkill

Chipkill-correct is commonly required for today's high-end servers and datacenters. Storage overhead of chipkill is typically around 12%. Prior NVRAM reliability proposals incur 12% storage overhead to overcome just wear-out failures [10, 22]. Supporting chipkill-correct with FREE-p, on the other hand, is cost-effective. We illustrate this using the following example design. We base our design on a configuration that supports chipkill in DRAM. We propose a 144-bit wide NVRAM channel (32 ×4 NVRAM devices for data and 4 ×4 devices for meta-data). Assuming burst-8 (like DDR3), a data block size is 128B and the size of the meta-data is 16B. We use two bus transfers as a coding unit to construct an 8-bit symbol out of a ×4 NVRAM device, and apply a GF (2^8) based 3-check symbol error code with chipkill-correct capability [6].

Table 1. Evaluated wear-out failure tolerance mechanisms.

	Overhead	Coding block	Tolerable bit failures	Map-out unit
No Correction	0%	N/A	0	4kB
SEC64	12.5% (8bit)	64 bit	1	4kB
ECP6	11.9% (61 bit)	512 bit	6	4kB
FREE-p	12. 5% (64 bit)	512 bit	4	64B

We use the remaining 4B of meta-data for D/P flag and another three 8-bit check symbols to tolerate wear-out failures. Also, the NMR (N modular redundancy) coded embedded pointer should be laid out such that NMR coding can tolerate a device failure, which is easy to do. We leave detailed investigation of this FREE-p+Chipkill design as well as integration with more advanced schemes such as [27, 33] to future work.

4. Evaluation

In this section, we evaluate our proposed NVRAM memory system. We first compare FREE-p with ECP [22], the current state-of-the-art mechanism, in Section 4.1, and then quantify the performance impact of FREE-p in Section 4.2.

4.1. Capacity vs. lifetime

Evaluating how the capacity of an NVRAM memory system changes over time is challenging because it requires simulating very long-term behavior (years) in reasonable time (hours). Since detailed cycle-based simulation is impractical in this case, we adopt a simplified methodology, which was also used in prior research [10, 22]. We use 4kB physical pages and a 64B granularity for reads and writes. In each failure simulation, we lay out 2000 pages, [3] each of which has 4kB memory cells as well as meta-data (ECC, D/P flag, and error correcting entries). To account for variability, the initial endurance of each NVRAM cell is a random variable. Unless noted otherwise, we use a normal distribution with an average of 10^8 writes and a standard deviation of 0.25×10^8 (CoV: coefficient of variation is 0.25), as in prior work [22]. We assume uniform random memory access patterns and random data values resulting in a bit-transition probability of 0.5. In addition, we model perfect wear-leveling, which makes our result conservative. Imperfect wear-leveling will increase CoV due to uneven distribution of failures. As we show later in the section FREE-p performs relatively better with high CoVs. We run failure simulations by applying writes to random locations while monitoring the remaining NVRAM cell lifetimes. Bit-error correction and coarse-grained/fine-grained remapping are accurately modeled. We use a constant write-back rate of 66.67M write-backs per second (1/3 peak data rate of a 12.8GB/s DDR3-like channel) for the failure simulations. The lifetime of NVRAM presented later in the section varies linearly with the write-back rate. Any deviation from our assumed data rate will still preserve the relative lifetime improvements between various designs.

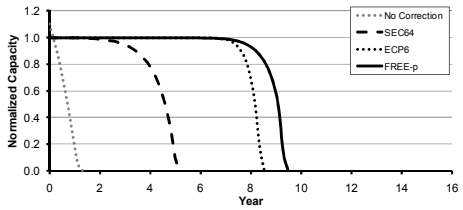

Figure 8. Capacity vs. lifetime (CoV=0.25).

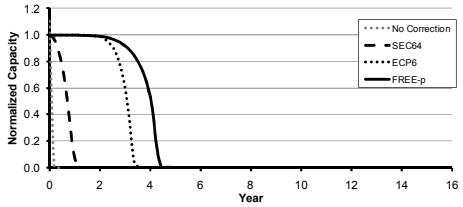

Figure 9. Capacity vs. lifetime (CoV=0.35).

Table 1 describes the configurations we use for these lifetime failure simulations. *No correction* [4] cannot tolerate any errors, and *SEC64* implements an 8-bit SEC-DED code per 64-bit data. Both schemes assume verify-after-write for wear-out detection and coarse-grained (4kB) remapping. *ECP6* [22] implements a 6-bit error correcting ECP within an NVRAM device, and uses coarse-grained remapping [5]. ECP6 uses verify-after-write for error detection; hence, a 7th failure within a 64B block triggers a page map-out.

Figure 8 compares capacity vs. lifetime of No-correction, SEC64, ECP6, and FREE-p. Both ECP6 and FREE-p show a much longer lifetime than the simpler mechanisms (No Correction and SEC64). FREE-p offers a 7.5% longer lifetime than ECP6 at 90% capacity and 11.5% longer at 50% capacity. This increase in lifetime is due to the flexibility provided by the fine-grained remapping compared to ECP6's coarse-grained approach. More importantly, FREE-p uses simple commodity NVRAM, can correct soft errors, and can provide chipkill protection without incurring any additional storage overhead (Section 3.6).

[3] When estimating the expected lifetime of a page (around 8 years) using the failure simulation, 2000 pages yield a confidence interval of ±10 days, at most, at 95% confidence level and ±15 days at 99%.

[4] No Correction uses 2240 pages, 12.5% more pages than other schemes, to account for 0% overhead.

[5] ECP can also be implemented at multiple levels – local ECP to correct errors in a block and a global ECP to correct bit errors at page level (layered ECP). This incurs non-deterministic access latency; hence, we do not consider it here.

Table 2. Simulated system parameters.

Processor core	4GHz in-order x86 core
L1 cache	Split I/D caches, each 32kB, 64B cache line 1-cycle latency
L2 cache	Unified 1MB, 64B cache line, 4-cycle latency
Memory controller	32 entry scheduling window FR-FCFS [20] with open-page policy XOR based bank interleaving [35]
NVRAM	Similar to DDR3-1600 channel 64-bit data and 8-bit meta-data (D/P flag and ECC)
NVRAM timing in cycles at 800MHz	tRCD: 44 cycles, tRP(dirty): 120 cycles tRP(clean): 10 cycles, tRR(dirty): 22 cycles tRR(clean): 3 cycles, tCL: 10 cycles, tBL: 4 cycles
Benchmarks	FFT, Radix, Ocean, Raytrace, Facesim, Canneal, and Streamcluster

Table 3. Summary of configurations in the evaluation.

FREE-p:Baseline	The baseline FREE-p system
FREE-p:Cache	FREE-p with a remap pointer cache presented in Section 3.3.1.
FREE-p:IndexCache	FREE-p with the index cache presented in Section 3.3.2.
FREE-p:PerfectCache+Locality	FREE-p with a perfect cache with spatial locality aware remapping

We also evaluate sensitivity to process variation and imperfect wear-leveling. At very low process variation and with perfect wear-leveling (CoV=0.15), all schemes show a lifetime longer than 8 years so we do not show the result here. As CoV increases, however, FREE-p achieves 21% longer lifetime than ECP6 at 90% capacity and 26% longer at 50% capacity, as shown in Figure 9(b).

4.2. Performance impact

We evaluate the performance impact of FREE-p using a cycle-level simulator with an in-order core processing up to 1 instruction and 1 memory access per cycle [2]. The simulator has detailed models of L1, L2, and main memory, including memory controllers, memory bank conflicts, and bus contention. We augmented the simulator to support an NVRAM device with a DDR3-like interface, and we used PCRAM timing parameters from prior work [13, 10]. Table 2 summarizes the system parameters used. We ran each application until 200 million memory instructions are processed. We use a mix of applications from the SPLASH 2 [30] and PARSEC [4] benchmark suites. We selected only applications that stress the memory system and that are potentially significantly impacted by the pointer-chasing operations of FR (Table 2). Other non-memory-intensive applications will not be affected adversely by FREE-p.

Configurations for the evaluation. We use the configuration presented in Section 3; FREE-p implements FR with a 6EC-7ED BCH code where a fifth hard bit failure remaps a block. We compare the

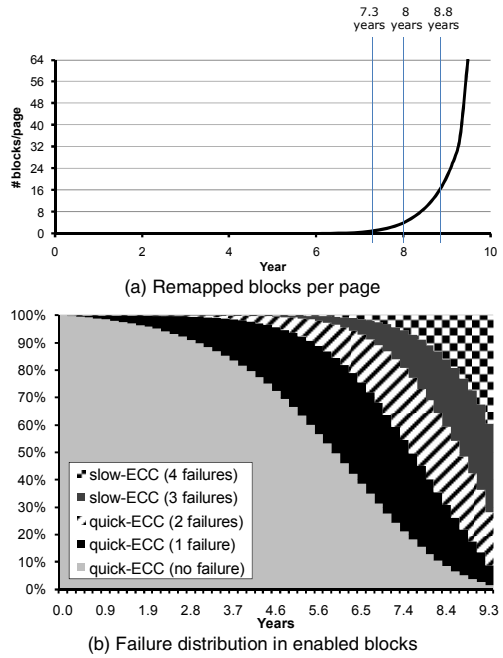

Figure 10. The number of remapped blocks per page and failures per block (CoV=0.25).

performance impact of FREE-p to an ideal memory system without wear-out failures.

ECP [22] has very little impact on performance but the true cost is the complexity and specialization of the NVRAM devices to include internal error-correcting logic and redundant storage, for which users pay the price throughout the entire system life, even when no fault exists. Hence, we do not directly compare the performance of FREE-p with that of ECP, and use ideal memory instead.

Table 3 summarizes the configurations used in the performance evaluation, including the optimization techniques presented in Section 3.3.1 (FREE-p:Cache) and Section 3.3.2 (FREE-p:IndexCache). We also evaluate an idealized implementation, FREE-p:PerfectCache+Locality; the memory controller has perfect knowledge of remapping so embedded pointers are never fetched. In this configuration, the remapped data blocks are allocated such that spatial locality is preserved, that is, all the blocks within a memory page are remapped to a single alternative memory page. This allows the memory controller to exploit row-buffer locality among remapped blocks. In all other FREE-p configurations, we assume blocks are remapped to random locations (or based on the hashing functions in FREE-p:IndexCache), potentially degrading spatial locality. We accurately model the performance impact of this loss of spatial locality.

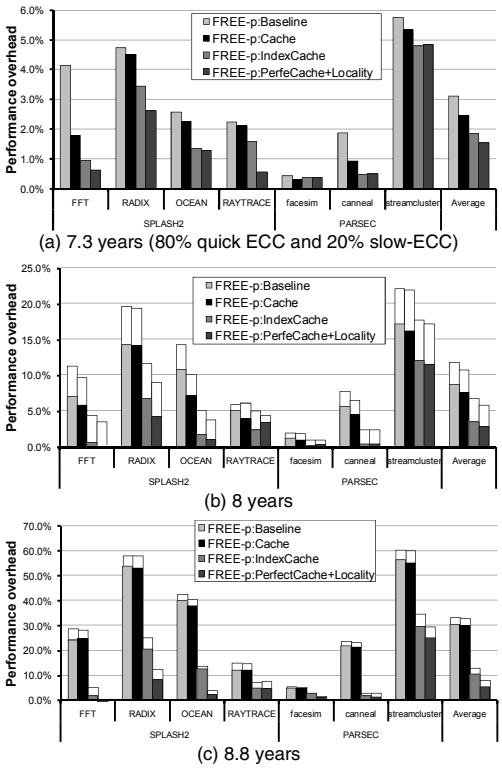

Figure 11. Performance overhead of the FREE-p schemes (CoV=0.25). Stacked white bars in (b) and (c) represent the additional overhead due to slow-ECC.

Fault injection. In order to measure the performance impact with failures, we use random failure injection to physical memory. At the beginning of each simulation, failures are injected with a certain probability (e.g., X failures per 4kB page on average), assuming no new wear-out failures occur during the relatively short program execution. The probability of injected faults is based on the capacity/lifetime simulations presented in Figure 8. Figure 10(a) shows the average number of blocks per page that need multiple accesses due to remappings. As shown, the performance impact in the first 7 years is negligible, and only 1 block per page is remapped on average even after 7.3 years. After 8 and 8.8 years, the number of remapped blocks grows to 4 and 16 respectively. We also present the distribution of failed bits per block in Figure 10(b), which shows that quick-ECC (2 or fewer failure bits) dominates. Even after 8 years, about 70% of active blocks use quick-ECC.

Performance impact. Figure 11 presents the performance impact of the FREE-p schemes. FREE-p's performance depends on the number of remapped blocks as well as the frequency of using quick-ECC. For the case of 7.3 years, when nearly all accesses use

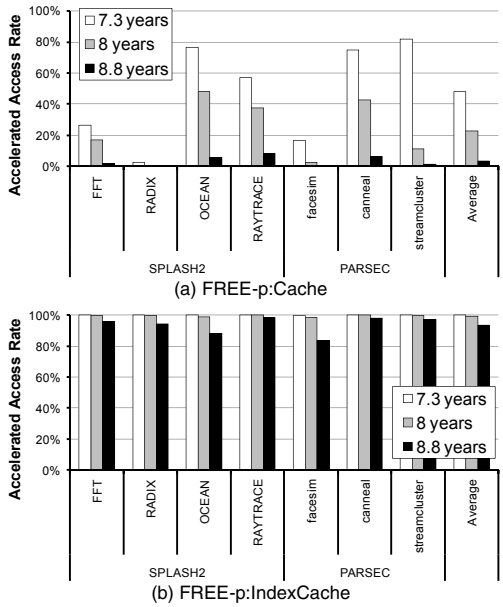

Figure 12. Accelerated access rate (AAR).

quick-ECC, we randomly use slow-ECC in 20% of accesses. For the case of 8 and 8.8 years, we bound the performance impact of using slow-ECC, by showing execution times of both quick-ECC only and slow-ECC only.

After 7.3 years, when FREE-p remaps one block per page on average, FREE-p:Baseline has a very small impact on performance; only 3.1% on average. FREE-p:Cache and FREE-p:IndexCache reduce this further to 2.5% and 1.8% respectively. After 8 years, as the number of failed blocks per page increases to four, FREE-p:Cache performs poorly, similar to the baseline, and incurs more than 10% overhead. FREE-p:IndexCache is very effective at mitigating the remapping penalty, degrading performance by only 3.5% with quick-ECC only and 6.7% with slow-ECC only (we expect real performance to fall in between these two numbers, likely closer to 3.5%). With further increase in errors (as shown in Figure 11(c), FREE-p:IndexCache is critical to avoid significant drop in performance. On average, FREE-p:IndexCache incurs a 13% drop in performance (with slow-ECC) compared to 31% drop in baseline and FREE-p:Cache.

Accelerated Access. We further investigate the effectiveness of the simple remap pointer cache and the index cache presented in Section 3.3. We define the *accelerated access rate* (AAR) as the percentage of accesses to the remapped blocks for which embedded pointer is readily available. The AAR of the simple cache (FREE-p:Cache) depends on its hit-rate. On the other hand, the index cache (FREE-p:IndexCache) can fail only when all remapping candidates are already used or because of rare hash collisions. The simple

cache works well only until 7.3 years and its AAR becomes very low after that (Figure 12(a)). The index cache, however, retains very high AAR even after 8.8 years (Figure 12(b)). Even with 16 blocks remapped per page, the AAR of the index cache is still 93% on average.

5. Conclusions and future work

In this paper, we presented a general NVRAM reliability mechanism that can tolerate both wear-out and soft errors and that can be extended to support chipkill-correct. We showed a low-cost wear-out tolerance mechanism, fine-grained remapping (FR), and described FREE-p that integrates FR and ECC. Our scheme, unlike prior work, implements all the necessary functionality including error detecting and correcting at the memory controller, leaving NVRAM devices as simple and cheap as possible. This also provides end-to-end reliability that protects not only memory cells but also wires, packages, and periphery circuits. Overall, FREE-p increases lifetime by up to 26%, and incurs less than 2% performance overhead for the initial 7 years and less than 10% even near end of life.

In future work, we will explore the optimal partitioning of functionality between the memory controller and NVRAM devices, adapting hard error tolerance levels to process variation, the design space of the index cache architecture, and a detailed design of more aggressive chipkill solutions.

6. Acknowledgments

This work is supported, in part, by the following organizations: NVIDIA Corp., The National Science Foundation under Grant #0954107.

7. References

[1] ITRS 2008 update. Tech. report, Int'l Tech. Roadmap for Semiconductors, 2008.
[2] J. H. Ahn, et al. Future scaling of processor-memory interfaces. In *SC*, 2009.
[3] AMD. BIOS and kernel developer's guide for AMD NPT family 0Fh processors, 2007.
[4] C. Bienia, et al. The PARSEC benchmark suite: Characterization and architectural implications. Tech. Report TR-811-08, Princeton Univ., 2008.
[5] G. W. Burr, et al. Phase change memory technology. *J. Vacuum Science and Tech. B*, 28 (2): 223–262, 2010.
[6] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM J. Res. and Dev.*, 28 (2): 124–134, 1984.
[7] S. Cho and H. Lee. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *MICRO*, 2009.
[8] T. J. Dell. System RAS implications of DRAM soft errors. *IBM J. Res. and Dev.*, 52 (3): 307–314, 2008.
[9] P. Friedberg, et al. Modeling within-die spatial correlation effects for process-design co-optimization. In *ISQED*, 2005.
[10] E. Ipek, et al. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *ASPLOS*, 2010.
[11] Y. Joo, et al. Energy- and endurance-aware design of phase change memory caches. In *DATE*, 2010.
[12] D.-H. Kang, et al. Two-bit cell operation in diode-switch phase change memory cells with 90nm technology. In *Symp. VLSI Tech.*, 2008.
[13] B. C. Lee, et al. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 2009.
[14] MICRON. *NAND Flash Translation Layer (NFTL) 4.5.0 User Guide*, 2010.
[15] J. A. Nerl, et al. System and method for controlling application of an error correction code. In *U.S. Patent*, #7,308,638, 2007.
[16] Numonyx. *128-Mbit Parallel Phase Change Memory*, 2010.
[17] A. Pirovano, et al. Reliability study of phase-change nonvolatile memories. *IEEE Tran. Device and Materials Reliability*, 4 (3): 422–427, 2004.
[18] M. K. Qureshi, et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO*, 2009.
[19] M. K. Qureshi, et al. Scalable high-performance main memory system using phase-change memory technology. In *ISCA*, 2009.
[20] S. Rixner, et al. Memory access scheduling. In *ISCA*, 2000.
[21] D. Roberts, et al. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *DSD*, 2007.
[22] S. Schechter, et al. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA*, 2010.
[23] B. Schroeder, et al. DRAM errors in the wild: A large-scale field study. In *SIGMETRICS*, 2009.
[24] N. H. Seong, et al. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA*, 2010.
[25] N. H. Seong, et al. SAFER: Stuck-at-fault error recovery for memories. In *MICRO*, 2010.
[26] D. Strukov. The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories. In *Asilomar Conf.*, 2006.
[27] A. Udipi, et al. Rethinking DRAM design and organization for energy-constrained multi-cores. In *ISCA*, 2010.
[28] C. Wilkerson, et al. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA*, 2010.
[29] C. Wilkerson, et al. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA*, 2008.
[30] S. C. Woo, et al. The SPLASH-2 programs: Characterization and methodological considerations. In *ISCA*, 1995.
[31] B.-D. Yang, et al. A low power phase-change random access memory using a data-comparison write scheme. In *ISCAS*, 2007.
[32] Y. Yokoyama, et al. A 1.8-V embedded 18-Mb DRAM macro with a 9-ns RAS access time and memory cell efficiency of 33%. In *CICC*, 2000.
[33] D. H. Yoon and M. Erez. Virtualized and flexible ECC for main memory. In *ASPLOS*, 2010.
[34] W. Zhang and T. Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO,* 2009.
[35] Z. Zhang, et al. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *MICRO*, 2000.
[36] P. Zhou, et al. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.